Problem 1: Patient Information Management System
Description: Create a menu-driven program to manage patient information, including basic details, medical history, and current medications.
Menu Options:
Add New Patient
View Patient Details
Update Patient Information
Delete Patient Record
List All Patients
Exit
Requirements:
Use variables to store patient details.
Utilize static and const for immutable data such as hospital name.
Implement switch case for menu selection.
Employ loops for iterative tasks like listing patients.
Use pointers for dynamic memory allocation.
Implement functions for CRUD operations.
Utilize arrays for storing multiple patient records.
Use structures for organizing patient data.
Apply nested structures for detailed medical history.
Use unions for optional data fields.
Employ nested unions for multi-type data entries.

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

// Define a const hospital name
const char HOSPITAL_NAME[] = "City HealthCare";

// Structure for medical history
struct MedicalHistory {
    char diagnosis[100];
    char treatment[100];
};

// Structure for patient data
struct Patient {
    int id;
    char name[50];
    int age;
    char gender[10];
    struct MedicalHistory history;
    char currentMedications[100];
    union {
        char additionalInfo[100];
        int allergyCount;
    } optionalData;
};

// Maximum number of patients
#define MAX_PATIENTS 10
static struct Patient *patients[MAX_PATIENTS];
static int patientCount = 0;

void addNewPatient();
void viewPatientDetails();
void updatePatientInformation();
void deletePatientRecord();
void listAllPatients();
void menu();

int main() {
    printf("Welcome to %s\n", HOSPITAL_NAME);
    menu();
    return 0;
}

void menu() {
    int choice;
    do {
        printf("\n--- Patient Information Management System ---\n");
        printf("1. Add New Patient\n");
        printf("2. View Patient Details\n");
        printf("3. Update Patient Information\n");
        printf("4. Delete Patient Record\n");
        printf("5. List All Patients\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: addNewPatient(); break;
            case 2: viewPatientDetails(); break;
            case 3: updatePatientInformation(); break;
            case 4: deletePatientRecord(); break;
            case 5: listAllPatients(); break;
            case 6: printf("Exiting...\n"); break;
            default: printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);
}

void addNewPatient() {
    if (patientCount >= MAX_PATIENTS) {
        printf("Patient limit reached!\n");
        return;
    }
    struct Patient *newPatient = (struct Patient *)malloc(sizeof(struct Patient));
    printf("Enter Patient ID: ");
    scanf("%d", &newPatient->id);
    printf("Enter Name: ");
    scanf(" %[^\n]", newPatient->name);
    printf("Enter Age: ");
    scanf("%d", &newPatient->age);
    printf("Enter Gender: ");
    scanf(" %[^\n]", newPatient->gender);
    printf("Enter Medical Diagnosis: ");
    scanf(" %[^\n]", newPatient->history.diagnosis);
    printf("Enter Treatment: ");
    scanf(" %[^\n]", newPatient->history.treatment);
    printf("Enter Current Medications: ");
    scanf(" %[^\n]", newPatient->currentMedications);

    // Union for additional optional data
    printf("Enter any additional information or number of allergies: ");
    scanf(" %[^\n]", newPatient->optionalData.additionalInfo);

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

```
void viewPatientDetails() {
    int id, found = 0;
    printf("Enter Patient ID to view details: ");
    scanf("%d", &id);

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            found = 1;
            printf("\nPatient ID: %d\n", patients[i]->id);
            printf("Name: %s\n", patients[i]->name);
            printf("Age: %d\n", patients[i]->age);
            printf("Gender: %s\n", patients[i]->gender);
            printf("Diagnosis: %s\n", patients[i]->history.diagnosis);
            printf("Treatment: %s\n", patients[i]->history.treatment);
            printf("Current Medications: %s\n", patients[i]->currentMedications);
            printf("Additional Info: %s\n", patients[i]->optionalData.additionalInfo);
            break;
        }
    }
    if (!found) printf("Patient not found!\n");
}

void updatePatientInformation() {
    int id, found = 0;
    printf("Enter Patient ID to update details: ");
    scanf("%d", &id);

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            found = 1;
            printf("Enter updated Name: ");
            scanf(" %[^\n]", patients[i]->name);
            printf("Enter updated Age: ");
            scanf("%d", &patients[i]->age);
            printf("Enter updated Gender: ");
            scanf(" %[^\n]", patients[i]->gender);
            printf("Enter updated Diagnosis: ");
            scanf(" %[^\n]", patients[i]->history.diagnosis);
            printf("Enter updated Treatment: ");
            scanf(" %[^\n]", patients[i]->history.treatment);
            printf("Enter updated Current Medications: ");
            scanf(" %[^\n]", patients[i]->currentMedications);
            printf("Enter updated Additional Info: ");
            scanf(" %[^\n]", patients[i]->optionalData.additionalInfo);
            printf("Patient information updated successfully!\n");
            break;
        }
    }
    if (!found) printf("Patient not found!\n");
}

void deletePatientRecord() {
    int id, found = 0;
    printf("Enter Patient ID to delete record: ");
    scanf("%d", &id);

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            found = 1;
            free(patients[i]);
            for (int j = i; j < patientCount - 1; j++) {
                patients[j] = patients[j + 1];
            }
            patientCount--;
            printf("Patient record deleted successfully!\n");
            break;
        }
    }
    if (!found) printf("Patient not found!\n");
}

void listAllPatients() {
    if (patientCount == 0) {
        printf("No patients found!\n");
        return;
    }
    printf("\n--- List of All Patients ---\n");
    for (int i = 0; i < patientCount; i++) {
        printf("ID: %d, Name: %s, Age: %d, Gender: %s\n",
               patients[i]->id, patients[i]->name, patients[i]->age, patients[i]->gender);
    }
}
```

Problem 2: Hospital Inventory Management
Description: Design a system to manage the inventory of medical supplies.
Menu Options:
Add Inventory Item
View Inventory Item
Update Inventory Item
Delete Inventory Item
List All Inventory Items
Exit
Requirements:
Declare variables for inventory details.
Use static and const for fixed supply details.
Implement switch case for different operations like adding, deleting, and viewing inventory.
Utilize loops for repetitive inventory checks.
Use pointers to handle inventory records.
Create functions for managing inventory.
Use arrays to store inventory items.
Define structures for each supply item.
Use nested structures for detailed item specifications.
Employ unions for variable item attributes.
Implement nested unions for complex item data types.

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

#define MAX_SUPPLY 10


struct itemspecification
{
  char category[50];
  char expiryDate[50];
};
struct Supply{
    int id;
    char name[50];
    float price;
    struct itemspecification specification;
    union{
        char additionalInfo[100];
    } optionalAttributes;
};


static struct Supply *supply[MAX_SUPPLY];
static int total=0;

void addItem();
void view();
void update();
void deletess();
void Listall();

int main(){
    int choice;

    while (1)
    {
        printf("\n--- Hospital Inventory Management System ---\n");
        printf("1. Add Inventory Item\n");
        printf("2. View Inventory Item\n");
        printf("3. Update Inventory Item\n");
        printf("4. Delete Inventory Item\n");
        printf("5. List All Inventory Items\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                addItem();
                break;
            case 2:
                view();
                break;
            case 3:
                update();
                break;
            case 4:
                deletess();
                break;
            case 5:
                Listall();
                break;
            case 6:
                printf("Exiting..........\n");
                return 0;
                break;
            default:
                printf("Invalid Choice \n");

        }

    }
    return 0;
}

void addItem(){
    if(total>=MAX_SUPPLY){
        printf("Limit reached cannot add \n");
        return;
    }
     struct Supply *newItem = (struct Supply *)malloc(sizeof(struct Supply));

    printf("Enter Item ID: ");
    scanf("%d", &newItem->id);
    getchar();
    printf("Enter Item Name: ");
    fgets(newItem->name, sizeof(newItem->name), stdin);
    newItem->name[strcspn(newItem->name, "\n")] = '\0';
    printf("Enter Item Price: ");
    scanf("%f", &newItem->price);
    getchar();
    printf("Enter Item Category: ");
    fgets(newItem->specification.category, sizeof(newItem->specification.category),
stdin);
```

```
    newItem->specification.category[strcspn(newItem->specification.category, "\n")] =
'\0'; // Remove newline character
    printf("Enter Expiry Date (DD/MM/YYYY): ");
    fgets(newItem->specification.expiryDate, sizeof(newItem->specification.expiryDate),
stdin);
    newItem->specification.expiryDate[strcspn(newItem->specification.expiryDate, "\n")]
= '\0'; // Remove newline character
    printf("Enter Additional Info: ");
    fgets(newItem->optionalAttributes.additionalInfo,
sizeof(newItem->optionalAttributes.additionalInfo), stdin);

newItem->optionalAttributes.additionalInfo[strcspn(newItem->optionalAttributes.additio
nalInfo, "\n")] = '\0'; // Remove newline character

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

void view() {
    int id, found = 0;
    printf("Enter Item ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (supply[i]->id == id) {
            found = 1;
            printf("\nItem ID: %d\n", supply[i]->id);
            printf("Name: %s\n", supply[i]->name);
            printf("Price: %.2f\n", supply[i]->price);
            printf("Category: %s\n", supply[i]->specification.category);
            printf("Expiry Date: %s\n", supply[i]->specification.expiryDate);
            printf("Additional Info: %s\n", supply[i]->optionalAttributes.additionalInfo);
            break;
        }
    }
    if (!found) {
        printf("Item not found!\n");
    }
}
void update(){
    int id, found=0;
    printf("Enter Item ID to update\n");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(supply[i]->id ==id){
            found =1;
            printf("Updating Item ID: %d\n", id);
            printf("Enter new Price: ");
            scanf("%f", &supply[i]->price);
            printf("Enter new Category: ");
            getchar();
            fgets(supply[i]->specification.category, sizeof(supply[i]->specification.category),
stdin);
            supply[i]->specification.category[strcspn(supply[i]->specification.category, "\n")]
= '\0';
            printf("Enter new Expiry Date: ");
            fgets(supply[i]->specification.expiryDate,
sizeof(supply[i]->specification.expiryDate), stdin);
            supply[i]->specification.expiryDate[strcspn(supply[i]->specification.expiryDate,
"\n")] = '\0';
            printf("Item updated successfully!\n");
            break;
        }
    }
     if (!found) {
        printf("Item not found!\n");
    }
}

void deletess() {
    int id, found = 0;
    printf("Enter Item ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (supply[i]->id == id) {
            found = 1;
            free(supply[i]);
            for (int j = i; j < total - 1; j++) {
                supply[j] = supply[j + 1];
            }
            total--;
            printf("Item deleted successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Item not found!\n");
    }
}


void Listall() {
    if (total == 0) {
        printf("No inventory items available.\n");
        return;
    }

    printf("\n--- Inventory List ---\n");
    for (int i = 0; i < total; i++) {
        printf("\nItem ID: %d\n", supply[i]->id);
        printf("Name: %s\n", supply[i]->name);
        printf("Price: %.2f\n", supply[i]->price);
        printf("Category: %s\n", supply[i]->specification.category);
        printf("Expiry Date: %s\n", supply[i]->specification.expiryDate);
        printf("Additional Info: %s\n", supply[i]->optionalAttributes.additionalInfo);
    }
}
```

Problem 3: Medical Appointment Scheduling System
Description: Develop a system to manage patient appointments.
Menu Options:
Schedule Appointment
View Appointment
Update Appointment
Cancel Appointment
List All Appointments
Exit
Requirements:
Use variables for appointment details.
Apply static and const for non-changing data like clinic hours.
Implement switch case for appointment operations.
Utilize loops for scheduling.

Use pointers for dynamic data manipulation.
Create functions for appointment handling.
Use arrays for storing appointments.
Define structures for appointment details.
Employ nested structures for detailed doctor and patient information.
Utilize unions for optional appointment data.
Apply nested unions for complex appointment data.

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

#define MAX 10
const char CLINIC_HOURS[] = "8:00 AM - 4:00 PM";

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

struct Patient {
    char name[50];
    int age;
    char contact[50];
    char gender[50];
};

struct Appointment {
    int id;
    char date[50];
    char time[50];
    struct Doctor doctor;
    struct Patient patient;
    union {
        char additionalInfo[100];
    } optional;
};

static struct Appointment *appointment[MAX];
static int total = 0;
```

```
void schedule();
void viewApp();
void updateApp();
void cancelApp();
void ListallApp();

int main() {
    int choice;
    while (1) {
        printf("\n--- Medical Appointment Scheduling System ---\n");
        printf("Clinic Hours: %s\n", CLINIC_HOURS);
        printf("1. Schedule Appointment\n");
        printf("2. View Appointment\n");
        printf("3. Update Appointment\n");
        printf("4. Cancel Appointment\n");
        printf("5. List All Appointments\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                schedule();
                break;
            case 2:
                viewApp();
                break;
            case 3:
                updateApp();
                break;
            case 4:
                cancelApp();
                break;
            case 5:
                ListallApp();
                break;
            case 6:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}

void schedule() {
    if (total >= MAX) {
        printf("Appointments limit reached. Cannot schedule more.\n");
        return;
    }

    struct Appointment *newappointment = (struct Appointment *)malloc(sizeof(struct
Appointment));
    newappointment->id = total + 1;

    getchar(); // Clear input buffer
    printf("Enter Doctor's Name: ");
    fgets(newappointment->doctor.name, sizeof(newappointment->doctor.name), stdin);
    newappointment->doctor.name[strcspn(newappointment->doctor.name, "\n")] = '\0';

    printf("Enter Doctor's Specialization: ");
    fgets(newappointment->doctor.specialization,
sizeof(newappointment->doctor.specialization), stdin);

newappointment->doctor.specialization[strcspn(newappointment->doctor.specialization,
"\n")] = '\0';

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

    printf("Enter Patient's Age: ");
    scanf("%d", &newappointment->patient.age);
    getchar(); // Clear input buffer

    printf("Enter Patient's Contact: ");
    fgets(newappointment->patient.contact, sizeof(newappointment->patient.contact),
stdin);
    newappointment->patient.contact[strcspn(newappointment->patient.contact, "\n")] =
'\0';
```

```
    printf("Enter Patient's Gender: ");
    fgets(newappointment->patient.gender, sizeof(newappointment->patient.gender),
stdin);
    newappointment->patient.gender[strcspn(newappointment->patient.gender, "\n")] =
'\0';

    printf("Enter Appointment Date (DD/MM/YYYY): ");
    fgets(newappointment->date, sizeof(newappointment->date), stdin);
    newappointment->date[strcspn(newappointment->date, "\n")] = '\0';

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

    printf("Enter Additional Notes (if any): ");
    fgets(newappointment->optional.additionalInfo,
sizeof(newappointment->optional.additionalInfo), stdin);

newappointment->optional.additionalInfo[strcspn(newappointment->optional.additionalI
nfo, "\n")] = '\0';

    appointment[total++] = newappointment;
    printf("Appointment scheduled successfully!\n");
}

void viewApp() {
    int id, found = 0;
    printf("Enter Appointment ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (appointment[i]->id == id) {
            found = 1;
            printf("Appointment ID: %d\n", appointment[i]->id);
            printf("Doctor: %s\n", appointment[i]->doctor.name);
            printf("Specialization: %s\n", appointment[i]->doctor.specialization);
            printf("Patient Name: %s\n", appointment[i]->patient.name);
            printf("Age: %d, Gender: %s\n", appointment[i]->patient.age,
appointment[i]->patient.gender);
            printf("Date: %s\n", appointment[i]->date);
            printf("Time: %s\n", appointment[i]->time);
            printf("Additional Notes: %s\n", appointment[i]->optional.additionalInfo);
            break;
        }
    }
    if (!found) {
        printf("Appointment ID not found!\n");
    }
}

void updateApp() {
    int id, found = 0;
    printf("Enter Appointment ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (appointment[i]->id == id) {
            found = 1;
            printf("Updating Appointment ID: %d\n", id);
            printf("Enter new Appointment Date (DD/MM/YYYY): ");
            getchar();
            fgets(appointment[i]->date, sizeof(appointment[i]->date), stdin);
            appointment[i]->date[strcspn(appointment[i]->date, "\n")] = '\0';

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

            printf("Enter new Additional Notes: ");
            fgets(appointment[i]->optional.additionalInfo,
sizeof(appointment[i]->optional.additionalInfo), stdin);

appointment[i]->optional.additionalInfo[strcspn(appointment[i]->optional.additionalInfo,
"\n")] = '\0';

            printf("Appointment updated successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Appointment not found!\n");
    }
```

```
}

void cancelApp() {
    int id, found = 0;
    printf("Enter Appointment ID to cancel: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (appointment[i]->id == id) {
            found = 1;
            free(appointment[i]);
            for (int j = i; j < total - 1; j++) {
                appointment[j] = appointment[j + 1];
            }
            total--;
            printf("Appointment cancelled successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Appointment ID not found!\n");
    }
}

void ListallApp() {
    if (total == 0) {
        printf("No appointments scheduled.\n");
        return;
    }

    for (int i = 0; i < total; i++) {
        printf("Appointment ID: %d\n", appointment[i]->id);
        printf("Doctor: %s (%s)\n", appointment[i]->doctor.name,
appointment[i]->doctor.specialization);
        printf("Patient: %s (Age: %d, Gender: %s)\n", appointment[i]->patient.name,
appointment[i]->patient.age, appointment[i]->patient.gender);
        printf("Date: %s, Time: %s\n", appointment[i]->date, appointment[i]->time);
        printf("Additional Notes: %s\n", appointment[i]->optional.additionalInfo);
        printf("----------------------\n");
    }
}
```

Problem 4: Patient Billing System
Description: Create a billing system for patients.
Menu Options:
Generate Bill
View Bill
Update Bill
Delete Bill
List All Bills
Exit
Requirements:
Declare variables for billing information.
Use static and const for fixed billing rates.
Implement switch case for billing operations.
Utilize loops for generating bills.
Use pointers for bill calculations.
Create functions for billing processes.
Use arrays for storing billing records.
Define structures for billing components.
Employ nested structures for detailed billing breakdown.
Use unions for variable billing elements.
Apply nested unions for complex billing scenarios.

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

#define MAX_BILLS 100
const double CONSULTATION_FEE = 500.0;
const double MEDICATION_FEE = 150.0;
const double LAB_TEST_FEE = 1000.0;

struct Patient {
   char name[50];
   int age;
   char contact[50];
};

struct BillingDetails {
   double consultationFee;
```

```
    double medicationFee;
    double labTestFee;
    union {
        double additionalCharges;
    } optional;
};

struct Bill {
    int id;
    struct Patient patient;
    struct BillingDetails details;
    double totalAmount;
};

static struct Bill *bills[MAX_BILLS];
static int totalBills = 0;

void generateBill();
void viewBill();
void updateBill();
void deleteBill();
void listAllBills();

int main() {
    int choice;
    while (1) {
        printf("\n--- Patient Billing System ---\n");
        printf("1. Generate Bill\n");
        printf("2. View Bill\n");
        printf("3. Update Bill\n");
        printf("4. Delete Bill\n");
        printf("5. List All Bills\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            viewBill();
            break;
        case 3:
            updateBill();
            break;
        case 4:
            deleteBill();
            break;
        case 5:
            listAllBills();
            break;
        case 6:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
void generateBill() {
    if (totalBills >= MAX_BILLS) {
        printf("Cannot generate more bills. Limit reached.\n");
        return;
    }

    struct Bill *newBill = (struct Bill *)malloc(sizeof(struct Bill));
    newBill->id = totalBills + 1;
    getchar(); // Clear input buffer before reading strings.

    printf("Enter Patient's Name: ");
    fgets(newBill->patient.name, sizeof(newBill->patient.name), stdin);
    newBill->patient.name[strcspn(newBill->patient.name, "\n")] = '\0'; // Remove newline
character.

    printf("Enter Patient's Age: ");
    while (scanf("%d", &newBill->patient.age) != 1 || newBill->patient.age <= 0) {
        printf("Invalid age. Please enter a valid number: ");
        while (getchar() != '\n'); // Clear input buffer.
    }
    getchar(); // Clear input buffer after reading the integer.
    printf("Enter Patient's Contact: ");
    fgets(newBill->patient.contact, sizeof(newBill->patient.contact), stdin);
    newBill->patient.contact[strcspn(newBill->patient.contact, "\n")] = '\0';

    printf("Enter Additional Charges (if any): ");
    while (scanf("%lf", &newBill->details.optional.additionalCharges) != 1) {
        printf("Invalid input. Please enter a valid amount: ");
        while (getchar() != '\n'); // Clear input buffer.
    }

    newBill->details.consultationFee = CONSULTATION_FEE;
    newBill->details.medicationFee = MEDICATION_FEE;
    newBill->details.labTestFee = LAB_TEST_FEE;

    newBill->totalAmount = newBill->details.consultationFee +
                           newBill->details.medicationFee +
                           newBill->details.labTestFee +
                           newBill->details.optional.additionalCharges;

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

void viewBill() {
    int id, found = 0;
    printf("Enter Bill ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < totalBills; i++) {
        if (bills[i]->id == id) {
            found = 1;
            printf("\n--- Bill Details ---\n");
            printf("Bill ID: %d\n", bills[i]->id);
            printf("Patient Name: %s\n", bills[i]->patient.name);
            printf("Patient Age: %d\n", bills[i]->patient.age);
            printf("Patient Contact: %s\n", bills[i]->patient.contact);
            printf("Consultation Fee: %.2f\n", bills[i]->details.consultationFee);
            printf("Medication Fee: %.2f\n", bills[i]->details.medicationFee);
            printf("Lab Test Fee: %.2f\n", bills[i]->details.labTestFee);
            printf("Additional Charges: %.2f\n", bills[i]->details.optional.additionalCharges);
            printf("Total Amount: %.2f\n", bills[i]->totalAmount);
            break;
        }
    }
    if (!found) {
        printf("Bill ID not found.\n");
    }
}

void updateBill() {
    int id, found = 0;
    printf("Enter Bill ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < totalBills; i++) {
        if (bills[i]->id == id) {
            found = 1;
            printf("Updating Bill ID: %d\n", id);
            printf("Enter new Additional Charges: ");
            scanf("%lf", &bills[i]->details.optional.additionalCharges);

            bills[i]->totalAmount = bills[i]->details.consultationFee +
                                    bills[i]->details.medicationFee +
                                    bills[i]->details.labTestFee +
                                    bills[i]->details.optional.additionalCharges;

            printf("Bill updated successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Bill ID not found.\n");
    }
}

void deleteBill() {
    int id, found = 0;
    printf("Enter Bill ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < totalBills; i++) {
        if (bills[i]->id == id) {
            found = 1;
            free(bills[i]);
            for (int j = i; j < totalBills - 1; j++) {
                bills[j] = bills[j + 1];
            }
            totalBills--;
            printf("Bill deleted successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Bill ID not found.\n");
    }
}

void listAllBills() {
    if (totalBills == 0) {
        printf("No bills to display.\n");
        return;
    }

    printf("\n--- All Bills ---\n");
    for (int i = 0; i < totalBills; i++) {
        printf("Bill ID: %d, Patient Name: %s, Total Amount: %.2f\n",
               bills[i]->id, bills[i]->patient.name, bills[i]->totalAmount);
    }
}
```

Problem 5: Medical Test Result Management
Description: Develop a system to manage and store patient test results
Menu Options:
Add Test Result
View Test Result
Update Test Result
Delete Test Result
List All Test Results
Exit
Requirements:
Declare variables for test results.
Use static and const for standard test ranges.

Implement switch case for result operations.
Utilize loops for result input and output.
Use pointers for handling result data.
Create functions for result management.
Use arrays for storing test results.
Define structures for test result details.
Employ nested structures for detailed test parameters.
Utilize unions for optional test data.
Apply nested unions for complex test result data.

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

#define MAX 10
const char STANDARD_TEST_RANGE[] = "Normal: 70 - 100 mg/dL";
struct TestDetails {
    char testName[50];
    char testDate[20];
    float testValue;
    union {
        char notes[100];
    } optionalData;
};

static struct TestDetails *testResults[MAX];
static int totalResults = 0;

void addTestResult();
void viewTestResult();
void updateTestResult();
void deleteTestResult();
void listAllTestResults();

int main() {
    int choice;
    while (1) {
        printf("\n--- Medical Test Result Management ---\n");
        printf("Standard Test Range: %s\n", STANDARD_TEST_RANGE);
        printf("1. Add Test Result\n");
        printf("2. View Test Result\n");
        printf("3. Update Test Result\n");
        printf("4. Delete Test Result\n");
        printf("5. List All Test Results\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTestResult();
                break;
            case 2:
                viewTestResult();
                break;
            case 3:
                updateTestResult();
                break;
            case 4:
                deleteTestResult();
                break;
            case 5:
                listAllTestResults();
                break;
            case 6:
                printf("Exiting ....\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

void addTestResult() {
    if (totalResults >= MAX) {
        printf("Test result limit reached. Cannot add more.\n");
        return;
    }

    struct TestDetails *newTestResult = (struct TestDetails *)malloc(sizeof(struct
TestDetails));
    getchar();

    printf("Enter Test Name: ");
    fgets(newTestResult->testName, sizeof(newTestResult->testName), stdin);
    newTestResult->testName[strcspn(newTestResult->testName, "\n")] = '\0';

    printf("Enter Test Date (DD/MM/YYYY): ");
    fgets(newTestResult->testDate, sizeof(newTestResult->testDate), stdin);
    newTestResult->testDate[strcspn(newTestResult->testDate, "\n")] = '\0';

    printf("Enter Test Value: ");
    scanf("%f", &newTestResult->testValue);

    getchar();
    printf("Enter Additional Notes (if any): ");
    fgets(newTestResult->optionalData.notes, sizeof(newTestResult->optionalData.notes),
stdin);
    newTestResult->optionalData.notes[strcspn(newTestResult->optionalData.notes, "\n")]
= '\0';

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

void viewTestResult() {
    int id, found = 0;
    printf("Enter Test ID to view: ");
    scanf("%d", &id);

    if (id < 1 || id > totalResults) {
        printf("Test result not found!\n");
        return;
    }

    printf("Test ID: %d\n", id);
    printf("Test Name: %s\n", testResults[id - 1]->testName);
    printf("Test Date: %s\n", testResults[id - 1]->testDate);
    printf("Test Value: %.2f\n", testResults[id - 1]->testValue);
    printf("Additional Notes: %s\n", testResults[id - 1]->optionalData.notes);
}
```

```
void updateTestResult() {
    int id, found = 0;
    printf("Enter Test ID to update: ");
    scanf("%d", &id);

    if (id < 1 || id > totalResults) {
        printf("Test result not found!\n");
        return;
    }

    printf("Updating Test ID: %d\n", id);
    printf("Enter new Test Value: ");
    scanf("%f", &testResults[id - 1]->testValue);

    getchar();
    printf("Enter new Additional Notes: ");
    fgets(testResults[id - 1]->optionalData.notes, sizeof(testResults[id - 
1]->optionalData.notes), stdin);
    testResults[id - 1]->optionalData.notes[strcspn(testResults[id - 1]->optionalData.notes, 
"\n")] = '\0';

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

void deleteTestResult() {
    int id, found = 0;
    printf("Enter Test ID to delete: ");
    scanf("%d", &id);

    if (id < 1 || id > totalResults) {
        printf("Test result not found!\n");
        return;
    }

    free(testResults[id - 1]);

    for (int i = id - 1; i < totalResults - 1; i++) {
        testResults[i] = testResults[i + 1];
    }
    totalResults--;
    printf("Test result deleted successfully!\n");
}

void listAllTestResults() {
    if (totalResults == 0) {
        printf("No test results available.\n");
        return;
    }

    for (int i = 0; i < totalResults; i++) {
        printf("Test ID: %d\n", i + 1);
        printf("Test Name: %s\n", testResults[i]->testName);
        printf("Test Date: %s\n", testResults[i]->testDate);
        printf("Test Value: %.2f\n", testResults[i]->testValue);
        printf("Additional Notes: %s\n", testResults[i]->optionalData.notes);
        printf("----------------------------------\n");
    }
}
```

Problem 6: Staff Duty Roster Management
Description: Create a system to manage hospital staff duty rosters
Menu Options:
Add Duty Roster
View Duty Roster
Update Duty Roster
Delete Duty Roster
List All Duty Rosters
Exit
Requirements:
Use variables for staff details.
Apply static and const for fixed shift timings.
Implement switch case for roster operations.
Utilize loops for roster generation.
Use pointers for dynamic staff data.
Create functions for roster management.
Use arrays for storing staff schedules.
Define structures for duty details.
Employ nested structures for detailed duty breakdowns.
Use unions for optional duty attributes.
Apply nested unions for complex duty data.

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

#define MAX 100

struct AdditionalInfo {
    char notes[100];
};

struct StaffDetails {
    char staffName[50];
    int staffID;
    char shift[10];
    char date[15];
    struct AdditionalInfo optionalData;
};

struct StaffDetails *staffRoster[MAX];
int totalStaff = 0;

void addDutyRoster();
void viewDutyRoster();
void updateDutyRoster();
void deleteDutyRoster();
void listAllDutyRosters();

int main() {
    int choice;

    while (1) {
        printf("\n--- Staff Duty Roster Management ---\n");
        printf("Shift Times: Morning: 9 AM - 5 PM, Night: 9 PM - 5 AM\n");
        printf("1. Add Duty Roster\n");
        printf("2. View Duty Roster\n");
        printf("3. Update Duty Roster\n");
        printf("4. Delete Duty Roster\n");
        printf("5. List All Duty Rosters\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addDutyRoster(); break;
            case 2: viewDutyRoster(); break;
            case 3: updateDutyRoster(); break;
            case 4: deleteDutyRoster(); break;
            case 5: listAllDutyRosters(); break;
            case 6: exit(0);
            default: printf("Invalid choice. Try again.\n");
        }
    }
}

void addDutyRoster() {
    if (totalStaff >= MAX) {
        printf("Roster limit reached. Cannot add more staff.\n");
        return;
    }

    struct StaffDetails *newStaff = (struct StaffDetails *)malloc(sizeof(struct
StaffDetails));
    getchar();

    printf("Enter Staff Name: ");
    fgets(newStaff->staffName, sizeof(newStaff->staffName), stdin);
    newStaff->staffName[strcspn(newStaff->staffName, "\n")] = '\0';

    printf("Enter Staff ID: ");
    scanf("%d", &newStaff->staffID);

    getchar();
    printf("Enter Shift (Morning/Night): ");
    fgets(newStaff->shift, sizeof(newStaff->shift), stdin);
    newStaff->shift[strcspn(newStaff->shift, "\n")] = '\0';

    printf("Enter Date (DD/MM/YYYY): ");
    fgets(newStaff->date, sizeof(newStaff->date), stdin);
    newStaff->date[strcspn(newStaff->date, "\n")] = '\0';
    getchar();
    printf("Enter Additional Notes (if any): ");
    fgets(newStaff->optionalData.notes, sizeof(newStaff->optionalData.notes), stdin);
    newStaff->optionalData.notes[strcspn(newStaff->optionalData.notes, "\n")] = '\0';

    staffRoster[totalStaff++] = newStaff;
    printf("Duty roster added successfully!\n");
}

void viewDutyRoster() {
    int id, found = 0;
    printf("Enter Staff ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < totalStaff; i++) {
        if (staffRoster[i]->staffID == id) {
            found = 1;
            printf("Staff ID: %d\n", staffRoster[i]->staffID);
            printf("Staff Name: %s\n", staffRoster[i]->staffName);
            printf("Shift: %s\n", staffRoster[i]->shift);
            printf("Date: %s\n", staffRoster[i]->date);
            printf("Additional Notes: %s\n", staffRoster[i]->optionalData.notes);
            break;
        }
    }
    if (!found) {
        printf("Staff not found!\n");
    }
}

void updateDutyRoster() {
    int id, found = 0;
    printf("Enter Staff ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < totalStaff; i++) {
        if (staffRoster[i]->staffID == id) {
            found = 1;
            getchar();
            printf("Enter new Staff Name: ");
            fgets(staffRoster[i]->staffName, sizeof(staffRoster[i]->staffName), stdin);
            staffRoster[i]->staffName[strcspn(staffRoster[i]->staffName, "\n")] = '\0';

            printf("Enter new Shift (Morning/Night): ");
            fgets(staffRoster[i]->shift, sizeof(staffRoster[i]->shift), stdin);
            staffRoster[i]->shift[strcspn(staffRoster[i]->shift, "\n")] = '\0';

            printf("Enter new Date (DD/MM/YYYY): ");
            fgets(staffRoster[i]->date, sizeof(staffRoster[i]->date), stdin);
            staffRoster[i]->date[strcspn(staffRoster[i]->date, "\n")] = '\0';

            printf("Enter new Additional Notes (if any): ");
            fgets(staffRoster[i]->optionalData.notes,
sizeof(staffRoster[i]->optionalData.notes), stdin);
            staffRoster[i]->optionalData.notes[strcspn(staffRoster[i]->optionalData.notes,
"\n")] = '\0';

            printf("Duty roster updated successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Staff not found!\n");
    }
}

void deleteDutyRoster() {
    int id, found = 0;
    printf("Enter Staff ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < totalStaff; i++) {
        if (staffRoster[i]->staffID == id) {
            found = 1;
            free(staffRoster[i]);
            for (int j = i; j < totalStaff - 1; j++) {
                staffRoster[j] = staffRoster[j + 1];
            }
            totalStaff--;
            printf("Duty roster deleted successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Staff not found!\n");
    }
}

void listAllDutyRosters() {
    if (totalStaff == 0) {
        printf("No staff found.\n");
    } else {
        for (int i = 0; i < totalStaff; i++) {
            printf("Staff ID: %d\n", staffRoster[i]->staffID);
            printf("Staff Name: %s\n", staffRoster[i]->staffName);
            printf("Shift: %s\n", staffRoster[i]->shift);
            printf("Date: %s\n", staffRoster[i]->date);
            printf("Additional Notes: %s\n", staffRoster[i]->optionalData.notes);
            printf("------------------------------\n");
        }
    }
}
```

Problem 7: Emergency Contact Management System
Description: Design a system to manage emergency contacts for patients.
Menu Options:
Add Emergency Contact
View Emergency Contact
Update Emergency Contact
Delete Emergency Contact
List All Emergency Contacts
Exit
Requirements:
Declare variables for contact details.
Use static and const for non-changing contact data.
Implement switch case for contact operations.
Utilize loops for contact handling.
Use pointers for dynamic memory allocation.
Create functions for managing contacts.
Use arrays for storing contacts.
Define structures for contact details.

Employ nested structures for detailed contact information.
Utilize unions for optional contact data.
Apply nested unions for complex contact entries.

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

#define MAX 10

struct detailedContact{
    char gaurdianName[50];
    char address[100];
    char phno[100];

};
struct Contact{
    int id;
    char Patientname[50];
    int age;
    char gender[50];
    struct  detailedContact contactdetails;
    union {
        char diagonis[100];
    }additionInfo;
};

static struct Contact *contact[MAX];
static  int total =0;

void addCon();
void viewCon();
void updateCon();
void deleteCon();
void listAll();

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

        switch(choice){
            case 1:
                addCon();
                break;
            case 2:
                viewCon();
                break;
            case 3:
                updateCon();
                break;
            case 4:
                deleteCon();
                break;
            case 5:
                listAll();
                break;
            case 6:
                printf("Existing.............\n");
                return 0;
            default:
                printf("Invalid choice\n");

        }
    }
    return 0;
}


void addCon(){
    if(total>=MAX){
        printf("Limit reached Cannot add\n");
        return;
    }
    struct Contact *newcontact = (struct Contact*)malloc (sizeof(struct Contact));
    printf("Enter Patient ID: ");
    scanf("%d", &newcontact->id);
    printf("Enter Patient Name: ");
    getchar();
    scanf("%[^\n]s", newcontact->Patientname);
    printf("Enter Age: ");
    scanf("%d", &newcontact->age);
    printf("Enter Gender: ");
    getchar();
    scanf("%[^\n]s", newcontact->gender);
    printf("Enter Guardian Name: ");
    getchar();
    scanf("%[^\n]s", newcontact->contactdetails.gaurdianName);
    printf("Enter Address: ");
    getchar();
    scanf("%[^\n]s", newcontact->contactdetails.address);
    printf("Enter Phone Number: ");
    getchar();
    scanf("%[^\n]s", newcontact->contactdetails.phno);
    printf("Enter Diagnosis: ");
    getchar();
    scanf("%[^\n]s", newcontact->additionInfo.diagonis);


    contact[total++]= newcontact;
    printf("Emergency contact added successfully!\n ");


}
void viewCon(){
    int id,found =0;
    printf("Enter patient Id  ");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(contact[i]->id==id){
            found =1;
            printf("Patient Name: %s \n",contact[i]->Patientname);
            printf("Age : %d\n",contact[i]->age);
            printf("Gaurdian: %s\n",contact[i]->contactdetails.gaurdianName);
            printf("Address: %s\n",contact[i]->contactdetails.address);
            printf("Phone Number: %s\n",contact[i]->contactdetails.phno);
        }
    }
    if(!found){
        printf("Patient ID not found\n");
    }
}
void updateCon(){
     int id,found =0;
    printf("Enter patient Id  ");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(contact[i]->id==id){
            found =1;
            printf("Enter current gaudian: ");
            scanf("%[^\n]s",contact[i]->contactdetails.gaurdianName);
            printf("Enter new Address ");
            scanf("%[^\n]s",contact[i]->contactdetails.address);
            printf("Enter new phone number ");
            scanf("%[^\n]s",contact[i]->contactdetails.phno);
            printf("Emergency contact updated successfully!\n");
            break;
        }
    }
    if(!found){
        printf("Patient ID not found\n");
    }
}
void deleteCon(){
    int id,found =0;
    printf("Enter patient Id  ");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(contact[i]->id ==id){
            found =1;
            free(contact[i]);
            for(int j=i;j<total-1;j++){
                contact[j] = contact[j+1];
            }
            total--;
            printf("Emergency contact deleted successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Emergency contact not found!\n");
    }
}
void listAll(){
    if(total ==0){
        printf("No emergency contact found\n");
    }else{
        for(int i=0;i<total;i++){
            printf("Patient Name: %s\n",contact[i]->Patientname);
            printf("Gaurdian Name : %s\n",contact[i]->contactdetails.gaurdianName);
            printf("Address : %s\n",contact[i]->contactdetails.address);
            printf("Phone Number : %s\n",contact[i]->contactdetails.phno);
            printf("-----------------------------------------------------\n");
        }
    }
}
```

Problem 8: Medical Record Update System
Description: Create a system for updating patient medical records.
Menu Options:
Add Medical Record
View Medical Record
Update Medical Record
Delete Medical Record
List All Medical Records
Exit
Requirements:
Use variables for record details.
Apply static and const for immutable data like record ID.
Implement switch case for update operations.
Utilize loops for record updating.
Use pointers for handling records.
Create functions for record management.
Use arrays for storing records.
Define structures for record details.
Employ nested structures for detailed medical history.

Utilize unions for optional record fields.
Apply nested unions for complex record data.

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

#define MAX 10

struct MedicalHistory{
    char disease[50];
    char doctor[50];
};
struct Record{
    const int id;
    char patientName[50];
    int age;
    char gender[50];
    struct MedicalHistory history;
    union{
        char Treatment[50];
    }addtionalInfo;
};

static struct Record *record [MAX];
static int total =0;

void addRecord();
void viewRecord();
void updateRecord();
void deleteRecord();
void listall();

int main(){
    int choice;
    while(1){
        printf("\n--- Medical Record Update System ---\n");
        printf("1. Add Medical Record\n");
        printf("2. View Medical Record\n");
        printf("3. Update Medical Record\n");
        printf("4. Delete Medical Record\n");
        printf("5. List All Medical Records\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                addRecord();
                break;
            case 2:
                viewRecord();
                break;
            case 3:
                updateRecord();
                break;
            case 4:
                deleteRecord();
                break;
            case 5:
                listall();
                break;
            case 6:
                printf("Exiting...........\n");
                return 0;
            default:
                printf("Invalid choice \n");
        }

    }
    return 0;
}


void addRecord(){
    if(total>=MAX){
        printf("Limit reached \n");
        return;
    }
    struct Record *newrecord = (struct Record*)malloc(sizeof(struct Record));
    printf("ENter patient id ");
    scanf("%d",newrecord->id);
    printf("Patient name ");
    getchar();
    scanf("%[^\n]s",newrecord->patientName);
    printf("Age: ");
    scanf("%d",&newrecord->age);
    printf("Gender : ");
    getchar();
    scanf("%[^\n]s",newrecord->gender);
    printf("Disease: ");
    getchar();
    scanf("%[^\n]s",newrecord->history.disease);
    printf("Treated Doctor: ");
    getchar();
    scanf("%[^\n]s",newrecord->history.doctor);
    printf("Treament : ");
    getchar();
    scanf("%[^\n]s",newrecord->addtionalInfo.Treatment);

    record[total++] = newrecord;
}
void viewRecord(){
    int id,found =0;
    printf("ENter record id to view: ");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(record[i]->id==id){
        found =1;
        printf("\n---------Medical Record Details-----------\n");
        printf("ID: %d\n",record[i]->id);
        printf("Patient Name: %s\n",record[i]->patientName);
        printf("Age : %d\n",record[i]->age);
        printf("gender : %s\n",record[i]->gender);
        printf("Disease : %s\n",record[i]->history.disease);
        printf("Treament : %s\n",record[i]->addtionalInfo.Treatment);
        printf("Doctor : %s\n",record[i]->history.doctor);
        break;
        }
    }
    if(!found){
        printf("Record with ID %d not found.\n", id);
    }
}
```

```
void updateRecord(){
    int id,found =0;
    printf("ENter record id to view: ");
    scanf("%d",&id);
    for(int i=0;i<total;i++){
        if(record[i]->id==id){
            found =1;
            printf("Enter new disease: ");
            getchar();
            scanf("%[^\n]s",record[i]->history.disease);
            printf("Enter new Doctor ");
            getchar();
            scanf("%[^\n]s",record[i]->history.doctor);
            printf("Medical record updated successfully!\n");
            break;
        }
    }
    if(!found){
        printf("Record with ID %d not found.\n", id);
    }
}
void deleteRecord(){
    int id, found = 0;

    printf("Enter record ID to delete: ");
    scanf("%d", &id);
    for(int i=0;i<total;i++){
        if(record[i]->id==id){
            found =1;
            free(record[i]);
            for(int j=i;j<total-1;j++){
                record[j] = record[j+1];
            }
            total--;
            printf("Medical record deleted successfully!\n");
            break;
        }
    }
     if (!found) {
        printf("Record with ID %d not found.\n", id);
    }
}
void listall(){
    if(total ==0){
        printf("No records available \n");
        return;
    }
    for(int i=0;i<total;i++){
        printf("\n---------Medical Record Details-----------\n");
        printf("ID: %d\n",record[i]->id);
        printf("Patient Name: %s\n",record[i]->patientName);
        printf("Age : %d\n",record[i]->age);
        printf("gender : %s\n",record[i]->gender);
        printf("Disease : %s\n",record[i]->history.disease);
        printf("Treament : %s\n",record[i]->addtionalInfo.Treatment);
        printf("Doctor : %s\n",record[i]->history.doctor);
        break;

    }

}
```

Problem 9: Patient Diet Plan Management
Description: Develop a system to manage diet plans for patients.
Menu Options:
Add Diet Plan
View Diet Plan
Update Diet Plan
Delete Diet Plan
List All Diet Plans
Exit
Requirements:
Declare variables for diet plan details.
Use static and const for fixed dietary guidelines.
Implement switch case for diet plan operations.
Utilize loops for diet plan handling.
Use pointers for dynamic diet data.
Create functions for diet plan management.
Use arrays for storing diet plans.
Define structures for diet plan details.
Employ nested structures for detailed dietary breakdowns.

Use unions for optional diet attributes.
Apply nested unions for complex diet plan data.

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

#define MAX_DIET_PLANS 100

typedef struct {
    char foodItems[200];
    char restrictions[200];
    char schedule[200];
} DetailedDiet;

typedef struct {
    int id;
    char patientName[50];
    int age;
    char gender[10];
    DetailedDiet dietDetails;
    union {
        char additionalInfo[200];
    } optionalInfo;
} DietPlan;

static DietPlan dietPlans[MAX_DIET_PLANS];
static int total = 0;

void addDietPlan();
void viewDietPlan();
void updateDietPlan();
void deleteDietPlan();
void listAllDietPlans();

int main() {
    int choice;

    while (1) {
        printf("\n--- Patient Diet Plan Management ---\n");
        printf("1. Add Diet Plan\n");
        printf("2. View Diet Plan\n");
        printf("3. Update Diet Plan\n");
        printf("4. Delete Diet Plan\n");
        printf("5. List All Diet Plans\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addDietPlan();
                break;
            case 2:
                viewDietPlan();
                break;
            case 3:
                updateDietPlan();
                break;
            case 4:
                deleteDietPlan();
                break;
            case 5:
                listAllDietPlans();
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

void addDietPlan() {
    if (total >= MAX_DIET_PLANS) {
        printf("Limit reached! Cannot add more diet plans.\n");
        return;
    }
    DietPlan *newPlan = &dietPlans[total];
    printf("Enter Patient ID: ");
    scanf("%d", &newPlan->id);
    printf("Enter Patient Name: ");
    getchar();
    scanf("%[^\n]", newPlan->patientName);
    printf("Enter Age: ");
    scanf("%d", &newPlan->age);
    printf("Enter Gender: ");
    getchar();
    scanf("%[^\n]", newPlan->gender);
    printf("Enter Food Items: ");
    getchar();
    scanf("%[^\n]", newPlan->dietDetails.foodItems);
    printf("Enter Restrictions: ");
    getchar();
    scanf("%[^\n]", newPlan->dietDetails.restrictions);
    printf("Enter Schedule: ");
    getchar();
    scanf("%[^\n]", newPlan->dietDetails.schedule);
    printf("Enter Additional Info (optional): ");
    getchar();
    scanf("%[^\n]", newPlan->optionalInfo.additionalInfo);

    total++;
    printf("Diet Plan added successfully!\n");
}

void viewDietPlan() {
    int id, found = 0;
    printf("Enter Patient ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (dietPlans[i].id == id) {
            found = 1;
            printf("\nPatient ID: %d\n", dietPlans[i].id);
            printf("Patient Name: %s\n", dietPlans[i].patientName);
            printf("Age: %d\n", dietPlans[i].age);
            printf("Gender: %s\n", dietPlans[i].gender);
            printf("Food Items: %s\n", dietPlans[i].dietDetails.foodItems);
            printf("Restrictions: %s\n", dietPlans[i].dietDetails.restrictions);
            printf("Schedule: %s\n", dietPlans[i].dietDetails.schedule);
            printf("Additional Info: %s\n", dietPlans[i].optionalInfo.additionalInfo);
            break;
        }
    }

    if (!found) {
        printf("Diet Plan not found for the given Patient ID.\n");
    }
}

void updateDietPlan() {
    int id, found = 0;
    printf("Enter Patient ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (dietPlans[i].id == id) {
            found = 1;
            printf("Enter New Food Items: ");
            getchar();
            scanf("%[^\n]", dietPlans[i].dietDetails.foodItems);
            printf("Enter New Restrictions: ");
            getchar();
            scanf("%[^\n]", dietPlans[i].dietDetails.restrictions);
            printf("Enter New Schedule: ");
            getchar();
            scanf("%[^\n]", dietPlans[i].dietDetails.schedule);
            printf("Diet Plan updated successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Diet Plan not found for the given Patient ID.\n");
    }
}

void deleteDietPlan() {
    int id, found = 0;
    printf("Enter Patient ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < total; i++) {
        if (dietPlans[i].id == id) {
            found = 1;
            for (int j = i; j < total - 1; j++) {
                dietPlans[j] = dietPlans[j + 1];
            }
            total--;
            printf("Diet Plan deleted successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Diet Plan not found for the given Patient ID.\n");
    }
}

void listAllDietPlans() {
    if (total == 0) {
        printf("No Diet Plans found.\n");
        return;
    }

    printf("\n--- List of All Diet Plans ---\n");
    for (int i = 0; i < total; i++) {
        printf("\nPatient ID: %d\n", dietPlans[i].id);
        printf("Patient Name: %s\n", dietPlans[i].patientName);
        printf("Age: %d\n", dietPlans[i].age);
        printf("Gender: %s\n", dietPlans[i].gender);
        printf("Food Items: %s\n", dietPlans[i].dietDetails.foodItems);
        printf("Restrictions: %s\n", dietPlans[i].dietDetails.restrictions);
        printf("Schedule: %s\n", dietPlans[i].dietDetails.schedule);
        printf("Additional Info: %s\n", dietPlans[i].optionalInfo.additionalInfo);
        printf("-----------------------------------");
    }
}
```

Problem 10: Surgery Scheduling System

Description: Design a system for scheduling surgeries.
Menu Options:
Schedule Surgery
View Surgery Schedule
Update Surgery Schedule
Cancel Surgery
List All Surgeries
Exit
Requirements:
Use variables for surgery details.
Apply static and const for immutable data like surgery types.
Implement switch case for scheduling operations.
Utilize loops for surgery scheduling.
Use pointers for handling surgery data.
Create functions for surgery management.
Use arrays for storing surgery schedules.
Define structures for surgery details.
Employ nested structures for detailed surgery information.
Utilize unions for optional surgery data.
Apply nested unions for complex surgery entries.

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

#define MAX_SURGERIES 100

typedef struct {
   char surgeryType[50];
   char surgeonName[50];
   char surgeryDate[20];
   char surgeryTime[20];
} SurgeryDetails;

typedef struct {
   int surgeryID;
   char patientName[50];
   int patientAge;
   char patientGender[10];
   SurgeryDetails details;
   union {
        char additionalNotes[200];
    } optionalData;
} Surgery;

static Surgery surgeries[MAX_SURGERIES];
static int totalSurgeries = 0;

void scheduleSurgery();
void viewSurgery();
void updateSurgery();
void cancelSurgery();
void listAllSurgeries();

int main() {
    int choice;

    while (1) {
        printf("\n--- Surgery Scheduling System ---\n");
        printf("1. Schedule Surgery\n");
        printf("2. View Surgery Schedule\n");
        printf("3. Update Surgery Schedule\n");
        printf("4. Cancel Surgery\n");
        printf("5. List All Surgeries\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleSurgery();
                break;
            case 2:
                viewSurgery();
                break;
            case 3:
                updateSurgery();
                break;
            case 4:
                cancelSurgery();
                break;
            case 5:
                listAllSurgeries();
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

void scheduleSurgery() {
    if (totalSurgeries >= MAX_SURGERIES) {
        printf("Limit reached! Cannot schedule more surgeries.\n");
        return;
    }

    Surgery *newSurgery = &surgeries[totalSurgeries];
    printf("Enter Surgery ID: ");
    scanf("%d", &newSurgery->surgeryID);
    printf("Enter Patient Name: ");
    getchar();
    scanf("%[^\n]", newSurgery->patientName);
    printf("Enter Patient Age: ");
    scanf("%d", &newSurgery->patientAge);
    printf("Enter Patient Gender: ");
    getchar();
    scanf("%[^\n]", newSurgery->patientGender);
    printf("Enter Surgery Type: ");
    getchar();
    scanf("%[^\n]", newSurgery->details.surgeryType);
    printf("Enter Surgeon Name: ");
    getchar();
    scanf("%[^\n]", newSurgery->details.surgeonName);
    printf("Enter Surgery Date (DD/MM/YYYY): ");
    getchar();
    scanf("%[^\n]", newSurgery->details.surgeryDate);
    printf("Enter Surgery Time (HH:MM AM/PM): ");
    getchar();
    scanf("%[^\n]", newSurgery->details.surgeryTime);
    printf("Enter Additional Notes (optional): ");
    getchar();
    scanf("%[^\n]", newSurgery->optionalData.additionalNotes);

    totalSurgeries++;
    printf("Surgery scheduled successfully!\n");
}

void viewSurgery() {
    int surgeryID, found = 0;
    printf("Enter Surgery ID to view: ");
    scanf("%d", &surgeryID);

    for (int i = 0; i < totalSurgeries; i++) {
        if (surgeries[i].surgeryID == surgeryID) {
            found = 1;
            printf("\nSurgery ID: %d\n", surgeries[i].surgeryID);
            printf("Patient Name: %s\n", surgeries[i].patientName);
            printf("Patient Age: %d\n", surgeries[i].patientAge);
            printf("Patient Gender: %s\n", surgeries[i].patientGender);
            printf("Surgery Type: %s\n", surgeries[i].details.surgeryType);
            printf("Surgeon Name: %s\n", surgeries[i].details.surgeonName);
            printf("Surgery Date: %s\n", surgeries[i].details.surgeryDate);
            printf("Surgery Time: %s\n", surgeries[i].details.surgeryTime);
            printf("Additional Notes: %s\n", surgeries[i].optionalData.additionalNotes);
            break;
        }
    }

    if (!found) {
        printf("Surgery not found for the given Surgery ID.\n");
    }
}

void updateSurgery() {
    int surgeryID, found = 0;
    printf("Enter Surgery ID to update: ");
    scanf("%d", &surgeryID);

    for (int i = 0; i < totalSurgeries; i++) {
        if (surgeries[i].surgeryID == surgeryID) {
            found = 1;
            printf("Enter New Surgery Type: ");
            getchar();
            scanf("%[^\n]", surgeries[i].details.surgeryType);
            printf("Enter New Surgeon Name: ");
            getchar();
            scanf("%[^\n]", surgeries[i].details.surgeonName);
            printf("Enter New Surgery Date (DD/MM/YYYY): ");
            getchar();
            scanf("%[^\n]", surgeries[i].details.surgeryDate);
            printf("Enter New Surgery Time (HH:MM AM/PM): ");
            getchar();
            scanf("%[^\n]", surgeries[i].details.surgeryTime);
            printf("Surgery schedule updated successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Surgery not found for the given Surgery ID.\n");
    }
}

void cancelSurgery() {
    int surgeryID, found = 0;
    printf("Enter Surgery ID to cancel: ");
    scanf("%d", &surgeryID);

    for (int i = 0; i < totalSurgeries; i++) {
        if (surgeries[i].surgeryID == surgeryID) {
            found = 1;
            for (int j = i; j < totalSurgeries - 1; j++) {
                surgeries[j] = surgeries[j + 1];
            }
            totalSurgeries--;
            printf("Surgery canceled successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Surgery not found for the given Surgery ID.\n");
    }
}

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

    printf("\n--- List of All Surgeries ---\n");
    for (int i = 0; i < totalSurgeries; i++) {
        printf("\nSurgery ID: %d\n", surgeries[i].surgeryID);
        printf("Patient Name: %s\n", surgeries[i].patientName);
        printf("Patient Age: %d\n", surgeries[i].patientAge);
        printf("Patient Gender: %s\n", surgeries[i].patientGender);
        printf("Surgery Type: %s\n", surgeries[i].details.surgeryType);
        printf("Surgeon Name: %s\n", surgeries[i].details.surgeonName);
        printf("Surgery Date: %s\n", surgeries[i].details.surgeryDate);
        printf("Surgery Time: %s\n", surgeries[i].details.surgeryTime);
        printf("Additional Notes: %s\n", surgeries[i].optionalData.additionalNotes);
        printf("-----------------------------------");
    }
}
```

Problem 11: Prescription Management System
Description: Develop a system to manage patient prescriptions.
Menu Options:
Add Prescription
View Prescription
Update Prescription
Delete Prescription
List All Prescriptions
Exit
Requirements:
Declare variables for prescription details.
Use static and const for fixed prescription guidelines.
Implement switch case for prescription operations.

Utilize loops for prescription handling.
Use pointers for dynamic prescription data.
Create functions for prescription management.
Use arrays for storing prescriptions.
Define structures for prescription details.
Employ nested structures for detailed prescription information.
Use unions for optional prescription fields.
Apply nested unions for complex prescription data.

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

#define MAX_PRESCRIPTIONS 100

// Define a nested structure to store detailed prescription information
struct dosage {
    char dose[50];
    char frequency[50];
};

// Union for optional prescription details (e.g., additional instructions)
union additionalInfo {
    char specialInstructions[100];
    int refillCount;
};

// Define the prescription structure
struct prescription {
    int prescriptionID;
    char patientName[50];
    char medication[50];
    struct dosage medDosage;
    union additionalInfo info;
    int isRefill;  // 0 for no refill, 1 for refill
};

// Array to store prescriptions
struct prescription prescriptions[MAX_PRESCRIPTIONS];

// Function to add a prescription
void addPrescription() {
    static int prescriptionCounter = 0; // Counter to track prescription IDs
    if (prescriptionCounter < MAX_PRESCRIPTIONS) {
        struct prescription *newPrescription = &prescriptions[prescriptionCounter];

        printf("Enter Patient Name: ");
        getchar(); // To consume the newline left by previous scanf
        fgets(newPrescription->patientName, sizeof(newPrescription->patientName), stdin);

        printf("Enter Medication Name: ");
        fgets(newPrescription->medication, sizeof(newPrescription->medication), stdin);

        printf("Enter Dosage (e.g., 500mg): ");
        fgets(newPrescription->medDosage.dose,
sizeof(newPrescription->medDosage.dose), stdin);

        printf("Enter Frequency (e.g., once a day): ");
        fgets(newPrescription->medDosage.frequency,
sizeof(newPrescription->medDosage.frequency), stdin);

        printf("Special Instructions (Leave blank if none): ");
        char buffer[100];
        fgets(buffer, sizeof(buffer), stdin);
        if (strlen(buffer) > 1) {
            strcpy(newPrescription->info.specialInstructions, buffer);
            newPrescription->isRefill = 0;
        } else {
            printf("Enter Refill Count (0 for no refill): ");
            scanf("%d", &newPrescription->info.refillCount);
            newPrescription->isRefill = 1;
        }

        newPrescription->prescriptionID = ++prescriptionCounter; // Increment prescription
ID

        printf("Prescription added successfully!\n");
    } else {
        printf("Prescription list is full!\n");
    }
}

// Function to view a prescription by ID
```

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

    if (id > 0 && id <= MAX_PRESCRIPTIONS && prescriptions[id-1].prescriptionID
!= 0) {
        struct prescription *p = &prescriptions[id - 1];

        printf("Prescription ID: %d\n", p->prescriptionID);
        printf("Patient Name: %s", p->patientName);
        printf("Medication: %s", p->medication);
        printf("Dosage: %s", p->medDosage.dose);
        printf("Frequency: %s", p->medDosage.frequency);

        if (p->isRefill) {
            printf("Refill Count: %d\n", p->info.refillCount);
        } else {
            printf("Special Instructions: %s", p->info.specialInstructions);
        }
    } else {
        printf("Prescription not found!\n");
    }
}

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

    if (id > 0 && id <= MAX_PRESCRIPTIONS && prescriptions[id-1].prescriptionID
!= 0) {
        struct prescription *p = &prescriptions[id - 1];

        printf("Enter new Patient Name: ");
        getchar();
        fgets(p->patientName, sizeof(p->patientName), stdin);

        printf("Enter new Medication Name: ");
        fgets(p->medication, sizeof(p->medication), stdin);
        printf("Enter new Dosage: ");
        fgets(p->medDosage.dose, sizeof(p->medDosage.dose), stdin);

        printf("Enter new Frequency: ");
        fgets(p->medDosage.frequency, sizeof(p->medDosage.frequency), stdin);

        printf("Special Instructions (Leave blank if none): ");
        char buffer[100];
        fgets(buffer, sizeof(buffer), stdin);
        if (strlen(buffer) > 1) {
            strcpy(p->info.specialInstructions, buffer);
            p->isRefill = 0;
        } else {
            printf("Enter Refill Count (0 for no refill): ");
            scanf("%d", &p->info.refillCount);
            p->isRefill = 1;
        }

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

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

    if (id > 0 && id <= MAX_PRESCRIPTIONS && prescriptions[id-1].prescriptionID 
!= 0) {
        prescriptions[id - 1].prescriptionID = 0;  // Mark as deleted by resetting ID
        printf("Prescription deleted successfully!\n");
    } else {
        printf("Prescription not found!\n");
    }
}

// Function to list all prescriptions
```

```
void listAllPrescriptions() {
    int found = 0;
    for (int i = 0; i < MAX_PRESCRIPTIONS; i++) {
        if (prescriptions[i].prescriptionID != 0) {
            printf("Prescription ID: %d\n", prescriptions[i].prescriptionID);
            printf("Patient Name: %s", prescriptions[i].patientName);
            printf("Medication: %s", prescriptions[i].medication);
            printf("Dosage: %s", prescriptions[i].medDosage.dose);
            printf("Frequency: %s", prescriptions[i].medDosage.frequency);

            if (prescriptions[i].isRefill) {
                printf("Refill Count: %d\n", prescriptions[i].info.refillCount);
            } else {
                printf("Special Instructions: %s", prescriptions[i].info.specialInstructions);
            }
            printf("\n");
            found = 1;
        }
    }
    if (!found) {
        printf("No prescriptions found!\n");
    }
}

int main() {
    int choice;
    do {
        printf("\nPrescription Management System\n");
        printf("1. Add Prescription\n");
        printf("2. View Prescription\n");
        printf("3. Update Prescription\n");
        printf("4. Delete Prescription\n");
        printf("5. List All Prescriptions\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

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

    return 0;
}
```

Problem 12: Doctor Consultation Management
Description: Create a system for managing doctor consultations.
Menu Options:
Schedule Consultation
View Consultation
Update Consultation
Cancel Consultation
List All Consultations
Exit
Requirements:
Use variables for consultation details.
Apply static and const for non-changing data like consultation fees.
Implement `

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

```
#define MAX_CONSULTATIONS 100

// Define a structure for consultation details
struct consultation {
    int consultationID;
    char patientName[50];
    char doctorName[50];
    char consultationTime[50];
    struct consultation *next;
};

struct consultation *first = NULL, *ptr;

// Function to schedule a consultation
void scheduleConsultation() {
    struct consultation *newConsultation = (struct consultation *)malloc(sizeof(struct
consultation));

    printf("Enter Patient Name: ");
    getchar();
    fgets(newConsultation->patientName, sizeof(newConsultation->patientName), stdin);

    printf("Enter Doctor Name: ");
    fgets(newConsultation->doctorName, sizeof(newConsultation->doctorName), stdin);

    printf("Enter Consultation Time: ");
    fgets(newConsultation->consultationTime,
sizeof(newConsultation->consultationTime), stdin);

    newConsultation->next = first;
    first = newConsultation;
    printf("Consultation scheduled successfully!\n");
}

// Function to view consultations
void viewConsultation() {
    int id;
    printf("Enter Consultation ID to view: ");
    scanf("%d", &id);
    struct consultation *temp = first;
    while (temp != NULL) {
        if (temp->consultationID == id) {
            printf("Consultation ID: %d\n", temp->consultationID);
            printf("Patient Name: %s", temp->patientName);
            printf("Doctor Name: %s", temp->doctorName);
            printf("Consultation Time: %s", temp->consultationTime);
            return;
        }
        temp = temp->next;
    }

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

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

    struct consultation *temp = first;
    while (temp != NULL) {
        if (temp->consultationID == id) {
            printf("Enter new Patient Name: ");
            getchar();
            fgets(temp->patientName, sizeof(temp->patientName), stdin);

            printf("Enter new Doctor Name: ");
            fgets(temp->doctorName, sizeof(temp->doctorName), stdin);

            printf("Enter new Consultation Time: ");
            fgets(temp->consultationTime, sizeof(temp->consultationTime), stdin);

            printf("Consultation updated successfully!\n");
            return;
        }
        temp = temp->next;
    }

    printf("Consultation not found!\n");
```

```
}

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

    struct consultation *temp = first, *prev = NULL;
    while (temp != NULL) {
        if (temp->consultationID == id) {
            if (prev == NULL) {
                first = temp->next;
            } else {
                prev->next = temp->next;
            }
            free(temp);
            printf("Consultation canceled successfully!\n");
            return;
        }
        prev = temp;
        temp = temp->next;
    }

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

// Function to list all consultations
void listAllConsultations() {
    struct consultation *temp = first;
    while (temp != NULL) {
        printf("Consultation ID: %d\n", temp->consultationID);
        printf("Patient Name: %s", temp->patientName);
        printf("Doctor Name: %s", temp->doctorName);
        printf("Consultation Time: %s", temp->consultationTime);
        temp = temp->next;
    }
}

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

        switch (choice) {
            case 1:
                scheduleConsultation();
                break;
            case 2:
                viewConsultation();
                break;
            case 3:
                updateConsultation();
                break;
            case 4:
                cancelConsultation();
                break;
            case 5:
                listAllConsultations();
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice, try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

*************Linked List *******************************

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

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

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

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

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

}

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

}

void Insert(struct Node *p,int index ,int x){
    struct Node *temp;
    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = x;  //x =6
    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 queue
```

```
{
   char name[10];
   struct queue *next;
};
struct queue *first=NULL,*ptr;
static int count=0;
void create()
{
   int n;
   struct queue *temp;
   printf("Enter queue size : ");
   scanf("%d",&n);
   for(int i=0;i<n;i++)
   {
      struct queue *newnode=(struct queue *)malloc(sizeof(struct queue));
   newnode->next=NULL;
   strcpy(newnode->name,"\0");
   if(first==NULL)
   {
      first=newnode;
      ptr=newnode;
      temp=newnode;

   }
   else
   {
      temp->next=newnode;
      temp=newnode;
   }
   }

}
void insert()
{
   if(ptr!=NULL)
   {
      printf("Enter name : ");
      scanf("%s",ptr->name);
      ptr=ptr->next;
      count++;
   }
   else
   printf("Queue full\n");
}
void display()
{
    struct queue *temp=first;
    while(temp!=NULL)
   {
      if(strcmp(temp->name,"\0")!=0)
      printf("%s ->",temp->name);
      temp=temp->next;

   }
   printf("\n");
}
void main()
{int choice;
do
{
   printf("1.Create queue\n");
   printf("2.Insert\n");
   printf("3.Display\n");
   printf("4.Exit\n");
   printf("Enter choice \n");
   scanf("%d",&choice);
   switch (choice)
   {
   case 1:
      create();
      break;
      case 2:
      insert();
      break;
      case 3:display();
      break;
      case 4: printf("Exiting ...");
      break;

   default:
      break;
   }
} while (choice !=4);

}
```

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>


struct Ward{
   int bedNumber;
   int occupied;
   struct Ward *next;
};
struct Ward *first = NULL,*ptr;

static int total =0;

void createBeds(){
   struct Ward *new;
   printf("Enter total number of beds: ");
   scanf("%d",&total);

   first = (struct Ward*) malloc (sizeof(struct Ward));
   first->bedNumber =1;
   first->occupied =0;
   first->next =NULL;
   ptr = first;


   for(int i=2;i<=total;i++){
       new = (struct Ward*) malloc (sizeof(struct Ward));
       new->bedNumber =i;
       new->occupied =0;
       new->next =NULL;
       ptr->next =new;
       ptr = new;
   }
}

void assign(){
   struct Ward *temp = first;
   while(temp!=NULL && temp->occupied ==1){
       temp = temp->next;
   }
   if(temp!=NULL){
       temp->occupied =1;
       printf("Patient assigned to Bed Number: %d\n", temp->bedNumber);
   } else {
       printf("No available beds.\n");
   }
}

void displayBeds() {
   struct Ward *temp = first;

   if (temp == NULL) {
       printf("No beds available.\n");
       return;
   }

   printf("Current Bed Allocation:\n");
   while (temp != NULL) {
       printf("-->Bed Number: %d, Status: %s-->", temp->bedNumber, (temp->occupied
== 1) ? "Occupied" : "Available");
       temp = temp->next;
   }
}


void main() {
   int choice;
   do {
       printf("\n1. Create Beds\n");
       printf("2. Assign Patient\n");
       printf("3. Display Bed Allocation\n");
       printf("4. Exit\n");
       printf("Enter choice: ");
       scanf("%d", &choice);

       switch (choice) {
           case 1:
               createBeds();
               break;
           case 2:
               assign();
               break;
           case 3:
               displayBeds();
               break;
           case 4:
               printf("Exiting...\n");
               break;
           default:
               printf("Invalid choice. Try again.\n");
       }
   } while (choice != 4);
}
```

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 inventory {
    int itemID;
    char itemName[50];
    int quantity;
    struct inventory *next;
};

struct inventory *first = NULL, *ptr;
static int count = 0;

void create() {
    int n;
    struct inventory *temp;
    printf("Enter total number of inventory items: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct inventory *newnode = (struct inventory *)malloc(sizeof(struct inventory));
        newnode->next = NULL;
        newnode->itemID = 0;
        newnode->quantity = 0;
        strcpy(newnode->itemName, "\0");

        if (first == NULL) {
            first = newnode;
            ptr = newnode;
            temp = newnode;
        } else {
            temp->next = newnode;
            temp = newnode;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Item ID: ");
        scanf("%d", &ptr->itemID);
        printf("Enter Item Name: ");
        scanf(" %[^\n]", ptr->itemName);  // To allow spaces in item name
        printf("Enter Quantity: ");
        scanf("%d", &ptr->quantity);
        ptr = ptr->next;
        count++;
    } else {
        printf("Inventory full\n");
    }
}

void display() {
    struct inventory *temp = first;
    while (temp != NULL) {
        if (temp->itemID != 0) {
            printf("%d, %s, %d\n", temp->itemID, temp->itemName, temp->quantity);
        }
        temp = temp->next;
    }
    printf("\n");
}

void main() {
    int choice;
    do {
        printf("1. Create Inventory\n");
        printf("2. Insert New Inventory Item\n");
        printf("3. Display Inventory\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                break;
        }
    } while (choice != 4);
}
```

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 {
    int ID;
    char patientName[50];
    char doctorName[50];
    char date[50];
    struct appointment *next;
};

struct appointment *first = NULL, *ptr;

void create() {
    struct appointment *temp;
    int n;
    printf("Enter total number of appointments: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct appointment *new = (struct appointment *)malloc(sizeof(struct
appointment));
        new->next = NULL;
        new->ID = 0;
        strcpy(new->patientName, "\0");
        strcpy(new->doctorName, "\0");
        strcpy(new->date, "\0");

        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Appointment ID: ");
        scanf("%d", &ptr->ID);
        printf("Enter Patient Name: ");
        getchar();  // To handle the newline character after scanf
        scanf(" %[^\n]", ptr->patientName);
        printf("Enter Doctor Name: ");
        scanf(" %[^\n]", ptr->doctorName);
        printf("Enter Appointment Date (DD/MM/YYYY): ");
        scanf(" %[^\n]", ptr->date);

        ptr = ptr->next;
    } else {
        printf("Appointment list is full.\n");
    }
}

void display() {
    struct appointment *temp = first;
    while (temp != NULL) {
        if (temp->ID != 0) {  // Ensure valid data
            printf("Appointment ID: %d\n", temp->ID);
            printf("Patient Name: %s\n", temp->patientName);
            printf("Doctor Name: %s\n", temp->doctorName);
            printf("Appointment Date: %s\n\n", temp->date);
        }
        temp = temp->next;
    }
}
```

```
int main() {
    int choice;
    do {
        printf("1. Create Appointment List\n");
        printf("2. Insert New Appointment\n");
        printf("3. Display All Appointments\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);

    return 0;
}
```

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 {
   int ID;
   char name[50];
   char phoneNumber[15];
   struct contact *next;
};

struct contact *first = NULL, *ptr;

void create() {
   struct contact *temp;
   int n;
   printf("Enter total number of emergency contacts: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      struct contact *new = (struct contact *)malloc(sizeof(struct contact));
      new->next = NULL;
      new->ID = 0;
      strcpy(new->name, "\0");
      strcpy(new->phoneNumber, "\0");

      if (first == NULL) {
         first = new;
         ptr = new;
         temp = new;
      } else {
         temp->next = new;
         temp = new;
      }
   }
}

void insert() {
   if (ptr != NULL) {
      printf("Enter Contact ID: ");
      scanf("%d", &ptr->ID);
      printf("Enter Name: ");
      getchar();  // To handle the newline character after scanf
      scanf(" %[^\n]", ptr->name);
      printf("Enter Phone Number: ");
      scanf(" %[^\n]", ptr->phoneNumber);

      ptr = ptr->next;
   } else {
      printf("Contact list is full.\n");
   }
}

void display() {
   struct contact *temp = first;
   while (temp != NULL) {
      if (temp->ID != 0) {  // Ensure valid data
         printf("Contact ID: %d\n", temp->ID);
         printf("Name: %s\n", temp->name);
         printf("Phone Number: %s\n\n", temp->phoneNumber);
      }
      temp = temp->next;
   }
}

int main() {
   int choice;
   do {
      printf("1. Create Emergency Contact List\n");
      printf("2. Insert New Contact\n");
      printf("3. Display All Emergency Contacts\n");
      printf("4. Exit\n");
      printf("Enter choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
             create();
             break;
         case 2:
             insert();
             break;
         case 3:
             display();
             break;
         case 4:
             printf("Exiting...\n");
             break;
         default:
             printf("Invalid choice. Please try again.\n");
             break;
      }
   } while (choice != 4);

   return 0;
}
```

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 {
    int surgeryID;
    char patientName[50];
    char surgeonName[50];
    char surgeryDate[50];
    struct surgery *next;
};

struct surgery *first = NULL, *ptr;

void create() {
    struct surgery *temp;
    int n;
    printf("Enter total number of surgeries: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct surgery *new = (struct surgery *)malloc(sizeof(struct surgery));
        new->next = NULL;
        new->surgeryID = 0;
        strcpy(new->patientName, "\0");
        strcpy(new->surgeonName, "\0");
        strcpy(new->surgeryDate, "\0");

        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Surgery ID: ");
        scanf("%d", &ptr->surgeryID);
        printf("Enter Patient Name: ");
        getchar();  // To handle the newline character after scanf
        scanf(" %[^\n]", ptr->patientName);
        printf("Enter Surgeon Name: ");
        scanf(" %[^\n]", ptr->surgeonName);
        printf("Enter Surgery Date (DD/MM/YYYY): ");
        scanf(" %[^\n]", ptr->surgeryDate);

        ptr = ptr->next;
    } else {
        printf("Surgery schedule list is full.\n");
    }
}
```

```
void display() {
    struct surgery *temp = first;
    while (temp != NULL) {
        if (temp->surgeryID != 0) {  // Ensure valid data
            printf("Surgery ID: %d\n", temp->surgeryID);
            printf("Patient Name: %s\n", temp->patientName);
            printf("Surgeon Name: %s\n", temp->surgeonName);
            printf("Surgery Date: %s\n\n", temp->surgeryDate);
        }
        temp = temp->next;
    }
}

int main() {
    int choice;
    do {
        printf("1. Create Surgery Schedule\n");
        printf("2. Insert New Surgery\n");
        printf("3. Display All Scheduled Surgeries\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);
    return 0;
}
```

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 patientHistory {
    int recordID;
    char patientName[50];
    char disease[50];
    char treatment[50];
    struct patientHistory *next;
};

struct patientHistory *first = NULL, *ptr;

void create() {
    struct patientHistory *temp;
    int n;
    printf("Enter total number of records: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct patientHistory *new = (struct patientHistory *)malloc(sizeof(struct
patientHistory));
        new->next = NULL;
        new->recordID = 0;
        strcpy(new->patientName, "\0");
        strcpy(new->disease, "\0");
        strcpy(new->treatment, "\0");
        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Record ID: ");
        scanf("%d", &ptr->recordID);
        printf("Enter Patient Name: ");
        getchar();
        scanf(" %[^\n]", ptr->patientName);
        printf("Enter Disease: ");
        scanf(" %[^\n]", ptr->disease);
        printf("Enter Treatment: ");
        scanf(" %[^\n]", ptr->treatment);

        ptr = ptr->next;
    } else {
        printf("History record list is full.\n");
    }
}

void display() {
    struct patientHistory *temp = first;
    while (temp != NULL) {
        if (temp->recordID != 0) {
            printf("Record ID: %d\n", temp->recordID);
            printf("Patient Name: %s\n", temp->patientName);
            printf("Disease: %s\n", temp->disease);
            printf("Treatment: %s\n\n", temp->treatment);
        }
        temp = temp->next;
    }
}
```

```
int main() {
    int choice;
    do {
        printf("1. Create History Record List\n");
        printf("2. Insert New Record\n");
        printf("3. Display All Records\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);

    return 0;
}
```

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 {
    int testID;
    char patientName[50];
    char testName[50];
    char result[50];
    struct medicalTest *next;
};

struct medicalTest *first = NULL, *ptr;

void create() {
    struct medicalTest *temp;
    int n;
    printf("Enter total number of tests: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct medicalTest *new = (struct medicalTest *)malloc(sizeof(struct medicalTest));
        new->next = NULL;
        new->testID = 0;
        strcpy(new->patientName, "\0");
        strcpy(new->testName, "\0");
        strcpy(new->result, "\0");

        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Test ID: ");
        scanf("%d", &ptr->testID);
        printf("Enter Patient Name: ");
        getchar();
        scanf(" %[^\n]", ptr->patientName);
        printf("Enter Test Name: ");
        scanf(" %[^\n]", ptr->testName);
        printf("Enter Test Result: ");
        scanf(" %[^\n]", ptr->result);

        ptr = ptr->next;
    } else {
        printf("Medical test list is full.\n");
    }
}

void display() {
    struct medicalTest *temp = first;
    while (temp != NULL) {
        if (temp->testID != 0) {
            printf("Test ID: %d\n", temp->testID);
            printf("Patient Name: %s\n", temp->patientName);
            printf("Test Name: %s\n", temp->testName);
            printf("Result: %s\n\n", temp->result);
        }
        temp = temp->next;
    }
}

int main() {
    int choice;
    do {
        printf("1. Create Medical Test List\n");
        printf("2. Insert New Test\n");
        printf("3. Display All Test Results\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);

    return 0;
}
```

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 {
    int prescriptionID;
    char patientName[50];
    char medication[50];
    char dosage[50];
    struct prescription *next;
};

struct prescription *first = NULL, *ptr;

void create() {
    struct prescription *temp;
    int n;
    printf("Enter total number of prescriptions: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct prescription *new = (struct prescription *)malloc(sizeof(struct prescription));
        new->next = NULL;
        new->prescriptionID = 0;
        strcpy(new->patientName, "\0");
        strcpy(new->medication, "\0");
        strcpy(new->dosage, "\0");

        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Prescription ID: ");
        scanf("%d", &ptr->prescriptionID);
        printf("Enter Patient Name: ");
        getchar();
        scanf(" %[^\n]", ptr->patientName);
        printf("Enter Medication: ");
        scanf(" %[^\n]", ptr->medication);
        printf("Enter Dosage: ");
        scanf(" %[^\n]", ptr->dosage);

        ptr = ptr->next;
    } else {
        printf("Prescription list is full.\n");
    }
}

void display() {
    struct prescription *temp = first;
    while (temp != NULL) {
        if (temp->prescriptionID != 0) {
            printf("Prescription ID: %d\n", temp->prescriptionID);
            printf("Patient Name: %s\n", temp->patientName);
            printf("Medication: %s\n", temp->medication);
            printf("Dosage: %s\n\n", temp->dosage);
        }
        temp = temp->next;
    }
}

int main() {
    int choice;
    do {
        printf("1. Create Prescription List\n");
        printf("2. Insert New Prescription\n");
        printf("3. Display All Prescriptions\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);
```

```c
    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.

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

struct staff {
    int staffID;
    char staffName[50];
    char position[50];
    struct staff *next;
};

struct staff *first = NULL, *ptr;

void create() {
    struct staff *temp;
    int n;
    printf("Enter total number of staff members: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        struct staff *new = (struct staff *)malloc(sizeof(struct staff));
        new->next = NULL;
        new->staffID = 0;
        strcpy(new->staffName, "\0");
        strcpy(new->position, "\0");

        if (first == NULL) {
            first = new;
            ptr = new;
            temp = new;
        } else {
            temp->next = new;
            temp = new;
        }
    }
}

void insert() {
    if (ptr != NULL) {
        printf("Enter Staff ID: ");
        scanf("%d", &ptr->staffID);
        printf("Enter Staff Name: ");
        getchar();
        scanf(" %[^\n]", ptr->staffName);
        printf("Enter Position: ");
        scanf(" %[^\n]", ptr->position);

        ptr = ptr->next;
    } else {
        printf("Staff roster is full.\n");
    }
}

void display() {
    struct staff *temp = first;
    while (temp != NULL) {
        if (temp->staffID != 0) {
            printf("Staff ID: %d\n", temp->staffID);
            printf("Staff Name: %s\n", temp->staffName);
            printf("Position: %s\n\n", temp->position);
        }
        temp = temp->next;
    }
}

int main() {
    int choice;
    do {
        printf("1. Create Staff Roster\n");
        printf("2. Insert New Staff\n");
        printf("3. Display Staff Roster\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 4);

    return 0;
}
```