1: Patient Information Management System

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

Menu Options:

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

Requirements:

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

<stdio.h>

#include <stdlib.h> #include

<string.h>

#define MAX_PATIENTS 100

const char *HOSPITAL_NAME = "City Hospital";

```
struct MedicalHistory { char
    pastDiseases[100]; char
    allergies[50]; union {
        char notes[200]; struct {
            char familyHistory[100]; char
            surgeries[100];
        } detailedHistory;
    } historyDetails;
};
struct Patient { int
    id;
    char name[50]; int
    age;
    char gender[10];
    char currentMedications[100]; struct
    MedicalHistory history;
};
struct Patient *patients; int
patientCount = 0;
```

```
void AddNewPatient(); void
ViewPatientDetails();
void UpdatePatientInformation(); void
DeletePatientRecord();
void ListAllPatients();

void initializePatients() {
    patients = (struct Patient *)malloc(MAX_PATIENTS * sizeof(struct Patient)); if (!patients) {
        printf("Memory allocation failed. Exiting.\n"); exit(1);
    }
}

void addNewPatient() {
    if (patientCount >= MAX_PATIENTS) {
        printf("\nPatient record is full. Cannot add more patients.\n"); return;
    }
    struct Patient *p = &patients[patientCount]; p->id =
    patientCount + 1;
```

```
    printf("\nEnter Name: "); scanf(" %s",
    p->name); printf("Enter Age: ");
    scanf("%d", &p->age); printf("Enter
    Gender: "); scanf("%s", p->gender);
    printf("Enter Current Medications: "); scanf("
    %s", p->currentMedications); printf("Enter Past
    Diseases: ");  scanf(" %s", p-
    >history.pastDiseases); printf("Enter Allergies:
    ");
    scanf(" %s", p->history.allergies); printf("Enter Family
    History: ");
    scanf(" %s", p->history.historyDetails.detailedHistory.familyHistory); printf("Enter Surgeries: ");
    scanf(" %s", p->history.historyDetails.detailedHistory.surgeries);

    patientCount++;
    printf("\nPatient added successfully with ID %d!\n", p->id);
}
void viewPatientDetails() {
```

```
    int id;
    printf("\nEnter Patient ID to view details: "); scanf("%d", &id);
    if (id <= 0 || id > patientCount) {
        printf("\nInvalid Patient ID.\n"); return;
    }
    struct Patient *p = &patients[id - 1]; printf("\nPatient
    ID: %d\n", p->id); printf("Name: %s\n", p->name);
    printf("Age: %d\n", p->age); printf("Gender: %s\n", p-
    >gender);
    printf("Current Medications: %s\n", p->currentMedications); printf("Past
    Diseases: %s\n", p->history.pastDiseases); printf("Allergies: %s\n", p-
    >history.allergies);
    printf("Family History: %s\n", p-
>history.historyDetails.detailedHistory.familyHistory);
    printf("Surgeries: %s\n", p->history.historyDetails.detailedHistory.surgeries);
}

void updatePatientInformation() { int id;
```

```
printf("\nEnter Patient ID to update: "); scanf("%d", &id);
if (id <= 0 || id > patientCount) {
    printf("\nInvalid Patient ID.\n"); return;
}
struct Patient *p = &patients[id - 1];
printf("\nUpdating information for Patient ID %d\n", p->id); printf("Enter New
Name: ");
scanf(" %s", p->name); printf("Enter
New Age: "); scanf("%d", &p->age);
printf("Enter New Gender: ");
scanf("%s", p->gender);
printf("Enter New Current Medications: "); scanf("
%s", p->currentMedications); printf("Enter New Past
Diseases: "); scanf(" %s", p->history.pastDiseases);
printf("Enter New Allergies: ");
scanf(" %s", p->history.allergies); printf("Enter New
Family History: ");
```

```
    scanf(" %s", p->history.historyDetails.detailedHistory.familyHistory); printf("Enter New
    Surgeries: ");
    scanf(" %s", p->history.historyDetails.detailedHistory.surgeries);

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

void deletePatientRecord() { int id;
    printf("\nEnter Patient ID to delete: "); scanf("%d", &id);
    if (id <= 0 || id > patientCount) {
        printf("\nInvalid Patient ID.\n"); return;
    }
    for (int i = id - 1; i < patientCount - 1; i++) { patients[i] =
        patients[i + 1];
    }
    patientCount--;
    printf("\nPatient record deleted successfully!\n");
}
```

```
void listAllPatients() {
    if (patientCount == 0) {
        printf("\nNo patient records available.\n"); return;
    }
    printf("\nListing 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);
    }
}

int main() { initializePatients();

    int choice;
    printf("Welcome to %s\n", HOSPITAL_NAME);

    do {
        printf("\nMenu:\n");
        printf("1. Add New Patient\n");
```

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


switch (choice) { case
    1:
        addNewPatient(); break;
    case 2:
        viewPatientDetails(); break;
    case 3:
        updatePatientInformation(); break;
    case 4:
        deletePatientRecord(); break;
```

```
            case 5:
                listAllPatients();
                break;
            case 6:
                printf("\nExiting the system. Goodbye!\n"); free(patients);
                break;
            default:
                printf("\nInvalid choice. Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

2: Hospital Inventory Management

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

Menu Options:

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

Requirements:

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

<stdio.h>

#include <stdlib.h> #include

<string.h>

#define MAX_ITEMS 100

const char *HOSPITAL_NAME = "City Hospital";

struct ItemDetails {

    char manufacturer[50]; char

    expirationDate[15]; union {

        int quantity;

        double weight;

    } unitInfo; // Union for quantity or weight.

};

```
struct InventoryItem { int id;
    char name[50]; char
    category[30]; double
    price;
    struct ItemDetails details;  // Nested structure for item details.
};

struct InventoryItem inventory[MAX_ITEMS];  // Array of inventory items. int itemCount = 0;

void addInventoryItem() {
    if (itemCount >= MAX_ITEMS) {
        printf("\nInventory is full. Cannot add more items.\n"); return;
    }
    struct InventoryItem *item = &inventory[itemCount]; item->id =
    itemCount + 1;
    printf("\nEnter Item Name: "); scanf("
    %s", item->name);
```

```
    printf("Enter Category: "); scanf("
    %s", item->category); printf("Enter
    Price: "); scanf("%lf", &item->price);
    printf("Enter Manufacturer: ");
    scanf(" %s", item->details.manufacturer); printf("Enter
    Expiration Date: ");
    scanf(" %s", item->details.expirationDate); printf("Enter
    Quantity (0 if N/A): "); scanf("%d", &item-
    >details.unitInfo.quantity);

    itemCount++;
    printf("\nItem added successfully with ID %d!\n", item->id);
}

void viewInventoryItem() { int id;
    printf("\nEnter Item ID to view: "); scanf("%d", &id);
    if (id <= 0 || id > itemCount) { printf("\nInvalid Item
        ID.\n");
```

```
        return;
    }
    struct InventoryItem *item = &inventory[id - 1];
    printf("\nID: %d, Name: %s, Category: %s, Price: %.2f\n", item->id, item->name, item->category, item->price);
    printf("Manufacturer: %s, Expiration Date: %s\n", item->details.manufacturer, item->details.expirationDate);
    printf("Quantity: %d\n", item->details.unitInfo.quantity);
}

void updateInventoryItem() { int id;
    printf("\nEnter Item ID to update: "); scanf("%d", &id);
    if (id <= 0 || id > itemCount) { printf("\nInvalid
        Item ID.\n"); return;
    }
    struct InventoryItem *item = &inventory[id - 1]; printf("\nEnter
    New Name: ");
    scanf(" %s", item->name); printf("Enter New
    Category: ");
```

```
    scanf(" %s", item->category); printf("Enter New
    Price: "); scanf("%lf", &item->price);
    printf("Enter New Manufacturer: ");
    scanf(" %s", item->details.manufacturer); printf("Enter
    New Expiration Date: "); scanf(" %s", item-
    >details.expirationDate); printf("Enter New Quantity:
    ");
    scanf("%d", &item->details.unitInfo.quantity); printf("\nItem
    updated successfully!\n");
}

void deleteInventoryItem() { int id;
    printf("\nEnter Item ID to delete: "); scanf("%d", &id);
    if (id <= 0 || id > itemCount) { printf("\nInvalid
        Item ID.\n"); return;
    }
    for (int i = id - 1; i < itemCount - 1; i++) {
```

```
        inventory[i] = inventory[i + 1];
    }
    itemCount--;
    printf("\nItem deleted successfully!\n");
}

void listAllInventoryItems() { if
    (itemCount == 0) {
        printf("\nNo items in the inventory.\n"); return;
    }
    printf("\nAll Inventory Items:\n"); for (int i =
    0; i < itemCount; i++) {
        printf("ID: %d, Name: %s, Category: %s, Price: %.2f\n", inventory[i].id, inventory[i].name,
inventory[i].category, inventory[i].price);
    }
}

int main() { int
    choice;
    printf("Welcome to %s Inventory Management System\n", HOSPITAL_NAME);
```

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

    switch (choice) { case
        1:
            addInventoryItem(); break;
        case 2:
            viewInventoryItem(); break;
        case 3:
            updateInventoryItem();
```

```
            break;
        case 4:
            deleteInventoryItem(); break;
        case 5:
            listAllInventoryItems(); break;
        case 6:
            printf("\nExiting the system. Goodbye!\n"); break;
        default:
            printf("\nInvalid choice. Please try again.\n");
      }
  } while (choice != 6);

  return 0;
}
```

3: Medical Appointment Scheduling System Description: Develop a system to manage patient appointments. Menu Options:

1. Schedule Appointment
2. View Appointment

3. Update Appointment
4. Cancel Appointment
5. List All Appointments
6. Exit

Requirements:

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

<stdio.h>

#include <stdlib.h> #include

<string.h>

#define MAX_APPOINTMENTS 100

const char *CLINIC_NAME = "City Health Clinic";

const char *CLINIC_HOURS = "Mon-Fri: 9 AM - 6 PM";

// Structure to represent patient details struct Patient

{

    int id;

    char name[50]; int

    age;

    char gender[10];

```
};
// Structure to represent doctor details struct Doctor
{
    int id;
    char name[50]; char
    specialty[50];
};
// Union to hold appointment-specific attributes (e.g., follow-up or consultation) union
AppointmentAttributes {
    char followUpDetails[200]; char
    consultationNotes[200];
};
// Structure to represent an appointment struct
Appointment {
    int id;
    struct Patient patient; struct
    Doctor doctor;
    char appointmentDate[20];  // Date of appointment (DD/MM/YYYY) char
    appointmentTime[10];  // Time of appointment (HH:MM)
    union AppointmentAttributes attributes;
    int isFollowUp; // 1 if follow-up, 0 if consultation
```

```
};

struct Appointment *appointments; int

appointmentCount = 0;

void initializeAppointments() {

    appointments = (struct Appointment *)malloc(MAX_APPOINTMENTS * sizeof(struct
Appointment));

    if (!appointments) {

        printf("Memory allocation failed. Exiting.\n"); exit(1);

    }

}

void scheduleAppointment() {

    if (appointmentCount >= MAX_APPOINTMENTS) { printf("\nAppointment

        schedule is full. Cannot schedule more
appointments.\n");

        return;

    }

  struct Appointment *app = &appointments[appointmentCount]; app->id =

    appointmentCount + 1;

  // Input patient details printf("\nEnter

    Patient Name: "); scanf(" %s", app-

    >patient.name);
```

```
    printf("Enter Patient Age: "); scanf("%d", &app-
    >patient.age); printf("Enter Patient Gender: ");
    scanf(" %s", app->patient.gender);
// Input doctor details printf("Enter Doctor
    Name: ");
    scanf(" %s", app->doctor.name); printf("Enter
    Doctor Specialty: "); scanf(" %s", app-
    >doctor.specialty);
  // Input appointment date and time
    printf("Enter Appointment Date (DD/MM/YYYY): "); scanf(" %s", app-
    >appointmentDate);
    printf("Enter Appointment Time (HH:MM): "); scanf(" %s",
    app->appointmentTime);
  // Choose appointment type (Follow-up or Consultation) int typeChoice;
    printf("\nIs this a follow-up appointment?\n1. Yes\n2. No (Consultation)\nEnter your choice: ");
    scanf("%d", &typeChoice); if
    (typeChoice == 1) {
        app->isFollowUp = 1;  printf("Enter Follow-
        up Details: ");
```

```
        scanf(" %s", app->attributes.followUpDetails);
    } else {
        app->isFollowUp = 0;
        printf("Enter Consultation Notes: ");
        scanf(" %s", app->attributes.consultationNotes);
    }

    appointmentCount++;
    printf("\nAppointment scheduled successfully with ID %d!\n", app->id);
}
void viewAppointment() { int id;
    printf("\nEnter Appointment ID to view details: "); scanf("%d",
    &id);
  if (id <= 0 || id > appointmentCount) { printf("\nInvalid
        Appointment ID.\n"); return;
    }
  struct Appointment *app = &appointments[id - 1];
    printf("\nAppointment ID: %d\n", app->id); printf("Patient
    Name: %s\n", app->patient.name);
```

```
    printf("Patient Age: %d\n", app->patient.age); printf("Patient Gender:
    %s\n", app->patient.gender); printf("Doctor Name: %s\n", app-
    >doctor.name); printf("Doctor Specialty: %s\n", app->doctor.specialty);
    printf("Appointment Date: %s\n", app->appointmentDate);
    printf("Appointment Time: %s\n", app->appointmentTime);

    // Display appointment type details (Follow-up or Consultation) if (app-
    >isFollowUp) {
        printf("\nFollow-up Details: %s\n", app->attributes.followUpDetails);
    } else {
        printf("\nConsultation Notes: %s\n", app->attributes.consultationNotes);
    }
}
void updateAppointment() { int id;
    printf("\nEnter Appointment ID to update: "); scanf("%d",
    &id);
if (id <= 0 || id > appointmentCount) { printf("\nInvalid
        Appointment ID.\n"); return;
```

```
    }
struct Appointment *app = &appointments[id - 1]; printf("\nUpdating details for
    Appointment ID %d\n", app->id);

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

    // Update doctor details printf("Enter New
    Doctor Name: "); scanf(" %s", app-
    >doctor.name);
    printf("Enter New Doctor Specialty: "); scanf(" %s",
    app->doctor.specialty);

    // Update appointment date and time
    printf("Enter New Appointment Date (DD/MM/YYYY): "); scanf(" %s", app-
    >appointmentDate);
```

```
    printf("Enter New Appointment Time (HH:MM): "); scanf(" %s",
    app->appointmentTime);


    // Update appointment type (Follow-up or Consultation) int typeChoice;
    printf("\nIs this a follow-up appointment?\n1. Yes\n2. No (Consultation)\nEnter your choice: ");
    scanf("%d", &typeChoice); if
 (typeChoice == 1) {
        app->isFollowUp = 1;
        printf("Enter New Follow-up Details: "); scanf(" %s", app-
        >attributes.followUpDetails);
    } else {
        app->isFollowUp = 0;
        printf("Enter New Consultation Notes: "); scanf(" %s", app-
        >attributes.consultationNotes);
    }
  printf("\nAppointment updated successfully!\n");
}
void cancelAppointment() { int id;
    printf("\nEnter Appointment ID to cancel: ");
```

```
    scanf("%d", &id);
    if (id <= 0 || id > appointmentCount) { printf("\nInvalid
        Appointment ID.\n"); return;
    }

    // Shift subsequent appointments to delete the canceled appointment for (int i = id - 1; i
    < appointmentCount - 1; i++) {
        appointments[i] = appointments[i + 1];
    }
    appointmentCount--;
    printf("\nAppointment canceled successfully!\n");
}
void listAllAppointments() {
    if (appointmentCount == 0) {
        printf("\nNo appointments scheduled.\n"); return;
    }
  printf("\nListing all appointments:\n");
    for (int i = 0; i < appointmentCount; i++) {
        printf("ID: %d, Patient Name: %s, Doctor: %s, Date: %s, Time: %s\n",
```

```
            appointments[i].id, appointments[i].patient.name, appointments[i].doctor.name,
            appointments[i].appointmentDate, appointments[i].appointmentTime);
    }
}
int main() { initializeAppointments();

    int choice;
    printf("Welcome to the Medical Appointment Scheduling System at %s\n", CLINIC_NAME);
    printf("Clinic Hours: %s\n\n", CLINIC_HOURS); do {
        printf("\nMenu:\n");
        printf("1. Schedule Appointment\n"); printf("2.
        View Appointment\n"); printf("3. Update
        Appointment\n"); printf("4. Cancel
        Appointment\n"); printf("5. List All
        Appointments\n"); printf("6. Exit\n");
        printf("Enter your choice: "); scanf("%d",
        &choice);
```

```
switch (choice) { case
    1:
        scheduleAppointment(); break;
    case 2:
        viewAppointment(); break;
    case 3:
        updateAppointment(); break;
    case 4:
        cancelAppointment(); break;
    case 5:
        listAllAppointments(); break;
    case 6:
        printf("\nExiting the system. Goodbye!\n"); free(appointments);
        break;
```

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

    return 0;
}
```

4: Patient Billing System Description: Create a billing system for patients. Menu Options:

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

Requirements:

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

Sol: #include <stdio.h> #include

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

#define MAX_BILLS 100

const float ROOM_CHARGE = 500.0;

const float CONSULTATION_FEE = 300.0; const float

MEDICINE_TAX_RATE = 0.05; struct BillDetails {

    float roomCharges; float

    consultationFees; float

    medicineCharges;

};

union AdditionalCharges { float

    tax;

    float discount;

};

struct Bill { int

    billId;

    char patientName[50]; struct

    BillDetails details;

    union AdditionalCharges additional;

```
    float totalAmount;
};
struct Bill *bills[MAX_BILLS]; int
billCount = 0;
void generateBill() {
    if (billCount >= MAX_BILLS) { printf("Maximum bill
        limit reached.\n"); return;
    }
   struct Bill *newBill = (struct Bill *)malloc(sizeof(struct Bill)); printf("Enter Bill
    ID: ");
    scanf("%d", &newBill->billId); printf("Enter
    Patient Name: "); scanf("%s", newBill-
    >patientName); printf("Enter Room Charges: ");
    scanf("%f", &newBill->details.roomCharges); printf("Enter
    Consultation Fees: ");
    scanf("%f", &newBill->details.consultationFees); printf("Enter
    Medicine Charges: ");
    scanf("%f", &newBill->details.medicineCharges);
```

```
    newBill->details.medicineCharges *= (1 + MEDICINE_TAX_RATE);  // Add tax on medicine

    newBill->additional.tax = newBill->details.medicineCharges * MEDICINE_TAX_RATE;

    newBill->totalAmount = newBill->details.roomCharges + newBill-
>details.consultationFees + newBill->details.medicineCharges; bills[billCount++] =

   newBill;

    printf("Bill generated successfully! Total Amount: %.2f\n", newBill-
>totalAmount);

}

void viewBill() { int

    billId;

    printf("Enter Bill ID to view: "); scanf("%d",

    &billId);

    for (int i = 0; i < billCount; i++) { if

        (bills[i]->billId == billId) {

            printf("\nBill ID: %d\n", bills[i]->billId); printf("Patient Name:

            %s\n", bills[i]->patientName);

            printf("Room Charges: %.2f\n", bills[i]->details.roomCharges); printf("Consultation Fees:

            %.2f\n", bills[i]->details.consultationFees);

            printf("Medicine Charges (after tax): %.2f\n", bills[i]-
>details.medicineCharges);

            printf("Total Amount: %.2f\n", bills[i]->totalAmount); return;
```

```
        }
    }
    printf("Bill with ID %d not found.\n", billId);
}
void updateBill() { int
    billId;
    printf("Enter Bill ID to update: "); scanf("%d",
    &billId);
 for (int i = 0; i < billCount; i++) { if
        (bills[i]->billId == billId) {
            printf("Enter New Room Charges: "); scanf("%f", &bills[i]-
            >details.roomCharges); printf("Enter New Consultation Fees:
            "); scanf("%f", &bills[i]->details.consultationFees);
            printf("Enter New Medicine Charges: "); scanf("%f",
            &bills[i]->details.medicineCharges);
            bills[i]->details.medicineCharges *= (1 + MEDICINE_TAX_RATE);
            bills[i]->totalAmount = bills[i]->details.roomCharges + bills[i]-
>details.consultationFees + bills[i]->details.medicineCharges;
            printf("Bill updated successfully! Total Amount: %.2f\n", bills[i]-
>totalAmount);
            return;
```

```
        }
    }
    printf("Bill with ID %d not found.\n", billId);
}
void deleteBill() { int
    billId;
    printf("Enter Bill ID to delete: "); scanf("%d", &billId);
  for (int i = 0; i < billCount; i++) { if
        (bills[i]->billId == billId) {
            free(bills[i]);
            for (int j = i; j < billCount - 1; j++) { bills[j] =
                bills[j + 1];
            }
            billCount--;
            printf("Bill deleted successfully!\n"); return;
        }
    }
    printf("Bill with ID %d not found.\n", billId);
}
```

```
void listAllBills() {
    if (billCount == 0) {
        printf("No bills available.\n"); return;
    }
 printf("\nList of Bills:\n");
    for (int i = 0; i < billCount; i++) {
        printf("ID: %d, Patient Name: %s, Total Amount: %.2f\n", bills[i]->billId, bills[i]-
                >patientName, bills[i]->totalAmount);
    }
}
int main() { int
    choice; do {
        printf("\nPatient 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");
```

```
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 6);
 for (int i = 0; i < billCount; i++) { free(bills[i]);
    }
 return 0;
}
```

Problem 5: Medical Test Result Management

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

Menu Options:

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

Requirements:

7. Declare variables for test results.
8. Use static and const for standard test ranges.
9. Implement switch case for result operations.
10. Utilize loops for result input and output.
11. Use pointers for handling result data.
12. Create functions for result management.
13. Use arrays for storing test results.

14. Define structures for test result details.
15. Employ nested structures for detailed test parameters.
16. Utilize unions for optional test data.
17. Apply nested unions for complex test result data. Sol:

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>


#define MAX_RESULTS 100


const char *STANDARD_RANGES = "Standard ranges vary by test.";


struct TestParameters { char

    testName[50]; float

    resultValue; char unit[20];

};


union AdditionalInfo { char

    comments[100];

    char technicianName[50];

};
```

```
struct TestResult { int
    resultId;
    char patientName[50];
    struct TestParameters parameters; union
    AdditionalInfo info;
};

struct TestResult *results[MAX_RESULTS]; int
resultCount = 0;

void addTestResult() {
    if (resultCount >= MAX_RESULTS) { printf("Maximum number of
        test results reached.\n"); return;
    }

    struct TestResult *newResult = (struct TestResult *)malloc(sizeof(struct TestResult));
    printf("Enter Result ID: ");  scanf("%d",
    &newResult->resultId); printf("Enter Patient
    Name: "); scanf("%s", newResult->patientName);
```

```
    printf("Enter Test Name: ");
    scanf("%s", newResult->parameters.testName); printf("Enter
    Test Result Value: ");
    scanf("%f", &newResult->parameters.resultValue); printf("Enter Unit: ");
    scanf("%s", newResult->parameters.unit); printf("Enter
    Technician Name or Comments: "); scanf("%s", newResult-
    >info.technicianName);

    results[resultCount++] = newResult; printf("Test result
    added successfully!\n");
}

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

    for (int i = 0; i < resultCount; i++) { if
        (results[i]->resultId == id) {
            printf("\nResult ID: %d\n", results[i]->resultId);
```

```
            printf("Patient Name: %s\n", results[i]->patientName); printf("Test Name:
            %s\n", results[i]->parameters.testName);
            printf("Result Value: %.2f %s\n", results[i]->parameters.resultValue, results[i]->parameters.unit);
            printf("Additional Info: %s\n", results[i]->info.technicianName); return;
        }
    }
    printf("Test result with ID %d not found.\n", id);
}

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

    for (int i = 0; i < resultCount; i++) { if
        (results[i]->resultId == id) {
            printf("Enter New Test Name: ");
            scanf("%s", results[i]->parameters.testName); printf("Enter New Result
            Value: ");
            scanf("%f", &results[i]->parameters.resultValue);
```

```
            printf("Enter New Unit: ");
            scanf("%s", results[i]->parameters.unit);
            printf("Enter New Technician Name or Comments: "); scanf("%s",
            results[i]->info.technicianName); printf("Test result updated
            successfully!\n");
            return;
        }
    }
    printf("Test result with ID %d not found.\n", id);
}

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

    for (int i = 0; i < resultCount; i++) { if
        (results[i]->resultId == id) {
            free(results[i]);
            for (int j = i; j < resultCount - 1; j++) { results[j] =
                results[j + 1];
```

```
            }
            resultCount--;
            printf("Test result deleted successfully!\n"); return;
        }
    }
    printf("Test result with ID %d not found.\n", id);
}

void listAllResults() {
    if (resultCount == 0) {
        printf("No test results to display.\n"); return;
    }

    printf("\nList of Test Results:\n"); for (int i =
    0; i < resultCount; i++) {
        printf("ID: %d, Patient Name: %s, Test Name: %s, Result: %.2f %s\n",
                results[i]->resultId, results[i]->patientName, results[i]-
>parameters.testName,
                results[i]->parameters.resultValue, results[i]->parameters.unit);
    }
```

```
}

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

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

```
            break;
        case 3:
            updateTestResult(); break;
        case 4:
            deleteTestResult(); break;
        case 5:
            listAllResults();
            break;
        case 6:
            printf("Exiting...\n"); break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 6);

for (int i = 0; i < resultCount; i++) { free(results[i]);
}
```

```
    return 0;
}
```

6: Staff Duty Roster Management

Description: Create a system to manage hospital staff duty rosters

Menu Options:

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

Requirements:

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

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>

#define MAX_ROSTERS 100
```

```
const char *SHIFT_MORNING = "08:00 AM - 04:00 PM"; const char
*SHIFT_EVENING = "04:00 PM - 12:00 AM"; const char *SHIFT_NIGHT =
"12:00 AM - 08:00 AM";
struct DutyDetails { char
    shift[20]; char
    date[15];
};
union OptionalDetails { char
    notes[100];
    char specialDuty[50];
};
struct DutyRoster { int
    rosterId;
    char staffName[50]; struct
    DutyDetails duty;
    union OptionalDetails extra;
};
struct DutyRoster *rosters[MAX_ROSTERS]; int rosterCount
= 0;
void addDutyRoster() {
    if (rosterCount >= MAX_ROSTERS) {
```

```
        printf("Maximum number of rosters reached.\n"); return;
    }
  struct DutyRoster *newRoster = (struct DutyRoster *)malloc(sizeof(struct DutyRoster));
    printf("Enter Roster ID: "); scanf("%d",
    &newRoster->rosterId); printf("Enter Staff
    Name: "); scanf("%s", newRoster->staffName);
    printf("Enter Shift (Morning, Evening, Night): "); scanf("%s",
    newRoster->duty.shift);  printf("Enter Date (DD/MM/YYYY):
    "); scanf("%s", newRoster->duty.date);
    printf("Enter Notes or Special Duty: "); scanf("%s", newRoster-
    >extra.notes);
 rosters[rosterCount++] = newRoster; printf("Duty roster
    added successfully!\n");
}
void viewDutyRoster() { int id;
    printf("Enter Roster ID to view: "); scanf("%d", &id);
```

```
    for (int i = 0; i < rosterCount; i++) { if
        (rosters[i]->rosterId == id) {
            printf("\nRoster ID: %d\n", rosters[i]->rosterId); printf("Staff Name:
            %s\n", rosters[i]->staffName); printf("Shift: %s\n", rosters[i]-
            >duty.shift); printf("Date: %s\n", rosters[i]->duty.date);
            printf("Additional Info: %s\n", rosters[i]->extra.notes); return;
        }
    }
    printf("Roster with ID %d not found.\n", id);
}
void updateDutyRoster() { int id;
    printf("Enter Roster ID to update: "); scanf("%d", &id);
for (int i = 0; i < rosterCount; i++) { if
        (rosters[i]->rosterId == id) {
            printf("Enter New Shift (Morning, Evening, Night): "); scanf("%s",
            rosters[i]->duty.shift);
```

```
            printf("Enter New Date (DD/MM/YYYY): "); scanf("%s",
            rosters[i]->duty.date); printf("Enter New Notes or Special
            Duty: "); scanf("%s", rosters[i]->extra.notes); printf("Duty
            roster updated successfully!\n"); return;
        }
    }
    printf("Roster with ID %d not found.\n", id);
}
void deleteDutyRoster() { int id;
    printf("Enter Roster ID to delete: "); scanf("%d", &id);
 for (int i = 0; i < rosterCount; i++) { if
        (rosters[i]->rosterId == id) {
            free(rosters[i]);
            for (int j = i; j < rosterCount - 1; j++) { rosters[j] =
                rosters[j + 1];
            }
            rosterCount--;
```

```
            printf("Duty roster deleted successfully!\n"); return;
        }
    }
    printf("Roster with ID %d not found.\n", id);
}
void listAllDutyRosters() { if
    (rosterCount == 0) {
        printf("No duty rosters available.\n"); return;
    }
  printf("\nList of Duty Rosters:\n"); for (int i =
    0; i < rosterCount; i++) {
        printf("ID: %d, Staff Name: %s, Shift: %s, Date: %s\n",
               rosters[i]->rosterId, rosters[i]->staffName, rosters[i]->duty.shift, rosters[i]->duty.date);
    }
}
int main() { int
    choice; do {
        printf("\nStaff Duty Roster Management\n");
```

```
printf("1. Add Duty Roster\n"); printf("2.
View Duty Roster\n"); printf("3. Update
Duty Roster\n"); printf("4. Delete Duty
Roster\n"); printf("5. List All Duty
Rosters\n"); printf("6. Exit\n");
printf("Enter your choice: "); scanf("%d",
&choice);

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

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

    for (int i = 0; i < rosterCount; i++) {
        free(rosters[i]);
    }

    return 0;
}
```

Problem 7: Emergency Contact Management System

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

Menu Options:

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

Requirements:

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

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>

#define MAX_CONTACTS 100


const char *HOSPITAL_NAME = "City Hospital";


struct ContactInfo { char

    phone[15]; char

    email[50];
```

```
    union {
        char address[100]; struct {
            char city[50]; char
            zip[10];
        } location;
    } details;
};

struct EmergencyContact { int id;
    char name[50];
    char relationship[20]; struct
    ContactInfo contact;
};

struct EmergencyContact *contacts; int
contactCount = 0;

void initializeContacts() {
    contacts = (struct EmergencyContact *)malloc(MAX_CONTACTS * sizeof(struct
EmergencyContact));
```

```
    if (!contacts) {
        printf("Memory allocation failed. Exiting.\n"); exit(1);
    }
}

void addEmergencyContact() {
    if (contactCount >= MAX_CONTACTS) {
        printf("\nContact list is full. Cannot add more contacts.\n"); return;
    }
    struct EmergencyContact *c = &contacts[contactCount]; c->id =
    contactCount + 1;
    printf("\nEnter Name: "); scanf(" %s",
    c->name); printf("Enter Relationship:
    "); scanf(" %s", c->relationship);
    printf("Enter Phone: ");
    scanf(" %s", c->contact.phone); printf("Enter Email: ");
    scanf(" %s", c->contact.email);
```

```
    printf("Enter City: ");
    scanf(" %s", c->contact.details.location.city); printf("Enter ZIP:
    ");
    scanf(" %s", c->contact.details.location.zip);

    contactCount++;
    printf("\nEmergency contact added successfully with ID %d!\n", c->id);
}

void viewEmergencyContact() { int id;
    printf("\nEnter Contact ID to view details: "); scanf("%d", &id);
    if (id <= 0 || id > contactCount) {
        printf("\nInvalid Contact ID.\n"); return;
    }
    struct EmergencyContact *c = &contacts[id - 1];
    printf("\nContact ID: %d\n", c->id); printf("Name: %s\n", c-
    >name); printf("Relationship: %s\n", c->relationship);
```

```
    printf("Phone: %s\n", c->contact.phone); printf("Email: %s\n", c-
    >contact.email); printf("City: %s\n", c-
    >contact.details.location.city); printf("ZIP: %s\n", c-
    >contact.details.location.zip);
}

void updateEmergencyContact() { int id;
    printf("\nEnter Contact ID to update: "); scanf("%d", &id);
    if (id <= 0 || id > contactCount) {
        printf("\nInvalid Contact ID.\n"); return;
    }
    struct EmergencyContact *c = &contacts[id - 1]; printf("\nUpdating information
    for Contact ID %d\n", c->id); printf("Enter New Name: ");
    scanf(" %s", c->name); printf("Enter New
    Relationship: "); scanf(" %s", c-
    >relationship); printf("Enter New Phone: ");
```

```
    scanf(" %s", c->contact.phone);
    printf("Enter New Email: "); scanf(" %s",
    c->contact.email); printf("Enter New
    City: ");
    scanf(" %s", c->contact.details.location.city); printf("Enter New
    ZIP: ");
    scanf(" %s", c->contact.details.location.zip);

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

void deleteEmergencyContact() { int id;
    printf("\nEnter Contact ID to delete: "); scanf("%d", &id);
    if (id <= 0 || id > contactCount) {
        printf("\nInvalid Contact ID.\n"); return;
    }
    for (int i = id - 1; i < contactCount - 1; i++) { contacts[i] =
        contacts[i + 1];
```

```
    }
    contactCount--;
    printf("\nContact record deleted successfully!\n");
}

void listAllContacts() {
    if (contactCount == 0) {
        printf("\nNo emergency contact records available.\n"); return;
    }
    printf("\nListing all contacts:\n");
    for (int i = 0; i < contactCount; i++) {
        printf("ID: %d, Name: %s, Relationship: %s, Phone: %s\n", contacts[i].id,
            contacts[i].name, contacts[i].relationship,
contacts[i].contact.phone);
    }
}

int main() { initializeContacts();

    int choice;
```

```
printf("Welcome to %s\n", HOSPITAL_NAME);


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

    switch (choice) { case
        1:
            addEmergencyContact(); break;
        case 2:
            viewEmergencyContact(); break;
        case 3:
```

```
            updateEmergencyContact(); break;
        case 4:
            deleteEmergencyContact(); break;
        case 5:
            listAllContacts();
            break;
        case 6:
            printf("\nExiting the system. Goodbye!\n"); free(contacts);
            break;
        default:
            printf("\nInvalid choice. Please try again.\n");
      }
  } while (choice != 6);

  return 0;
}
```

Problem 8: Medical Record Update System

Description: Create a system for updating patient medical records.

Menu Options:

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

Requirements:

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

```
#include <stdio.h>

#include <string.h>

#define MAX_RECORDS 100

// Structure for medical history typedef

struct {

    char diagnosis[50]; char

    treatment[50];

} MedicalHistory;
```

```
// Union for optional fields typedef union
{
    char bloodType[4]; float
    weight;
} OptionalField;
// Structure for patient record typedef
struct {
    int id;
    char name[50]; int
    age;
    char gender[10]; MedicalHistory
    history; OptionalField optional;
} MedicalRecord;
MedicalRecord records[MAX_RECORDS]; int
recordCount = 0;
// Function prototypes void
addMedicalRecord();
void viewMedicalRecord(); void
updateMedicalRecord(); void
deleteMedicalRecord();
```

```
void listAllMedicalRecords();

// Function to add a medical record void

addMedicalRecord() {

    if (recordCount >= MAX_RECORDS) {

        printf("\nRecord limit reached. Cannot add more records.\n"); return;

    }

    MedicalRecord *record = &records[recordCount]; record->id =

    recordCount + 1;

    printf("Enter patient name: "); scanf("

    %s", record->name); printf("Enter

    age: "); scanf("%d", &record->age);

    printf("Enter gender: "); scanf(" %s",

    record->gender); printf("Enter

    diagnosis: ");

    scanf(" %s", record->history.diagnosis); printf("Enter

    treatment: ");

    scanf(" %s", record->history.treatment); printf("Enter blood

    type: ");

    scanf(" %s", record->optional.bloodType);
```

```
    recordCount++;
    printf("Record added successfully with ID: %d\n", record->id);
}
// Function to view a medical record void
viewMedicalRecord() {
    int id;
    printf("Enter record ID to view: "); scanf("%d", &id);
    if (id < 1 || id > recordCount) {
        printf("Invalid record ID.\n"); return;
    }
    MedicalRecord *record = &records[id - 1];
    printf("\nID: %d\nName: %s\nAge: %d\nGender: %s\nDiagnosis:
%s\nTreatment: %s\nBlood Type: %s\n",
            record->id, record->name, record->age, record->gender, record-
            >history.diagnosis, record->history.treatment, record-
>optional.bloodType);
}
// Function to update a medical record void
updateMedicalRecord() {
    int id;
```

```
    printf("Enter record ID to update: "); scanf("%d", &id);
    if (id < 1 || id > recordCount) {
        printf("Invalid record ID.\n"); return;
    }
    MedicalRecord *record = &records[id - 1]; int choice;
    do {
        printf("\nUpdate Menu:\n1. Name\n2. Age\n3. Gender\n4. Diagnosis\n5.
Treatment\n6. Exit\nEnter choice: "); scanf("%d",
        &choice);
        switch (choice) { case
            1:
                printf("Enter new name: "); scanf("
                %s", record->name); break;
            case 2:
                printf("Enter new age: "); scanf("%d",
                &record->age); break;
            case 3:
```

```
            printf("Enter new gender: "); scanf("
            %s", record->gender); break;
        case 4:
            printf("Enter new diagnosis: ");
            scanf(" %s", record->history.diagnosis); break;
        case 5:
            printf("Enter new treatment: ");
            scanf(" %s", record->history.treatment); break;
        case 6:
            printf("Exiting update menu.\n"); break;
        default:
            printf("Invalid choice.\n");
    }
  } while (choice != 6);
}

// Function to delete a medical record
```

```
void deleteMedicalRecord() { int id;
    printf("Enter record ID to delete: "); scanf("%d", &id);
    if (id < 1 || id > recordCount) {
        printf("Invalid record ID.\n"); return;
    }
    for (int i = id - 1; i < recordCount - 1; i++) { records[i] = records[i
        + 1];
    }
    recordCount--;
    printf("Record deleted successfully.\n");
}

// Function to list all medical records void
listAllMedicalRecords() {
    if (recordCount == 0) {
        printf("No records to display.\n"); return;
    }
```

```
    for (int i = 0; i < recordCount; i++) {
        printf("ID: %d, Name: %s, Age: %d, Gender: %s\n",
               records[i].id, records[i].name, records[i].age, records[i].gender);
    }
}

int main() { int
    choice; do {
        printf("\nMedical Record System:\n");
        printf("1. Add Medical Record\n2. View Medical Record\n3. Update Medical Record\n4. Delete
Medical Record\n5. List All Medical Records\n6. Exit\nEnter your choice: ");
        scanf("%d", &choice); switch
        (choice) {
            case 1:
                addMedicalRecord(); break;
            case 2:
                viewMedicalRecord(); break;
            case 3:
```

```
            updateMedicalRecord(); break;
        case 4:
            deleteMedicalRecord(); break;
        case 5:
            listAllMedicalRecords(); break;
        case 6:
            printf("Exiting system.\n"); break;
        default:
            printf("Invalid choice.\n");
      }
  } while (choice != 6); return
  0;
}
```

Problem 9: Patient Diet Plan Management

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

Menu Options:

1. Add Diet Plan
2. View Diet Plan

3. Update Diet Plan
4. Delete Diet Plan
5. List All Diet Plans
6. Exit

Requirements:

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

```
#include <stdio.h>

#include <string.h>

#define MAX_DIET_PLANS 100

// Structure for meal details typedef struct

{

    char breakfast[50]; char

    lunch[50]; char

    dinner[50];

} MealPlan;

// Union for optional diet attributes typedef union {

    int calorieLimit;
```

```
    char notes[100];
} OptionalDietData;

// Structure for diet plan typedef
struct {
    int id;
    char patientName[50]; MealPlan
    meals; OptionalDietData optional;
} DietPlan;

DietPlan dietPlans[MAX_DIET_PLANS]; int
dietPlanCount = 0;

// Function to add a diet plan void
addDietPlan() {
    if (dietPlanCount >= MAX_DIET_PLANS) { printf("\nCannot
        add more diet plans.\n"); return;
    }
    DietPlan *plan = &dietPlans[dietPlanCount];
```

```
    plan->id = dietPlanCount + 1; printf("Enter
    patient name: "); scanf(" %s", plan-
    >patientName); printf("Enter breakfast plan:
    "); scanf(" %s", plan->meals.breakfast);
    printf("Enter lunch plan: ");
    scanf(" %s", plan->meals.lunch);
    printf("Enter dinner plan: "); scanf(" %s",
    plan->meals.dinner);
    printf("Enter calorie limit (0 to skip): "); scanf("%d",
    &plan->optional.calorieLimit); dietPlanCount++;
    printf("Diet plan added successfully with ID: %d\n", plan->id);
}
// Function to view a diet plan void
viewDietPlan() {
    int id;
    printf("Enter diet plan ID to view: "); scanf("%d", &id);
    if (id < 1 || id > dietPlanCount) { printf("Invalid diet
        plan ID.\n");
```

```
        return;
    }
    DietPlan *plan = &dietPlans[id - 1];
    printf("\nID: %d\nPatient Name: %s\nBreakfast: %s\nLunch: %s\nDinner:
%s\n",
           plan->id, plan->patientName, plan->meals.breakfast, plan->meals.lunch, plan-
>meals.dinner);
    if (plan->optional.calorieLimit != 0) {
        printf("Calorie Limit: %d\n", plan->optional.calorieLimit);
    } else {
        printf("No calorie limit set.\n");
    }
}
// Function to update a diet plan void
updateDietPlan() {
    int id;
    printf("Enter diet plan ID to update: "); scanf("%d", &id);
    if (id < 1 || id > dietPlanCount) { printf("Invalid
        diet plan ID.\n"); return;
    }
```

```
    DietPlan *plan = &dietPlans[id - 1]; int choice;
    do {
        printf("\nUpdate Menu:\n1. Patient Name\n2. Breakfast\n3. Lunch\n4.
Dinner\n5. Calorie Limit\n6. Exit\nEnter choice: "); scanf("%d",
        &choice);
        switch (choice) { case
            1:
                printf("Enter new patient name: "); scanf("
                %s", plan->patientName); break;
            case 2:
                printf("Enter new breakfast plan: "); scanf("
                %s", plan->meals.breakfast); break;
            case 3:
                printf("Enter new lunch plan: "); scanf("
                %s", plan->meals.lunch); break;
            case 4:
                printf("Enter new dinner plan: "); scanf("
                %s", plan->meals.dinner);
```

```
                break;
            case 5:
                printf("Enter new calorie limit (0 to skip): "); scanf("%d",
                &plan->optional.calorieLimit); break;
            case 6:
                printf("Exiting update menu.\n"); break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);
}
// Function to delete a diet plan void
deleteDietPlan() {
    int id;
    printf("Enter diet plan ID to delete: "); scanf("%d",
    &id);
    if (id < 1 || id > dietPlanCount) { printf("Invalid
        diet plan ID.\n"); return;
```

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

// Function to list all diet plans void
listAllDietPlans() {
    if (dietPlanCount == 0) {
        printf("No diet plans available.\n"); return;
    }
    for (int i = 0; i < dietPlanCount; i++) {
        printf("ID: %d, Patient Name: %s\n", dietPlans[i].id, dietPlans[i].patientName);
    }
}
int main() { int
    choice; do {
```

```
        printf("\nDiet Plan Management System\n");

        printf("1. Add Diet Plan\n2. View Diet Plan\n3. Update Diet Plan\n4. Delete Diet Plan\n5. List All Diet Plans\n6. Exit\nEnter your choice: ");

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

        (choice) {

            case 1:

                addDietPlan();

                break;

            case 2:

                viewDietPlan();

                break;

            case 3:

                updateDietPlan(); break;

            case 4:

                deleteDietPlan();

                break;

            case 5:

                listAllDietPlans(); break;

            case 6:

                printf("Exiting system.\n");
```

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

return 0;
}
```

Problem 10: Surgery Scheduling System Description: Design a system for scheduling surgeries. Menu Options:

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

Requirements:

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

17. Apply nested unions for complex surgery entries. Sol: #include

<stdio.h>

#include <string.h>

#define MAX_SURGERIES 100

// Structure for surgery details typedef

struct {

    char surgeon[50]; char

    patient[50]; char

    date[20]; char

    time[10];

} SurgeryDetails;

// Union for optional surgery data typedef

union {

    char notes[100];

    int estimatedDuration; // Duration in minutes union {

        char roomNumber[10]; char

        assistantName[50];

    } nestedOptional;

} OptionalSurgeryData;

```
// Structure for surgery schedule typedef
struct {
    int id;
    const char surgeryType[30];
    SurgeryDetails details;
    OptionalSurgeryData optional;
} Surgery;
Surgery surgeries[MAX_SURGERIES]; int
surgeryCount = 0;
// Function to schedule a surgery void
scheduleSurgery() {
    if (surgeryCount >= MAX_SURGERIES) { printf("\nCannot
        schedule more surgeries.\n"); return;
    }
    Surgery *surgery = &surgeries[surgeryCount]; surgery->id
    = surgeryCount + 1;
    printf("Enter surgery type: ");
    scanf(" %29[^"]", surgery->surgeryType); printf("Enter
    surgeon's name: ");
    scanf(" %49[^"]", surgery->details.surgeon);
```

```
    printf("Enter patient's name: "); scanf(" %s",
    surgery->details.patient);
    printf("Enter date (YYYY-MM-DD): "); scanf(" %s",
    surgery->details.date); printf("Enter time (HH:MM):
    ");
    scanf(" %s", surgery->details.time);
    printf("Enter estimated duration in minutes (0 to skip): "); scanf("%d", &surgery-
    >optional.estimatedDuration);
    if (surgery->optional.estimatedDuration != 0) { printf("Enter
        room number: ");
        scanf(" %s", surgery->optional.nestedOptional.roomNumber);
    } else {
        printf("Enter assistant name: ");
        scanf(" %s", surgery->optional.nestedOptional.assistantName);
    }
    surgeryCount++;
    printf("Surgery scheduled successfully with ID: %d\n", surgery->id);
}
// Function to view a surgery schedule void
viewSurgerySchedule() {
    int id;
```

```
    printf("Enter surgery ID to view: "); scanf("%d", &id);
    if (id < 1 || id > surgeryCount) { printf("Invalid
        surgery ID.\n"); return;
    }
    Surgery *surgery = &surgeries[id - 1];
    printf("\nID: %d\nSurgery Type: %s\nSurgeon: %s\nPatient: %s\nDate:
%s\nTime: %s\n",
            surgery->id, surgery->surgeryType, surgery->details.surgeon, surgery->details.patient,
            surgery->details.date, surgery->details.time);
    if (surgery->optional.estimatedDuration != 0) {
                                printf("Estimated Duration: %d minutes\nRoom Number: %s\n", surgery-
                            >optional.estimatedDuration,  surgery->optional.nestedOptional.roomNumber);
    } else {
        printf("Assistant Name: %s\n", surgery-
>optional.nestedOptional.assistantName);
    }
}
// Function to update a surgery schedule void
updateSurgerySchedule() {
    int id;
    printf("Enter surgery ID to update: ");
```

```
    scanf("%d", &id);
    if (id < 1 || id > surgeryCount) { printf("Invalid
        surgery ID.\n"); return;
    }
    Surgery *surgery = &surgeries[id - 1]; int choice;
    do {
        printf("\nUpdate Menu:\n1. Surgeon\n2. Patient\n3. Date\n4. Time\n5.
Estimated Duration\n6. Nested Optional Data\n7. Exit\nEnter choice: "); scanf("%d", &choice);
        switch (choice) { case
            1:
                printf("Enter new surgeon's name: "); scanf(" %s",
                surgery->details.surgeon); break;
            case 2:
                printf("Enter new patient's name: "); scanf(" %s",
                surgery->details.patient); break;
            case 3:
                printf("Enter new date (YYYY-MM-DD): ");
```

```
    scanf(" %s", surgery->details.date); break;
case 4:
    printf("Enter new time (HH:MM): "); scanf("
    %s", surgery->details.time); break;
case 5:
    printf("Enter new estimated duration in minutes (0 to skip): "); scanf("%d", &surgery-
    >optional.estimatedDuration);
    break;
case 6:
    if (surgery->optional.estimatedDuration != 0) { printf("Enter new room
        number: ");
        scanf(" %s", surgery->optional.nestedOptional.roomNumber);
    } else {
        printf("Enter new assistant name: ");
        scanf(" %s", surgery->optional.nestedOptional.assistantName);
    }
    break;
case 7:
    printf("Exiting update menu.\n");
```

```
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 7);
}

// Function to cancel a surgery void
cancelSurgery() {
    int id;
    printf("Enter surgery ID to cancel: "); scanf("%d",
    &id);
    if (id < 1 || id > surgeryCount) { printf("Invalid
        surgery ID.\n"); return;
    }
    for (int i = id - 1; i < surgeryCount - 1; i++) { surgeries[i] =
        surgeries[i + 1];
    }
    surgeryCount--;
    printf("Surgery canceled successfully.\n");
```

```
}

// Function to list all surgeries void
listAllSurgeries() {
    if (surgeryCount == 0) {
        printf("No surgeries scheduled.\n"); return;
    }
    for (int i = 0; i < surgeryCount; i++) {
        printf("ID: %d, Surgery Type: %s, Surgeon: %s, Patient: %s\n", surgeries[i].id,
                surgeries[i].surgeryType, surgeries[i].details.surgeon,
surgeries[i].details.patient);
    }
}

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

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

```
        }
    } while (choice != 6);

    return 0;
}
```

Problem 11: Prescription Management System

Description: Develop a system to manage patient prescriptions.

Menu Options:

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

Requirements:

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

```
<stdio.h>

#include <stdlib.h>
```

```
#include <string.h>

#define MAX_PRESCRIPTIONS 100

// Structure for nested prescription details struct
PrescriptionDetail {
    char medicineName[50]; int
    quantity;
    float dosage;
};

// Union for optional prescription fields union
OptionalDetails {
    char notes[100]; int
    followUpDays;
};

// Nested union for complex prescription data struct
Prescription {
    int id;
    char patientName[50];
```

```
    struct PrescriptionDetail details; union
    OptionalDetails optional; int
    hasFollowUp;
};

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

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

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 your choice: "); scanf("%d",
&choice);

switch (choice) { case
    1:
        addPrescription(); break;
    case 2:
        viewPrescription(); break;
    case 3:
        updatePrescription(); break;
    case 4:
        deletePrescription();
```

```
            break;
        case 5:
            listAllPrescriptions(); break;
        case 6:
            printf("Exiting system.\n"); break;
        default:
            printf("Invalid choice. Try again.\n");
        }
    } while (choice != 6);

    return 0;
}

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

```
struct Prescription *p = &prescriptions[prescriptionCount]; p->id =
prescriptionCount + 1;

printf("Enter patient name: "); scanf("%s", p-
>patientName); printf("Enter medicine name: ");
scanf("%s", p->details.medicineName);
printf("Enter quantity: ");
scanf("%d", &p->details.quantity);
printf("Enter dosage: "); scanf("%f", &p-
>details.dosage);

printf("Is there a follow-up? (1 for Yes, 0 for No): "); scanf("%d", &p-
>hasFollowUp);

if (p->hasFollowUp) { printf("Enter follow-
    up days: ");
    scanf("%d", &p->optional.followUpDays);
} else {
    printf("Enter any notes: "); scanf("%s", p-
    >optional.notes);
```

```
    }

    prescriptionCount++;
    printf("Prescription added successfully.\n");
}

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

    if (id < 1 || id > prescriptionCount) { printf("Invalid
        prescription ID.\n"); return;
    }

    struct Prescription *p = &prescriptions[id - 1];
    printf("\nPrescription ID: %d\n", p->id); printf("Patient Name:
    %s\n", p->patientName);
    printf("Medicine Name: %s\n", p->details.medicineName); printf("Quantity: %d\n", p-
    >details.quantity);
```

```
    printf("Dosage: %.2f\n", p->details.dosage);

    if (p->hasFollowUp) {
        printf("Follow-Up Days: %d\n", p->optional.followUpDays);
    } else {
        printf("Notes: %s\n", p->optional.notes);
    }
}

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

    if (id < 1 || id > prescriptionCount) { printf("Invalid
        prescription ID.\n"); return;
    }

    struct Prescription *p = &prescriptions[id - 1]; printf("Enter
    new medicine name: ");
```

```
    scanf("%s", p->details.medicineName);
    printf("Enter new quantity: "); scanf("%d", &p-
    >details.quantity); printf("Enter new dosage: ");
    scanf("%f", &p->details.dosage);

    printf("Is there a follow-up? (1 for Yes, 0 for No): "); scanf("%d", &p-
    >hasFollowUp);

    if (p->hasFollowUp) { printf("Enter follow-
        up days: ");
        scanf("%d", &p->optional.followUpDays);
    } else {
        printf("Enter any notes: "); scanf("%s", p-
        >optional.notes);
    }

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

void deletePrescription() {
```

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

    if (id < 1 || id > prescriptionCount) { printf("Invalid
        prescription ID.\n"); return;
    }

    for (int i = id - 1; i < prescriptionCount - 1; i++) {
        prescriptions[i] = prescriptions[i + 1];
    }
    prescriptionCount--;
    printf("Prescription deleted successfully.\n");
}

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

```
    for (int i = 0; i < prescriptionCount; i++) { struct
        Prescription *p = &prescriptions[i]; printf("\nPrescription
        ID: %d\n", p->id);
        printf("Patient Name: %s\n", p->patientName); printf("Medicine Name:
        %s\n", p->details.medicineName); printf("Quantity: %d\n", p-
        >details.quantity); printf("Dosage: %.2f\n", p->details.dosage);


        if (p->hasFollowUp) {
            printf("Follow-Up Days: %d\n", p->optional.followUpDays);
        } else {
            printf("Notes: %s\n", p->optional.notes);
        }
    }
}
```

Problem 12: Doctor Consultation Management

Description: Create a system for managing doctor consultations.

Menu Options:

1. Schedule Consultation
2. View Consultation
3. Update Consultation
4. Cancel Consultation

5. List All Consultations
6. Exit

Requirements:

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

Sol: #include <stdio.h> #include

<string.h>

#define MAX_CONSULTATIONS 100

#define CONSULTATION_FEE 50.0

// Structure to hold consultation details typedef

struct {

char patient_name[50]; char

doctor_name[50]; char

date[20];

char time[10];

int is_active; // 1 for active consultation, 0 for canceled

} Consultation;

Consultation consultations[MAX_CONSULTATIONS]; int

consultation_count = 0;

```
// Function to schedule consultation void
scheduleConsultation() {
    if (consultation_count < MAX_CONSULTATIONS) { printf("Enter Patient
        Name: ");
        scanf(" %[^\n]s", consultations[consultation_count].patient_name); printf("Enter Doctor Name: ");
        scanf(" %[^\n]s", consultations[consultation_count].doctor_name); printf("Enter Date
        (DD/MM/YYYY): ");
        scanf(" %[^\n]s", consultations[consultation_count].date); printf("Enter Time
        (HH:MM): ");
        scanf(" %[^\n]s", consultations[consultation_count].time);
        consultations[consultation_count].is_active = 1; consultation_count++;
        printf("Consultation Scheduled Successfully.\n");
    } else {
        printf("Max number of consultations reached.\n");
    }
}

// Function to view consultation details
```

```
void viewConsultation() { char
    patient_name[50];
    printf("Enter Patient Name to View Consultation: "); scanf("
    %[^\n]s", patient_name);

    for (int i = 0; i < consultation_count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0 && consultations[i].is_active == 1) {
            printf("Consultation Details:\n");
            printf("Patient Name: %s\n", consultations[i].patient_name); printf("Doctor
            Name: %s\n", consultations[i].doctor_name); printf("Date: %s\n",
            consultations[i].date);
            printf("Time: %s\n", consultations[i].time); printf("Consultation Fee: %.2f\n",
            CONSULTATION_FEE); return;
        }
    }
    printf("Consultation not found for %s.\n", patient_name);
}

// Function to update consultation details void
updateConsultation() {
```

```
    char patient_name[50];
    printf("Enter Patient Name to Update Consultation: "); scanf(" %[^\n]s",
    patient_name);

    for (int i = 0; i < consultation_count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0 && consultations[i].is_active == 1) {
            printf("Enter New Doctor Name: ");
            scanf(" %[^\n]s", consultations[i].doctor_name); printf("Enter
            New Date (DD/MM/YYYY): "); scanf(" %[^\n]s",
            consultations[i].date); printf("Enter New Time (HH:MM): ");
            scanf(" %[^\n]s", consultations[i].time); printf("Consultation
            Updated Successfully.\n"); return;
        }
    }
    printf("Consultation not found for %s.\n", patient_name);
}

// Function to cancel consultation void
cancelConsultation() {
```

```
    char patient_name[50];
    printf("Enter Patient Name to Cancel Consultation: "); scanf(" %[^\n]s",
    patient_name);

    for (int i = 0; i < consultation_count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0 && consultations[i].is_active == 1) {
            consultations[i].is_active = 0;
            printf("Consultation for %s has been canceled.\n", patient_name); return;
        }
    }
    printf("Consultation not found for %s.\n", patient_name);
}

// Function to list all consultations void
listAllConsultations() {
    printf("All Consultations:\n");
    for (int i = 0; i < consultation_count; i++) { if
        (consultations[i].is_active == 1) {
            printf("Patient Name: %s\n", consultations[i].patient_name); printf("Doctor Name:
            %s\n", consultations[i].doctor_name);
```

```
            printf("Date: %s\n", consultations[i].date); printf("Time:
            %s\n", consultations[i].time);
            printf("Consultation Fee: %.2f\n\n", CONSULTATION_FEE);
        }
    }
}

// Main function to display menu and handle user input 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 your choice: "); scanf("%d",
        &choice);
```

```
switch (choice) { case
    1:
        scheduleConsultation(); break;
    case 2:
        viewConsultation(); break;
    case 3:
        updateConsultation(); break;
    case 4:
        cancelConsultation(); break;
    case 5:
        listAllConsultations(); break;
    case 6:
        printf("Exiting...\n"); break;
    default:
```

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

    return 0;
}
```

Problem:

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

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

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

int main()
{
```

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

    return 0;
}

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

```
        temp->data = A[i]; temp-
        >next = NULL; last->next
        = temp; last = temp;
    }

}

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

}

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

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

Problem 1: Patient Queue Management

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

1. Create a new patient queue.
2. Insert a patient into the queue.
3. Display the current queue of patients.

Sol:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> struct
PatientNode
{
char name[50];
```

```
struct PatientNode *next;
} *first = NULL;
// Function prototypes
void createPatientQueue(char names[][50], int n); void
displayPatientQueue(struct PatientNode *p);
void insertPatient(struct PatientNode *p, char name[]); int
main()
{
char patientNames[][50] = {"Nanditha M", "Niharika C L", "Shama M G"};
createPatientQueue(patientNames, 3);
printf("Initial patient queue:\n");
displayPatientQueue(first);
printf("\nAdding a new patient to the queue:\n");
insertPatient(first, "Ram");
displayPatientQueue(first);
return 0;
}
void createPatientQueue(char names[][50], int n)
{
int i;
struct PatientNode *temp, *last;
first = (struct PatientNode *)malloc(sizeof(struct PatientNode));
strcpy(first->name, names[0]);
```

```
first->next = NULL; last
= first;
for (i = 1; i < n; i++)
{
temp = (struct PatientNode *)malloc(sizeof(struct PatientNode));
strcpy(temp->name, names[i]);
temp->next = NULL;
last->next = temp; last
= temp;
}
}
void displayPatientQueue(struct PatientNode *p)
{
while (p != NULL)
{
printf("Name: %s\n", p->name); p =
p->next;
}
}
void insertPatient(struct PatientNode *p, char name[])
{
struct PatientNode *temp, *last = p;
temp = (struct PatientNode *)malloc(sizeof(struct PatientNode));
```

```
strcpy(temp->name, name);

temp->next = NULL;

while (last->next != NULL) last

= last->next;

last->next = temp;

}
```

Problem 2: Hospital Ward Allocation

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

1. Create a list of available beds.
2. Insert a patient into an available bed.
3. Display the current bed allocation.

Sol:

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>

// Define a structure for the bed

struct BedNode

{

int bedNumber;

char patientName[50];

struct BedNode *next;

} *first = NULL, *last = NULL;

// Function Prototypes

void createNode(int bedCount);

void displayBedAllocation(struct BedNode *p);
```

```
void allocateBed(struct BedNode *p, int bedNumber, char patientName[]); int
main()
{
}
int bedCount = 5;
createNode(bedCount); printf("Initial
Bed Allocation:\n");
displayBedAllocation(first);
printf("\nAllocating bed 2 to patient 'John Smith'\n");
allocateBed(first, 2, "John Smith");  printf("\nUpdated
Bed Allocation:\n"); displayBedAllocation(first);
return 0;
void createNode(int bedCount)
{
int i;
struct BedNode *temp;
first = (struct BedNode *)malloc(sizeof(struct BedNode)); first-
>bedNumber = 1;
strcpy(first->patientName, "Available");
first->next = NULL;
last = first;
for (i = 2; i <= bedCount; i++)
```

```
{
temp = (struct BedNode *)malloc(sizeof(struct BedNode));
temp->bedNumber = i;
strcpy(temp->patientName, "Available");
temp->next = NULL;
last->next = temp;
last = temp;
}
}
// Function to allocate a bed to a patient
void allocateBed(struct BedNode *p, int bedNumber, char patientName[])
{
while (p != NULL)
{
if (p->bedNumber == bedNumber && strcmp(p->patientName, "Available") == 0)
{
strcpy(p->patientName, patientName); // Assign the bed to the patient
printf("Bed %d allocated to %s\n", p->bedNumber, p->patientName);
return;
}
p = p->next;
}
// If the bed is not found or not available
```

```
printf("Bed %d is not available or invalid.\n", bedNumber);

}

// Function to display the current bed allocation void

displayBedAllocation(struct BedNode *p)

{

if (p == NULL)

{

printf("No beds have been created.\n");

return;

}

// Traverse through the list and display bed details printf("Current

Bed Allocation:\n");

while (p != NULL)

{

printf("Bed Number: %d, Patient: %s\n", p->bedNumber, p->patientName); p = p-

>next;

}

}
```

Problem 3: Medical Inventory Tracking

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

1. Create an inventory list.
2. Insert a new inventory item.
3. Display the current inventory.

Sol:

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

```
#include <stdlib.h>
#include <string.h>
struct InventoryNode
{
int itemID;
char itemName[50];
int quantity;
struct InventoryNode *next;
} *first = NULL;
// Function prototypes
void createInventoryList(int itemCount);
void displayInventory(struct InventoryNode *p);
void insertInventoryItem(struct InventoryNode *p, int itemID, char itemName[], int quantity); int
main()
{
}
int itemCount = 3;
createInventoryList(itemCount);
printf("Initial Inventory List:\n");
displayInventory(first);
printf("\nAdding a new inventory item:\n");
insertInventoryItem(first, 4, "Bandage", 200);
displayInventory(first);
```

```
return 0;
// Function to create an initial inventory list void
createInventoryList(int itemCount)
{
int i;
struct InventoryNode *temp, *last;
// Create first inventory item
first = (struct InventoryNode *)malloc(sizeof(struct InventoryNode)); first-
>itemID = 1;
strcpy(first->itemName, "Paracetamol");
first->quantity = 50;
first->next = NULL; last
= first;
// Create remaining inventory items for
(i = 2; i <= itemCount; i++)
{
temp = (struct InventoryNode *)malloc(sizeof(struct InventoryNode)); temp-
>itemID = i;
if (i == 2)
strcpy(temp->itemName, "Aspirin");
else
strcpy(temp->itemName, "Cough Syrup");
temp->quantity = 100;
```

```
temp->next = NULL;
last->next = temp; last
= temp;
}
}
void insertInventoryItem(struct InventoryNode *p, int itemID, char itemName[], int quantity)
{
struct InventoryNode *temp, *last = p;
while (last->next != NULL)
last = last->next;
temp = (struct InventoryNode *)malloc(sizeof(struct InventoryNode)); temp-
>itemID = itemID;
strcpy(temp->itemName, itemName);
temp->quantity = quantity;
temp->next = NULL;
last->next = temp;
}
// Function to display the current inventory list void
displayInventory(struct InventoryNode *p)
{
while (p != NULL)
{
printf("Item ID: %d, Item Name: %s, Quantity: %d\n", p->itemID, p->itemName, p->quantity);
```

```
p = p->next;

}

}
```

Problem 4: Doctor Appointment Scheduling

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

1. Create an appointment list.
2. Insert a new appointment.
3. Display all scheduled appointments.

Sol:

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>

// Define structure for appointments struct

AppointmentNode

{

char patientName[50]; // Name of the patient

char appointmentDate[20]; // Appointment date (e.g., "2025-01-15") char

appointmentTime[20]; // Appointment time (e.g., "10:30 AM") struct

AppointmentNode *next; // Pointer to the next appointment

} *first = NULL;

// Function prototypes

void createAppointmentList(int count);

void insertAppointment(struct AppointmentNode *p, char patientName[], char appointmentDate[],

char appointmentTime[]);

void displayAppointments(struct AppointmentNode *p);
```

```
int main()
{
int count = 3; createAppointmentList(count);
printf("Initial Appointment List:\n");
displayAppointments(first); printf("\nAdding
a new appointment:\n");
insertAppointment(first, "John Smith", "2025-01-20", "11:00 AM");
displayAppointments(first);
return 0;
}
// Function to create an initial appointment list void
createAppointmentList(int count)
{
int i;
struct AppointmentNode *temp, *last;
// Create the first appointment
first = (struct AppointmentNode *)malloc(sizeof(struct AppointmentNode));
strcpy(first->patientName, "Alice Brown");
strcpy(first->appointmentDate, "2025-01-18");
strcpy(first->appointmentTime, "9:30 AM");
first->next = NULL;
last = first;
```

```
// Create remaining appointments for
(i = 2; i <= count; i++)
{
temp = (struct AppointmentNode *)malloc(sizeof(struct AppointmentNode)); if (i ==
2)
{
strcpy(temp->patientName, "Bob White");
strcpy(temp->appointmentDate, "2025-01-19");
strcpy(temp->appointmentTime, "10:00 AM");
}
else
{
strcpy(temp->patientName, "Charlie Green");
strcpy(temp->appointmentDate, "2025-01-19");
strcpy(temp->appointmentTime, "10:30 AM");
}
temp->next = NULL;
last->next = temp; last
= temp;
}
}
// Function to insert a new appointment
void insertAppointment(struct AppointmentNode *p, char patientName[], char appointmentDate[],
```

```
char appointmentTime[])
{
struct AppointmentNode *temp, *last = p;
// Traverse to the last node while
(last->next != NULL) last = last-
>next;
// Create a new node for the new appointment
temp = (struct AppointmentNode *)malloc(sizeof(struct AppointmentNode));
strcpy(temp->patientName, patientName);
strcpy(temp->appointmentDate, appointmentDate);
strcpy(temp->appointmentTime, appointmentTime);
temp->next = NULL;
// Link the new node to the last node
last->next = temp;
}
// Function to display all scheduled appointments
void displayAppointments(struct AppointmentNode *p)
{
if (p == NULL)
{
printf("No appointments scheduled.\n");
return;
}
```

```
// Traverse through the list and display appointment details while

(p != NULL)

{

printf("Patient: %s, Date: %s, Time: %s\n", p->patientName, p->appointmentDate, p

>appointmentTime); p

= p->next;

}

}
```

Problem 5: Emergency Contact List

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

1. Create a contact list.
2. Insert a new contact.
3. Display all emergency contacts.

Sol:

```
#include <stdio.h>

#include <stdlib.h> #include

<string.h>


// Define structure for emergency contact

struct EmergencyContact

{

    char name[50];

    char phoneNumber[15];

    struct EmergencyContact *next;

} *first = NULL;
```

```
// Function prototypes
void createContactList(char contacts[][2][50], int n);
void insertContact(struct EmergencyContact *p, char name[], char phoneNumber[]); void
displayContacts(struct EmergencyContact *p);


int main()
{
    char emergencyContacts[][2][50] = {{"John Doe", "123-456-7890"}, {"Jane Smith", "987- 654-
3210"}};
    createContactList(emergencyContacts,       2);
    printf("Initial    emergency    contact    list:\n");
    displayContacts(first);
    printf("\nAdding a new emergency contact:\n");
    insertContact(first, "Alex Brown", "555-555-5555");
    displayContacts(first);
    return 0;
}


void createContactList(char contacts[][2][50], int n)
{
    int i;
    struct EmergencyContact *temp, *last;
    first = (struct EmergencyContact *)malloc(sizeof(struct EmergencyContact));
```

```
    strcpy(first->name, contacts[0][0]); strcpy(first-
    >phoneNumber, contacts[0][1]); first->next =
    NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct EmergencyContact *)malloc(sizeof(struct EmergencyContact));
        strcpy(temp->name, contacts[i][0]);
        strcpy(temp->phoneNumber, contacts[i][1]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertContact(struct EmergencyContact *p, char name[], char phoneNumber[])
{
    struct EmergencyContact *temp, *last = p;
    temp = (struct EmergencyContact *)malloc(sizeof(struct EmergencyContact));
    strcpy(temp->name, name);
    strcpy(temp->phoneNumber, phoneNumber);
    temp->next = NULL;
```

```
    while (last->next != NULL)
        last = last->next;
    last->next = temp;
}

void displayContacts(struct EmergencyContact *p)
{
    while (p != NULL)
    {
        printf("Name: %s, Phone: %s\n", p->name, p->phoneNumber); p = p-
        >next;
    }
}
```

Problem 6: Surgery Scheduling System

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

1. Create a surgery schedule.
2. Insert a new surgery into the schedule.
3. Display all scheduled surgeries.

Sol: #include <stdio.h> #include

<stdlib.h>

```
struct Surgery {
    int surgeryID;
    char patientName[50];
```

```
    char surgeryType[50];
    struct Surgery *next;
};

struct Surgery *first = NULL, *ptr;

void createSurgery() {
    struct Surgery *newnode = (struct Surgery *)malloc(sizeof(struct Surgery)); printf("Enter
    Surgery ID: ");
    scanf("%d", &newnode->surgeryID);
    printf("Enter Patient Name: ");
    scanf(" %s", newnode->patientName);
    printf("Enter Surgery Type: ");  scanf("
    %s", newnode->surgeryType); newnode-
    >next = NULL;

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

```
}

void displaySurgeries() { struct
    Surgery *temp = first; if (temp
    == NULL) {
        printf("No surgeries scheduled.\n");
        return;
    }

    printf("Scheduled Surgeries:\n");
    while (temp != NULL) {
        printf("ID: %d, Patient: %s, Surgery Type: %s -> ", temp->surgeryID, temp->patientName, temp-
>surgeryType);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() { int
    n;
    printf("Enter the number of surgeries to schedule: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        createSurgery();
```

```
    }

    displaySurgeries();

    return 0;

}
```

Problem 7: Patient History Record

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

1. Create a history record list.
2. Insert a new record.
3. Display all patient history records.

Sol: #include <stdio.h> #include

<stdlib.h>

```
struct HistoryRecord {

    int recordID;

    char patientName[50];

    char diagnosis[100];

    struct HistoryRecord *next;

};


struct HistoryRecord *first = NULL, *ptr;


void createHistoryRecord() {

    struct HistoryRecord *newnode = (struct HistoryRecord *)malloc(sizeof(struct HistoryRecord));

    printf("Enter Record ID: ");
```

```
    scanf("%d", &newnode->recordID);
    printf("Enter Patient Name: ");
    scanf(" %s", newnode->patientName);
    printf("Enter Diagnosis: ");
    scanf(" %s", newnode->diagnosis);
    newnode->next = NULL;

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

void displayHistoryRecords() {
    struct HistoryRecord *temp = first; if
    (temp == NULL) {
        printf("No patient history records available.\n");
        return;
    }
```

```
    printf("Patient History Records:\n");

    while (temp != NULL) {

        printf("Record ID: %d, Patient: %s, Diagnosis: %s -> ", temp->recordID, temp-
>patientName, temp->diagnosis);

        temp = temp->next;

    }

    printf("NULL\n");

}



int main() { int

    n;

    printf("Enter the number of history records to create: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        createHistoryRecord();

    }

    displayHistoryRecords();

    return 0;

}
```

Problem 8: Medical Test Tracking

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

1. Create a list of medical tests.
2. Insert a new test result.
3. Display all test results.

Sol: #include <stdio.h>

```
#include <stdlib.h>

struct MedicalTest { int
    testID;
    char patientName[50];
    char testName[50]; char
    result[100];
    struct MedicalTest *next;
};

struct MedicalTest *first = NULL, *ptr;

void createMedicalTest() {
    struct MedicalTest *newnode = (struct MedicalTest *)malloc(sizeof(struct MedicalTest)); printf("Enter
    Test ID: ");
    scanf("%d", &newnode->testID);
    printf("Enter Patient Name: ");
    scanf(" %s", newnode->patientName);
    printf("Enter Test Name: ");
    scanf(" %s", newnode->testName);
    printf("Enter Test Result: "); scanf("
    %s", newnode->result); newnode-
    >next = NULL;
```

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

void displayMedicalTests() {   struct
    MedicalTest *temp = first; if (temp
    == NULL) {
        printf("No medical test records available.\n"); return;
    }

    printf("Medical Test Records:\n");
    while (temp != NULL) {
        printf("Test ID: %d, Patient: %s, Test: %s, Result: %s -> ", temp->testID, temp-
>patientName, temp->testName, temp->result);
        temp = temp->next;
    }
    printf("NULL\n");
```

```
}


int main() { int
    n;
    printf("Enter the number of medical test records to create: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        createMedicalTest();
    }
    displayMedicalTests(); return
    0;
}
```

Problem 9: Prescription Management System

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

1. Create a prescription list.
2. Insert a new prescription.
3. Display all prescriptions.

Sol:

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


// Define structure for prescription struct
PrescriptionNode
{
```

```
    char patientName[50];
    char medication[50];
    char dosage[50];
    struct PrescriptionNode *next;
} *first = NULL;

// Function prototypes
void createPrescriptionList(char prescriptions[][3][50], int n);
void insertPrescription(struct PrescriptionNode *p, char patientName[], char medication[], char dosage[]);
void displayPrescriptions(struct PrescriptionNode *p);

int main()
{
    char prescriptions[][3][50] = {{"Alice Brown", "Paracetamol", "500mg"}, {"Bob White", "Aspirin",
"100mg"}};
    createPrescriptionList(prescriptions, 2);
    printf("Initial prescription list:\n");
    displayPrescriptions(first); printf("\nAdding a
    new prescription:\n");
    insertPrescription(first, "Charlie Green", "Cough Syrup", "10ml");
    displayPrescriptions(first);
    return 0;
}
```

```
void createPrescriptionList(char prescriptions[][3][50], int n)
{
    int i;
    struct PrescriptionNode *temp, *last;
    first = (struct PrescriptionNode *)malloc(sizeof(struct PrescriptionNode));
    strcpy(first->patientName, prescriptions[0][0]);
    strcpy(first->medication, prescriptions[0][1]);
    strcpy(first->dosage, prescriptions[0][2]); first-
    >next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct PrescriptionNode *)malloc(sizeof(struct PrescriptionNode));
        strcpy(temp->patientName, prescriptions[i][0]);
        strcpy(temp->medication, prescriptions[i][1]);
        strcpy(temp->dosage, prescriptions[i][2]); temp-
        >next = NULL;
        last->next = temp;
        last = temp;
    }
}
```

```
void insertPrescription(struct PrescriptionNode *p, char patientName[], char medication[], char dosage[])
{
    struct PrescriptionNode *temp, *last = p;
    temp = (struct PrescriptionNode *)malloc(sizeof(struct PrescriptionNode));
    strcpy(temp->patientName, patientName);
    strcpy(temp->medication, medication);
    strcpy(temp->dosage, dosage);
    temp->next = NULL;
    while (last->next != NULL)
        last = last->next;
    last->next = temp;
}

void displayPrescriptions(struct PrescriptionNode *p)
{
    while (p != NULL)
    {
        printf("Patient: %s, Medication: %s, Dosage: %s\n", p->patientName, p->medication, p-
>dosage);
        p = p->next;
    }
}
```