Structures

Problem 1: Patient Information Management System

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

Menu Options:

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

Requirements:

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

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

```
#define MAX_PATIENTS 50

typedef struct
{
  char doctor[50];
  char disease[50];
  char treatment[100];
} MedicalHistory;

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

typedef struct
{
   char name[50];
   int age;
   char address[100];
   char phone[20];
   MedicalHistory history;
   OptionalData optional;
   int op;
} Patient;

Patient *patients[MAX_PATIENTS];
int patientCount = 0;
```

```
// Function prototypes
void addPatient();
void viewPatientDetails();
void updatePatientInfo();
void deletePatientRecord();
void listAllPatients();

int main()
{
   int option;
   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 the option: ");
      scanf("%d", &option);

      switch(option)
      {
         case 1: addPatient();
                 break;
         case 2: viewPatientDetails();
```

```
            break;
        case 3: updatePatientInfo();
            break;
        case 4: deletePatientRecord();
            break;
        case 5: listAllPatients();
            break;
        case 6: printf("Exiting the system\n");
            break;
        default:printf("Invalid option\n");
            break;
      }
   } while(option != 6);
   for (int i = 0; i < patientCount; i++)
      free(patients[i]);
   return 0;
}

void addPatient()
{
   if(patientCount >= MAX_PATIENTS)
   {
      printf("MAximum patient reached\n");
      return;
   }
   Patient *newPatient = (Patient*)malloc(sizeof(Patient));
   printf("Enter the patient name: ");
```

```
scanf(" %[^\n]", newPatient->name);

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

printf("Enter the patient address: ");
scanf(" %[^\n]", newPatient->address);

printf("Enter the patient phone number: ");
scanf(" %[^\n]", newPatient->phone);

printf("Enter medical history - doctor: ");
scanf(" %[^\n]", newPatient->history.doctor);

printf("Enter medical history - disease: ");
scanf(" %[^\n]", newPatient->history.disease);

printf("Enter medical history - treatment: ");
scanf(" %[^\n]", newPatient->history.treatment);

printf("Add optional data? (1 for Yes, 0 for No): ");
scanf("%d", &newPatient->op);
if (newPatient->op)
{
   printf("Enter the allergies if any present: ");
   scanf(" %[^\n]", newPatient->optional.allergies);
}
```

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

void viewPatientDetails()
{
    char name[50];
    printf("Enter patient name to view the details: ");
    scanf(" %[^\n]", name);

    for(int i = 0; i < patientCount; i++)
    {
        if(strcmp(patients[i]->name, name) == 0)
        {
            printf("Patient Details\n");
            printf("Name: %[^\n]\n", patients[i]->name);
            printf("Age: %d\n", patients[i]->age);
            printf("Address: %[^\n]\n", patients[i]->address);
            printf("Phone number: %[^\n]\n", patients[i]->phone);
            printf("Medical History - \n");
            printf("Doctor: %[^\n]\n", patients[i]->history.doctor);
            printf("Disease: %[^\n]\n", patients[i]->history.disease);
            printf("Treatment: %[^\n]\n", patients[i]->history.treatment);
            if(patients[i]->op)
                printf("Optional data: %[^\n]\n", patients[i]->optional.allergies);
            return;
        }
```

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

void updatePatientInfo()
{
    char name[50];
    printf("Enter patient name to update the details: ");
    scanf(" %[^\n]", name);

    for(int i = 0; i < patientCount; i++)
    {
        if(strcmp(patients[i]->name, name) == 0)
        {
            printf("Enter the patient's new name: ");
            scanf("%[^\n]", patients[i]->name);

            printf("Enter the patient's new age: ");
            scanf(" %d", &patients[i]->age);

            printf("Enter the patient's new address: ");
            scanf(" %[^\n]", patients[i]->address);

            printf("Enter the patient's new phone number: ");
            scanf(" %[^\n]", patients[i]->phone);

            printf("Enter new medical history - doctor: ");
```

```
            scanf(" %[^\n]", patients[i]->history.doctor);

            printf("Enter new medical history - disease: ");
            scanf(" %[^\n]", patients[i]->history.disease);

            printf("Enter new medical history - treatment: ");
            scanf(" %[^\n]", patients[i]->history.treatment);

            printf("Update optional data? (1 for Yes, 0 for No): ");
            scanf("%d", &patients[i]->op);
            if (patients[i]->op)
            {
               printf("Enter the new allergies if any present: ");
               scanf(" %[^\n]", patients[i]->optional.allergies);
            }
            printf("Patient data updated successfully\n");
            return;
        }
    }
    printf("Patient not found\n");
}

void deletePatientRecord()
{
    char name[50];
    printf("Enter patient name to delete the details: ");
    scanf(" %[^\n]", name);
```

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

void listAllPatients()
{
    if (patientCount == 0)
    {
        printf("No patients found\n");
        return;
    }
    printf("\nList of all patients\n");
    for (int i = 0; i < patientCount; i++)
        printf("Name: %s  Age: %d", patients[i]->name, patients[i]->age);
}
```

Problem 2: Hospital Inventory Management

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

Menu Options:

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

Requirements:

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

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

#define MAX_ITEMS 100
```

```
typedef struct
{
    char manufacturer[50];
    char manufactureDate[20];
    char expiryDate[50];
} ItemDetails;

typedef union
{
    int quantity;
    float weight;
}Attributes;

typedef struct
{
    char name[50];
    char category[50];
    float price;
    ItemDetails details;
    Attributes attributes;
    int j; // 1-Quantity and 0-Weight
}InventoryItem;
InventoryItem *inventory[MAX_ITEMS];
int itemCount = 0;

// Function prototypes
void addInventoryItem();
```

```
void viewInventoryItem();
void updateInventoryItem();
void deleteInventoryItem();
void listAllInventoryItems();

int main()
{
   int option;
   do
   {
      printf("\n--- Hospital Inventory Management ---\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 the option: ");
      scanf("%d", &option);
      switch(option)
      {
         case 1: addInventoryItem(); break;
         case 2: viewInventoryItem(); break;
         case 3: updateInventoryItem(); break;
         case 4: deleteInventoryItem(); break;
         case 5: listAllInventoryItems(); break;
         case 6: printf("Exit the system\n"); break;
```

```
            default:printf("Onvalid option\n");
        }
    }
    while (option != 6);
    for (int i = 0; i < itemCount; i++)
        free(inventory[i]);
    return 0;
}

void addInventoryItem()
{
    if(itemCount >= MAX_ITEMS)
    {
        printf("MAximum items reached\n");
        return;
    }

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

    printf("Enter the item name: ");
    scanf(" %[^\n]", newInv->name);
    printf("Enter the category: ");
    scanf(" %[^\n]", newInv->category);

    printf("Enter the price: ");
    scanf(" %f", &newInv->price);
```

```
    printf("Enter the manufacturer: ");
    scanf(" %[^\n]", newInv->details.manufacturer);

    printf("Enter the manufacture date: ");
    scanf(" %[^\n]", newInv->details.manufactureDate);

    printf("Enter the expiry date: ");
    scanf(" %[^\n]", newInv->details.expiryDate);

    printf("Is item quantity based(1) or weight based(0): ");
    scanf(" %d", &newInv->j);

    if (newInv->j)
    {
        printf("Enter the quantity: ");
        scanf(" %d", &newInv->attributes.quantity);
    }
    else
    {
        printf("Enter the weight: ");
        scanf(" %f", &newInv->attributes.weight);
    }
    inventory[itemCount++] = newInv;
    printf("Item added successfully\n");
}

void viewInventoryItem()
```

```
{
    char name[50];
    printf("Enter item name to view the details: ");
    scanf(" %[^\n]", name);

    for(int i = 0; i < itemCount; i++)
    {
        if(strcmp(inventory[i]->name, name) == 0)
        {
            printf("Item Details\n");
            printf("Name: %s\n", inventory[i]->name);
            printf("Category: %s\n", inventory[i]->category);
            printf("Price: %.2f\n", inventory[i]->price);
            printf("Manufacturer: %s\n", inventory[i]->details.manufacturer);
            printf("Manufactuer Date: %s\n",
                                inventory[i]>details.manufactureDate);
            printf("Expiry Date: %s\n", inventory[i]->details.expiryDate);
            if(inventory[i]->j)
                printf("Quantity: %d\n", inventory[i]->attributes.quantity);
            else
                printf("Weight: %f\n", inventory[i]->attributes.weight);
            return;
        }
    }
    printf("Item not found\n");
}
```

```
void updateInventoryItem()
{
    char name[50];
    printf("Enter item name to update: ");
    scanf(" %[^\n]", name);

    for(int i = 0; i < itemCount; i++)
    {
        if(strcmp(inventory[i]->name, name) == 0)
        {
            printf("Enter the item's new name: ");
            scanf("%[^\n]", inventory[i]->name);

            printf("Enter the item's new category: ");
            scanf(" %[\n]", inventory[i]->category);

            printf("Enter the item's new price: ");
            scanf(" %f", &inventory[i]->price);

            printf("Enter the item's new manufacturer: ");
            scanf(" %[^\n]", inventory[i]->details.manufacturer);
            printf("Enter the item's new manufactuer date: ");
            scanf(" %[^\n]", inventory[i]->details.manufactureDate);

            printf("Enter the item's new expiry date: ");
            scanf(" %[^\n]", inventory[i]->details.expiryDate);
```

```
            printf("Is item quantity based(1) or weight based(0): ");
            scanf(" %d", &inventory[i]->j);
            if (inventory[i]->j)
            {
               printf("Enter the new quantity: ");
               scanf(" %d", &inventory[i]->attributes.quantity);
            }
            else
            {
               printf("Enter the new weight: ");
               scanf(" %f", &inventory[i]->attributes.weight);
            }
            printf("Item updated successfully\n");
            return;
         }
      }
      printf("Item not found\n");
}

void deleteInventoryItem()
{
   char name[50];
   printf("Enter item name to delete: ");
   scanf(" %[^\n]", name);
   for(int i = 0; i < itemCount; i++)
   {
      if(strcmp(inventory[i]->name, name) == 0)
```

```
        {
            free(inventory[i]);
            for (int j = i; j < itemCount - 1; j++)
                inventory[j] = inventory[j + 1];
            itemCount--;
            printf("Item deleted successfully\n");
            return;
        }
    }
    printf("Item not found\n");
}

void listAllInventoryItems()
{
    if (itemCount == 0)
    {
        printf("No items found\n");
        return;
    }
    printf("\nList of all inventory items\n");
    for (int i = 0; i < itemCount; i++)
        printf("Name: %s  Category: %s  Price: %.2f\n", inventory[i]->name,
            inventory[i]->category, inventory[i]->price);
}
```

Problem 3: Medical Appointment Scheduling System

Description: Develop a system to manage patient appointments.

Menu Options:

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

Requirements:

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

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

#define MAX_APPOINTMENTS 50
```

```
typedef struct
{
  char doctorName[50];
  char specialization[50];
} DoctorInfo;

typedef struct
{
   char patientName[50];
   int patientAge;
   char patientContact[20];
} PatientInfo;

typedef union
{
   char additionalSuggestions[200];
} OptionalData;

typedef struct
{
   int appointmentID;
   char date[15];
   char time[10];
   DoctorInfo doctor;
   PatientInfo patient;
   OptionalData optional;
   int m; // 1 for yes, 0 for no
```

```
} Appointment;

Appointment *appointments[MAX_APPOINTMENTS];
int appointmentCount = 0;

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

int main()
{
   int option;
   do
   {
      printf("\n--- Medical Appointment Scheduling System ---\n");
      printf("1. Schedule Appointment\n");
      printf("2. View Appointment\n");
      printf("3. Update Appointment\n");
      printf("4. Cancel Appointment\n");
      printf("5. List All Appointments\n");
      printf("6. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
```

```
        switch(option)
        {
            case 1: scheduleAppointment();
                    break;
            case 2: viewAppointment();
                    break;
            case 3: updateAppointment();
                    break;
            case 4: cancelAppointment();
                    break;
            case 5: listAllAppointments();
                    break;
            case 6: printf("Exiting the system\n");
                    break;
            default:printf("Invalid option\n");
                    break;
        }
    } while(option != 6);
    for (int i = 0; i < appointmentCount; i++)
        free(appointments[i]);
    return 0;
}

void scheduleAppointment()
{
    if(appointmentCount >= MAX_APPOINTMENTS)
    {
```

```
    printf("MAximum appointments reached\n");
    return;
}
Appointment*newAppointment=(Appointment*)
                                        malloc(sizeof(Appointment));

printf("Enter the appointment ID: ");
scanf(" %d", &newAppointment->appointmentID);

printf("Enter the appointment date: ");
scanf(" %[^\n]", newAppointment->date);

printf("Enter the appointment time: ");
scanf(" %[^\n]", newAppointment->time);

printf("Enter the doctor's name: ");
scanf(" %[^\n]", newAppointment->doctor.doctorName);

printf("Enter the doctor's specialization: ");
scanf(" %[^\n]", newAppointment->doctor.specialization);

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

printf("Enter patient's age: ");
scanf(" %d", &newAppointment->patient.patientAge);
```

```
    printf("Enter patient's contact number: ");
    scanf(" %[^\n]", newAppointment->patient.patientContact);

    printf("Add optional appointment? (1 for Yes, 0 for No): ");
    scanf(" %d", &newAppointment->m);
    if (newAppointment->m)
    {
        printf("Enter the additional suggestions: ");
        scanf(" %[^\n]", newAppointment->optional.additionalSuggestions);
    }

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

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

    for(int i = 0; i < appointmentCount; i++)
    {
        if(appointments[i]->appointmentID == id)
        {
            found = 1;
           printf("\nAppointment ID: %d\n",
```

```
                                appointments[i]->appointmentID);
            printf("Date: %s\n", appointments[i]->date);
            printf("Time: %s\n", appointments[i]->time);
            printf("Doctor's Name: %s\n",
                                appointments[i]->doctor.doctorName);
           printf("Doctor's Specialization: %s\n",
                                    appointments[i]->doctor.specialization);
           printf("Patient's Name: %s\n",
                                    appointments[i]->patient.patientName);
            printf("Patient's Age: %d\n", appointments[i]->patient.patientAge);
           printf("Patient's Contact: %s\n",
                                    appointments[i]->patient.patientContact);
             if (appointments[i]->m)
                    printf("Optional Data: %s\n",
                            appointments[i]->optional.additionalSuggestions);
        }
        break;
    }
    if(!found)
        printf("Appointment with ID %d not found\n", id);
}

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

```
for(int i = 0; i < appointmentCount; i++)
{
   if(appointments[i]->appointmentID == id)
   {
      found = 1;
      printf("Enter new appointment date: ");
      scanf(" %[^\n]", appointments[i]->date);

      printf("Enter new appointment time: ");
      scanf(" %[^\n]", appointments[i]->time);

      printf("Enter new doctor's name: ");
      scanf(" %[^\n]", appointments[i]->doctor.doctorName);

      printf("Enter new doctor's specialization: ");
      scanf(" %[^\n]", appointments[i]->doctor.specialization);

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

      printf("Enter new patient's age: ");
      scanf("%d", &appointments[i]->patient.patientAge);

      printf("Enter new patient's contact number: ");
      scanf(" %[^\n]", appointments[i]->patient.patientContact);

     printf("Update optional appointment data? (1 for Yes, 0 for No): ");
```

```
            scanf("%d", &appointments[i]->m);

            if (appointments[i]->m)
            {
               printf("Enter new additional suggestions: ");
               scanf(" %[^\n]",
                           appointments[i]->optional.additionalSuggestions);
            }
            printf("Appointment updated successfully\n");
            break;
         }
      }
      if(!found)
         printf("Appointment with ID %d not found\n", id);
}

void cancelAppointment()
{
   int id, found = 0;
   printf("Enter appointment ID to view the details: ");
   scanf(" %d", &id);
   for(int i = 0; i < appointmentCount; i++)
   {
      if(appointments[i]->appointmentID == id)
      {
         free(appointments[i]);
         for (int j = i; j < appointmentCount - 1; j++)
```

```
                appointments[j] = appointments[j + 1];
            appointmentCount--;
            printf("Appointment cancelled successfully\n");
            break;
        }
    }
    if(!found)
        printf("Appointment with ID %d not found\n", id);
}

void listAllAppointments()
{
    if (appointmentCount == 0)
    {
        printf("No appointmnets found\n");
        return;
    }
    printf("\nList of all appointmnets\n");
    for (int i = 0; i < appointmentCount; i++)
        printf("ID: %d  Date: %s  Time: %s  Doctor: %s  Patient: %s\n",
         appointments[i]->appointmentID,appointments[i]->date,
         appointments[i]->time,appointments[i]->doctor.doctorName,
         appointments[i]->patient.patientName);
}
```

Problem 4: Patient Billing System

Description: Create a billing system for patients.

Menu Options:

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

Requirements:

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

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

#define MAX_BILLS 100
const float CONSULTATION_FEE = 500.0;
const float ROOM_CHARGE_PER_DAY = 2000.0;
```

```
const float MEDICINE_TAX = 0.1; // 10% tax

typedef struct
{
   char medicineName[50];
   float medicineCost;
   int quantity;
} MedicineDetails;

typedef struct
{
   float consultationFee;
   int daysInRoom;
   float roomCharges;
   MedicineDetails medicines[10];
   int medicineCount;
} BillingBreakdown;

typedef union
{
   float discount;
} OptionalBillingData;

typedef struct
{
   int billID;
   char patientName[50];
```

```
    char billingDate[15];
    BillingBreakdown breakdown;
    OptionalBillingData optionalData;
    int m; // 1 for yes, 0 for no
    float totalBill;
} PatientBill;

PatientBill *bills[MAX_BILLS];
int billCount = 0;

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

int main()
{
    int option;
    printf("\n--- Patient Billing System ---\n");
    printf("Consultation Fee: %.2f, Room Charge per Day: %.2f, Medicine
Tax: %.2f%%\n",
                    CONSULTATION_FEE,    ROOM_CHARGE_PER_DAY,
MEDICINE_TAX * 100);
    do
    {
        printf("\n1. 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 the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: generateBill(); break;
            case 2: viewBill(); break;
            case 3: updateBill(); break;
            case 4: deleteBill(); break;
            case 5: listAllBills(); break;
            case 6: printf("Exit the system\n"); break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    for (int i = 0; i < billCount; i++)
        free(bills[i]);
    return 0;
}

void generateBill()
{
    if (billCount >= MAX_BILLS)
    {
        printf("Maximum limit reached\n");
```

```
    return;
}

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

printf("Enter bill ID: ");
scanf("%d", &newBill->billID);

printf("Enter patient name: ");
scanf(" %[^\n]", newBill->patientName);

printf("Enter billing date: ");
scanf(" %[^\n]",newBill->billingDate);

newBill->breakdown.consultationFee = CONSULTATION_FEE;

printf("Enter number of days in room: ");
scanf("%d", &newBill->breakdown.daysInRoom);

newBill->breakdown.roomCharges =
   newBill->breakdown.daysInRoom * ROOM_CHARGE_PER_DAY;

printf("Enter number of medicines: ");
scanf("%d", &newBill->breakdown.medicineCount);

for (int i = 0; i < newBill->breakdown.medicineCount; i++)
{
```

```
    printf("Enter medicine name: ");
    scanf(" %[^\n]", newBill->breakdown.medicines[i].medicineName);

    printf("Enter medicine cost: ");
    scanf("%f", &newBill->breakdown.medicines[i].medicineCost);

    printf("Enter quantity: ");
    scanf("%d", &newBill->breakdown.medicines[i].quantity);
}

printf("Additional billing data? (1 for Yes, 0 for No): ");
scanf("%d", &newBill->m);

if (newBill->m)
{
    printf("Enter discount percentage: ");
    scanf("%f", &newBill->optionalData.discount);
}

float medicineTotal = 0.0;
for (int i = 0; i < newBill->breakdown.medicineCount; i++)
{
    medicineTotal += newBill->breakdown.medicines[i].medicineCost *
                     newBill->breakdown.medicines[i].quantity;
}
medicineTotal += medicineTotal * MEDICINE_TAX;
```

```
    newBill->totalBill =
        newBill->breakdown.consultationFee +
        newBill->breakdown.roomCharges + medicineTotal;

    if (!newBill->m)
    {
        newBill->totalBill -=
            (newBill->optionalData.discount / 100.0) *
            newBill->totalBill;
    }

    bills[billCount++] = 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 < billCount; i++)
    {
        if (bills[i]->billID == id)
        {
            found = 1;
            printf("\nBill ID: %d\n", bills[i]->billID);
            printf("Patient Name: %s\n", bills[i]->patientName);
```

```
            printf("Billing Date: %s\n", bills[i]->billingDate);
            printf("Consultation Fee: %.2f\n",
                        bills[i]->breakdown.consultationFee);
            printf("Room Charges: %.2f\n",
                        bills[i]->breakdown.roomCharges);
            printf("Medicines:\n");
            for (int j = 0; j < bills[i]->breakdown.medicineCount; j++)
            {
               printf("- %s (Cost: %.2f, Quantity: %d)\n",
                     bills[i]->breakdown.medicines[j].medicineName,
                     bills[i]->breakdown.medicines[j].medicineCost,
                     bills[i]->breakdown.medicines[j].quantity);
            }
            if (bills[i]->m)
                  printf("Discount Applied: %.2f%%\n",
                              bills[i]->optionalData.discount);
            printf("Total Bill: %.2f\n", bills[i]->totalBill);
            break;
         }
      }
      if (!found)
         printf("Bill with ID %d not found\n", id);
}

void updateBill()
{
   int id, found = 0;
```

```
printf("Enter bill ID to update: ");
scanf("%d", &id);
for (int i = 0; i < billCount; i++)
{
   if (bills[i]->billID == id)
   {
      found = 1;
      printf("Enter new patient's name: ");
      scanf(" %[^\n]", bills[i]->patientName);

      printf("Enter new billing date: ");
      scanf(" %[^\n]", bills[i]->billingDate);

      printf("Enter new number of days in room: ");
      scanf("%d", &bills[i]->breakdown.daysInRoom);

      bills[i]->breakdown.roomCharges =
     bills[i]->breakdown.daysInRoom * ROOM_CHARGE_PER_DAY;

      printf("Enter new number of medicines: ");
      scanf("%d", &bills[i]->breakdown.medicineCount);

      for (int j = 0; j < bills[i]->breakdown.medicineCount; j++)
      {
         printf("Enter medicine name: ");
         scanf(" %[^\n]",
                  bills[i]->breakdown.medicines[j].medicineName);
```

```
            printf("Enter medicine cost: ");
            scanf("%f", &bills[i]->breakdown.medicines[j].medicineCost);

            printf("Enter quantity: ");
            scanf("%d", &bills[i]->breakdown.medicines[j].quantity);
        }

        printf("Additional data? (1 for Yes, 0 for No): ");
        scanf("%d", &bills[i]->m);
        if (bills[i]->m)
        {
            printf("Enter new discount percentage: ");
            scanf("%f", &bills[i]->optionalData.discount);
        }
        printf("Bill updated successfully\n");
        break;
      }
   }
   if (!found)
      printf("Bill with ID %d not found\n", id);
}

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

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

void listAllBills()
{
    if (billCount == 0)
    {
        printf("No bills found\n");
        return;
    }
    printf("\nAll bills\n");
    for (int i = 0; i < billCount; i++)
        printf("ID: %d  Patient: %s  Total: %.2f\n",
```

```
            bills[i]->billID, bills[i]->patientName, bills[i]->totalBill);
    }
```

Problem 5: Medical Test Result Management

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

Menu Options:

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

Requirements:

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

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

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

#define MAX_TESTS 100

typedef struct
{
   float bloodPressure;
   float heartRate;
} TestResultsDetails;

typedef union
{
   float discount;
} OptionalTestData;

typedef struct
{
   int testID;
   char patientName[50];
   char testDate[15];
   TestResultsDetails results;
   OptionalTestData optionalData;
   int d; // 1 for yes, 0 for no
} PatientTestResult;

PatientTestResult *testResults[MAX_TESTS];
```

```
int testCount = 0;

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

int main()
{
   int option;
   do
   {
      printf("\n--- Medical Test Result Management System ---\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 the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: addTestResult(); break;
         case 2: viewTestResult(); break;
```

```
            case 3: updateTestResult(); break;
            case 4: deleteTestResult(); break;
            case 5: listAllTestResults(); break;
            case 6: printf("Exit the system\n"); break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);

    for (int i = 0; i < testCount; i++)
        free(testResults[i]);
    return 0;
}

// Function to add a test result
void addTestResult()
{
    if (testCount >= MAX_TESTS)
    {
        printf("Test result limit reached\n");
        return;
    }
   PatientTestResult *newTest = (PatientTestResult *)
                          malloc(sizeof(PatientTestResult));
    printf("Enter test ID: ");
    scanf("%d", &newTest->testID);

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

```
    scanf(" %[^\n]", newTest->patientName);

    printf("Enter test date: ");
    scanf(" %[^\n]", newTest->testDate);

    printf("Enter blood pressure: ");
    scanf("%f", &newTest->results.bloodPressure);

    printf("Enter heart rate: ");
    scanf("%f", &newTest->results.heartRate);

    printf("Does this test have a discount? (1 for Yes, 0 for No): ");
    scanf("%d", &newTest->d);
    if (newTest->d)
    {
        printf("Enter discount percentage: ");
        scanf("%f", &newTest->optionalData.discount);
    }
    testResults[testCount++] = newTest;
    printf("Test result added successfully\n");
}

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

```
    for (int i = 0; i < testCount; i++)
    {
        if (testResults[i]->testID == testID)
        {
            found = 1;
            printf("\nTest ID: %d\n", testResults[i]->testID);
            printf("Patient Name: %s\n", testResults[i]->patientName);
            printf("Test Date: %s\n", testResults[i]->testDate);
            printf("Blood Pressure: %.2f\n",
                            testResults[i]->results.bloodPressure);
            printf("Heart Rate: %.2f\n", testResults[i]->results.heartRate);
            if (testResults[i]->d)
                printf("Discount Applied: %.2f%%\n",
                            testResults[i]->optionalData.discount);
            break;
        }
    }
    if (!found)
        printf("Test result with ID %d not found\n", testID);
}

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

```
    for (int i = 0; i < testCount; i++)
    {
        if (testResults[i]->testID == testID)
        {
            found = 1;
            printf("Enter new patient name: ");
            scanf(" %[^\n]", testResults[i]->patientName);

            printf("Enter new test date: ");
            scanf(" %[^\n]", testResults[i]->testDate);

            printf("Enter new blood pressure: ");
            scanf("%f", &testResults[i]->results.bloodPressure);

            printf("Enter new heart rate: ");
            scanf("%f", &testResults[i]->results.heartRate);

             printf("Does this test have a new discount? (1 for Yes, 0 for No):
");
            scanf("%d", &testResults[i]->d);
            if (testResults[i]->d)
            {
                printf("Enter new discount percentage: ");
                scanf("%f", &testResults[i]->optionalData.discount);
            }
            printf("Test result updated successfully\n");
            break;
        }
```

```
    }
    if (!found)
        printf("Test result with ID %d not found\n", testID);
}

void deleteTestResult()
{
    int testID, found = 0;
    printf("Enter test ID to delete: ");
    scanf("%d", &testID);
    for (int i = 0; i < testCount; i++)
    {
        if (testResults[i]->testID == testID)
        {
            found = 1;
            free(testResults[i]);
            for (int j = i; j < testCount - 1; j++)
                testResults[j] = testResults[j + 1];
            testCount--;
            printf("Test result deleted successfully\n");
            break;
        }
    }
    if (!found)
        printf("Test result with ID %d not found.\n", testID);
}
```

```
void listAllTestResults()
{
   if (testCount == 0)
   {
      printf("No test results found\n");
      return;
   }
   printf("\nAll Test Results\n");
   for (int i = 0; i < testCount; i++)
       printf("Test ID: %d  Patient: %s  Blood Pressure: %.2f  Heart Rate: %.2f\n",testResults[i]->testID, testResults[i]->patientName,
testResults[i]->results.bloodPressure, testResults[i]->results.heartRate);
}
```

Problem 6: Staff Duty Roster Management

Description: Create a system to manage hospital staff duty rosters

Menu Options:

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

Requirements:

7. Use variables for staff details.
8. Apply static and const for fixed shift timings.
9. Implement switch case for roster operations.

10.Utilize loops for roster generation.

11.Use pointers for dynamic staff data.

12.Create functions for roster management.

13.Use arrays for storing staff schedules.

14.Define structures for duty details.

15.Employ nested structures for detailed duty breakdowns.

16.Use unions for optional duty attributes.

17.Apply nested unions for complex duty data.

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

#define MAX_STAFF 100
const char *MORNING_SHIFT = "08:00 - 16:00";
const char *EVENING_SHIFT = "16:00 - 00:00";
const char *NIGHT_SHIFT = "00:00 - 08:00";

typedef struct
{
   char shift[20];
   char department[50];
} DutyDetails;

typedef union
{
   float overtime;
```

```
    int leaveDays;
} OptionalDutyAttributes;

typedef struct
{
    int staffID;
    char staffName[50];
    DutyDetails duty;
    OptionalDutyAttributes optional;
    int m; // 1 for overtime, 0 for leave
} StaffDutyRoster;

StaffDutyRoster *staffRosters[MAX_STAFF];
int staffCount = 0;
// Function prototypes
void addDutyRoster();
void viewDutyRoster();
void updateDutyRoster();
void deleteDutyRoster();
void listAllDutyRosters();

int main()
{
    int option;
    do
    {
        printf("\n--- Staff Duty Roster Management ---\n");
```

```
        printf("1. Add Duty Roster\n");
        printf("2. View Duty Roster\n");
        printf("3. Update Duty Roster\n");
        printf("4. Delete Duty Roster\n");
        printf("5. List All Duty Rosters\n");
        printf("6. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: addDutyRoster(); break;
            case 2: viewDutyRoster(); break;
            case 3: updateDutyRoster(); break;
            case 4: deleteDutyRoster(); break;
            case 5: listAllDutyRosters(); break;
            case 6: printf("Exiting the system...\n"); break;
            default: printf("Invalid option!\n");
        }
    } while (option != 6);

    for (int i = 0; i < staffCount; i++)
        free(staffRosters[i]);
    return 0;
}

// Function to add a duty roster
void addDutyRoster()
```

```
{
    if (staffCount >= MAX_STAFF)
    {
        printf("Staff limit reached\n");
        return;
    }
  StaffDutyRoster *newRoster =
                        (StaffDutyRoster *)malloc(sizeof(StaffDutyRoster));

    printf("Enter staff ID: ");
    scanf("%d", &newRoster->staffID);
    printf("Enter staff name: ");
    scanf(" %[^\n]", newRoster->staffName);

    printf("Enter shift (1 for Morning, 2 for Evening, 3 for Night): ");
    int shiftOption;
    scanf("%d", &shiftOption);
    if (shiftOption == 1)
        strcpy(newRoster->duty.shift, MORNING_SHIFT);
    else if (shiftOption == 2)
        strcpy(newRoster->duty.shift, EVENING_SHIFT);
    else if (shiftOption == 3)
        strcpy(newRoster->duty.shift, NIGHT_SHIFT);
    else
    {
        printf("Invalid shift option, setting to morning shift\n");
        strcpy(newRoster->duty.shift, MORNING_SHIFT);
```

```
    }

    printf("Enter department: ");
    scanf(" %[^\n]", newRoster->duty.department);

    printf("Does this staff have overtime or leave? (1 for overtime, 0 for leave):
");
    scanf("%d", &newRoster->m);
    if (newRoster->m)
    {
        printf("Enter overtime hours: ");
        scanf("%f", &newRoster->optional.overtime);
    }
    else
    {
        printf("Enter leave days: ");
        scanf("%d", &newRoster->optional.leaveDays);
    }
    staffRosters[staffCount++] = newRoster;
    printf("Duty roster added successfully\n");
}

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

```
    for (int i = 0; i < staffCount; i++)
    {
        if (staffRosters[i]->staffID == staffID)
        {
            found = 1;
            printf("\nStaff ID: %d\n", staffRosters[i]->staffID);
            printf("Staff Name: %s\n", staffRosters[i]->staffName);
            printf("Shift: %s\n", staffRosters[i]->duty.shift);
            printf("Department: %s\n", staffRosters[i]->duty.department);
            if (staffRosters[i]->m)
                        printf("Overtime Hours: %.2f\n",
                                    staffRosters[i]->optional.overtime);
            else
                printf("Leave Days: %d\n", staffRosters[i]->optional.leaveDays);
            break;
        }
    }
    if (!found)
        printf("Staff with ID %d not found\n", staffID);
}

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

```
    for (int i = 0; i < staffCount; i++)
    {
        if (staffRosters[i]->staffID == staffID)
        {
            found = 1;
            printf("Enter new staff name: ");
            scanf(" %[^\n]", staffRosters[i]->staffName);
            printf("Enter new shift (1 for Morning, 2 for Evening, 3 for Night): ");
            int shiftOption;
            scanf("%d", &shiftOption);
            if (shiftOption == 1)
                strcpy(staffRosters[i]->duty.shift, MORNING_SHIFT);
            else if (shiftOption == 2)
                strcpy(staffRosters[i]->duty.shift, EVENING_SHIFT);
            else if (shiftOption == 3)
                strcpy(staffRosters[i]->duty.shift, NIGHT_SHIFT);
            else
            {
                printf("Invalid shift option, setting to morning shift\n");
                strcpy(staffRosters[i]->duty.shift, MORNING_SHIFT);
            }

            printf("Enter new department: ");
            scanf(" %[^\n]", staffRosters[i]->duty.department);

             printf("Does this staff have overtime or leave? (1 for overtime, 0 for
leave): ");
            scanf("%d", &staffRosters[i]->m);
```

```
            if (staffRosters[i]->m)
            {
                printf("Enter new overtime hours: ");
                scanf("%f", &staffRosters[i]->optional.overtime);
            }
            else
            {
                printf("Enter new leave days: ");
                scanf("%d", &staffRosters[i]->optional.leaveDays);
            }
            printf("Duty roster updated successfully\n");
            break;
        }
    }
    if (!found)
        printf("Staff with ID %d not found\n", staffID);
}

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

    for (int i = 0; i < staffCount; i++)
    {
        if (staffRosters[i]->staffID == staffID)
```

```
        {
            found = 1;
            free(staffRosters[i]);
            for (int j = i; j < staffCount - 1; j++)
                staffRosters[j] = staffRosters[j + 1];
            staffCount--;
            printf("Duty roster deleted successfully\n");
            break;
        }
    }
    if (!found)
        printf("Staff with ID %d not found\n", staffID);
}

void listAllDutyRosters()
{
    if (staffCount == 0)
    {
        printf("No duty rosters found\n");
        return;
    }
    printf("\nAll Duty Rosters\n");
    for (int i = 0; i < staffCount; i++)
        printf("Staff ID: %d  Name: %s  Shift: %s  Department: %s\n",
            staffRosters[i]->staffID, staffRosters[i]->staffName,
            staffRosters[i]->duty.shift, staffRosters[i]->duty.department);
}
```

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 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:

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

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

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

struct BedNode
{
    char patientName[50];
    int bedNumber;
    struct BedNode *next;
} *first = NULL;

// Function prototypes
void createBeds(int totalBeds);
void allocateBed(int bedNumber, char patientName[]);
void displayBedAllocation();

int main()
{
    createBeds(3);
    printf("Initial bed allocation:\n");
    displayBedAllocation();
    printf("\nAllocating beds:\n");
    allocateBed(2, "Nanditha");
    allocateBed(1, "Monisha");
    printf("\nUpdated bed allocation:\n");
    displayBedAllocation();
    return 0;
}
```

```
void createBeds(int totalBeds)
{
    struct BedNode *temp, *last;
    first = (struct BedNode *)malloc(sizeof(struct BedNode));
    first->bedNumber = 1;
    strcpy(first->patientName, "Available");
    first->next = NULL;
    last = first;

    for (int i = 2; i <= totalBeds; i++)
    {
        temp = (struct BedNode *)malloc(sizeof(struct BedNode));
        temp->bedNumber = i;
        strcpy(temp->patientName, "Available");
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void allocateBed(int bedNumber, char patientName[])
{
    struct BedNode *p = first;
    while (p != NULL)
    {
        if (p->bedNumber == bedNumber)
        {
```

```
            if (strcmp(p->patientName, "Available") == 0)
            {
                strcpy(p->patientName, patientName);
                printf("Bed %d allocated to %s\n", bedNumber, patientName);
            }
            else
                printf("Bed %d is already occupied by %s\n", bedNumber, p->patientName);
            return;
        }
        p = p->next;
    }
    printf("Bed %d does not exist\n", bedNumber);
}

void displayBedAllocation()
{
    struct BedNode *p = first;
    while (p != NULL)
    {
        printf("Bed %d: %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:

- Create an inventory list.
- Insert a new inventory item.
- Display the current inventory.

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

struct InventoryNode
{
   char itemName[50];
   int quantity;
   struct InventoryNode *next;
} *first = NULL;

// Function prototypes
void createInventory(char items[][50], int quantities[], int n);
void insertInventoryItem(char itemName[], int quantity);
void displayInventory();

int main()
{
   char itemNames[][50] = {"Paracetamol", "Cough Syrup"};
   int quantities[] = {100, 30};
```

```
    createInventory(itemNames, quantities, 2);

    printf("Initial inventory\n");
    displayInventory();

    printf("\nAdding a new item to the inventory:\n");
    insertInventoryItem("Antibiotic", 20);

    printf("\nUpdated inventory:\n");
    displayInventory();

    return 0;
}

void createInventory(char items[][50], int quantities[], int n)
{
    struct InventoryNode *temp, *last;
    first = (struct InventoryNode *)malloc(sizeof(struct InventoryNode));
    strcpy(first->itemName, items[0]);
    first->quantity = quantities[0];
    first->next = NULL;
    last = first;

    for (int i = 1; i < n; i++)
    {
        temp = (struct InventoryNode *)malloc(sizeof(struct InventoryNode));
        strcpy(temp->itemName, items[i]);
```

```
        temp->quantity = quantities[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertInventoryItem(char itemName[], int quantity)
{
    struct InventoryNode *temp, *last = first;
    temp = (struct InventoryNode *)malloc(sizeof(struct InventoryNode));
    strcpy(temp->itemName, itemName);
    temp->quantity = quantity;
    temp->next = NULL;

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

    printf("Item '%s' added to the inventory\n", itemName);
}

void displayInventory()
{
    struct InventoryNode *p = first;
    printf("Item Name\t\tQuantity\n");
    while (p != NULL)
```

```
    {
        printf("%s\t\t%d\n", p->itemName, p->quantity);
        p = p->next;
    }
}
```

Problem 4: Doctor Appointment Scheduling

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

- Create an appointment list.
- Insert a new appointment.
- Display all scheduled appointments.

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

struct Appointment
{
    int appointmentID;
    char patientName[50];
    char appointmentTime[20];
    struct Appointment *next;
} *first = NULL;

// Function prototypes
void createAppointmentList(int appointmentIDs[], char patientNames[][50], char appointmentTimes[][20], int n);
```

```
void insertAppointment(int index, int appointmentID, char patientName[], char appointmentTime[]);
void displayAppointments();

int main()
{
   int appointmentIDs[] = {1, 2, 3, 4};
   char patientNames[][50] = {"XYZ", "ABC", "MNO", "PQR"};
    char appointmentTimes[][20] = {"11:00 AM", "12:00 AM", "5:00 PM", "6:00 PM"};

   createAppointmentList(appointmentIDs,
                                    patientNames, appointmentTimes, 4);

   printf("Initial scheduled appointments:\n");
   displayAppointments();

   printf("\nInserting a new appointment:\n");
   insertAppointment(2, 5, "UVW", "4:30 PM");

   printf("\nUpdated scheduled appointments:\n");
   displayAppointments();

   return 0;
}

void createAppointmentList(int appointmentIDs[], char patientNames[][50], char appointmentTimes[][20], int n)
```

```
{
    int i;
    struct Appointment *temp, *last;
    first = (struct Appointment *)malloc(sizeof(struct Appointment));
    first->appointmentID = appointmentIDs[0];
    strcpy(first->patientName, patientNames[0]);
    strcpy(first->appointmentTime, appointmentTimes[0]);
    first->next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct Appointment *)malloc(sizeof(struct Appointment));
        temp->appointmentID = appointmentIDs[i];
        strcpy(temp->patientName, patientNames[i]);
        strcpy(temp->appointmentTime, appointmentTimes[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertAppointment(int index, int appointmentID, char patientName[],
char appointmentTime[])
{
    struct Appointment *temp, *p = first;
    int i;
    temp = (struct Appointment *)malloc(sizeof(struct Appointment));
```

```
    temp->appointmentID = appointmentID;
    strcpy(temp->patientName, patientName);
    strcpy(temp->appointmentTime, appointmentTime);
    for (i = 0; i < index - 1 && p != NULL; i++)
        p = p->next;
    if (p != NULL)
    {
        temp->next = p->next;
        p->next = temp;
    }
    else
        printf("Invalid index\n");
}

void displayAppointments()
{
    struct Appointment *p = first;
    while (p != NULL)
    {
            printf("Appointment ID: %d Patient: %s Time: %s\n", p->appointmentID, p->patientName, p->appointmentTime);
        p = p->next;
    }
}
```

Problem 5: Emergency Contact List

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

- Create a contact list.
- Insert a new contact.
- Display all emergency contacts.

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

struct Contact
{
   int contactID;
   char name[50];
   char phoneNumber[15];
   struct Contact *next;
} *first = NULL;

// Function prototypes
void createContactList(int contactIDs[], char names[][50], char phoneNumbers[][15], int n);
void insertContact(int index, int contactID, char name[], char phoneNumber[]);
void displayContacts();

int main()
{
```

```
    int contactIDs[] = {1, 2, 3};
    char names[][50] = {"ABC", "DEF", "GHI"};
        char phoneNumbers[][15] = {"1234567891", "2345678912", "3456789123"};

    createContactList(contactIDs, names, phoneNumbers, 3);

    printf("Initial emergency contact list:\n");
    displayContacts();

    printf("\nInserting a new emergency contact at the middle:\n");
    insertContact(2, 4, "JKL", "4567891234");

    printf("\nUpdated emergency contact list:\n");
    displayContacts();

    return 0;
}

void createContactList(int contactIDs[], char names[][50], char phoneNumbers[][15], int n)
{
    int i;
    struct Contact *temp, *last;
    first = (struct Contact *)malloc(sizeof(struct Contact));
    first->contactID = contactIDs[0];
    strcpy(first->name, names[0]);
    strcpy(first->phoneNumber, phoneNumbers[0]);
```

```
    first->next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct Contact *)malloc(sizeof(struct Contact));
        temp->contactID = contactIDs[i];
        strcpy(temp->name, names[i]);
        strcpy(temp->phoneNumber, phoneNumbers[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertContact(int index, int contactID, char name[], char phoneNumber[])
{
    struct Contact *temp, *p = first;
    int i;
    temp = (struct Contact *)malloc(sizeof(struct Contact));
    temp->contactID = contactID;
    strcpy(temp->name, name);
    strcpy(temp->phoneNumber, phoneNumber);
    for (i = 0; i < index - 1 && p != NULL; i++)
        p = p->next;
    if (p != NULL)
    {
```

```
        temp->next = p->next;
        p->next = temp;
    }
    else
        printf("Invalid index\n");
}

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

Problem 6: Surgery Scheduling System

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

- Create a surgery schedule.
- Insert a new surgery into the schedule.
- Display all scheduled surgeries.

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

```
struct Surgery
{
   int surgeryID;
   char patientName[50];
   char surgeonName[50];
   char surgeryTime[20];
   struct Surgery *next;
} *first = NULL;

// Function prototypes
void createSurgerySchedule(int surgeryIDs[], char patientNames[][50], char surgeonNames[][50], char surgeryTimes[][20], int n);
void insertSurgery(int index, int surgeryID, char patientName[], char surgeonName[], char surgeryTime[]);
void displaySurgerySchedule();

int main()
{
   int surgeryIDs[] = {1, 2, 3, 4};
   char patientNames[][50] = {"ABC", "DEF", "GHI", "JKL"};
    char surgeonNames[][50] = {"Dr. MNO", "Dr. PQR", "Dr. STU", "Dr. VWX"};
    char surgeryTimes[][20] = {"11:00 AM", "11:30 AM", "4:00 PM", "5:30 PM"};

        createSurgerySchedule(surgeryIDs, patientNames, surgeonNames, surgeryTimes, 4);

   printf("Initial surgery schedule:\n");
```

```
    displaySurgerySchedule();

    printf("\nInserting a new surgery at the middle:\n");
    insertSurgery(3, 5, "YZA ", "Dr. BCD", "5:00 PM");

    printf("\nUpdated surgery schedule:\n");
    displaySurgerySchedule();

    return 0;
}

void createSurgerySchedule(int surgeryIDs[], char patientNames[][50], char
surgeonNames[][50], char surgeryTimes[][20], int n)
{
    int i;
    struct Surgery *temp, *last;
    first = (struct Surgery *)malloc(sizeof(struct Surgery));
    first->surgeryID = surgeryIDs[0];
    strcpy(first->patientName, patientNames[0]);
    strcpy(first->surgeonName, surgeonNames[0]);
    strcpy(first->surgeryTime, surgeryTimes[0]);
    first->next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct Surgery *)malloc(sizeof(struct Surgery));
        temp->surgeryID = surgeryIDs[i];
```

```
        strcpy(temp->patientName, patientNames[i]);
        strcpy(temp->surgeonName, surgeonNames[i]);
        strcpy(temp->surgeryTime, surgeryTimes[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertSurgery(int index, int surgeryID, char patientName[], char surgeonName[], char surgeryTime[])
{
    struct Surgery *temp, *p = first;
    int i;

    temp = (struct Surgery *)malloc(sizeof(struct Surgery));
    temp->surgeryID = surgeryID;
    strcpy(temp->patientName, patientName);
    strcpy(temp->surgeonName, surgeonName);
    strcpy(temp->surgeryTime, surgeryTime);
    for (i = 0; i < index - 1 && p != NULL; i++)
        p = p->next;
    if (p != NULL)
    {
        temp->next = p->next;
        p->next = temp;
    }
    else
```

```
        printf("Invalid index\n");
    }

    void displaySurgerySchedule()
    {
       struct Surgery *p = first;
       while (p != NULL)
       {
            printf("Surgery ID: %d  Patient: %s  Surgeon: %s  Time: %s\n", p->surgeryID, p->patientName, p->surgeonName, p->surgeryTime);
          p = p->next;
       }
    }
```

Problem 7: Patient History Record

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

- Create a history record list.
- Insert a new record.
- Display all patient history records.

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

    struct PatientHistory
    {
       int patientID;
```

```
    char patientName[50];
    char diagnosis[100];
    char treatment[100];
    struct PatientHistory *next;
} *first = NULL;

// Function prototypes
void createPatientHistoryRecord(int patientIDs[], char patientNames[][50], char diagnoses[][100], char treatments[][100], int n);
void insertPatientHistory(int patientID, char patientName[], char diagnosis[], char treatment[]);
void displayPatientHistoryRecords();

int main()
{
    int patientIDs[] = {1, 2, 3};
    char patientNames[][50] = {"ABC", "DEF", "GHI"};
    char diagnoses[][100] = {"Flu", "Back Pain", "Cold"};
      char treatments[][100] = {"Rest and Fluids", "Physical Therapy", "Medication"};

        createPatientHistoryRecord(patientIDs, patientNames, diagnoses, treatments, 3);

    printf("Initial patient history records:\n");
    displayPatientHistoryRecords();

    printf("\nInserting a new patient history record:\n");
    insertPatientHistory(4, "JKL", "Headache", "Painkillers");
```

```
    printf("\nUpdated patient history records:\n");
    displayPatientHistoryRecords();

    return 0;
}

void createPatientHistoryRecord(int patientIDs[], char patientNames[][50], char diagnoses[][100], char treatments[][100], int n)
{
    int i;
    struct PatientHistory *temp, *last;
    first = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
    first->patientID = patientIDs[0];
    strcpy(first->patientName, patientNames[0]);
    strcpy(first->diagnosis, diagnoses[0]);
    strcpy(first->treatment, treatments[0]);
    first->next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
        temp->patientID = patientIDs[i];
        strcpy(temp->patientName, patientNames[i]);
        strcpy(temp->diagnosis, diagnoses[i]);
        strcpy(temp->treatment, treatments[i]);
        temp->next = NULL;
        last->next = temp;
```

```
        last = temp;
    }
}

void insertPatientHistory(int patientID, char patientName[], char diagnosis[],
char treatment[])
{
    struct PatientHistory *temp;
    temp = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
    temp->patientID = patientID;
    strcpy(temp->patientName, patientName);
    strcpy(temp->diagnosis, diagnosis);
    strcpy(temp->treatment, treatment);
    temp->next = first;
    first = temp;
}

void displayPatientHistoryRecords()
{
    struct PatientHistory *p = first;
    while (p != NULL)
    {
        printf("Patient ID: %d  Name: %s  Diagnosis: %s  Treatment: %s\n", p-
>patientID, p->patientName, p->diagnosis, p->treatment);
        p = p->next;
    }
}
```

Problem 8: Medical Test Tracking

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

- Create a list of medical tests.
- Insert a new test result.
- Display all test results.

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

struct MedicalTest
{
    int patientID;
    char testName[100];
    char result[100];
    struct MedicalTest *next;
} *first = NULL;

// Function prototypes
void createMedicalTestList(int patientIDs[], char testNames[][100], char results[][100], int n);
void insertTestResult(int patientID, char testName[], char result[]);
void displayTestResults();

int main()
{
    int patientIDs[] = {1, 2, 3};
    char testNames[][100] = {"Blood Test", "X-Ray", "MRI"};
```

```
    char results[][100] = {"Normal", "Abnormal", "Normal"};

    createMedicalTestList(patientIDs, testNames, results, 3);

    printf("Initial medical test results:\n");
    displayTestResults();

    printf("\nInserting a new test result:\n");
    insertTestResult(4, "CT Scan", "Normal");

    printf("\nUpdated medical test results:\n");
    displayTestResults();

    return 0;
}

void createMedicalTestList(int patientIDs[], char testNames[][100], char results[][100], int n)
{
    int i;
    struct MedicalTest *temp, *last;
    first = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));
    first->patientID = patientIDs[0];
    strcpy(first->testName, testNames[0]);
    strcpy(first->result, results[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
```

```
    {
        temp = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));
        temp->patientID = patientIDs[i];
        strcpy(temp->testName, testNames[i]);
        strcpy(temp->result, results[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertTestResult(int patientID, char testName[], char result[])
{
    struct MedicalTest *temp;
    temp = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));
    temp->patientID = patientID;
    strcpy(temp->testName, testName);
    strcpy(temp->result, result);
    temp->next = first;
    first = temp;
}

void displayTestResults()
{
    struct MedicalTest *p = first;
    while (p != NULL)
    {
```

```
        printf("Patient ID: %d  Test Name: %s  Result: %s\n",
            p->patientID, p->testName, p->result);
        p = p->next;
    }
}
```

Problem 9: Prescription Management System

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

- Create a prescription list.
- Insert a new prescription.
- Display all prescriptions.

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

struct Prescription
{
    int prescriptionID;
    char medicine[100];
    char dosage[100];
    struct Prescription *next;
} *first = NULL;

// Function prototypes
void createPrescriptionList(int prescriptionIDs[], char medicines[][100], char
dosages[][100], int n);
```

```
void insertPrescription(int prescriptionID, char medicine[], char dosage[]);
void displayPrescriptions();

int main()
{
    int prescriptionIDs[] = {1, 2, 3};
    char medicines[][100] = {"Paracetamol", "Ibuprofen", "Amoxicillin"};
    char dosages[][100] = {"500mg", "200mg", "250mg"};

    createPrescriptionList(prescriptionIDs, medicines, dosages, 3);

    printf("Initial prescriptions:\n");
    displayPrescriptions();

    printf("\nInserting a new prescription:\n");
    insertPrescription(4, "Cough Syrup", "10ml thrice daily");

    printf("\nUpdated prescriptions:\n");
    displayPrescriptions();

    return 0;
}

void createPrescriptionList(int prescriptionIDs[], char medicines[][100], char
dosages[][100], int n)
{
    int i;
    struct Prescription *temp, *last;
```

```
    first = (struct Prescription *)malloc(sizeof(struct Prescription));
    first->prescriptionID = prescriptionIDs[0];
    strcpy(first->medicine, medicines[0]);
    strcpy(first->dosage, dosages[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Prescription *)malloc(sizeof(struct Prescription));
        temp->prescriptionID = prescriptionIDs[i];
        strcpy(temp->medicine, medicines[i]);
        strcpy(temp->dosage, dosages[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertPrescription(int prescriptionID, char medicine[], char dosage[])
{
    struct Prescription *temp;
    temp = (struct Prescription *)malloc(sizeof(struct Prescription));
    temp->prescriptionID = prescriptionID;
    strcpy(temp->medicine, medicine);
    strcpy(temp->dosage, dosage);
    temp->next = first;
    first = temp;
```

```
}

void displayPrescriptions()
{
   struct Prescription *p = first;
   while (p != NULL)
   {
      printf("Prescription ID: %d  Medicine: %s  Dosage: %s\n",
            p->prescriptionID, p->medicine, p->dosage);
      p = p->next;
   }
}
```

Problem 10: Hospital Staff Roster

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

- Create a staff roster.
- Insert a new staff member into the roster.
- Display the current staff roster.

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

struct Staff
{
   int staffID;
   char name[100];
```

```
    char role[50];
    char department[50];
    struct Staff *next;
} *first = NULL;

// Function prototypes
void createStaffRoster(int staffIDs[], char names[][100], char roles[][50], char departments[][50], int n);
void insertStaffMember(int staffID, char name[], char role[], char department[], int position);
void displayStaffRoster();

int main()
{
    int staffIDs[] = {1, 2, 3};
    char names[][100] = {"Dr. ABC", "Nurse DEF", "Technician GHI"};
    char roles[][50] = {"Doctor", "Nurse", "Technician"};
    char departments[][50] = {"Cardiology", "Emergency", "Radiology"};

    createStaffRoster(staffIDs, names, roles, departments, 3);

    printf("Initial staff roster:\n");
    displayStaffRoster();

    printf("\nInserting a new staff member at position 2:\n");
    insertStaffMember(4, "Dr. JKL", "Surgeon", "Orthopedics", 2);
    displayStaffRoster();
```

```
    printf("\nInserting a new staff member at the end:\n");
    insertStaffMember(5, "Nurse MNO", "Head Nurse", "Pediatrics", 5);
    displayStaffRoster();

    return 0;
}

void createStaffRoster(int staffIDs[], char names[][100], char roles[][50], char
departments[][50], int n)
{
    int i;
    struct Staff *temp, *last;
    first = (struct Staff *)malloc(sizeof(struct Staff));
    first->staffID = staffIDs[0];
    strcpy(first->name, names[0]);
    strcpy(first->role, roles[0]);
    strcpy(first->department, departments[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Staff *)malloc(sizeof(struct Staff));
        temp->staffID = staffIDs[i];
        strcpy(temp->name, names[i]);
        strcpy(temp->role, roles[i]);
        strcpy(temp->department, departments[i]);
        temp->next = NULL;
        last->next = temp;
```

```
        last = temp;
    }
}

void insertStaffMember(int staffID, char name[], char role[], char department[], int position)
{
    struct Staff *temp, *current, *prev;
    int count = 1;
    temp = (struct Staff *)malloc(sizeof(struct Staff));
    temp->staffID = staffID;
    strcpy(temp->name, name);
    strcpy(temp->role, role);
    strcpy(temp->department, department);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
        return;
    }
    current = first;
    while (current != NULL && count < position)
    {
        prev = current;
        current = current->next;
        count++;
    }
```

```
    prev->next = temp;
    temp->next = current;
}

void displayStaffRoster()
{
    struct Staff *p = first;
    while (p != NULL)
    {
        printf("Staff ID: %d  Name: %s  Role: %s  Department: %s\n",
                p->staffID, p->name, p->role, p->department);
        p = p->next;
    }
}
```