```
//1.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_PATIENTS 100
const char *HOSPITAL_NAME = "National 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 = NULL;
int patientCount = 0;
```

```c
void addNewPatient();
void viewPatientDetails();
void updatePatientInformation();
void deletePatientRecord();
void listAllPatients();

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



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

  do {
    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("\nExit\n");
                free(patients);
                break;
            default:
                printf("\nInvalid  choice\n");
        }
    } while (choice  != 6);

    return 0;
}

void addNewPatient() {
    if (patientCount  >= MAX_PATIENTS) {
        printf("\nPatient  record  is full.\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 %d!\n",  p->id);
}

void viewPatientDetails() {
    int id;
    printf("\nEnter  Patient  ID: ");
    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);
    getchar(); // To handle newline character
    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!\n");
}

void deletePatientRecord() {
    int id;
    printf("\nEnter Patient ID: ");
    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!\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);
    }
}



//2.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ITEMS 100
const char *HOSPITAL_NAME = "National Hospital";

struct ItemDetails {
    char manufacturer[50];
    char expirationDate[15];
    union {
        int quantity;
        double weight;
    } unitInfo;
};

struct InventoryItem {
    int id;
    char name[50];

    char category[30];

    double price;

    struct ItemDetails details;

};


struct InventoryItem inventory[MAX_ITEMS];

int itemCount = 0;


// Function Prototypes

void addInventoryItem();

void viewInventoryItem();

void updateInventoryItem();

void deleteInventoryItem();

void listAllInventoryItems();


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;
}


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!\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!\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);
    }
}



//3.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
const char *CLINIC_NAME = "National Health Clinic";
const char *CLINIC_HOURS = "Mon-Fri: 9 AM - 6 PM";

struct Patient {
   int id;
   char name[50];
   int age;
   char gender[10];
};

struct Doctor {
   int id;
   char name[50];
   char specialty[50];
};

union AppointmentAttributes {
   char followUpDetails[200];
   char consultationNotes[200];
};

struct Appointment {
   int id;
   struct Patient patient;
   struct Doctor doctor;
   char appointmentDate[20];
   char appointmentTime[10];
   union AppointmentAttributes attributes;
   int isFollowUp;
};
```

```
struct Appointment *appointments = NULL;
int appointmentCount = 0;
int capacity = 2;

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

int main() {
  appointments = malloc(capacity * sizeof(struct Appointment));
  if (!appointments) {
    printf("Memory allocation failed. Exiting.\n");
    return 1;
  }

  int choice;
  printf("Welcome to %s\n", CLINIC_NAME);
  printf("Clinic Hours: %s\n\n", CLINIC_HOURS);

  do {
    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;
}
```

```
void scheduleAppointment() {
    if (appointmentCount >= capacity) {
        capacity *= 2;
        appointments = realloc(appointments, capacity * sizeof(struct Appointment));
        if (!appointments) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }

    struct Appointment *app = &appointments[appointmentCount];
    app->id = appointmentCount + 1;

    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);



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

    printf("Enter Appointment Date (DD/MM/YYYY): ");
    scanf(" %s", app->appointmentDate);
    printf("Enter Appointment Time (HH:MM): ");
    scanf(" %s", app->appointmentTime);
    int typeChoice;
    printf("\nfollow-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 with ID %d!\n", app->id);
}

void viewAppointment() {
    int id;
    printf("\nEnter Appointment ID: ");
    scanf("%d", &id);

    if (id <= 0 || id > appointmentCount) {
        printf("\nInvalid.\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);


    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 : ");
    scanf("%d", &id);

    if (id <= 0 || id > appointmentCount) {
        printf("\nInvalid.\n");
        return;
    }

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



    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);



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



printf("Enter New Appointment Date (DD/MM/YYYY): ");
scanf(" %s", app->appointmentDate);
printf("Enter New Appointment Time (HH:MM): ");
scanf(" %s", app->appointmentTime);



int typeChoice;
printf("\nfollow-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!\n");
}

void cancelAppointment() {
    int id;
    printf("\nEnter Appointment ID: ");
    scanf("%d", &id);

    if (id <= 0 || id > appointmentCount) {
        printf("\nInvalid\n");
        return;
    }

    for (int i = id - 1; i < appointmentCount - 1; i++) {
        appointments[i] = appointments[i + 1];
    }
    appointmentCount--;
    printf("\nAppointment canceled!\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);
    }
}
//4.
#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;

// Function Prototypes
void generateBill();
void viewBill();
void updateBill();
void deleteBill();
void listAllBills();

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

    for (int i = 0; i < billCount; i++) {
        free(bills[i]);
    }

    return 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);
    }
}
//5.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TESTS 100
#define MAX_NAME_LENGTH 50

const float BLOOD_SUGAR_RANGE_LOW = 70.0;
const float BLOOD_SUGAR_RANGE_HIGH = 110.0;
const float CHOLESTEROL_RANGE_LOW = 150.0;
const float CHOLESTEROL_RANGE_HIGH = 200.0;



struct TestResultDetails {
    float bloodSugar;
    float cholesterol;
    char testDate[15];
};

union OptionalTestData {
    float bloodPressure;
    char comments[100];
};



struct TestResult {
    int testId;
    char patientName[MAX_NAME_LENGTH];
    struct TestResultDetails details;
    union OptionalTestData optionalData;
    int hasBloodPressure;
};

struct TestResult *testResults[MAX_TESTS];
int testCount = 0;

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

int main() {
    int choice;
    do {
        printf("\nMedical 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 your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTestResult();
                break;
            case 2:
                viewTestResult();
                break;
            case 3:
                updateTestResult();
                break;
            case 4:
                deleteTestResult();
                break;
            case 5:
                listAllTestResults();
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 6);

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

    return 0;
}

void addTestResult() {
    if (testCount >= MAX_TESTS) {
        printf("Maximum test limit reached.\n");
        return;
    }

    struct TestResult *newTest = (struct TestResult *)malloc(sizeof(struct TestResult));

    printf("Enter Test ID: ");
    scanf("%d", &newTest->testId);
    printf("Enter Patient Name: ");
    scanf("%s", newTest->patientName);

    printf("Enter Blood Sugar Level: ");
    scanf("%f", &newTest->details.bloodSugar);
    printf("Enter Cholesterol Level: ");
    scanf("%f", &newTest->details.cholesterol);
    printf("Enter Test Date (DD-MM-YYYY): ");
    scanf("%s", newTest->details.testDate);

    printf("Do you want to enter Blood Pressure? (1 for Yes, 0 for No): ");
    scanf("%d", &newTest->hasBloodPressure);
    if (newTest->hasBloodPressure) {
        printf("Enter Blood Pressure: ");
        scanf("%f", &newTest->optionalData.bloodPressure);
    } else {
        printf("Enter any additional comments: ");
        scanf(" %[^\n]%*c", newTest->optionalData.comments);
    }
    testResults[testCount++] = newTest;
    printf("Test Result added successfully!\n");
}
```

```
void viewTestResult() {
    int testId;
    printf("Enter Test ID to view: ");
    scanf("%d", &testId);

    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->testId == testId) {
            printf("\nTest ID: %d\n", testResults[i]->testId);
            printf("Patient Name: %s\n", testResults[i]->patientName);
            printf("Blood Sugar Level: %.2f (Range: %.2f - %.2f)\n",
                   testResults[i]->details.bloodSugar,
                   BLOOD_SUGAR_RANGE_LOW,
                   BLOOD_SUGAR_RANGE_HIGH);
            printf("Cholesterol Level: %.2f (Range: %.2f - %.2f)\n",
                   testResults[i]->details.cholesterol,
                   CHOLESTEROL_RANGE_LOW,
                   CHOLESTEROL_RANGE_HIGH);
            printf("Test Date: %s\n", testResults[i]->details.testDate);

            if (testResults[i]->hasBloodPressure) {
                printf("Blood Pressure: %.2f\n", testResults[i]->optionalData.bloodPressure);
            } else {
                printf("Comments: %s\n", testResults[i]->optionalData.comments);
            }

            return;
        }
    }
    printf("Test Result with ID %d not found.\n", testId);
}
```

```
void updateTestResult() {
    int testId;
    printf("Enter Test ID to update: ");
    scanf("%d", &testId);
    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->testId == testId) {
            printf("Enter new Blood Sugar Level: ");
            scanf("%f", &testResults[i]->details.bloodSugar);
            printf("Enter new Cholesterol Level: ");
            scanf("%f", &testResults[i]->details.cholesterol);
            printf("Enter new Test Date (DD-MM-YYYY): ");
            scanf("%s", testResults[i]->details.testDate);



            printf("Do you want to update Blood Pressure? (1 for Yes, 0 for No): ");
            scanf("%d", &testResults[i]->hasBloodPressure);
            if (testResults[i]->hasBloodPressure) {
                printf("Enter new Blood Pressure: ");
                scanf("%f", &testResults[i]->optionalData.bloodPressure);
            } else {
                printf("Enter new comments: ");
                scanf(" %[^\n]%*c", testResults[i]->optionalData.comments);
            }


            printf("Test Result updated successfully!\n");
            return;
        }
    }
    printf("Test Result with ID %d not found.\n", testId);
}
void deleteTestResult() {
    int testId;
    printf("Enter Test ID to delete: ");
    scanf("%d", &testId);

    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->testId == testId) {
            free(testResults[i]);
            for (int j = i; j < testCount - 1; j++) {
                testResults[j] = testResults[j + 1];
            }
            testCount--;
            printf("Test Result deleted successfully!\n");
            return;
        }
    }
    printf("Test Result with ID %d not found.\n", testId);
}
void listAllTestResults() {
    if (testCount == 0) {
        printf("No test results available.\n");
        return;
    }
    printf("\nList of All Test Results:\n");
    for (int i = 0; i < testCount; i++) {
        printf("Test ID: %d, Patient Name: %s, Blood Sugar: %.2f, Cholesterol: %.2f, Test Date: %s\n",
               testResults[i]->testId,
               testResults[i]->patientName,
               testResults[i]->details.bloodSugar,
               testResults[i]->details.cholesterol,
               testResults[i]->details.testDate);
    }
```

```
}
//6.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_STAFF 100
#define MAX_NAME_LEN 50

static const int SHIFT_TIMES[3] = {8, 16, 24};

union OptionalDutyAttributes {
    char location[MAX_NAME_LEN];
    int specialShiftHours;
};

struct DutyDetails {
    int shiftType;
    union OptionalDutyAttributes optionalDetails;
};

struct Staff {
    char name[MAX_NAME_LEN];
    int staffID;
    struct DutyDetails duty;
};

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

struct Staff* roster[MAX_STAFF];
int totalStaff = 0;

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

      switch (choice) {
         case 1:
            addDutyRoster();
            break;
         case 2:
            viewDutyRoster();
            break;
         case 3:
            updateDutyRoster();
            break;
         case 4:
            deleteDutyRoster();
            break;
            case 5:
                listAllDutyRosters();
                break;
            case 6:
                printf("Exiting the system...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);

    for (int i = 0; i < totalStaff; i++) {
        free(roster[i]);
    }

    return 0;
}

void addDutyRoster() {
    if (totalStaff >= MAX_STAFF) {
        printf("Roster is full.\n");
        return;
    }

    struct Staff* newStaff = (struct Staff*)malloc(sizeof(struct Staff));
    if (newStaff == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

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

    printf("Enter shift type: ");
    scanf("%d", &newStaff->duty.shiftType);

    if (newStaff->duty.shiftType == 0 || newStaff->duty.shiftType == 1) {
        printf("Enter duty location: ");
        scanf(" %[^\n]s", newStaff->duty.optionalDetails.location);
    } else if (newStaff->duty.shiftType == 2) {
        printf("Enter special shift hours: ");
        scanf("%d", &newStaff->duty.optionalDetails.specialShiftHours);
    }

    roster[totalStaff++] = newStaff;
    printf("Staff duty roster added\n");
}

void viewDutyRoster() {
    int staffID;
    printf("Enter staff ID : ");
    scanf("%d", &staffID);

    for (int i = 0; i < totalStaff; i++) {
        if (roster[i]->staffID == staffID) {
            printf("\nStaff ID: %d\n", roster[i]->staffID);
            printf("Staff Name: %s\n", roster[i]->name);

            printf("Shift: ");
            switch (roster[i]->duty.shiftType) {
                case 0:
                    printf("Morning\n");
                    break;
                case 1:
                    printf("Evening\n");
                    break;
                case 2:
                    printf("Night\n");
                    break;
                default:
                    printf("Unknown\n");
            }

            if (roster[i]->duty.shiftType == 0 || roster[i]->duty.shiftType == 1) {
                printf("Duty Location: %s\n", roster[i]->duty.optionalDetails.location);
            } else if (roster[i]->duty.shiftType == 2) {
                printf("Special Shift Hours: %d hours\n", roster[i]->duty.optionalDetails.specialShiftHours);
            }
            return;
        }
    }
    printf("Staff with ID %d not found.\n", staffID);
}

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

    for (int i = 0; i < totalStaff; i++) {
        if (roster[i]->staffID == staffID) {
            printf("Updating duty roster for staff ID %d...\n", staffID);

            printf("Enter new shift type: ");
            scanf("%d", &roster[i]->duty.shiftType);

            if (roster[i]->duty.shiftType == 0 || roster[i]->duty.shiftType == 1) {
                printf("Enter new duty location: ");
                scanf(" %[^\n]s", roster[i]->duty.optionalDetails.location);
            } else if (roster[i]->duty.shiftType == 2) {
                printf("Enter new special shift hours: ");
                scanf("%d", &roster[i]->duty.optionalDetails.specialShiftHours);
            }

            printf("Duty roster updated.\n");
            return;
        }
    }
    printf("Staff with ID %d not found.\n", staffID);
}

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

    for (int i = 0; i < totalStaff; i++) {
        if (roster[i]->staffID == staffID) {
            free(roster[i]);
            for (int j = i; j < totalStaff - 1; j++) {
                roster[j] = roster[j + 1];
            }
            totalStaff--;
            printf("Staff duty roster deleted.\n");
            return;
        }
    }
    printf("Staff with ID %d not found.\n", staffID);
}

void listAllDutyRosters() {
    if (totalStaff == 0) {
        printf("No duty rosters available.\n");
        return;
    }

    printf("\nList of All Staff Duty Rosters:\n");
    for (int i = 0; i < totalStaff; i++) {
        printf("\nStaff ID: %d\n", roster[i]->staffID);
        printf("Staff Name: %s\n", roster[i]->name);

        printf("Shift: ");
        switch (roster[i]->duty.shiftType) {
            case 0:
                printf("Morning\n");
                break;
            case 1:
                printf("Evening\n");
                break;
            case 2:
                printf("Night\n");
                break;
            default:
```

```
            printf("Unknown\n");
        }

        if (roster[i]->duty.shiftType == 0 || roster[i]->duty.shiftType == 1) {
            printf("Duty Location: %s\n", roster[i]->duty.optionalDetails.location);
        } else if (roster[i]->duty.shiftType == 2) {
            printf("Special Shift Hours: %d hours\n", roster[i]->duty.optionalDetails.specialShiftHours);
        }
    }
}

//7.
#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;
}

//8.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_RECORDS 100
const char *HOSPITAL_NAME = "National Hospital";

struct MedicalHistory {
    char pastDiseases[100];
    char allergies[50];
    union {
        char notes[200];
        struct {
            char familyHistory[100];
            char surgeries[100];
        } detailedHistory;
    } historyDetails;
};

struct MedicalRecord {
    int recordID;
    char patientName[50];
    int age;
    char gender[10];
    struct MedicalHistory history;
};
```

```
struct MedicalRecord *records = NULL;
int recordCount = 0;

void addNewRecord();
void viewRecordDetails();
void updateRecordInformation();
void deleteRecord();
void listAllRecords();

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

  records = (struct MedicalRecord *)malloc(MAX_RECORDS * sizeof(struct MedicalRecord));
  if (!records) {
    printf("Memory allocation failed.\n");
    return 1;
  }

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

    switch (choice) {
            case 1:
                addNewRecord();
                break;
            case 2:
                viewRecordDetails();
                break;
            case 3:
                updateRecordInformation();
                break;
            case 4:
                deleteRecord();
                break;
            case 5:
                listAllRecords();
                break;
            case 6:
                printf("\nExit\n");
                free(records);
                break;
            default:
                printf("\nInvalid choice\n");
        }
    } while (choice != 6);

    return 0;
}

void addNewRecord() {
    if (recordCount >= MAX_RECORDS) {
        printf("\nRecord limit reached.\n");
        return;
    }

    struct MedicalRecord *record = &records[recordCount];
    record->recordID = recordCount + 1;

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

    recordCount++;
    printf("\nRecord added successfully with ID %d!\n", record->recordID);
}

void viewRecordDetails() {
    int id;
    printf("\nEnter Record ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > recordCount) {
        printf("\nInvalid Record ID.\n");
        return;
    }
    struct MedicalRecord *record = &records[id - 1];

    printf("\nRecord ID: %d\n", record->recordID);
    printf("Patient Name: %s\n", record->patientName);
    printf("Age: %d\n", record->age);
    printf("Gender: %s\n", record->gender);
    printf("Past Diseases: %s\n", record->history.pastDiseases);
    printf("Allergies: %s\n", record->history.allergies);
    printf("Family History: %s\n", record->history.historyDetails.detailedHistory.familyHistory);
    printf("Surgeries: %s\n", record->history.historyDetails.detailedHistory.surgeries);
}

void updateRecordInformation() {
    int id;
    printf("\nEnter Record ID to update: ");
    scanf("%d", &id);
    if (id <= 0 || id > recordCount) {
        printf("\nInvalid Record ID.\n");
        return;
    }
    struct MedicalRecord *record = &records[id - 1];

    printf("\nUpdating information for Record ID %d\n", record->recordID);
    printf("Enter New Patient Name: ");
    scanf(" %s", record->patientName);
    printf("Enter New Age: ");
    scanf("%d", &record->age);
    printf("Enter New Gender: ");
    scanf(" %s", record->gender);
    printf("Enter New Past Diseases: ");
    scanf(" %s", record->history.pastDiseases);
    printf("Enter New Allergies: ");
    scanf(" %s", record->history.allergies);
    printf("Enter New Family History: ");
    scanf(" %s", record->history.historyDetails.detailedHistory.familyHistory);
    printf("Enter New Surgeries: ");
    scanf(" %s", record->history.historyDetails.detailedHistory.surgeries);

    printf("\nRecord updated!\n");
}

void deleteRecord() {
    int id;
    printf("\nEnter Record ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > recordCount) {
        printf("\nInvalid Record ID.\n");
        return;
    }

    for (int i = id - 1; i < recordCount - 1; i++) {
        records[i] = records[i + 1];
    }
    recordCount--;
    printf("\nRecord deleted!\n");
}

void listAllRecords() {
    if (recordCount == 0) {
        printf("\nNo records available.\n");
        return;
    }
    printf("\nListing all records:\n");
    for (int i = 0; i < recordCount; i++) {
        printf("Record ID: %d, Patient Name: %s, Age: %d, Gender: %s\n", records[i].recordID,
records[i].patientName, records[i].age, records[i].gender);
    }
}

//9.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_DIET_PLANS 100
const char *HOSPITAL_NAME = "National Hospital";

struct DietPlan {
    int planID;
    char patientName[50];
    char dietType[50];
    char duration[20];
    char recommendedFoods[200];
};

struct DietPlan *dietPlans = NULL;
int dietPlanCount = 0;

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

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

   dietPlans = (struct DietPlan *)malloc(MAX_DIET_PLANS * sizeof(struct DietPlan));
   if (!dietPlans) {
      printf("Memory allocation failed.\n");
      return 1;
   }

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

      switch (choice) {
         case 1:
            addDietPlan();
            break;
         case 2:
            viewDietPlan();
            break;
         case 3:
            updateDietPlan();
            break;
          case 4:
            deleteDietPlan();
            break;
          case 5:
            listAllDietPlans();
            break;
          case 6:
            printf("\nExit\n");
            free(dietPlans);
            break;
          default:
            printf("\nInvalid  choice\n");
        }
    } while (choice  != 6);

    return 0;
}

void addDietPlan() {
    if (dietPlanCount  >= MAX_DIET_PLANS) {
        printf("\nDiet  plan limit reached.\n");
        return;
    }

    struct DietPlan *plan = &dietPlans[dietPlanCount];
    plan->planID = dietPlanCount  + 1;

    printf("\nEnter  Patient Name: ");
    scanf(" %s", plan->patientName);
    printf("Enter Diet Type: ");
    scanf(" %s", plan->dietType);
    printf("Enter Duration: ");
    scanf(" %s", plan->duration);
    printf("Enter Recommended Foods: ");
    scanf(" %s", plan->recommendedFoods);

    dietPlanCount++;
    printf("\nDiet plan added successfully with ID %d!\n", plan->planID);
}

void viewDietPlan() {
    int id;
    printf("\nEnter Diet Plan ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > dietPlanCount) {
        printf("\nInvalid Diet Plan ID.\n");
        return;
    }
    struct DietPlan *plan = &dietPlans[id - 1];

    printf("\nDiet Plan ID: %d\n", plan->planID);
    printf("Patient Name: %s\n", plan->patientName);
    printf("Diet Type: %s\n", plan->dietType);
    printf("Duration: %s\n", plan->duration);
    printf("Recommended Foods: %s\n", plan->recommendedFoods);
}

void updateDietPlan() {
    int id;
    printf("\nEnter Diet Plan ID to update: ");
    scanf("%d", &id);
    if (id <= 0 || id > dietPlanCount) {
        printf("\nInvalid Diet Plan ID.\n");
        return;
    }
    struct DietPlan *plan = &dietPlans[id - 1];

    printf("\nUpdating information for Diet Plan ID %d\n", plan->planID);
    printf("Enter New Patient Name: ");
    scanf(" %s", plan->patientName);
    printf("Enter New Diet Type: ");
    scanf(" %s", plan->dietType);
    printf("Enter New Duration: ");
    scanf(" %s", plan->duration);
    printf("Enter New Recommended Foods: ");
    scanf(" %s", plan->recommendedFoods);

    printf("\nDiet plan updated!\n");
}

void deleteDietPlan() {
    int id;
    printf("\nEnter Diet Plan ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > dietPlanCount) {
        printf("\nInvalid Diet Plan ID.\n");
        return;
    }

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

void listAllDietPlans() {
    if (dietPlanCount == 0) {
        printf("\nNo diet plans available.\n");
        return;
    }
    printf("\nListing all diet plans:\n");
    for (int i = 0; i < dietPlanCount; i++) {
        printf("Diet Plan ID: %d, Patient Name: %s, Diet Type: %s, Duration: %s\n", dietPlans[i].planID,
dietPlans[i].patientName, dietPlans[i].dietType, dietPlans[i].duration);
    }
}


//10.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SURGERIES 100
const char *HOSPITAL_NAME = "National Hospital";

struct Surgery {
    int surgeryID;
    char patientName[50];
    char surgeryType[50];
    char surgeryDate[20];
    char surgeon[50];
```

```
};

struct Surgery *surgeries = NULL;
int surgeryCount = 0;

void scheduleSurgery();
void viewSurgerySchedule();
void updateSurgerySchedule();
void cancelSurgery();
void listAllSurgeries();

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

  surgeries = (struct Surgery *)malloc(MAX_SURGERIES * sizeof(struct Surgery));
  if (!surgeries) {
    printf("Memory allocation failed.\n");
    return 1;
  }

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

```
        switch (choice) {
            case 1:
                scheduleSurgery();
                break;
            case 2:
                viewSurgerySchedule();
                break;
            case 3:
                updateSurgerySchedule();
                break;
            case 4:
                cancelSurgery();
                break;
            case 5:
                listAllSurgeries();
                break;
            case 6:
                printf("\nExit\n");
                free(surgeries);
                break;
            default:
                printf("\nInvalid choice\n");
        }
    } while (choice != 6);

    return 0;
}

void scheduleSurgery() {
    if (surgeryCount >= MAX_SURGERIES) {
        printf("\nSurgery schedule is full.\n");
        return;
    }

    struct Surgery *surgery = &surgeries[surgeryCount];
    surgery->surgeryID = surgeryCount + 1;

    printf("\nEnter Patient Name: ");
    scanf(" %s", surgery->patientName);
    printf("Enter Surgery Type: ");
    scanf(" %s", surgery->surgeryType);
    printf("Enter Surgery Date: ");
    scanf(" %s", surgery->surgeryDate);
    printf("Enter Surgeon Name: ");
    scanf(" %s", surgery->surgeon);

    surgeryCount++;
    printf("\nSurgery scheduled successfully with ID %d!\n", surgery->surgeryID);
}

void viewSurgerySchedule() {
    int id;
    printf("\nEnter Surgery ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > surgeryCount) {
        printf("\nInvalid Surgery ID.\n");
        return;
    }
    struct Surgery *surgery = &surgeries[id - 1];

    printf("\nSurgery ID: %d\n", surgery->surgeryID);
    printf("Patient Name: %s\n", surgery->patientName);
    printf("Surgery Type: %s\n", surgery->surgeryType);
    printf("Surgery Date: %s\n", surgery->surgeryDate);
    printf("Surgeon: %s\n", surgery->surgeon);
}

void updateSurgerySchedule() {
    int id;
    printf("\nEnter Surgery ID to update: ");
    scanf("%d", &id);
    if (id <= 0 || id > surgeryCount) {
        printf("\nInvalid Surgery ID.\n");
        return;
    }
    struct Surgery *surgery = &surgeries[id - 1];

    printf("\nUpdating Surgery Schedule for ID %d\n", surgery->surgeryID);
    printf("Enter New Patient Name: ");
    scanf(" %s", surgery->patientName);
    printf("Enter New Surgery Type: ");
    scanf(" %s", surgery->surgeryType);
    printf("Enter New Surgery Date: ");
    scanf(" %s", surgery->surgeryDate);
    printf("Enter New Surgeon Name: ");
    scanf(" %s", surgery->surgeon);

    printf("\nSurgery schedule updated!\n");
}

void cancelSurgery() {
    int id;
    printf("\nEnter Surgery ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > surgeryCount) {
        printf("\nInvalid Surgery ID.\n");
        return;
    }

    for (int i = id - 1; i < surgeryCount - 1; i++) {
        surgeries[i] = surgeries[i + 1];
    }
    surgeryCount--;
    printf("\nSurgery cancelled!\n");
}

void listAllSurgeries() {
    if (surgeryCount == 0) {
        printf("\nNo surgeries scheduled.\n");
        return;
    }
    printf("\nListing all surgeries:\n");
    for (int i = 0; i < surgeryCount; i++) {
        printf("Surgery ID: %d, Patient Name: %s, Surgery Type: %s, Surgery Date: %s\n",
surgeries[i].surgeryID, surgeries[i].patientName, surgeries[i].surgeryType, surgeries[i].surgeryDate);
    }
}

//11.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_PRESCRIPTIONS 100
```

```
const char *HOSPITAL_NAME = "National Hospital";

struct Prescription {
    int prescriptionID;
    char patientName[50];
    char medication[100];
    char dosage[50];
    char doctor[50];
};

struct Prescription *prescriptions = NULL;
int prescriptionCount = 0;

void addPrescription();
void viewPrescription();
void updatePrescription();
void deletePrescription();
void listAllPrescriptions();

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

    prescriptions = (struct Prescription *)malloc(MAX_PRESCRIPTIONS * sizeof(struct Prescription));
    if (!prescriptions) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    do {
        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("\nExit\n");
              free(prescriptions);
              break;
           default:
              printf("\nInvalid choice\n");
        }
```

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

    return 0;
}

void addPrescription() {
    if (prescriptionCount >= MAX_PRESCRIPTIONS) {
        printf("\nPrescription limit reached.\n");
        return;
    }

    struct Prescription *prescription = &prescriptions[prescriptionCount];
    prescription->prescriptionID = prescriptionCount + 1;

    printf("\nEnter Patient Name: ");
    scanf(" %s", prescription->patientName);
    printf("Enter Medication: ");
    scanf(" %s", prescription->medication);
    printf("Enter Dosage: ");
    scanf(" %s", prescription->dosage);
    printf("Enter Doctor's Name: ");
    scanf(" %s", prescription->doctor);

    prescriptionCount++;
    printf("\nPrescription added successfully with ID %d!\n", prescription->prescriptionID);
}

void viewPrescription() {
    int id;
    printf("\nEnter Prescription ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > prescriptionCount) {
        printf("\nInvalid Prescription ID.\n");
        return;
    }
    struct Prescription *prescription = &prescriptions[id - 1];

    printf("\nPrescription ID: %d\n", prescription->prescriptionID);
    printf("Patient Name: %s\n", prescription->patientName);
    printf("Medication: %s\n", prescription->medication);
    printf("Dosage: %s\n", prescription->dosage);
    printf("Doctor: %s\n", prescription->doctor);
}

void updatePrescription() {
    int id;
    printf("\nEnter Prescription ID to update: ");
    scanf("%d", &id);
    if (id <= 0 || id > prescriptionCount) {
        printf("\nInvalid Prescription ID.\n");
        return;
    }
    struct Prescription *prescription = &prescriptions[id - 1];

    printf("\nUpdating Prescription ID %d\n", prescription->prescriptionID);
    printf("Enter New Patient Name: ");
    scanf(" %s", prescription->patientName);
    printf("Enter New Medication: ");
    scanf(" %s", prescription->medication);
    printf("Enter New Dosage: ");
    scanf(" %s", prescription->dosage);
    printf("Enter New Doctor's Name: ");
    scanf(" %s", prescription->doctor);

    printf("\nPrescription updated!\n");
}

void deletePrescription() {
    int id;
    printf("\nEnter Prescription ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > prescriptionCount) {
        printf("\nInvalid Prescription ID.\n");
        return;
    }

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

void listAllPrescriptions() {
    if (prescriptionCount == 0) {
        printf("\nNo prescriptions available.\n");
        return;
    }
    printf("\nListing all prescriptions:\n");
    for (int i = 0; i < prescriptionCount; i++) {
        printf("Prescription ID: %d, Patient Name: %s, Medication: %s, Dosage: %s\n",
prescriptions[i].prescriptionID, prescriptions[i].patientName, prescriptions[i].medication,
prescriptions[i].dosage);
    }
```

```
}

//12.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CONSULTATIONS 100
const char *HOSPITAL_NAME = "National Hospital";

struct Consultation {
    int consultationID;
    char patientName[50];
    char doctorName[50];
    char consultationDate[20];
    char diagnosis[100];
};

struct Consultation *consultations = NULL;
int consultationCount = 0;

void scheduleConsultation();
void viewConsultation();
void updateConsultation();
void cancelConsultation();
void listAllConsultations();

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

```
consultations = (struct Consultation *)malloc(MAX_CONSULTATIONS * sizeof(struct Consultation));
if (!consultations) {
   printf("Memory allocation failed.\n");
   return 1;
}

do {
   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("\nExit\n");
            free(consultations);
            break;
          default:
            printf("\nInvalid choice\n");
        }
    } while (choice != 6);

    return 0;
}

void scheduleConsultation() {
    if (consultationCount >= MAX_CONSULTATIONS) {
        printf("\nConsultation schedule is full.\n");
        return;
    }

    struct Consultation *consultation = &consultations[consultationCount];
    consultation->consultationID = consultationCount + 1;

    printf("\nEnter Patient Name: ");
    scanf(" %s", consultation->patientName);
    printf("Enter Doctor Name: ");
    scanf(" %s", consultation->doctorName);
    printf("Enter Consultation Date: ");
    scanf(" %s", consultation->consultationDate);
    printf("Enter Diagnosis: ");
    scanf(" %s", consultation->diagnosis);
    consultationCount++;
    printf("\nConsultation scheduled successfully with ID %d!\n", consultation->consultationID);
}

void viewConsultation() {
    int id;
    printf("\nEnter Consultation ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > consultationCount) {
        printf("\nInvalid Consultation ID.\n");
        return;
    }
    struct Consultation *consultation = &consultations[id - 1];

    printf("\nConsultation ID: %d\n", consultation->consultationID);
    printf("Patient Name: %s\n", consultation->patientName);
    printf("Doctor Name: %s\n", consultation->doctorName);
    printf("Consultation Date: %s\n", consultation->consultationDate);
    printf("Diagnosis: %s\n", consultation->diagnosis);
}

void updateConsultation() {
    int id;
    printf("\nEnter Consultation ID to update: ");
    scanf("%d", &id);
    if (id <= 0 || id > consultationCount) {
        printf("\nInvalid Consultation ID.\n");
        return;
    }
    struct Consultation *consultation = &consultations[id - 1];
    printf("\nUpdating Consultation ID %d\n", consultation->consultationID);
    printf("Enter New Patient Name: ");
    scanf(" %s", consultation->patientName);
    printf("Enter New Doctor Name: ");
    scanf(" %s", consultation->doctorName);
    printf("Enter New Consultation Date: ");
    scanf(" %s", consultation->consultationDate);
    printf("Enter New Diagnosis: ");
    scanf(" %s", consultation->diagnosis);

    printf("\nConsultation updated!\n");
}

void cancelConsultation() {
    int id;
    printf("\nEnter Consultation ID: ");
    scanf("%d", &id);
    if (id <= 0 || id > consultationCount) {
        printf("\nInvalid Consultation ID.\n");
        return;
    }

    for (int i = id - 1; i < consultationCount - 1; i++) {
        consultations[i] = consultations[i + 1];
    }
    consultationCount--;
    printf("\nConsultation cancelled!\n");
}

void listAllConsultations() {
    if (consultationCount == 0) {
        printf("\nNo consultations available.\n");
        return;
    }
    printf("\nListing all consultations:\n");
    for (int i = 0; i < consultationCount; i++) {
        printf("Consultation ID: %d, Patient Name: %s, Doctor Name: %s, Consultation Date: %s\n",
consultations[i].consultationID, consultations[i].patientName, consultations[i].doctorName,
consultations[i].consultationDate);
    }
}
```

```
// 1.
#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;
}*/




//2.
*#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;
    }
}*/


// 3.
#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;
    }
}



//4.
#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;
    }
}



//5.
#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;
    }
}


//6.
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

// Define structure for surgery schedule
struct SurgeryNode
{
    char patientName[50];
    char surgeryType[50];
    char surgeryDate[20];
    struct SurgeryNode *next;
} *first = NULL;

// Function prototypes
void createSurgerySchedule(char schedules[][3][50], int n);
void insertSurgery(struct SurgeryNode *p, char patientName[], char surgeryType[], char
surgeryDate[]);
void displaySurgerySchedule(struct SurgeryNode *p);

int main()
{
    char surgerySchedules[][3][50] = {{"Alice Brown", "Appendectomy", "2025-02-15"}, {"Bob White",
"Knee Replacement", "2025-02-16"}};
    createSurgerySchedule(surgerySchedules, 2);
    printf("Initial surgery schedule:\n");
    displaySurgerySchedule(first);
    printf("\nAdding a new surgery to the schedule:\n");
    insertSurgery(first, "Charlie Green", "Heart Bypass", "2025-02-17");
    displaySurgerySchedule(first);
    return 0;
}

void createSurgerySchedule(char schedules[][3][50], int n)
{
    int i;
    struct SurgeryNode *temp, *last;
    first = (struct SurgeryNode *)malloc(sizeof(struct SurgeryNode));
    strcpy(first->patientName, schedules[0][0]);
    strcpy(first->surgeryType, schedules[0][1]);
    strcpy(first->surgeryDate, schedules[0][2]);
    first->next = NULL;
    last = first;

    for (i = 1; i < n; i++)
    {
        temp = (struct SurgeryNode *)malloc(sizeof(struct SurgeryNode));
        strcpy(temp->patientName, schedules[i][0]);
        strcpy(temp->surgeryType, schedules[i][1]);
        strcpy(temp->surgeryDate, schedules[i][2]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void insertSurgery(struct SurgeryNode *p, char patientName[], char surgeryType[], char
surgeryDate[])
{
    struct SurgeryNode *temp, *last = p;
    temp = (struct SurgeryNode *)malloc(sizeof(struct SurgeryNode));
    strcpy(temp->patientName, patientName);
    strcpy(temp->surgeryType, surgeryType);
    strcpy(temp->surgeryDate, surgeryDate);
    temp->next = NULL;
    while (last->next != NULL)
```

```
        last = last->next;
    last->next = temp;
}

void displaySurgerySchedule(struct SurgeryNode *p)
{
    while (p != NULL)
    {
        printf("Patient: %s, Surgery: %s, Date: %s\n", p->patientName, p->surgeryType, p->surgeryDate);
        p = p->next;
    }
}

//7.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define structure for patient history record
struct PatientHistoryNode
{
    char patientName[50];
    char diagnosis[100];
    char treatment[100];
    struct PatientHistoryNode *next;
} *first = NULL;

// Function prototypes
void createHistoryRecordList(char records[][3][50], int n);
void insertHistoryRecord(struct PatientHistoryNode *p, char patientName[], char diagnosis[], char
treatment[]);
```

```
void displayHistoryRecords(struct PatientHistoryNode *p);


int main()
{
   char historyRecords[][3][50] = {{"Alice Brown", "Fever", "Paracetamol"}, {"Bob White", "Knee
Injury", "Surgery"}};
   createHistoryRecordList(historyRecords, 2);
   printf("Initial patient history records:\n");
   displayHistoryRecords(first);
   printf("\nAdding a new patient history record:\n");
   insertHistoryRecord(first, "Charlie Green", "Cold", "Cough Syrup");
   displayHistoryRecords(first);
   return 0;
}


void createHistoryRecordList(char records[][3][50], int n)
{
   int i;
   struct PatientHistoryNode *temp, *last;
   first = (struct PatientHistoryNode *)malloc(sizeof(struct PatientHistoryNode));
   strcpy(first->patientName, records[0][0]);
   strcpy(first->diagnosis, records[0][1]);
   strcpy(first->treatment, records[0][2]);
   first->next = NULL;
   last = first;


   for (i = 1; i < n; i++)
   {
      temp = (struct PatientHistoryNode *)malloc(sizeof(struct PatientHistoryNode));
      strcpy(temp->patientName, records[i][0]);
      strcpy(temp->diagnosis, records[i][1]);
```

```
        strcpy(temp->treatment, records[i][2]);

        temp->next = NULL;

        last->next = temp;

        last = temp;

    }

}


void insertHistoryRecord(struct PatientHistoryNode *p, char patientName[], char diagnosis[], char
treatment[])

{

    struct PatientHistoryNode *temp, *last = p;

    temp = (struct PatientHistoryNode *)malloc(sizeof(struct PatientHistoryNode));

    strcpy(temp->patientName, patientName);

    strcpy(temp->diagnosis, diagnosis);

    strcpy(temp->treatment, treatment);

    temp->next = NULL;

    while (last->next != NULL)

        last = last->next;

    last->next = temp;

}


void displayHistoryRecords(struct PatientHistoryNode *p)

{

    while (p != NULL)

    {

        printf("Patient: %s, Diagnosis: %s, Treatment: %s\n", p->patientName, p->diagnosis, p-
>treatment);

        p = p->next;

    }

}


//8.
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define structure for medical test

struct MedicalTestNode

{

    char patientName[50];

    char testName[50];

    char testDate[20];

    struct MedicalTestNode *next;

} *first = NULL;


// Function prototypes

void createMedicalTestList(char tests[][3][50], int n);

void insertMedicalTest(struct MedicalTestNode *p, char patientName[], char testName[], char
testDate[]);

void displayMedicalTests(struct MedicalTestNode *p);


int main()

{

    char medicalTests[][3][50] = {{"Alice Brown", "Blood Test", "2025-02-01"}, {"Bob White", "X-Ray",
"2025-02-05"}};

    createMedicalTestList(medicalTests, 2);

    printf("Initial medical test list:\n");

    displayMedicalTests(first);

    printf("\nAdding a new medical test result:\n");

    insertMedicalTest(first, "Charlie Green", "MRI", "2025-02-10");

    displayMedicalTests(first);

    return 0;

}
```

```
void createMedicalTestList(char tests[][3][50], int n)
{
   int i;
   struct MedicalTestNode *temp, *last;
   first = (struct MedicalTestNode *)malloc(sizeof(struct MedicalTestNode));
   strcpy(first->patientName, tests[0][0]);
   strcpy(first->testName, tests[0][1]);
   strcpy(first->testDate, tests[0][2]);
   first->next = NULL;
   last = first;

   for (i = 1; i < n; i++)
   {
      temp = (struct MedicalTestNode *)malloc(sizeof(struct MedicalTestNode));
      strcpy(temp->patientName, tests[i][0]);
      strcpy(temp->testName, tests[i][1]);
      strcpy(temp->testDate, tests[i][2]);
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void insertMedicalTest(struct MedicalTestNode *p, char patientName[], char testName[], char
testDate[])
{
   struct MedicalTestNode *temp, *last = p;
   temp = (struct MedicalTestNode *)malloc(sizeof(struct MedicalTestNode));
   strcpy(temp->patientName, patientName);
   strcpy(temp->testName, testName);
   strcpy(temp->testDate, testDate);
```

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

void displayMedicalTests(struct MedicalTestNode *p)
{
    while (p != NULL)
    {
        printf("Patient: %s, Test: %s, Date: %s\n", p->patientName, p->testName, p->testDate);
        p = p->next;
    }
}

//9.
#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;
    }
}
```

```
// 10.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define structure for hospital staff
struct StaffNode
{
    char name[50];
    char position[50];
    struct StaffNode *next;
} *first = NULL;

// Function prototypes
void createStaffRoster(char staff[][2][50], int n);
void insertStaffMember(struct StaffNode *p, char name[], char position[]);
void displayStaffRoster(struct StaffNode *p);

int main()
{
    char staffRoster[][2][50] = {{"Dr. Smith", "Surgeon"}, {"Nurse Mary", "Nurse"}};
    createStaffRoster(staffRoster, 2);
    printf("Initial hospital staff roster:\n");
    displayStaffRoster(first);
    printf("\nAdding a new staff member:\n");
    insertStaffMember(first, "Dr. John", "Cardiologist");
    displayStaffRoster(first);
    return 0;
}

void createStaffRoster(char staff[][2][50], int n)
```

```
{
   int i;
   struct StaffNode *temp, *last;
   first = (struct StaffNode *)malloc(sizeof(struct  StaffNode));
   strcpy(first->name,  staff[0][0]);
   strcpy(first->position,  staff[0][1]);
   first->next = NULL;
   last = first;

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

void insertStaffMember(struct  StaffNode *p, char name[], char position[])
{
   struct StaffNode *temp, *last = p;
   temp = (struct StaffNode *)malloc(sizeof(struct  StaffNode));
   strcpy(temp->name,  name);
   strcpy(temp->position,  position);
   temp->next = NULL;
   while (last->next  != NULL)
      last = last->next;
   last->next = temp;
}
```

```
void displayStaffRoster(struct  StaffNode *p)
{
   while (p != NULL)
   {
      printf("Name: %s, Position: %s\n", p->name, p->position);
      p = p->next;
   }
}
```