CREATING A LINKED LIST
==========

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

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

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

int main()
{
    int A[] = {1,2,3,4,5};
    create(A,5);
    display(first);

    return 0;
}

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

}

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

}
```

INSERTION
==========

1.VALUE AT BEGINING
==========

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

struct Node{
```

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

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

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

    return 0;
}

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

}

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

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
```

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

2.INSERTING BETWEEN
==========

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

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

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

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

    return 0;
}

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

}

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

```
        p = p->next;
    }

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

3.INSERTING AT LAST

==========

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

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

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

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

    return 0;
}

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

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

}

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

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

USING INSERT FUNCTION ONLY

==========

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

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

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

int main()
{
```

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

    printf("\n");
    display(first);

    return 0;
}

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

}

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

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

SET OF PROBLEMS
==========
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>

// Structure to represent a patient
struct Patient {
    char name[50];
    int age;
    char condition[100];
    struct Patient* next;
};

// Queue structure
struct Queue {
    struct Patient* front;
    struct Patient* rear;
};

// Function to create a new queue
struct Queue* createQueue() {
    struct Queue* newQueue = (struct Queue*)malloc(sizeof(struct Queue));
    newQueue->front = newQueue->rear = NULL;
    return newQueue;
}

// Function to create a new patient node
struct Patient* createPatient(char* name, int age, char* condition) {
    struct Patient* newPatient = (struct Patient*)malloc(sizeof(struct Patient));
    strcpy(newPatient->name, name);
    newPatient->age = age;
    strcpy(newPatient->condition, condition);
    newPatient->next = NULL;
    return newPatient;
}

// Function to insert a patient into the queue
void enqueue(struct Queue* queue, char* name, int age, char* condition) {
    struct Patient* newPatient = createPatient(name, age, condition);

    // If queue is empty, the new patient is both front and rear
    if (queue->rear == NULL) {
        queue->front = queue->rear = newPatient;
        return;
    }

    // Otherwise, add the new patient at the end and update the rear
```

```
    queue->rear->next = newPatient;
    queue->rear = newPatient;
}

// Function to display the queue of patients
void displayQueue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("No patients in the queue.\n");
        return;
    }

    struct Patient* temp = queue->front;
    printf("Patient Queue:\n");
    while (temp != NULL) {
        printf("Name: %s, Age: %d, Condition: %s\n", temp->name, temp->age, temp->condition);
        temp = temp->next;
    }
}

// Main function to demonstrate the queue operations
int main() {
    struct Queue* patientQueue = createQueue();

    // Insert patients into the queue
    enqueue(patientQueue, "John Doe", 30, "Cold");
    enqueue(patientQueue, "Jane Smith", 25, "Fever");
    enqueue(patientQueue, "Tom Brown", 45, "Headache");

    // Display the current queue of patients
    displayQueue(patientQueue);

    return 0;
}
```

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>

// Structure to represent a bed in the hospital ward
typedef struct Bed {
    int bedNumber;
    char patientName[50];
    struct Bed *next;  // Pointer to the next bed
} Bed;

// Function to create an initial list of available beds
Bed* createBedList(int totalBeds) {
    Bed *head = NULL, *temp, *newBed;
```

```
    for (int i = 1; i <= totalBeds; i++) {
        newBed = (Bed*)malloc(sizeof(Bed));
        newBed->bedNumber = i;
        strcpy(newBed->patientName, "Available");  // Mark as available
        newBed->next = NULL;

        if (head == NULL) {
            head = newBed;
        } else {
            temp->next = newBed;
        }
        temp = newBed;
    }

    return head;
}

// Function to insert a patient into an available bed
void allocateBed(Bed *head, int bedNumber, const char *patientName) {
    Bed *temp = head;

    // Find the bed with the given bed number
    while (temp != NULL) {
        if (temp->bedNumber == bedNumber && strcmp(temp->patientName, "Available") == 0) {
            strcpy(temp->patientName, patientName);  // Allocate the bed to the patient
            printf("Bed %d allocated to %s.\n", bedNumber, patientName);
            return;
        }
        temp = temp->next;
    }
    printf("Bed %d is either not available or doesn't exist.\n", bedNumber);
}

// Function to display the current bed allocation
void displayBedAllocation(Bed *head) {
    Bed *temp = head;
    printf("Current Bed Allocation:\n");
    printf("-------------------------------------------------\n");
    while (temp != NULL) {
        printf("Bed %d: %s\n", temp->bedNumber, temp->patientName);
        temp = temp->next;
    }
    printf("-------------------------------------------------\n");
}

// Main function to demonstrate the allocation system
int main() {
    int totalBeds = 5;  // Let's assume there are 5 beds
    Bed *bedList = createBedList(totalBeds);

    // Display initial bed allocation
    displayBedAllocation(bedList);

    // Allocate beds to patients
    allocateBed(bedList, 2, "John Doe");
```

```
    allocateBed(bedList, 4, "Jane Smith");

    // Display updated bed allocation
    displayBedAllocation(bedList);

    // Attempt to allocate a non-available bed
    allocateBed(bedList, 2, "Michael Johnson");

    // Final display of bed allocation
    displayBedAllocation(bedList);

    return 0;
}
```

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>

// Define a structure to represent an inventory item
struct InventoryItem {
    int itemID;
    char itemName[50];
    int quantity;
    float price;
    struct InventoryItem *next; // Pointer to the next item in the list
};

// Function to create an inventory list (initialize the list as NULL)
struct InventoryItem* createInventory() {
    return NULL;
}

// Function to insert a new inventory item at the end of the list
void insertInventoryItem(struct InventoryItem **head, int itemID, const char *itemName, int quantity, float
price) {
    // Create a new inventory item
    struct InventoryItem *newItem = (struct InventoryItem*)malloc(sizeof(struct InventoryItem));
    newItem->itemID = itemID;
    strncpy(newItem->itemName, itemName, sizeof(newItem->itemName) - 1);
    newItem->itemName[sizeof(newItem->itemName) - 1] = '\0'; // Ensure null-termination
    newItem->quantity = quantity;
    newItem->price = price;
    newItem->next = NULL;

    // If the list is empty, the new item becomes the first item
    if (*head == NULL) {
        *head = newItem;
    } else {
        // Traverse to the last item and insert the new item there
```

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

// Function to display the current inventory
void displayInventory(struct InventoryItem *head) {
    if (head == NULL) {
        printf("Inventory is empty.\n");
        return;
    }

    struct InventoryItem *temp = head;
    printf("Current Inventory:\n");
    printf("-----------------------------------------------\n");
    printf("ItemID\tItemName\tQuantity\tPrice\n");
    printf("-----------------------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\t\t%d\t\t%.2f\n", temp->itemID, temp->itemName, temp->quantity, temp->price);
        temp = temp->next;
    }
    printf("-----------------------------------------------\n");
}

int main() {
    struct InventoryItem *inventory = createInventory();  // Start with an empty inventory

    // Insert a few items into the inventory
    insertInventoryItem(&inventory, 101, "Aspirin", 50, 5.99);
    insertInventoryItem(&inventory, 102, "Bandages", 100, 2.49);
    insertInventoryItem(&inventory, 103, "Cough Syrup", 30, 7.89);

    // Display the current inventory
    displayInventory(inventory);

    return 0;
}
```

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>

// Define a structure for a doctor appointment
struct Appointment {
    char patientName[50];
    char appointmentDate[20];
```

```
    struct Appointment *next;
};

// Function to create a new appointment
struct Appointment* createAppointment(char patientName[], char appointmentDate[]) {
    struct Appointment *newAppointment = (struct Appointment*)malloc(sizeof(struct Appointment));
    strcpy(newAppointment->patientName, patientName);
    strcpy(newAppointment->appointmentDate, appointmentDate);
    newAppointment->next = NULL;
    return newAppointment;
}

// Function to insert a new appointment at the end of the list
void insertAppointment(struct Appointment **head, char patientName[], char appointmentDate[]) {
    struct Appointment *newAppointment = createAppointment(patientName, appointmentDate);
    if (*head == NULL) {
        *head = newAppointment;
    } else {
        struct Appointment *temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newAppointment;
    }
}

// Function to display all scheduled appointments
void displayAppointments(struct Appointment *head) {
    if (head == NULL) {
        printf("No appointments scheduled.\n");
    } else {
        struct Appointment *temp = head;
        printf("\nScheduled Appointments:\n");
        while (temp != NULL) {
            printf("Patient: %s, Appointment Date: %s\n", temp->patientName, temp->appointmentDate);
            temp = temp->next;
        }
    }
}

// Main function to test the program
int main() {
    struct Appointment *appointmentList = NULL;
    int choice;
    char patientName[50], appointmentDate[20];

    while (1) {
        printf("\nDoctor Appointment Scheduling System\n");
        printf("1. Insert New Appointment\n");
        printf("2. Display All Appointments\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // To consume the newline character left by scanf
```

```
        switch (choice) {
            case 1:
                printf("Enter Patient Name: ");
                fgets(patientName, sizeof(patientName), stdin);
                patientName[strcspn(patientName, "\n")] = 0;  // Remove trailing newline
                printf("Enter Appointment Date (e.g., YYYY-MM-DD): ");
                fgets(appointmentDate, sizeof(appointmentDate), stdin);
                appointmentDate[strcspn(appointmentDate, "\n")] = 0;  // Remove trailing newline
                insertAppointment(&appointmentList, patientName, appointmentDate);
                break;
            case 2:
                displayAppointments(appointmentList);
                break;
            case 3:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Problem 5: Emergency Contact List
Description: Implement a linked list to manage emergency contacts for hospital staff. Operations:
Create a contact list.
Insert a new contact.
Display all emergency contacts.

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

// Define the structure for a contact
struct Contact {
    char name[50];
    char phone[15];
    struct Contact* next;
};

// Function to create a new contact
struct Contact* createContact(char* name, char* phone) {
    struct Contact* newContact = (struct Contact*)malloc(sizeof(struct Contact));
    if (newContact == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    strcpy(newContact->name, name);
    strcpy(newContact->phone, phone);
    newContact->next = NULL;
    return newContact;
}

// Function to insert a new contact at the end of the list
```

```
void insertContact(struct Contact** head, char* name, char* phone) {
    struct Contact* newContact = createContact(name, phone);
    if (*head == NULL) {
        *head = newContact; // If the list is empty, make the new contact the head
    } else {
        struct Contact* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next; // Traverse to the last contact
        }
        temp->next = newContact; // Insert the new contact at the end
    }
}

// Function to display all emergency contacts
void displayContacts(struct Contact* head) {
    if (head == NULL) {
        printf("No contacts available.\n");
        return;
    }
    struct Contact* temp = head;
    printf("Emergency Contact List:\n");
    while (temp != NULL) {
        printf("Name: %s, Phone: %s\n", temp->name, temp->phone);
        temp = temp->next;
    }
}

// Main function
int main() {
    struct Contact* contactList = NULL; // Initialize an empty list
    int choice;
    char name[50], phone[15];

    while (1) {
        printf("\nEmergency Contact List Menu:\n");
        printf("1. Insert a new contact\n");
        printf("2. Display all contacts\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // To consume the newline character left by scanf

        switch (choice) {
            case 1:
                printf("Enter the name of the contact: ");
                fgets(name, sizeof(name), stdin);
                name[strcspn(name, "\n")] = '\0'; // Remove trailing newline from input
                printf("Enter the phone number: ");
                fgets(phone, sizeof(phone), stdin);
                phone[strcspn(phone, "\n")] = '\0'; // Remove trailing newline from input
                insertContact(&contactList, name, phone);
                printf("Contact added successfully!\n");
                break;

            case 2:
```

```
            displayContacts(contactList);
            break;

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

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

   return 0;
}
```

Problem 6: Surgery Scheduling System
Description: Use a linked list to manage surgery schedules. Operations:
Create a surgery schedule.
Insert a new surgery into the schedule.
Display all scheduled surgeries.

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

// Define the structure for a surgery node
struct Surgery {
   char name[100];
   char date[20];
   struct Surgery* next;
};

// Function to create a new surgery node
struct Surgery* createSurgery(char name[], char date[]) {
   struct Surgery* newSurgery = (struct Surgery*)malloc(sizeof(struct Surgery));
   strcpy(newSurgery->name, name);
   strcpy(newSurgery->date, date);
   newSurgery->next = NULL;
   return newSurgery;
}

// Function to insert a new surgery into the schedule
void insertSurgery(struct Surgery** head, char name[], char date[]) {
   struct Surgery* newSurgery = createSurgery(name, date);
   if (*head == NULL) {
      *head = newSurgery;  // If the list is empty, set the head to the new surgery
   } else {
      struct Surgery* temp = *head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newSurgery;  // Add the new surgery at the end of the list
   }
}
```

```
// Function to display all scheduled surgeries
void displaySurgeries(struct Surgery* head) {
    if (head == NULL) {
        printf("No surgeries scheduled.\n");
        return;
    }
    struct Surgery* temp = head;
    while (temp != NULL) {
        printf("Surgery: %s, Date: %s\n", temp->name, temp->date);
        temp = temp->next;
    }
}

// Main function to test the scheduling system
int main() {
    struct Surgery* schedule = NULL;
    int choice;
    char name[100], date[20];

    while (1) {
        printf("\nSurgery Scheduling System\n");
        printf("1. Create a new surgery schedule\n");
        printf("2. Insert a new surgery into the schedule\n");
        printf("3. Display all scheduled surgeries\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                schedule = NULL; // Reset the schedule (creating a new one)
                printf("New surgery schedule created.\n");
                break;

            case 2:
                printf("Enter surgery name: ");
                scanf(" %[^\n]s", name);  // Read string with spaces
                printf("Enter surgery date (YYYY-MM-DD): ");
                scanf(" %[^\n]s", date);  // Read string with spaces
                insertSurgery(&schedule, name, date);
                printf("Surgery added to the schedule.\n");
                break;

            case 3:
                printf("Scheduled surgeries:\n");
                displaySurgeries(schedule);
                break;

            case 4:
                printf("Exiting the system.\n");
                exit(0);
                break;

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

```
        }
    }

    return 0;
}
```

Problem 7: Patient History Record
Description: Maintain a linked list to keep track of patient history records. Operations:
Create a history record list.
Insert a new record.
Display all patient history records.

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

// Define the structure for a patient history record
struct PatientHistory {
    int patientID;
    char name[100];
    char disease[100];
    struct PatientHistory *next;
};

// Function to create a new record
struct PatientHistory* createHistoryRecord(int patientID, char* name, char* disease) {
    struct PatientHistory* newRecord = (struct PatientHistory*)malloc(sizeof(struct PatientHistory));

    newRecord->patientID = patientID;
    strcpy(newRecord->name, name);
    strcpy(newRecord->disease, disease);
    newRecord->next = NULL;

    return newRecord;
}

// Function to insert a new record at the end of the list
void insertRecord(struct PatientHistory** head, int patientID, char* name, char* disease) {
    struct PatientHistory* newRecord = createHistoryRecord(patientID, name, disease);

    if (*head == NULL) {
        *head = newRecord;
    } else {
        struct PatientHistory* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newRecord;
    }
}

// Function to display all patient history records
void displayRecords(struct PatientHistory* head) {
    if (head == NULL) {
        printf("No patient history records available.\n");
```

```
        return;
    }

    struct PatientHistory* temp = head;
    while (temp != NULL) {
        printf("Patient ID: %d\n", temp->patientID);
        printf("Name: %s\n", temp->name);
        printf("Disease: %s\n", temp->disease);
        printf("------------------------------\n");
        temp = temp->next;
    }
}

int main() {
    struct PatientHistory* head = NULL;

    // Insert patient records
    insertRecord(&head, 1, "John Doe", "Flu");
    insertRecord(&head, 2, "Jane Smith", "Cold");
    insertRecord(&head, 3, "Alice Johnson", "Headache");

    // Display patient records
    displayRecords(head);

    return 0;
}
```

Problem 8: Medical Test Tracking
Description: Implement a linked list to track medical tests for patients. Operations:
Create a list of medical tests.
Insert a new test result.
Display all test results.

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

// Define a structure for a medical test
struct TestResult {
    char testName[50];
    char result[50];
    struct TestResult* next;
};

// Function to create a new test result node
struct TestResult* createTestResult(char* testName, char* result) {
    struct TestResult* newTest = (struct TestResult*)malloc(sizeof(struct TestResult));
    if (newTest == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    strcpy(newTest->testName, testName);
    strcpy(newTest->result, result);
    newTest->next = NULL;
    return newTest;
```

```
}

// Function to insert a new test result at the end of the list
void insertTestResult(struct TestResult** head, char* testName, char* result) {
    struct TestResult* newTest = createTestResult(testName, result);
    if (*head == NULL) {
        *head = newTest;
    } else {
        struct TestResult* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTest;
    }
}

// Function to display all test results
void displayTestResults(struct TestResult* head) {
    if (head == NULL) {
        printf("No test results available.\n");
        return;
    }

    struct TestResult* temp = head;
    while (temp != NULL) {
        printf("Test Name: %s, Result: %s\n", temp->testName, temp->result);
        temp = temp->next;
    }
}

int main() {
    struct TestResult* head = NULL;

    // Inserting some test results
    insertTestResult(&head, "Blood Test", "Normal");
    insertTestResult(&head, "X-Ray", "Clear");
    insertTestResult(&head, "ECG", "Irregular");

    // Displaying all test results
    printf("Medical Test Results:\n");
    displayTestResults(head);

    // Free memory (important in real-world applications)
    struct TestResult* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

Problem 9: Prescription Management System
Description: Use a linked list to manage patient prescriptions. Operations:

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

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

// Structure to store prescription data
struct Prescription {
    char patientName[100];
    char medication[100];
    char dosage[100];
    struct Prescription *next;
};

// Function to create a new prescription node
struct Prescription* createPrescription(char* patientName, char* medication, char* dosage) {
    struct Prescription* newPrescription = (struct Prescription*)malloc(sizeof(struct Prescription));
    strcpy(newPrescription->patientName, patientName);
    strcpy(newPrescription->medication, medication);
    strcpy(newPrescription->dosage, dosage);
    newPrescription->next = NULL;
    return newPrescription;
}

// Function to insert a prescription at the end of the list
void insertPrescription(struct Prescription** head, char* patientName, char* medication, char* dosage) {
    struct Prescription* newPrescription = createPrescription(patientName, medication, dosage);
    if (*head == NULL) {
        *head = newPrescription;
    } else {
        struct Prescription* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPrescription;
    }
}

// Function to display all prescriptions in the list
void displayPrescriptions(struct Prescription* head) {
    if (head == NULL) {
        printf("No prescriptions found.\n");
        return;
    }

    struct Prescription* temp = head;
    while (temp != NULL) {
        printf("Patient Name: %s\n", temp->patientName);
        printf("Medication: %s\n", temp->medication);
        printf("Dosage: %s\n", temp->dosage);
        printf("--------------------------\n");
        temp = temp->next;
    }
```

```
}

int main() {
    struct Prescription* head = NULL;  // Initialize the head of the linked list as NULL
    int choice;
    char patientName[100], medication[100], dosage[100];

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

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

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

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

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

            case 2:
                displayPrescriptions(head);
                break;

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

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

    return 0;
}
```

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

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

```
// Define a structure to hold staff information
struct Staff {
    char name[50];
    int id;
    struct Staff *next;
};

// Function to create a new staff node
struct Staff* createStaffNode(int id, const char* name) {
    struct Staff* newStaff = (struct Staff*)malloc(sizeof(struct Staff));
    if (newStaff == NULL) {
        printf("Memory allocation failed\n");
        return NULL;
    }
    newStaff->id = id;
    strcpy(newStaff->name, name);
    newStaff->next = NULL;
    return newStaff;
}

// Function to insert a new staff member into the roster
void insertStaff(struct Staff** head, int id, const char* name) {
    struct Staff* newStaff = createStaffNode(id, name);
    if (newStaff == NULL) return;

    newStaff->next = *head;
    *head = newStaff;
}

// Function to display the current staff roster
void displayRoster(struct Staff* head) {
    if (head == NULL) {
        printf("The staff roster is empty.\n");
        return;
    }

    printf("Current Staff Roster:\n");
    struct Staff* temp = head;
    while (temp != NULL) {
        printf("ID: %d, Name: %s\n", temp->id, temp->name);
        temp = temp->next;
    }
}

int main() {
    struct Staff* roster = NULL;

    // Insert some staff members into the roster
    insertStaff(&roster, 1, "Dr. Alice");
    insertStaff(&roster, 2, "Nurse Bob");
    insertStaff(&roster, 3, "Dr. Charlie");

    // Display the current roster
    displayRoster(roster);
```

```
    // Insert more staff members
    insertStaff(&roster, 4, "Dr. Dana");
    insertStaff(&roster, 5, "Nurse Eve");

    // Display the updated roster
    displayRoster(roster);

    return 0;
}
```