Problem 1: Inventory Management System
Description: Implement a linked list to manage the inventory of raw materials.
Operations:
Create an inventory list.
Insert a new raw material.
Delete a raw material from the inventory.
Display the current inventory.

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

struct Node {
    char material[100];
    int quantity;
    struct Node *next;
} *first = NULL;

void create();
void insertMaterial(char material[], int quantity);
int deleteMaterial(char material[]);
void displayAll();

int main() {
    int choice, deletes;
    char material[100];
    int quantity;

    while (1) {
        printf("\nInventory Management System\n");
        printf("1. Create Inventory\n");
        printf("2. Insert Raw Material\n");
        printf("3. Delete Raw Material\n");
        printf("4. Display Inventory\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
```

```
            break;
        case 2:
            printf("Enter material name: ");
            scanf("%s", material);
            printf("Enter quantity: ");
            scanf("%d", &quantity);
            insertMaterial(material, quantity);
            break;
        case 3:
            printf("Enter material name to delete: ");
            scanf("%s", material);
            deletes = deleteMaterial(material);
            if (deletes) {
                printf("Material '%s' deleted successfully.\n", material);
            } else {
                printf("Material '%s' not found.\n", material);
            }
            break;
        case 4:
            displayAll();
            break;
        case 5:
            printf("Exiting the system.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
   return 0;
}

void create() {
   if (first != NULL) {
      printf("Inventory already exists.\n");
   } else {
      first = NULL;
      printf("Inventory created successfully.\n");
   }
}

void insertMaterial(char material[], int quantity) {
```

```
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    struct Node *current = first;

    strcpy(temp->material, material);
    temp->quantity = quantity;
    temp->next = NULL;

    if (first == NULL) {
        first = temp;
    } else {
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = temp;
    }

    printf("Material '%s' with quantity %d added successfully.\n", material, quantity);
}

int deleteMaterial(char material[]) {
    struct Node *current = first;
    struct Node *previous = NULL;

    while (current != NULL) {
        if (strcmp(current->material, material) == 0) {
            if (previous == NULL) { // Material is in the first node
                first = current->next;
            } else {
                previous->next = current->next;
            }
            free(current);
            return 1; // Material deleted successfully
        }
        previous = current;
        current = current->next;
    }
    return 0;
}

void displayAll() {
    struct Node *current = first;
```

```
    if (current == NULL) {
        printf("Inventory is empty.\n");
        return;
    }

    printf("\nCurrent Inventory:\n");
    printf("Material\tQuantity\n");
    printf("------------------------\n");
    while (current != NULL) {
        printf("%s\t\t%d\n", current->material, current->quantity);
        current = current->next;
    }
}
```

Problem 2: Production Line Queue
Description: Use a linked list to manage the queue of tasks on a production line.
Operations:
Create a production task queue.
Insert a new task into the queue.
Delete a completed task.
Display the current task queue.

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

struct Node {
    int id;
    char taskName[100];
    struct Node *next;
} *front = NULL, *rear = NULL;

void createQueue();
void InsertTask(char taskName[], int id);
void deleteTask();
```

```
void displayTask();

int main() {
    int id, choice;
    char taskName[100];

    while (1) {
        printf("\nProduction Line Queue Management\n");
        printf("1. Create Task Queue\n");
        printf("2. Insert New Task\n");
        printf("3. Delete Completed Task\n");
        printf("4. Display Task Queue\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createQueue();
                break;
            case 2:
                printf("Enter Task ID: ");
                scanf("%d", &id);
                printf("Enter Task Name: ");
                getchar();
                scanf("%[^\n]s", taskName);
                InsertTask(taskName, id);
                break;
            case 3:
                deleteTask();
                break;
            case 4:
                displayTask();
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid Choice. Please try again.\n");
        }
    }
```

```
    return 0;
}

void createQueue() {
    if (front != NULL) {
        printf("Task queue already exists.\n");
    } else {
        front = rear = NULL;
        printf("Task queue created successfully.\n");
    }
}

void InsertTask(char taskName[], int id) {
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->id = id;
    strcpy(temp->taskName, taskName);
    temp->next = NULL;

    if (front == NULL) {
        front = rear = temp;
    } else {
        rear->next = temp;
        rear = temp;
    }
    printf("Task ID %d => '%s' added to the queue successfully.\n", id, taskName);
}

void deleteTask() {
    if (front == NULL) {
        printf("Task queue is empty.\n");
    } else {
        struct Node *temp = front;
        printf("Completed task '%s' (ID: %d) is removed from the queue.\n",
front->taskName, front->id);
        front = front->next;

        if (front == NULL) {
            rear = NULL;
        }
        free(temp);
    }
```

```
}

void displayTask() {
    struct Node *current = front;

    if (current == NULL) {
        printf("Task queue is empty.\n");
        return;
    }

    printf("\nCurrent Task Queue:\n");
    printf("Task ID\t\tTask Name\n");
    printf("------------------------------\n");
    while (current != NULL) {
        printf("%d\t\t%s\n", current->id, current->taskName);
        current = current->next;
    }
}
```

Problem 3: Machine Maintenance Schedule
Description: Develop a linked list to manage the maintenance schedule of machines.
Operations:
Create a maintenance schedule.
Insert a new maintenance task.
Delete a completed maintenance task.
Display the maintenance schedule.

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

struct Node
{
    char date[100];
    char task[100];
    struct Node *next;
```

```
}*head =NULL;

void createSchedule();
void inserttask(char date[],char task[]);
void deletetask(char task[]);
void display();

int main(){
   int choice;
   char date[100],task[100];

   while(1){
       printf("\nMachine Maintenance Schedule Management\n");
      printf("1. Create Maintenance Schedule\n");
      printf("2. Insert New Maintenance Task\n");
      printf("3. Delete Completed Maintenance Task\n");
      printf("4. Display Maintenance Schedule\n");
      printf("5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch(choice){
         case 1:
            createSchedule();
            break;
         case 2:
            printf("Enter Maintenane Date (YYYY-MM-DD): ");
            getchar();
            scanf("%[^\n]s", date);
            printf("Enter Maintenance Task: ");
            getchar();
            scanf("%[^\n]s", task);
            inserttask(date,task);
            break;
         case 3:
            printf("Enter the Task to Delete: ");
            getchar();
            scanf("%[^\n]s", task);
            deletetask(task);
            break;
         case 4:
```

```
            display();
            break;
        case 5:
            printf("Exiting.....\n");
            exit(0);
        default:
            printf("Invalid choice ");

    }

  }
  return 0;
}


void createSchedule(){
  if(head!=NULL){
    printf("Maintenance schedule already exists.\n");
  }
  else{
    head = NULL;
    printf("Maintenance schedule created successfully.\n");
  }
}
void inserttask(char date[],char task[]){
  struct Node *temp = (struct Node*)malloc(sizeof(struct Node ));
  struct Node *current;

  strcpy(temp->date,date);
  strcpy(temp->task,task);
  temp->next =NULL;

  if(head ==NULL){
    head =temp;
  }else{
    current =head;
    while(current->next !=NULL){
      current = current->next;
    }
    current->next = temp;
  }
```

```
    printf("Task '%s' scheduled for %s added successfully.\n", task, date);
}
void deletetask(char task[]){
    struct Node *current = head;
    struct Node *previous = NULL;

    while(current != NULL){
        if(strcmp(current->task,task)==0){
            if(previous==NULL){
                head = current->next;
            }else{
                previous->next = current->next;
            }
            free(current);
            printf("Task '%s' removed from the schedule.\n", task);
            return;
        }
        previous = current;
        current = current->next;
    }
    printf("Task '%s' not found in the schedule.\n", task);
}
void display(){
    struct Node *current = head;
    if(current ==NULL){
        printf("The maintenance schedule is empty.\n");
        return;
    }
    printf("\n Maintenance Schedule \n");
    printf("Date\t\tTask\n");
    printf("----------------------------\n");
    while(current!=NULL){
        printf("%s\t%s\n",current->date,current->task);
        current = current->next;
    }
}
```

Problem 4: Employee Shift Management
Description: Use a linked list to manage employee shifts in a manufacturing plant.
Operations:
Create a shift schedule.
Insert a new shift.
Delete a completed or canceled shift.
Display the current shift schedule.

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

struct Node {
    int employeeID;
    char employeeName[100];
    char shiftTime[100];
    struct Node *next;
} *head = NULL;

void createShiftSchedule();
void insertShift(int employeeID, char employeeName[], char shiftTime[]);
void deleteShift(int employeeID);
void displayShiftSchedule();

int main() {
    int choice, employeeID;
    char employeeName[100], shiftTime[100];

    while (1) {
        printf("\nEmployee Shift Management\n");
        printf("1. Create Shift Schedule\n");
        printf("2. Insert New Shift\n");
        printf("3. Delete Completed or Canceled Shift\n");
        printf("4. Display Current Shift Schedule\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                createShiftSchedule();
                break;
            case 2:
                printf("Enter Employee ID: ");
                scanf("%d", &employeeID);
                printf("Enter Employee Name: ");
                getchar();
                scanf("%[^\n]s", employeeName);
                printf("Enter Shift Time (e.g., 9 AM - 5 PM): ");
                getchar();
                scanf("%[^\n]s", shiftTime);
                insertShift(employeeID, employeeName, shiftTime);
                break;
            case 3:
                printf("Enter Employee ID to Delete Shift: ");
                scanf("%d", &employeeID);
                deleteShift(employeeID);
                break;
            case 4:
                displayShiftSchedule();
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid Choice! Please try again.\n");
        }
    }

    return 0;
}

void createShiftSchedule() {
    if (head != NULL) {
        printf("Shift schedule already exists.\n");
    } else {
        head = NULL;
        printf("Shift schedule created successfully.\n");
    }
}
```

```
void insertShift(int employeeID, char employeeName[], char shiftTime[]) {
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    struct Node *current;

    temp->employeeID = employeeID;
    strcpy(temp->employeeName, employeeName);
    strcpy(temp->shiftTime, shiftTime);
    temp->next = NULL;

    if (head == NULL) {
        head = temp;
    } else {
        current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = temp;
    }
    printf("Shift for Employee ID %d ('%s') at '%s' added successfully.\n", employeeID,
employeeName, shiftTime);
}

void deleteShift(int employeeID) {
    struct Node *current = head;
    struct Node *previous = NULL;

    while (current != NULL) {
        if (current->employeeID == employeeID) {
            if (previous == NULL) {
                head = current->next;
            } else {
                previous->next = current->next;
            }
            free(current);
            printf("Shift for Employee ID %d removed from the schedule.\n", employeeID);
            return;
        }
        previous = current;
        current = current->next;
    }
```

```
    printf("Shift for Employee ID %d not found in the schedule.\n", employeeID);
}

void displayShiftSchedule() {
    struct Node *current = head;

    if (current == NULL) {
        printf("The shift schedule is empty.\n");
        return;
    }

    printf("\nCurrent Shift Schedule:\n");
    printf("Employee ID\tEmployee Name\t\tShift Time\n");
    printf("-------------------------------------------------\n");
    while (current != NULL) {
        printf("%d\t\t%s\t\t%s\n", current->employeeID, current->employeeName,
current->shiftTime);
        current = current->next;
    }
}
```

Problem 5: Order Processing System
Description: Implement a linked list to track customer orders.
Operations:
Create an order list.
Insert a new customer order.
Delete a completed or canceled order.
Display all current orders.

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

struct Node {
    int orderID;
    char customerName[100];
    char orderDetails[200];
    struct Node *next;
} *head = NULL;

void createOrderList();
void insertOrder(int orderID, char customerName[], char orderDetails[]);
void deleteOrder(int orderID);
void displayOrders();

int main() {
    int choice, orderID;
    char customerName[100], orderDetails[200];

    while (1) {
        printf("\nOrder Processing System\n");
        printf("1. Create Order List\n");
        printf("2. Insert New Order\n");
        printf("3. Delete Completed or Canceled Order\n");
        printf("4. Display Current Orders\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createOrderList();
                break;
            case 2:
                printf("Enter Order ID: ");
                scanf("%d", &orderID);
                printf("Enter Customer Name: ");
                getchar();
```

```
            scanf("%[^\n]s", customerName);
            printf("Enter Order Details: ");
            getchar();
            scanf("%[^\n]s", orderDetails);
            insertOrder(orderID, customerName, orderDetails);
            break;
        case 3:
            printf("Enter Order ID to Delete: ");
            scanf("%d", &orderID);
            deleteOrder(orderID);
            break;
        case 4:
            displayOrders();
            break;
        case 5:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid Choice! Please try again.\n");
      }
   }

   return 0;
}

void createOrderList() {
   if (head != NULL) {
      printf("Order list already exists.\n");
   } else {
      head = NULL;
      printf("Order list created successfully.\n");
   }
}

void insertOrder(int orderID, char customerName[], char orderDetails[]) {
   struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
   struct Node *current;

   temp->orderID = orderID;
   strcpy(temp->customerName, customerName);
   strcpy(temp->orderDetails, orderDetails);
```

```
    temp->next = NULL;

    if (head == NULL) {
        head = temp;
    } else {
        current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = temp;
    }
    printf("Order ID %d for '%s' added successfully.\n", orderID, customerName);
}

void deleteOrder(int orderID) {
    struct Node *current = head;
    struct Node *previous = NULL;

    while (current != NULL) {
        if (current->orderID == orderID) {
            if (previous == NULL) {
                head = current->next;
            } else {
                previous->next = current->next;
            }
            free(current);
            printf("Order ID %d removed from the list.\n", orderID);
            return;
        }
        previous = current;
        current = current->next;
    }
    printf("Order ID %d not found in the list.\n", orderID);
}

void displayOrders() {
    struct Node *current = head;

    if (current == NULL) {
        printf("No orders in the list.\n");
        return;
```

```
    }

    printf("\nCurrent Orders:\n");
    printf("Order ID\tCustomer Name\t\tOrder Details\n");
    printf("-----------------------------------------------\n");
    while (current != NULL) {
        printf("%d\t\t%s\t\t%s\n", current->orderID, current->customerName,
current->orderDetails);
        current = current->next;
    }
}
```

Problem 6: Tool Tracking System
Description: Maintain a linked list to track tools used in the manufacturing process.
Operations:
Create a tool tracking list.
Insert a new tool entry.
Delete a tool that is no longer in use.
Display all tools currently tracked.

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

struct Node {
    char toolName[100];
    int toolID;
    struct Node* next;
} *first = NULL;

void create();
void insertTool(char toolName[], int toolID);
void deleteTool(int toolID);
void displayTools();

int main() {
```

```
    int toolID, choice;
    char toolName[100];

    while (1) {
        printf("\nTool Tracking System\n");
        printf("1. Create Tool List\n");
        printf("2. Insert New Tool\n");
        printf("3. Delete Tool\n");
        printf("4. Display All Tools\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Tool Name: ");
                scanf("%s", toolName);
                printf("Enter Tool ID: ");
                scanf("%d", &toolID);
                insertTool(toolName, toolID);
                break;
            case 3:
                printf("Enter Tool ID to delete: ");
                scanf("%d", &toolID);
                deleteTool(toolID);
                break;
            case 4:
                displayTools();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
    return 0;
}
```

```
void create() {
    if (first != NULL) {
        printf("Tool list already exists.\n");
    } else {
        first = NULL;
        printf("Tool list created successfully.\n");
    }
}

void insertTool(char toolName[], int toolID) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->toolID = toolID;
    strcpy(temp->toolName, toolName);
    temp->next = first;
    first = temp;
    printf("Tool '%s' with ID %d added successfully.\n", toolName, toolID);
}

void deleteTool(int toolID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("Tool list is empty.\n");
        return;
    }
    if (first->toolID == toolID) {
        first = first->next;
        free(temp);
        printf("Tool with ID %d deleted.\n", toolID);
        return;
    }
    while (temp != NULL && temp->toolID != toolID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Tool with ID %d not found.\n", toolID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Tool with ID %d deleted.\n", toolID);
```

```
}

void displayTools() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No tools tracked.\n");
        return;
    }
    printf("Tool Name\tTool ID\n");
    printf("----------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%d\n", temp->toolName, temp->toolID);
        temp = temp->next;
    }
}
```

Problem 7: Product Assembly Line
Description: Use a linked list to manage the assembly stages of a product.
Operations:
Create an assembly line stage list.
Insert a new stage.
Delete a completed stage.
Display the current assembly stages.

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

struct Node {
    int stageID;
    char stageName[100];
    struct Node* next;
} *first = NULL;

void create();
void insertStage(int stageID, char stageName[]);
void deleteStage(int stageID);
void displayStages();

int main() {
    int stageID, choice;
```

```
    char stageName[100];

    while (1) {
        printf("\nProduct Assembly Line\n");
        printf("1. Create Assembly Line\n");
        printf("2. Insert New Stage\n");
        printf("3. Delete Completed Stage\n");
        printf("4. Display All Stages\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Stage ID: ");
                scanf("%d", &stageID);
                printf("Enter Stage Name: ");
                scanf("%s", stageName);
                insertStage(stageID, stageName);
                break;
            case 3:
                printf("Enter Stage ID to delete: ");
                scanf("%d", &stageID);
                deleteStage(stageID);
                break;
            case 4:
                displayStages();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
    return 0;
}

void create() {
```

```
    if (first != NULL) {
        printf("Assembly line already exists.\n");
    } else {
        first = NULL;
        printf("Assembly line created successfully.\n");
    }
}

void insertStage(int stageID, char stageName[]) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->stageID = stageID;
    strcpy(temp->stageName, stageName);
    temp->next = first;
    first = temp;
    printf("Stage '%s' with ID %d added successfully.\n", stageName, stageID);
}

void deleteStage(int stageID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No stages available.\n");
        return;
    }
    if (first->stageID == stageID) {
        first = first->next;
        free(temp);
        printf("Stage with ID %d deleted.\n", stageID);
        return;
    }
    while (temp != NULL && temp->stageID != stageID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Stage with ID %d not found.\n", stageID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Stage with ID %d deleted.\n", stageID);
}
```

```
void displayStages() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No stages in the assembly line.\n");
        return;
    }
    printf("Stage ID\tStage Name\n");
    printf("-------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->stageID, temp->stageName);
        temp = temp->next;
    }
}
```

Problem 8: Quality Control Checklist
Description: Implement a linked list to manage a quality control checklist.
Operations:
Create a quality control checklist.
Insert a new checklist item.
Delete a completed or outdated checklist item.
Display the current quality control checklist.

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

struct Node {
    int itemID;
    char itemName[100];
    struct Node* next;
} *first = NULL;

void create();
void insertItem(int itemID, char itemName[]);
void deleteItem(int itemID);
```

```
void displayItems();

int main() {
    int itemID, choice;
    char itemName[100];

    while (1) {
        printf("\nQuality Control Checklist\n");
        printf("1. Create Checklist\n");
        printf("2. Insert New Item\n");
        printf("3. Delete Item\n");
        printf("4. Display All Items\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Item ID: ");
                scanf("%d", &itemID);
                printf("Enter Item Name: ");
                scanf("%s", itemName);
                insertItem(itemID, itemName);
                break;
            case 3:
                printf("Enter Item ID to delete: ");
                scanf("%d", &itemID);
                deleteItem(itemID);
                break;
            case 4:
                displayItems();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
```

```
    return 0;
}

void create() {
    if (first != NULL) {
        printf("Checklist already exists.\n");
    } else {
        first = NULL;
        printf("Checklist created successfully.\n");
    }
}

void insertItem(int itemID, char itemName[]) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->itemID = itemID;
    strcpy(temp->itemName, itemName);
    temp->next = first;
    first = temp;
    printf("Item '%s' with ID %d added successfully.\n", itemName, itemID);
}

void deleteItem(int itemID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No items available.\n");
        return;
    }
    if (first->itemID == itemID) {
        first = first->next;
        free(temp);
        printf("Item with ID %d deleted.\n", itemID);
        return;
    }
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Item with ID %d not found.\n", itemID);
        return;
    }
```

```
    prev->next = temp->next;
    free(temp);
    printf("Item with ID %d deleted.\n", itemID);
}

void displayItems() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No items in the checklist.\n");
        return;
    }
    printf("Item ID\tItem Name\n");
    printf("----------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->itemID, temp->itemName);
        temp = temp->next;
    }
}
```

Problem 9: Supplier Management System
Description: Use a linked list to manage a list of suppliers.
Operations:
Create a supplier list.
Insert a new supplier.
Delete an inactive or outdated supplier.
Display all current suppliers.

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

struct Node {
    int supplierID;
    char supplierName[100];
    struct Node* next;
} *first = NULL;

void create();
void insertSupplier(int supplierID, char supplierName[]);
void deleteSupplier(int supplierID);
void displaySuppliers();
```

```
int main() {
    int supplierID, choice;
    char supplierName[100];

    while (1) {
        printf("\nSupplier Management System\n");
        printf("1. Create Supplier List\n");
        printf("2. Insert New Supplier\n");
        printf("3. Delete Supplier\n");
        printf("4. Display All Suppliers\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Supplier ID: ");
                scanf("%d", &supplierID);
                printf("Enter Supplier Name: ");
                scanf("%s", supplierName);
                insertSupplier(supplierID, supplierName);
                break;
            case 3:
                printf("Enter Supplier ID to delete: ");
                scanf("%d", &supplierID);
                deleteSupplier(supplierID);
                break;
            case 4:
                displaySuppliers();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
    return 0;
}
```

```
void create() {
    if (first != NULL) {
        printf("Supplier list already exists.\n");
    } else {
        first = NULL;
        printf("Supplier list created successfully.\n");
    }
}

void insertSupplier(int supplierID, char supplierName[]) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->supplierID = supplierID;
    strcpy(temp->supplierName, supplierName);
    temp->next = first;
    first = temp;
    printf("Supplier '%s' with ID %d added successfully.\n", supplierName, supplierID);
}

void deleteSupplier(int supplierID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No suppliers available.\n");
        return;
    }
    if (first->supplierID == supplierID) {
        first = first->next;
        free(temp);
        printf("Supplier with ID %d deleted.\n", supplierID);
        return;
    }
    while (temp != NULL && temp->supplierID != supplierID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Supplier with ID %d not found.\n", supplierID);
        return;
    }
    prev->next = temp->next;
    free(temp);
```

```
    printf("Supplier with ID %d deleted.\n", supplierID);
}

void displaySuppliers() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No suppliers in the list.\n");
        return;
    }
    printf("Supplier ID\tSupplier Name\n");
    printf("----------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->supplierID, temp->supplierName);
        temp = temp->next;
    }
}
```

Problem 10: Manufacturing Project Timeline
Description: Develop a linked list to manage the timeline of a manufacturing project.
Operations:
Create a project timeline.
Insert a new project milestone.
Delete a completed milestone.
Display the current project timeline.

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

struct Node {
    int milestoneID;
    char milestoneName[100];
    struct Node* next;
} *first = NULL;

void create();
void insertMilestone(int milestoneID, char milestoneName[]);
void deleteMilestone(int milestoneID);
void displayMilestones();

int main() {
```

```
    int milestoneID, choice;
    char milestoneName[100];

    while (1) {
        printf("\nManufacturing Project Timeline\n");
        printf("1. Create Project Timeline\n");
        printf("2. Insert New Milestone\n");
        printf("3. Delete Completed Milestone\n");
        printf("4. Display All Milestones\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Milestone ID: ");
                scanf("%d", &milestoneID);
                printf("Enter Milestone Name: ");
                scanf("%s", milestoneName);
                insertMilestone(milestoneID, milestoneName);
                break;
            case 3:
                printf("Enter Milestone ID to delete: ");
                scanf("%d", &milestoneID);
                deleteMilestone(milestoneID);
                break;
            case 4:
                displayMilestones();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
    return 0;
}
```

```
void create() {
    if (first != NULL) {
        printf("Project timeline already exists.\n");
    } else {
        first = NULL;
        printf("Project timeline created successfully.\n");
    }
}

void insertMilestone(int milestoneID, char milestoneName[]) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->milestoneID = milestoneID;
    strcpy(temp->milestoneName, milestoneName);
    temp->next = first;
    first = temp;
    printf("Milestone '%s' with ID %d added successfully.\n", milestoneName,
milestoneID);
}

void deleteMilestone(int milestoneID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No milestones available.\n");
        return;
    }
    if (first->milestoneID == milestoneID) {
        first = first->next;
        free(temp);
        printf("Milestone with ID %d deleted.\n", milestoneID);
        return;
    }
    while (temp != NULL && temp->milestoneID != milestoneID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Milestone with ID %d not found.\n", milestoneID);
        return;
    }
    prev->next = temp->next;
    free(temp);
```

```
    printf("Milestone with ID %d deleted.\n", milestoneID);
}

void displayMilestones() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No milestones in the timeline.\n");
        return;
    }
    printf("Milestone ID\tMilestone Name\n");
    printf("------------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->milestoneID, temp->milestoneName);
        temp = temp->next;
    }
}
```

Problem 11: Warehouse Storage Management
Description: Implement a linked list to manage the storage of goods in a warehouse.
Operations:
Create a storage list.
Insert a new storage entry.
Delete a storage entry when goods are shipped.
Display the current warehouse storage.

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

struct Node {
    int itemID;
    char itemName[100];
    int quantity;
    struct Node* next;
} *first = NULL;

void create();
void insertStorage(int itemID, char itemName[], int quantity);
void deleteStorage(int itemID);
void displayStorage();
```

```
int main() {
    int itemID, choice, quantity;
    char itemName[100];

    while (1) {
        printf("\nWarehouse Storage Management\n");
        printf("1. Create Storage List\n");
        printf("2. Insert New Storage Entry\n");
        printf("3. Delete Storage Entry\n");
        printf("4. Display Current Storage\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Item ID: ");
                scanf("%d", &itemID);
                printf("Enter Item Name: ");
                scanf("%s", itemName);
                printf("Enter Quantity: ");
                scanf("%d", &quantity);
                insertStorage(itemID, itemName, quantity);
                break;
            case 3:
                printf("Enter Item ID to delete: ");
                scanf("%d", &itemID);
                deleteStorage(itemID);
                break;
            case 4:
                displayStorage();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
```

```
    return 0;
}

void create() {
    if (first != NULL) {
        printf("Storage list already exists.\n");
    } else {
        first = NULL;
        printf("Storage list created successfully.\n");
    }
}

void insertStorage(int itemID, char itemName[], int quantity) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->itemID = itemID;
    strcpy(temp->itemName, itemName);
    temp->quantity = quantity;
    temp->next = first;
    first = temp;
    printf("Item '%s' with ID %d and Quantity %d added successfully.\n", itemName,
itemID, quantity);
}

void deleteStorage(int itemID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No items in storage.\n");
        return;
    }
    if (first->itemID == itemID) {
        first = first->next;
        free(temp);
        printf("Item with ID %d deleted from storage.\n", itemID);
        return;
    }
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Item with ID %d not found in storage.\n", itemID);
```

```
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Item with ID %d deleted from storage.\n", itemID);
}

void displayStorage() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No items in storage.\n");
        return;
    }
    printf("Item ID\tItem Name\tQuantity\n");
    printf("-------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\t\t%d\n", temp->itemID, temp->itemName, temp->quantity);
        temp = temp->next;
    }
}
```

Problem 12: Machine Parts Inventory
Description: Use a linked list to track machine parts inventory.
Operations:
Create a parts inventory list.
Insert a new part.
Delete a part that is used up or obsolete.
Display the current parts inventory.

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

struct Node {
    int partID;
    char partName[100];
    int quantity;
```

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

void create();
void insertPart(int partID, char partName[], int quantity);
void deletePart(int partID);
void displayParts();

int main() {
    int partID, choice, quantity;
    char partName[100];

    while (1) {
        printf("\nMachine Parts Inventory\n");
        printf("1. Create Parts Inventory List\n");
        printf("2. Insert New Part\n");
        printf("3. Delete Part\n");
        printf("4. Display Current Inventory\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Part ID: ");
                scanf("%d", &partID);
                printf("Enter Part Name: ");
                scanf("%s", partName);
                printf("Enter Quantity: ");
                scanf("%d", &quantity);
                insertPart(partID, partName, quantity);
                break;
            case 3:
                printf("Enter Part ID to delete: ");
                scanf("%d", &partID);
                deletePart(partID);
                break;
            case 4:
```

```
            displayParts();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid Choice\n");
      }
   }
   return 0;
}

void create() {
   if (first != NULL) {
      printf("Parts inventory list already exists.\n");
   } else {
      first = NULL;
      printf("Parts inventory list created successfully.\n");
   }
}

void insertPart(int partID, char partName[], int quantity) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->partID = partID;
   strcpy(temp->partName, partName);
   temp->quantity = quantity;
   temp->next = first;
   first = temp;
   printf("Part '%s' with ID %d and Quantity %d added successfully.\n", partName,
partID, quantity);
}

void deletePart(int partID) {
   struct Node* temp = first, * prev = NULL;
   if (first == NULL) {
      printf("No parts in inventory.\n");
      return;
   }
   if (first->partID == partID) {
      first = first->next;
      free(temp);
      printf("Part with ID %d deleted from inventory.\n", partID);
```

```
        return;
    }
    while (temp != NULL && temp->partID != partID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Part with ID %d not found in inventory.\n", partID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Part with ID %d deleted from inventory.\n", partID);
}

void displayParts() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No parts in inventory.\n");
        return;
    }
    printf("Part ID\tPart Name\tQuantity\n");
    printf("-------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\t\t%d\n", temp->partID, temp->partName, temp->quantity);
        temp = temp->next;
    }
}
```

Problem 13: Packaging Line Schedule
Description: Manage the schedule of packaging tasks using a linked list.
Operations:
Create a packaging task schedule.
Insert a new packaging task.
Delete a completed packaging task.
Display the current packaging schedule.

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

struct Node {
    int taskID;
    char taskName[100];
    struct Node* next;
} *first = NULL;

void create();
void insertTask(int taskID, char taskName[]);
void deleteTask(int taskID);
void displayTasks();

int main() {
    int taskID, choice;
    char taskName[100];

    while (1) {
        printf("\nPackaging Line Schedule\n");
        printf("1. Create Packaging Task Schedule\n");
        printf("2. Insert New Packaging Task\n");
        printf("3. Delete Completed Packaging Task\n");
        printf("4. Display Current Packaging Tasks\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Task ID: ");
                scanf("%d", &taskID);
                printf("Enter Task Name: ");
                scanf("%s", taskName);
                insertTask(taskID, taskName);
```

```
            break;
        case 3:
            printf("Enter Task ID to delete: ");
            scanf("%d", &taskID);
            deleteTask(taskID);
            break;
        case 4:
            displayTasks();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid Choice\n");
      }
   }
   return 0;
}

void create() {
   if (first != NULL) {
      printf("Packaging task schedule already exists.\n");
   } else {
      first = NULL;
      printf("Packaging task schedule created successfully.\n");
   }
}

void insertTask(int taskID, char taskName[]) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->taskID = taskID;
   strcpy(temp->taskName, taskName);
   temp->next = first;
   first = temp;
   printf("Task '%s' with ID %d added successfully.\n", taskName, taskID);
}

void deleteTask(int taskID) {
   struct Node* temp = first, * prev = NULL;
   if (first == NULL) {
      printf("No tasks in the schedule.\n");
      return;
```

```
    }
    if (first->taskID == taskID) {
        first = first->next;
        free(temp);
        printf("Task with ID %d deleted from schedule.\n", taskID);
        return;
    }
    while (temp != NULL && temp->taskID != taskID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Task with ID %d not found in schedule.\n", taskID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Task with ID %d deleted from schedule.\n", taskID);
}

void displayTasks() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No tasks in the schedule.\n");
        return;
    }
    printf("Task ID\tTask Name\n");
    printf("----------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->taskID, temp->taskName);
        temp = temp->next;
    }
}
```

Problem 14: Production Defect Tracking
Description: Implement a linked list to track defects in the production process.
Operations:
Create a defect tracking list.
Insert a new defect report.
Delete a resolved defect.
Display all current defects.

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

struct Node {
    int defectID;
    char defectDescription[100];
    struct Node* next;
} *first = NULL;

void create();
void insertDefect(int defectID, char defectDescription[]);
void deleteDefect(int defectID);
void displayDefects();

int main() {
    int defectID, choice;
    char defectDescription[100];

    while (1) {
        printf("\nProduction Defect Tracking\n");
        printf("1. Create Defect Tracking List\n");
        printf("2. Insert New Defect Report\n");
        printf("3. Delete Resolved Defect\n");
        printf("4. Display Current Defects\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Defect ID: ");
                scanf("%d", &defectID);
                printf("Enter Defect Description: ");
                scanf("%s", defectDescription);
                insertDefect(defectID, defectDescription);
                break;
```

```
            case 3:
                printf("Enter Defect ID to delete: ");
                scanf("%d", &defectID);
                deleteDefect(defectID);
                break;
            case 4:
                displayDefects();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid Choice\n");
        }
    }
    return 0;
}

void create() {
    if (first != NULL) {
        printf("Defect tracking list already exists.\n");
    } else {
        first = NULL;
        printf("Defect tracking list created successfully.\n");
    }
}

void insertDefect(int defectID, char defectDescription[]) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->defectID = defectID;
    strcpy(temp->defectDescription, defectDescription);
    temp->next = first;
    first = temp;
    printf("Defect '%s' with ID %d added successfully.\n", defectDescription, defectID);
}

void deleteDefect(int defectID) {
    struct Node* temp = first, * prev = NULL;
    if (first == NULL) {
        printf("No defects reported.\n");
        return;
    }
```

```
    if (first->defectID == defectID) {
        first = first->next;
        free(temp);
        printf("Defect with ID %d deleted from tracking.\n", defectID);
        return;
    }
    while (temp != NULL && temp->defectID != defectID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Defect with ID %d not found in tracking.\n", defectID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Defect with ID %d deleted from tracking.\n", defectID);
}

void displayDefects() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No defects reported.\n");
        return;
    }
    printf("Defect ID\tDefect Description\n");
    printf("--------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->defectID, temp->defectDescription);
        temp = temp->next;
    }
}
```

Problem 15: Finished Goods Dispatch System
Description: Use a linked list to manage the dispatch schedule of finished goods.
Operations:
Create a dispatch schedule.
Insert a new dispatch entry.

Delete a dispatched or canceled entry.
Display the current dispatch schedule.

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

struct Node {
    int dispatchID;
    char dispatchDetails[100];
    struct Node* next;
} *first = NULL;

void create();
void insertDispatch(int dispatchID, char dispatchDetails[]);
void deleteDispatch(int dispatchID);
void displayDispatches();

int main() {
    int dispatchID, choice;
    char dispatchDetails[100];

    while (1) {
        printf("\nFinished Goods Dispatch System\n");
        printf("1. Create Dispatch Schedule\n");
        printf("2. Insert New Dispatch Entry\n");
        printf("3. Delete Dispatched Entry\n");
        printf("4. Display Current Dispatch Schedule\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                printf("Enter Dispatch ID: ");
                scanf("%d", &dispatchID);
                printf("Enter Dispatch Details: ");
```

```
            scanf("%s", dispatchDetails);
            insertDispatch(dispatchID, dispatchDetails);
            break;
        case 3:
            printf("Enter Dispatch ID to delete: ");
            scanf("%d", &dispatchID);
            deleteDispatch(dispatchID);
            break;
        case 4:
            displayDispatches();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid Choice\n");
      }
   }
   return 0;
}

void create() {
   if (first != NULL) {
      printf("Dispatch schedule already exists.\n");
   } else {
      first = NULL;
      printf("Dispatch schedule created successfully.\n");
   }
}

void insertDispatch(int dispatchID, char dispatchDetails[]) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->dispatchID = dispatchID;
   strcpy(temp->dispatchDetails, dispatchDetails);
   temp->next = first;
   first = temp;
   printf("Dispatch entry '%s' with ID %d added successfully.\n", dispatchDetails,
dispatchID);
}

void deleteDispatch(int dispatchID) {
   struct Node* temp = first, * prev = NULL;
```

```
    if (first == NULL) {
        printf("No dispatch entries.\n");
        return;
    }
    if (first->dispatchID == dispatchID) {
        first = first->next;
        free(temp);
        printf("Dispatch entry with ID %d deleted from schedule.\n", dispatchID);
        return;
    }
    while (temp != NULL && temp->dispatchID != dispatchID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Dispatch entry with ID %d not found in schedule.\n", dispatchID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Dispatch entry with ID %d deleted from schedule.\n", dispatchID);
}

void displayDispatches() {
    struct Node* temp = first;
    if (temp == NULL) {
        printf("No dispatch entries.\n");
        return;
    }
    printf("Dispatch ID\tDispatch Details\n");
    printf("------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->dispatchID, temp->dispatchDetails);
        temp = temp->next;
    }
}
```

Problem 1: Team Roster Management
Description: Implement a linked list to manage the roster of players in a sports team.Operations:
Create a team roster.
Insert a new player.
Delete a player who leaves the team.
Display the current team roster.

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

struct Node{
    int playerID;
    char playerName[50];
    struct Node *next;
}*head = NULL;

void createTeam();
void insertPlayer(int playerID,char playerName[]);
void deletePlayer(int playerID);
void displayRoster();

int main(){
    int choice,playerID;
    char playerName[50];
     while (1) {
        printf("\nTeam Roster Management\n");
        printf("1. Create Team Roster\n");
        printf("2. Insert New Player\n");
        printf("3. Delete Player\n");
        printf("4. Display Team Roster\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                createTeam();
                break;
```

```
            case 2:
                printf("Enter Player ID: ");
                scanf("%d", &playerID);
                printf("Enter Player Name: ");
                scanf(" %[^\n]s", playerName);
                insertPlayer(playerID,playerName);
                break;
            case 3:
                printf("Enter Player ID: ");
                scanf("%d", &playerID);
                deletePlayer(playerID);
                break;
            case 4:
                displayRoster();
                break;
            case 5:
                printf("Exiting.....\n");
                exit(0);
            default:
                printf("Invalid choice\n");

        }
    }
    return 0;
}

void createTeam(){
    if(head!=NULL){
        printf("Team roster already exists.\n");
    }else{
        head =NULL;
        printf("Team roster created successfully.\n");
    }
}
void insertPlayer(int playerID,char playerName[]){
    struct Node *temp = (struct Node *) malloc(sizeof(struct Node ));
    temp->playerID =playerID;
    strcpy(temp->playerName,playerName);
    temp->next = head;
    head = temp;
    printf("Player '%s' with ID %d added successfully.\n", playerName, playerID);
```

```
}
void deletePlayer(int playerID){
    struct Node *temp =head;
    struct Node *prev = NULL;

    if(head == NULL){
        printf("No players in the roster.\n");
        return;
    }
    if(head->playerID==playerID){
        head = head->next;
        free(temp);
        printf("Player with ID %d deleted successfully.\n", playerID);
        return;
    }
    while(temp !=NULL && temp->playerID != playerID){
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Player with ID %d not found in the roster.\n", playerID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Player with ID %d deleted successfully.\n", playerID);
}
void displayRoster()
{
    struct Node* temp = head;

    if (temp == NULL) {
        printf("No players in the team roster.\n");
        return;
    }

    printf("Player ID\tPlayer Name\n");
    printf("-----------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->playerID, temp->playerName);
        temp = temp->next;
```

```
    }
}
```

Problem 2: Tournament Match Scheduling
Description: Use a linked list to schedule matches in a tournament.Operations:
Create a match schedule.
Insert a new match.
Delete a completed or canceled match.
Display the current match schedule.

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

struct Node {
    int matchNo;
    char date[50];
    struct Node* next;
} *head = NULL;

void createSchedule();
void insertMatch(int matchNo, char date[]);
void deleteMatch(int matchNo);
void displaySchedule();

int main() {
    int choice, matchNo;
    char date[50];

    while (1) {
        printf("\nTournament Match Scheduling\n");
        printf("1. Create Match Schedule\n");
        printf("2. Insert New Match\n");
        printf("3. Delete Completed or Canceled Match\n");
        printf("4. Display Match Schedule\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
```

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

        switch (choice) {
            case 1:
                createSchedule();
                break;
            case 2:
                printf("Enter Match Number: ");
                scanf("%d", &matchNo);
                printf("Enter Match Date (DD-MM-YYYY): ");
                scanf(" %[^\n]s", date);
                insertMatch(matchNo, date);
                break;
            case 3:
                printf("Enter Match Number to delete: ");
                scanf("%d", &matchNo);
                deleteMatch(matchNo);
                break;
            case 4:
                displaySchedule();
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

void createSchedule() {
    if (head != NULL) {
        printf("Match schedule already exists.\n");
    } else {
        head = NULL;
        printf("Match schedule created successfully.\n");
    }
}
```

```
void insertMatch(int matchNo, char date[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->matchNo = matchNo;
    strcpy(newNode->date, date);
    newNode->next = head;
    head = newNode;
    printf("Match %d scheduled on %s added successfully.\n", matchNo, date);
}

void deleteMatch(int matchNo) {
    struct Node* temp = head;
    struct Node* prev = NULL;

    if (head == NULL) {
        printf("No matches in the schedule.\n");
        return;
    }

    if (head->matchNo == matchNo) {
        head = head->next;
        free(temp);
        printf("Match %d removed from the schedule.\n", matchNo);
        return;
    }

    while (temp != NULL && temp->matchNo != matchNo) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Match %d not found in the schedule.\n", matchNo);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Match %d removed from the schedule.\n", matchNo);
}

void displaySchedule() {
```

```
    struct Node* temp = head;

    if (temp == NULL) {
        printf("No matches in the schedule.\n");
        return;
    }

    printf("\nCurrent Match Schedule:\n");
    printf("Match Number\tMatch Date\n");
    printf("---------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->matchNo, temp->date);
        temp = temp->next;
    }
}
```

Problem 3: Athlete Training Log
Description: Develop a linked list to log training sessions for athletes.Operations:
Create a training log.
Insert a new training session.
Delete a completed or canceled session.
Display the training log.

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

struct TrainingSession {
    int sessionID;
    char date[50];
    char description[100];
    struct TrainingSession* next;
} *headTraining = NULL;

void createTrainingLog();
void insertTrainingSession(int sessionID, char date[], char description[]);
void deleteTrainingSession(int sessionID);
void displayTrainingLog();
```

```
void createTrainingLog() {
    if (headTraining != NULL) {
        printf("Training log already exists.\n");
    } else {
        headTraining = NULL;
        printf("Training log created successfully.\n");
    }
}

void insertTrainingSession(int sessionID, char date[], char description[]) {
    struct TrainingSession* newNode = (struct TrainingSession*)malloc(sizeof(struct
TrainingSession));
    newNode->sessionID = sessionID;
    strcpy(newNode->date, date);
    strcpy(newNode->description, description);
    newNode->next = headTraining;
    headTraining = newNode;
    printf("Training session %d on %s added successfully.\n", sessionID, date);
}

void deleteTrainingSession(int sessionID) {
    struct TrainingSession* temp = headTraining;
    struct TrainingSession* prev = NULL;

    if (headTraining == NULL) {
        printf("No training sessions in the log.\n");
        return;
    }

    if (headTraining->sessionID == sessionID) {
        headTraining = headTraining->next;
        free(temp);
        printf("Training session %d deleted.\n", sessionID);
        return;
    }

    while (temp != NULL && temp->sessionID != sessionID) {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Training session %d not found.\n", sessionID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Training session %d deleted.\n", sessionID);
}

void displayTrainingLog() {
    struct TrainingSession* temp = headTraining;

    if (temp == NULL) {
        printf("No training sessions in the log.\n");
        return;
    }

    printf("\nTraining Log:\n");
    printf("Session ID\tDate\t\tDescription\n");
    printf("----------------------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\t%s\n", temp->sessionID, temp->date, temp->description);
        temp = temp->next;
    }
}
```

Problem 4: Sports Equipment Inventory
Description: Use a linked list to manage the inventory of sports equipment.Operations:
Create an equipment inventory list.
Insert a new equipment item.
Delete an item that is no longer usable.
Display the current equipment inventory.

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

struct Equipment {
    int equipmentID;
    char name[50];
```

```
    int quantity;
    struct Equipment* next;
} *headEquipment = NULL;

void createEquipmentInventory();
void insertEquipment(int equipmentID, char name[], int quantity);
void deleteEquipment(int equipmentID);
void displayEquipmentInventory();

void createEquipmentInventory() {
    if (headEquipment != NULL) {
        printf("Equipment inventory already exists.\n");
    } else {
        headEquipment = NULL;
        printf("Equipment inventory created successfully.\n");
    }
}

void insertEquipment(int equipmentID, char name[], int quantity) {
    struct Equipment* newNode = (struct Equipment*)malloc(sizeof(struct Equipment));
    newNode->equipmentID = equipmentID;
    strcpy(newNode->name, name);
    newNode->quantity = quantity;
    newNode->next = headEquipment;
    headEquipment = newNode;
    printf("Equipment %s (ID: %d) added successfully.\n", name, equipmentID);
}

void deleteEquipment(int equipmentID) {
    struct Equipment* temp = headEquipment;
    struct Equipment* prev = NULL;

    if (headEquipment == NULL) {
        printf("No equipment in the inventory.\n");
        return;
    }

    if (headEquipment->equipmentID == equipmentID) {
        headEquipment = headEquipment->next;
        free(temp);
        printf("Equipment ID %d deleted.\n", equipmentID);
```

```
        return;
    }

    while (temp != NULL && temp->equipmentID != equipmentID) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Equipment ID %d not found.\n", equipmentID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Equipment ID %d deleted.\n", equipmentID);
}

void displayEquipmentInventory() {
    struct Equipment* temp = headEquipment;

    if (temp == NULL) {
        printf("No equipment in the inventory.\n");
        return;
    }

    printf("\nEquipment Inventory:\n");
    printf("Equipment ID\tName\t\tQuantity\n");
    printf("----------------------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\t\t%d\n", temp->equipmentID, temp->name, temp->quantity);
        temp = temp->next;
    }
}
```

Problem 5: Player Performance Tracking
Description: Implement a linked list to track player performance over the season.Operations:
Create a performance record list.

Insert a new performance entry.
Delete an outdated or erroneous entry.
Display all performance records.

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

struct Performance {
    int playerID;
    char name[50];
    int score;
    struct Performance* next;
} *headPerformance = NULL;

void createPerformanceRecord();
void insertPerformance(int playerID, char name[], int score);
void deletePerformance(int playerID);
void displayPerformanceRecords();

void createPerformanceRecord() {
    if (headPerformance != NULL) {
        printf("Performance record already exists.\n");
    } else {
        headPerformance = NULL;
        printf("Performance record created successfully.\n");
    }
}

void insertPerformance(int playerID, char name[], int score) {
    struct Performance* newNode = (struct Performance*)malloc(sizeof(struct
Performance));
    newNode->playerID = playerID;
    strcpy(newNode->name, name);
    newNode->score = score;
    newNode->next = headPerformance;
    headPerformance = newNode;
    printf("Performance record for %s (ID: %d) added successfully.\n", name, playerID);
}

void deletePerformance(int playerID) {
```

```
    struct Performance* temp = headPerformance;
    struct Performance* prev = NULL;

    if (headPerformance == NULL) {
        printf("No performance records available.\n");
        return;
    }

    if (headPerformance->playerID == playerID) {
        headPerformance = headPerformance->next;
        free(temp);
        printf("Performance record for Player ID %d deleted.\n", playerID);
        return;
    }

    while (temp != NULL && temp->playerID != playerID) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Performance record for Player ID %d not found.\n", playerID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Performance record for Player ID %d deleted.\n", playerID);
}

void displayPerformanceRecords() {
    struct Performance* temp = headPerformance;

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

    printf("\nPerformance Records:\n");
    printf("Player ID\tName\t\tScore\n");
    printf("----------------------------------------\n");
```

```
    while (temp != NULL) {
        printf("%d\t\t%s\t\t%d\n", temp->playerID, temp->name, temp->score);
        temp = temp->next;
    }
}
```

Problem 6: Event Registration System
Description: Use a linked list to manage athlete registrations for sports events.Operations:
Create a registration list.
Insert a new registration.
Delete a canceled registration.
Display all current registrations.

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

struct Registration {
    int regID;
    char athleteName[50];
    char eventName[50];
    struct Registration* next;
} *headRegistration = NULL;

void createRegistrationList();
void insertRegistration(int regID, char athleteName[], char eventName[]);
void deleteRegistration(int regID);
void displayRegistrations();

void createRegistrationList() {
    if (headRegistration != NULL) {
        printf("Registration list already exists.\n");
    } else {
        headRegistration = NULL;
        printf("Registration list created successfully.\n");
    }
}
```

```
void insertRegistration(int regID, char athleteName[], char eventName[]) {
    struct Registration* newNode = (struct Registration*)malloc(sizeof(struct
Registration));
    newNode->regID = regID;
    strcpy(newNode->athleteName, athleteName);
    strcpy(newNode->eventName, eventName);
    newNode->next = headRegistration;
    headRegistration = newNode;
    printf("Registration for %s in %s (ID: %d) added successfully.\n", athleteName,
eventName, regID);
}

void deleteRegistration(int regID) {
    struct Registration* temp = headRegistration;
    struct Registration* prev = NULL;

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

    if (headRegistration->regID == regID) {
        headRegistration = headRegistration->next;
        free(temp);
        printf("Registration ID %d canceled successfully.\n", regID);
        return;
    }

    while (temp != NULL && temp->regID != regID) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Registration ID %d not found.\n", regID);
        return;
    }

    prev->next = temp->next;
    free(temp);
```

```
    printf("Registration ID %d canceled successfully.\n", regID);
}

void displayRegistrations() {
    struct Registration* temp = headRegistration;

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

    printf("\nEvent Registrations:\n");
    printf("Reg ID\tAthlete Name\t\tEvent Name\n");
    printf("-----------------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\t\t%s\n", temp->regID, temp->athleteName, temp->eventName);
        temp = temp->next;
    }
}

int main() {
    createRegistrationList();
    insertRegistration(1, "John Doe", "100m Sprint");
    insertRegistration(2, "Jane Smith", "Long Jump");
    insertRegistration(3, "Alice Brown", "Marathon");
    displayRegistrations();
    deleteRegistration(2);
    displayRegistrations();
    return 0;
}
```

Problem 7: Sports League Standings
Description: Develop a linked list to manage the standings of teams in a sports league.Operations:
Create a league standings list.
Insert a new team.
Delete a team that withdraws.
Display the current league standings.

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

struct Team {
    char teamName[50];
    int points;
    struct Team* next;
} *headLeague = NULL;

void createStandingsList();
void insertTeam(char teamName[], int points);
void deleteTeam(char teamName[]);
void displayStandings();

void createStandingsList() {
    headLeague = NULL;
    printf("League standings list created successfully.\n");
}

void insertTeam(char teamName[], int points) {
    struct Team* newTeam = (struct Team*)malloc(sizeof(struct Team));
    strcpy(newTeam->teamName, teamName);
    newTeam->points = points;
    newTeam->next = headLeague;
    headLeague = newTeam;
    printf("Team %s with %d points added to the standings.\n", teamName, points);
}

void deleteTeam(char teamName[]) {
    struct Team* temp = headLeague;
    struct Team* prev = NULL;

    while (temp != NULL && strcmp(temp->teamName, teamName) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Team %s not found in the standings.\n", teamName);
```

```
        return;
    }

    if (prev == NULL)
        headLeague = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Team %s removed from the standings.\n", teamName);
}

void displayStandings() {
    struct Team* temp = headLeague;

    if (temp == NULL) {
        printf("No teams in the standings.\n");
        return;
    }

    printf("\nLeague Standings:\n");
    printf("Team Name\tPoints\n");
    printf("-------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%d\n", temp->teamName, temp->points);
        temp = temp->next;
    }
}

int main() {
    createStandingsList();
    insertTeam("Tigers", 15);
    insertTeam("Eagles", 20);
    insertTeam("Panthers", 12);
    displayStandings();
    deleteTeam("Eagles");
    displayStandings();
    return 0;
}
```

Problem 8: Match Result Recording

Description: Implement a linked list to record results of matches.Operations:
Create a match result list.
Insert a new match result.
Delete an incorrect or outdated result.
Display all recorded match results.

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

struct Match {
    char matchDetails[100];
    struct Match* next;
} *headMatch = NULL;

void createMatchResultList();
void insertMatchResult(char matchDetails[]);
void deleteMatchResult(char matchDetails[]);
void displayMatchResults();

void createMatchResultList() {
    headMatch = NULL;
    printf("Match result list created successfully.\n");
}

void insertMatchResult(char matchDetails[]) {
    struct Match* newMatch = (struct Match*)malloc(sizeof(struct Match));
    strcpy(newMatch->matchDetails, matchDetails);
    newMatch->next = headMatch;
    headMatch = newMatch;
    printf("Match result '%s' added successfully.\n", matchDetails);
}

void deleteMatchResult(char matchDetails[]) {
    struct Match* temp = headMatch;
    struct Match* prev = NULL;

    while (temp != NULL && strcmp(temp->matchDetails, matchDetails) != 0) {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Match result '%s' not found.\n", matchDetails);
        return;
    }

    if (prev == NULL)
        headMatch = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Match result '%s' deleted successfully.\n", matchDetails);
}

void displayMatchResults() {
    struct Match* temp = headMatch;

    if (temp == NULL) {
        printf("No match results recorded.\n");
        return;
    }

    printf("\nMatch Results:\n");
    while (temp != NULL) {
        printf("%s\n", temp->matchDetails);
        temp = temp->next;
    }
}

int main() {
    createMatchResultList();
    insertMatchResult("Team A vs Team B: 2-1");
    insertMatchResult("Team C vs Team D: 0-0");
    insertMatchResult("Team E vs Team F: 3-2");
    displayMatchResults();
    deleteMatchResult("Team C vs Team D: 0-0");
    displayMatchResults();
    return 0;
}
```

Problem 9: Player Injury Tracker
Description: Use a linked list to track injuries of players.Operations:
Create an injury tracker list.
Insert a new injury report.
Delete a resolved or erroneous injury report.
Display all current injury reports.

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

struct Injury {
    char playerName[50];
    char injuryDetails[100];
    struct Injury* next;
} *headInjury = NULL;

void createInjuryTrackerList();
void insertInjuryReport(char playerName[], char injuryDetails[]);
void deleteInjuryReport(char playerName[]);
void displayInjuryReports();

void createInjuryTrackerList() {
    headInjury = NULL;
    printf("Injury tracker list created successfully.\n");
}

void insertInjuryReport(char playerName[], char injuryDetails[]) {
    struct Injury* newInjury = (struct Injury*)malloc(sizeof(struct Injury));
    strcpy(newInjury->playerName, playerName);
    strcpy(newInjury->injuryDetails, injuryDetails);
    newInjury->next = headInjury;
    headInjury = newInjury;
    printf("Injury report for %s added successfully.\n", playerName);
}

void deleteInjuryReport(char playerName[]) {
    struct Injury* temp = headInjury;
    struct Injury* prev = NULL;
```

```
    while (temp != NULL && strcmp(temp->playerName, playerName) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Injury report for %s not found.\n", playerName);
        return;
    }

    if (prev == NULL)
        headInjury = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Injury report for %s removed successfully.\n", playerName);
}

void displayInjuryReports() {
    struct Injury* temp = headInjury;

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

    printf("\nInjury Reports:\n");
    printf("Player Name\tInjury Details\n");
    printf("---------------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%s\n", temp->playerName, temp->injuryDetails);
        temp = temp->next;
    }
}

int main() {
    createInjuryTrackerList();
    insertInjuryReport("John", "Knee Injury");
    insertInjuryReport("Alice", "Sprained Ankle");
```

```
    insertInjuryReport("Bob", "Wrist Fracture");
    displayInjuryReports();
    deleteInjuryReport("Alice");
    displayInjuryReports();
    return 0;
}
```

Problem 10: Sports Facility Booking System
Description: Manage bookings for sports facilities using a linked list.Operations:
Create a booking list.
Insert a new booking.
Delete a canceled or completed booking.
Display all current bookings.

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

struct Booking {
    char facilityName[50];
    char bookingDate[20];
    char timeSlot[20];
    struct Booking* next;
} *headBooking = NULL;

void createBookingList();
void insertBooking(char facilityName[], char bookingDate[], char timeSlot[]);
void deleteBooking(char facilityName[], char bookingDate[], char timeSlot[]);
void displayBookings();

void createBookingList() {
    headBooking = NULL;
    printf("Booking list created successfully.\n");
}

void insertBooking(char facilityName[], char bookingDate[], char timeSlot[]) {
    struct Booking* newBooking = (struct Booking*)malloc(sizeof(struct Booking));
    strcpy(newBooking->facilityName, facilityName);
    strcpy(newBooking->bookingDate, bookingDate);
    strcpy(newBooking->timeSlot, timeSlot);
```

```
    newBooking->next = headBooking;
    headBooking = newBooking;
    printf("Booking for %s on %s at %s added successfully.\n", facilityName,
bookingDate, timeSlot);
}

void deleteBooking(char facilityName[], char bookingDate[], char timeSlot[]) {
    struct Booking* temp = headBooking;
    struct Booking* prev = NULL;

    while (temp != NULL && (strcmp(temp->facilityName, facilityName) != 0 ||
           strcmp(temp->bookingDate, bookingDate) != 0 || strcmp(temp->timeSlot,
timeSlot) != 0)) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Booking not found for %s on %s at %s.\n", facilityName, bookingDate,
timeSlot);
        return;
    }

    if (prev == NULL)
        headBooking = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Booking for %s on %s at %s removed successfully.\n", facilityName,
bookingDate, timeSlot);
}

void displayBookings() {
    struct Booking* temp = headBooking;

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

```
    printf("\nCurrent Bookings:\n");
    printf("Facility Name\tDate\t\tTime Slot\n");
    printf("-----------------------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%s\t%s\n", temp->facilityName, temp->bookingDate,
temp->timeSlot);
        temp = temp->next;
    }
}

int main() {
    createBookingList();
    insertBooking("Tennis Court", "2025-01-20", "10:00 AM - 11:00 AM");
    insertBooking("Swimming Pool", "2025-01-21", "3:00 PM - 4:00 PM");
    displayBookings();
    deleteBooking("Tennis Court", "2025-01-20", "10:00 AM - 11:00 AM");
    displayBookings();
    return 0;
}
```

Problem 11: Coaching Staff Management
Description: Use a linked list to manage the coaching staff of a sports team.Operations:
Create a coaching staff list.
Insert a new coach.
Delete a coach who leaves the team.
Display the current coaching staff.

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

struct Coach {
    char coachName[50];
    char specialization[50];
    struct Coach* next;
} *headCoach = NULL;

void createCoachingStaffList();
void insertCoach(char coachName[], char specialization[]);
void deleteCoach(char coachName[]);
void displayCoachingStaff();
```

```
void createCoachingStaffList() {
    headCoach = NULL;
    printf("Coaching staff list created successfully.\n");
}

void insertCoach(char coachName[], char specialization[]) {
    struct Coach* newCoach = (struct Coach*)malloc(sizeof(struct Coach));
    strcpy(newCoach->coachName, coachName);
    strcpy(newCoach->specialization, specialization);
    newCoach->next = headCoach;
    headCoach = newCoach;
    printf("Coach %s (%s) added to the coaching staff.\n", coachName, specialization);
}

void deleteCoach(char coachName[]) {
    struct Coach* temp = headCoach;
    struct Coach* prev = NULL;

    while (temp != NULL && strcmp(temp->coachName, coachName) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Coach %s not found.\n", coachName);
        return;
    }

    if (prev == NULL)
        headCoach = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Coach %s removed from the coaching staff.\n", coachName);
}

void displayCoachingStaff() {
    struct Coach* temp = headCoach;
```

```
    if (temp == NULL) {
        printf("No coaches in the coaching staff.\n");
        return;
    }

    printf("\nCoaching Staff:\n");
    printf("Coach Name\tSpecialization\n");
    printf("---------------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%s\n", temp->coachName, temp->specialization);
        temp = temp->next;
    }
}

int main() {
    createCoachingStaffList();
    insertCoach("John", "Fitness Trainer");
    insertCoach("Alice", "Strategy Coach");
    displayCoachingStaff();
    deleteCoach("John");
    displayCoachingStaff();
    return 0;
}
```

Problem 12: Fan Club Membership Management
Description: Implement a linked list to manage memberships in a sports team's fan club.Operations:
Create a membership list.
Insert a new member.
Delete a member who cancels their membership.
Display all current members.

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

struct Member {
    char memberName[50];
    int memberId;
    struct Member* next;
```

```
} *headMember = NULL;

void createMembershipList();
void insertMember(char memberName[], int memberId);
void deleteMember(int memberId);
void displayMembers();

void createMembershipList() {
    headMember = NULL;
    printf("Membership list created successfully.\n");
}

void insertMember(char memberName[], int memberId) {
    struct Member* newMember = (struct Member*)malloc(sizeof(struct Member));
    strcpy(newMember->memberName, memberName);
    newMember->memberId = memberId;
    newMember->next = headMember;
    headMember = newMember;
    printf("Member %s (ID: %d) added to the fan club.\n", memberName, memberId);
}

void deleteMember(int memberId) {
    struct Member* temp = headMember;
    struct Member* prev = NULL;

    while (temp != NULL && temp->memberId != memberId) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Member with ID %d not found.\n", memberId);
        return;
    }

    if (prev == NULL)
        headMember = temp->next;
    else
        prev->next = temp->next;

    free(temp);
```

```
    printf("Member with ID %d removed from the fan club.\n", memberId);
}

void displayMembers() {
    struct Member* temp = headMember;

    if (temp == NULL) {
        printf("No members in the fan club.\n");
        return;
    }

    printf("\nFan Club Members:\n");
    printf("Member ID\tMember Name\n");
    printf("----------------------------\n");
    while (temp != NULL) {
        printf("%d\t\t%s\n", temp->memberId, temp->memberName);
        temp = temp->next;
    }
}

int main() {
    createMembershipList();
    insertMember("Emma", 101);
    insertMember("Liam", 102);
    displayMembers();
    deleteMember(101);
    displayMembers();
    return 0;
}
```

Problem 13: Sports Event Scheduling
Description: Use a linked list to manage the schedule of sports events.Operations:
Create an event schedule.
Insert a new event.
Delete a completed or canceled event.
Display the current event schedule.

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

```
struct Event {
    char eventName[50];
    char eventDate[20];
    struct Event* next;
} *headEvent = NULL;

void createEventSchedule();
void insertEvent(char eventName[], char eventDate[]);
void deleteEvent(char eventName[]);
void displayEventSchedule();

void createEventSchedule() {
    headEvent = NULL;
    printf("Event schedule created successfully.\n");
}

void insertEvent(char eventName[], char eventDate[]) {
    struct Event* newEvent = (struct Event*)malloc(sizeof(struct Event));
    strcpy(newEvent->eventName, eventName);
    strcpy(newEvent->eventDate, eventDate);
    newEvent->next = headEvent;
    headEvent = newEvent;
    printf("Event %s on %s added to the schedule.\n", eventName, eventDate);
}

void deleteEvent(char eventName[]) {
    struct Event* temp = headEvent;
    struct Event* prev = NULL;

    while (temp != NULL && strcmp(temp->eventName, eventName) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Event %s not found in the schedule.\n", eventName);
        return;
    }

    if (prev == NULL)
        headEvent = temp->next;
```

```
    else
        prev->next = temp->next;

    free(temp);
    printf("Event %s removed from the schedule.\n", eventName);
}

void displayEventSchedule() {
    struct Event* temp = headEvent;

    if (temp == NULL) {
        printf("No events in the schedule.\n");
        return;
    }

    printf("\nCurrent Event Schedule:\n");
    printf("Event Name\t\tDate\n");
    printf("---------------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%s\n", temp->eventName, temp->eventDate);
        temp = temp->next;
    }
}

int main() {
    createEventSchedule();
    insertEvent("Football Finals", "2025-02-10");
    insertEvent("Basketball Semifinals", "2025-01-30");
    displayEventSchedule();
    deleteEvent("Football Finals");
    displayEventSchedule();
    return 0;
}
```

Problem 14: Player Transfer Records
Description: Maintain a linked list to track player transfers between teams.Operations:
Create a transfer record list.
Insert a new transfer record.
Delete an outdated or erroneous transfer record.
Display all current transfer records.

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

struct Transfer {
    char playerName[50];
    char fromTeam[50];
    char toTeam[50];
    struct Transfer* next;
} *headTransfer = NULL;

void createTransferRecordList();
void insertTransferRecord(char playerName[], char fromTeam[], char toTeam[]);
void deleteTransferRecord(char playerName[]);
void displayTransferRecords();

void createTransferRecordList() {
    headTransfer = NULL;
    printf("Transfer record list created successfully.\n");
}

void insertTransferRecord(char playerName[], char fromTeam[], char toTeam[]) {
    struct Transfer* newTransfer = (struct Transfer*)malloc(sizeof(struct Transfer));
    strcpy(newTransfer->playerName, playerName);
    strcpy(newTransfer->fromTeam, fromTeam);
    strcpy(newTransfer->toTeam, toTeam);
    newTransfer->next = headTransfer;
    headTransfer = newTransfer;
    printf("Transfer record for %s from %s to %s added successfully.\n", playerName,
fromTeam, toTeam);
}

void deleteTransferRecord(char playerName[]) {
    struct Transfer* temp = headTransfer;
    struct Transfer* prev = NULL;

    while (temp != NULL && strcmp(temp->playerName, playerName) != 0) {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Transfer record for %s not found.\n", playerName);
        return;
    }

    if (prev == NULL)
        headTransfer = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Transfer record for %s removed successfully.\n", playerName);
}

void displayTransferRecords() {
    struct Transfer* temp = headTransfer;

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

    printf("\nCurrent Transfer Records:\n");
    printf("Player Name\tFrom Team\tTo Team\n");
    printf("-----------------------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%s\t\t%s\n", temp->playerName, temp->fromTeam, temp->toTeam);
        temp = temp->next;
    }
}

int main() {
    createTransferRecordList();
    insertTransferRecord("John Doe", "Team A", "Team B");
    insertTransferRecord("Alice Smith", "Team C", "Team D");
    displayTransferRecords();
    deleteTransferRecord("John Doe");
    displayTransferRecords();
    return 0;
}
```

Problem 15: Championship Points Tracker
Description: Implement a linked list to track championship points for teams.Operations:
Create a points tracker list.
Insert a new points entry.
Delete an incorrect or outdated points entry.
Display all current points standings.

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

struct Points {
    char teamName[50];
    int points;
    struct Points* next;
} *headPoints = NULL;

void createPointsTracker();
void insertPointsEntry(char teamName[], int points);
void deletePointsEntry(char teamName[]);
void displayPointsStandings();

void createPointsTracker() {
    headPoints = NULL;
    printf("Points tracker created successfully.\n");
}

void insertPointsEntry(char teamName[], int points) {
    struct Points* newPoints = (struct Points*)malloc(sizeof(struct Points));
    strcpy(newPoints->teamName, teamName);
    newPoints->points = points;
    newPoints->next = headPoints;
    headPoints = newPoints;
    printf("Points entry for %s with %d points added successfully.\n", teamName, points);
}

void deletePointsEntry(char teamName[]) {
    struct Points* temp = headPoints;
    struct Points* prev = NULL;
```

```
    while (temp != NULL && strcmp(temp->teamName, teamName) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Points entry for %s not found.\n", teamName);
        return;
    }

    if (prev == NULL)
        headPoints = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Points entry for %s removed successfully.\n", teamName);
}

void displayPointsStandings() {
    struct Points* temp = headPoints;

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

    printf("\nCurrent Points Standings:\n");
    printf("Team Name\tPoints\n");
    printf("-------------------------\n");
    while (temp != NULL) {
        printf("%s\t\t%d\n", temp->teamName, temp->points);
        temp = temp->next;
    }
}

int main() {
    createPointsTracker();
    insertPointsEntry("Team A", 30);
    insertPointsEntry("Team B", 25);
    displayPointsStandings();
```

```
    deletePointsEntry("Team B");
    displayPointsStandings();
    return 0;
}
```