Problem 1: Inventory Management System

Description: Implement a linked list to manage the inventory of raw materials.

Operations:

1. Create an inventory list.
2. Insert a new raw material.
3. Delete a raw material from the inventory.
4. Display the current inventory.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct Inventory{

   int id;

   char name[50];

   int quantity;

   struct Inventory *next;

};



void createInventory();

void insertRawmaterial(int id, char name[], int quantity);

void deleteRawMaterial(int id);

void displayInventory(void);



struct Inventory *head=NULL;



int main(){

   int choice,id,quantity;
```

```
char name[50];


while(1){

    printf("\nInventory Management System:\n ");

    printf("1.Create Inventory List\n2.Insert Raw Material\n");

    printf("3.Delete raw Material\n4.Display Inventory\n5.Exit\n");

    printf("Enter the Choice: ");

    scanf("%d",&choice);

    switch (choice){

        case 1:

        createInventory();

        break;

        case 2:

        printf("Enter the Material ID: ");

        scanf("%d",&id);

        printf("Enter material name: ");

        scanf(" %[^\n]s",name);

        printf("Enter Quantity: ");

        scanf("%d",&quantity);

        insertRawmaterial(id,name,quantity);

        break;


        case 3:

        printf("Enter the material ID to Delete: ");

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

```
            deleteRawMaterial(id);
            break;

            case 4:
            displayInventory();
            break;

            case 5:
            printf("Exiting!");
            exit(0);

            default:
            printf("Invalid Choice!\n");

        }
    }
    return 0;
}
void createInventory(){
    if(head!=NULL){
        printf("Inventory Already Exists.\n");

    }
    else{
        head=NULL;
        printf("Inventory list created");
    }
}
void insertRawmaterial(int id, char name[], int quantity){
    struct Inventory *newnode=(struct Inventory *)malloc(sizeof(struct Inventory));
    newnode->id=id;
    strcpy(newnode->name,name);
    newnode->quantity=quantity;
    newnode->next=NULL;

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

    printf("Rawmaterial %s is added to Inventory.\n",name);
}

void deleteRawMaterial(int id){
```

```
struct Inventory *temp=head, *prev=NULL;


if(temp==NULL){
    printf("Inventory is Empty!\n");
    return;
}


 // Check if head node needs to be deleted
if(temp->id==id){
    head=temp->next;
    free(temp);
    printf("Raw material with ID %d is deleted! \n",id);
    return;
}
else{
    while(temp!=NULL && temp->id!=id){
        prev=temp;
        temp=temp->next;
    }
    if(temp==NULL){
        printf("Raw material with %d doesnot exist.\n",id);
    }
    else{
        prev->next=temp->next;
        free(temp);
```

```
            printf("Raw material with ID %d id deleted!\n",id);
        }
    }

}
void displayInventory(){
    struct Inventory *temp=head;

    if(temp==NULL){
        printf("Inventory Empty.\n");
        return;
    }
    printf("\nCurrent Inventory\n");
    while(temp!=NULL){
        printf("%d\t%s\t%d\n",temp->id,temp->name,temp->quantity);
        temp=temp->next;
    }
}
```

Problem 2: Production Line Queue

Description: Use a linked list to manage the queue of tasks on a production line.

Operations:

1. Create a production task queue.
2. Insert a new task into the queue.
3. Delete a completed task.
4. Display the current task queue.

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

```
#include<string.h>

struct Task{
    int id;
    char description[100];
    struct Task *next;
};

void createQueue();
void inserttask(int id, char description[]);
void deleteTask(int id);
void displayQueue(void);

struct Task *head=NULL;

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

    while(1){
        printf("\nProduction Line Queue:\n ");
        printf("1.Create Task Queue\n2.Insert New Task\n");
        printf("3.Delete Completed Task\n4. Display Task Queue\n5.Exit\n");
        printf("Enter the Choice: ");
        scanf("%d",&choice);
```

```
switch (choice){
    case 1:
    createQueue();
    break;
    case 2:
    printf("Enter Task ID: ");
    scanf("%d",&id);
    printf("Enter Description: ");
    scanf(" %[^\n]s",description);
    inserttask(id,description);
    break;

    case 3:
    printf("Enter the Task ID to Delete: ");
    scanf("%d",&id);
    deleteTask(id);
    break;

    case 4:
    displayQueue();
    break;

    case 5:
    printf("Exiting!");
    exit(0);
```

```
            default:

            printf("Invalid Choice!\n");


        }

    }

    return 0;

}


void createQueue(){

    if(head!=NULL){

        printf("Queue already created!\n");

    }else{

        head=NULL;

        printf("Queue Created!");

    }

}

void inserttask(int id, char description[]){

    struct Task *newTask=(struct Task*)malloc(sizeof(struct Task));

    newTask->id=id;

    strcpy(newTask->description,description);

    newTask->next=NULL;


    if(head==NULL){

        head=newTask;
    }
    else{
        struct Task *temp=head;
        while(temp->next!=NULL){
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("New Task with ID %d added\n",id);
}
void deleteTask(int id){
    struct Task *temp=head,*prev=NULL;

    if(head==NULL){
        printf("No task Exist");
        return;
    }
    if(temp->id==id){
        head=temp->next;
        free(temp);
        printf("Task ID %d has deleted!\n",id);
    }
    else{
        while(temp!=NULL && temp->id != id){
            prev=temp;
            temp=temp->next;

        }
        if(temp==NULL){
            printf("Task with ID %d doesnot Exist!\n",id);
        }
        else{
            prev->next=temp->next;
            free(temp);
            printf("Task with ID %d deleted Successfully!\n",id);
        }
    }

}

void displayQueue(void){
    struct Task *temp=head;
    if(temp==NULL){
        printf("No task Exist!\n");
    }
    printf("Display Entire Task Details");
    while(temp!=NULL){
        printf("%d\t%s\n",temp->id,temp->description);
        temp=temp->next;
    }
```

```
}
```

Problem 3: Machine Maintenance Schedule

Description: Develop a linked list to manage the maintenance schedule of machines.

Operations:

1. Create a maintenance schedule.
2. Insert a new maintenance task.
3. Delete a completed maintenance task.
4. Display the maintenance schedule.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct Maintenance {

    int id;

    char description[100];

    struct Maintenance *next;

};



void createSchedule();

void insertTask(int id, char description[]);

void deleteTask(int id);

void displaySchedule(void);



struct Maintenance *head = NULL;



int main() {

    int choice, id;
```

```
char description[100];


while (1) {
    printf("\nMachine Maintenance Schedule:\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 Task ID: ");
            scanf("%d", &id);
            printf("Enter Description: ");
            scanf(" %[^\n]s", description);
            insertTask(id, description);
            break;
        case 3:
            printf("Enter the Task ID to Delete: ");
}

        scanf("%d", &id);

        deleteTask(id);

        break;

      case 4:

        displaySchedule();

        break;

      case 5:

        printf("Exiting!\n");

        exit(0);

      default:

        printf("Invalid choice!\n");

    }

  }

  return 0;

}


void createSchedule() {

  if (head != NULL) {

    printf("Schedule already created!\n");

  } else {

    head = NULL;

    printf("Schedule created successfully!\n");

  }

}
```

```
void insertTask(int id, char description[]) {
    struct Maintenance *newTask = (struct Maintenance *)malloc(sizeof(struct Maintenance));
    newTask->id = id;
    strcpy(newTask->description, description);
    newTask->next = NULL;

    if (head == NULL) {
        head = newTask;
    } else {
        struct Maintenance *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("New maintenance task with ID %d added successfully.\n", id);
}

void deleteTask(int id) {
    struct Maintenance *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No tasks exist in the schedule!\n");
        return;
    }
    if (temp->id == id) {
        head = temp->next;
        free(temp);
        printf("Task with ID %d deleted successfully!\n", id);
    } else {
        while (temp != NULL && temp->id != id) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Task with ID %d does not exist in the schedule!\n", id);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Task with ID %d deleted successfully!\n", id);
        }
    }
}

void displaySchedule(void) {
    struct Maintenance *temp = head;
    if (temp == NULL) {
        printf("No tasks exist in the maintenance schedule!\n");
        return;
    }

    printf("Displaying all maintenance tasks:\n");
    while (temp != NULL) {
        printf("ID: %d, Description: %s\n", temp->id, temp->description);
        temp = temp->next;
    }
}
```

Problem 4: Employee Shift Management

Description: Use a linked list to manage employee shifts in a manufacturing plant.

Operations:

1. Create a shift schedule.
2. Insert a new shift.
3. Delete a completed or canceled shift.
4. Display the current shift schedule.

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

struct Shift {
    int id;
    char description[100];
    struct Shift *next;
};

void createShiftSchedule();
void insertShift(int id, char description[]);
void deleteShift(int id);
void displayShiftSchedule(void);

struct Shift *head = NULL;

int main() {
    int choice, id;
    char description[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 Shift Schedule\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createShiftSchedule();
                break;
            case 2:
                printf("Enter Shift ID: ");
                scanf("%d", &id);
                printf("Enter Shift Description: ");
                scanf(" %[^\n]s", description);
                insertShift(id, description);
                break;
            case 3:
                printf("Enter the Shift ID to Delete: ");
                scanf("%d", &id);
                deleteShift(id);
                break;
            case 4:
                displayShiftSchedule();
                break;
            case 5:
                printf("Exiting!\n");
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}

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

void insertShift(int id, char description[]) {
    struct Shift *newShift = (struct Shift *)malloc(sizeof(struct Shift));
    newShift->id = id;
    strcpy(newShift->description, description);
    newShift->next = NULL;

    if (head == NULL) {
        head = newShift;
    } else {
        struct Shift *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newShift;
    }
    printf("New shift with ID %d added successfully.\n", id);
}
```

```
void deleteShift(int id) {
    struct Shift *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No shifts exist in the schedule!\n");
        return;
    }

    if (temp->id == id) {
        head = temp->next;
        free(temp);
        printf("Shift with ID %d deleted successfully!\n", id);
    } else {
        while (temp != NULL && temp->id != id) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Shift with ID %d does not exist in the schedule!\n", id);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Shift with ID %d deleted successfully!\n", id);
        }
    }
}


void displayShiftSchedule(void) {

    struct Shift *temp = head;

    if (temp == NULL) {

        printf("No shifts exist in the schedule!\n");

        return;

    }


    printf("Displaying all shifts in the schedule:\n");

    while (temp != NULL) {

        printf("ID: %d, Description: %s\n", temp->id, temp->description);

        temp = temp->next;

    }

}
```

Problem 5: Order Processing System

Description: Implement a linked list to track customer orders.

Operations:

1. Create an order list.
2. Insert a new customer order.
3. Delete a completed or canceled order.
4. Display all current orders.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



struct Order {
    int orderId;

    char customerName[100];

    struct Order *next;

};


void createOrderList();

void insertOrder(int orderId, char customerName[]);

void deleteOrder(int orderId);

void displayOrders();


struct Order *head = NULL;


int main() {

    int choice, orderId;

    char customerName[100];


    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 All 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: ");
        scanf(" %[^\n]s", customerName);
        insertOrder(orderId, customerName);
        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[]) {
    struct Order *newOrder = (struct Order *)malloc(sizeof(struct Order));
    newOrder->orderId = orderId;
    strcpy(newOrder->customerName, customerName);
    newOrder->next = NULL;

    if (head == NULL) {
        head = newOrder;
    } else {
        struct Order *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newOrder;
    }
    printf("New order with ID %d for customer '%s' added successfully.\n", orderId, customerName);
}

void deleteOrder(int orderId) {
    struct Order *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No orders exist in the list!\n");
        return;
    }

    if (temp->orderId == orderId) {
        head = temp->next;
        free(temp);
        printf("Order with ID %d deleted successfully.\n", orderId);
    } else {
        while (temp != NULL && temp->orderId != orderId) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Order with ID %d does not exist!\n", orderId);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Order with ID %d deleted successfully.\n", orderId);
        }
    }
}


void displayOrders() {
    struct Order *temp = head;

    if (temp == NULL) {
        printf("No orders exist in the list!\n");
        return;
    }

    printf("Current Orders:\n");
    while (temp != NULL) {
        printf("Order ID: %d, Customer Name: %s\n", temp->orderId, temp->customerName);
        temp = temp->next;
    }
}
```

Problem 6: Tool Tracking System

Description: Maintain a linked list to track tools used in the manufacturing process.

Operations:

1. Create a tool tracking list.
2. Insert a new tool entry.
3. Delete a tool that is no longer in use.
4. Display all tools currently tracked.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



struct Tool {

    int toolId;

    char toolName[100];

    struct Tool *next;

};



void createToolList();

void insertTool(int toolId, char toolName[]);

void deleteTool(int toolId);

void displayTools();



struct Tool *head = NULL;



int main() {

    int choice, toolId;

    char toolName[100];



    while (1) {
```

```
printf("\nTool Tracking System:\n");
printf("1. Create Tool Tracking List\n");
printf("2. Insert New Tool Entry\n");
printf("3. Delete Tool No Longer in Use\n");
printf("4. Display All Tools\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        createToolList();
        break;
    case 2:
        printf("Enter Tool ID: ");
        scanf("%d", &toolId);
        printf("Enter Tool Name: ");
        scanf(" %[^\n]s", toolName);
        insertTool(toolId, toolName);
        break;
    case 3:
        printf("Enter Tool ID to Delete: ");
        scanf("%d", &toolId);
        deleteTool(toolId);
        break;
```

```
            case 4:
                displayTools();
                break;
            case 5:
                printf("Exiting!\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}


void createToolList() {
    if (head != NULL) {
        printf("Tool tracking list already exists!\n");
    } else {
        head = NULL;
        printf("Tool tracking list created successfully!\n");
    }
}


void insertTool(int toolId, char toolName[]) {
    struct Tool *newTool = (struct Tool *)malloc(sizeof(struct Tool));
    newTool->toolId = toolId;
    strcpy(newTool->toolName, toolName);
    newTool->next = NULL;

    if (head == NULL) {
        head = newTool;
    } else {
        struct Tool *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTool;
    }
    printf("New tool with ID %d ('%s') added successfully.\n", toolId, toolName);
}

void deleteTool(int toolId) {
    struct Tool *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No tools are currently being tracked!\n");
        return;
    }

    if (temp->toolId == toolId) {
        head = temp->next;
        free(temp);
        printf("Tool with ID %d has been deleted successfully.\n", toolId);
    } else {
        while (temp != NULL && temp->toolId != toolId) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Tool with ID %d does not exist in the list!\n", toolId);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Tool with ID %d has been deleted successfully.\n", toolId);
        }
    }
}

void displayTools() {
    struct Tool *temp = head;

    if (temp == NULL) {
        printf("No tools are currently being tracked!\n");
        return;
    }
    printf("Current Tools in the Tracking List:\n");
    while (temp != NULL) {
        printf("Tool ID: %d, Tool Name: %s\n", temp->toolId, temp->toolName);
        temp = temp->next;
    }
}
```

Problem 7: Product Assembly Line

Description: Use a linked list to manage the assembly stages of a product.

Operations:

1. Create an assembly line stage list.
2. Insert a new stage.
3. Delete a completed stage.
4. Display the current assembly stages.

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

struct AssemblyStage {
    int stageId;
    char stageName[100];
    struct AssemblyStage *next;
};

void createAssemblyLine();
void insertStage(int stageId, char stageName[]);
void deleteStage(int stageId);
void displayStages();
```

```
struct AssemblyStage *head = NULL;


int main() {
    int choice, stageId;
    char stageName[100];


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


        switch (choice) {
            case 1:
                createAssemblyLine();
                break;
            case 2:
                printf("Enter Stage ID: ");
                scanf("%d", &stageId);
                printf("Enter Stage Name: ");
                scanf(" %[^\n]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:

                printf("Exiting!\n");

                exit(0);

            default:

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

        }

    }

    return 0;

}



void createAssemblyLine() {

    if (head != NULL) {

        printf("Assembly line already exists!\n");

    } else {
        head = NULL;
        printf("Assembly line stages list created successfully!\n");
    }
}

void insertStage(int stageId, char stageName[]) {
    struct AssemblyStage *newStage = (struct AssemblyStage *)malloc(sizeof(struct AssemblyStage));
    newStage->stageId = stageId;
    strcpy(newStage->stageName, stageName);
    newStage->next = NULL;

    if (head == NULL) {
        head = newStage;
    } else {
        struct AssemblyStage *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newStage;
    }
    printf("New stage with ID %d ('%s') added successfully.\n", stageId, stageName);
}

void deleteStage(int stageId) {
    struct AssemblyStage *temp = head, *prev = NULL;
    if (head == NULL) {
        printf("No stages in the assembly line!\n");
        return;
    }

    if (temp->stageId == stageId) {
        head = temp->next;
        free(temp);
        printf("Stage with ID %d has been deleted successfully.\n", stageId);
    } else {
        while (temp != NULL && temp->stageId != stageId) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Stage with ID %d does not exist!\n", stageId);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Stage with ID %d has been deleted successfully.\n", stageId);
        }
    }
}
```

```
void displayStages() {
    struct AssemblyStage *temp = head;

    if (temp == NULL) {
        printf("No stages in the assembly line!\n");
        return;
    }

    printf("Current Assembly Stages in the Line:\n");
    while (temp != NULL) {
        printf("Stage ID: %d, Stage Name: %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:

1. Create a quality control checklist.
2. Insert a new checklist item.
3. Delete a completed or outdated checklist item.
4. Display the current quality control checklist.
5.

Problem 9: Supplier Management System

Description: Use a linked list to manage a list of suppliers.

Operations:

1. Create a supplier list.
2. Insert a new supplier.
3. Delete an inactive or outdated supplier.
4. Display all current suppliers.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct ChecklistItem {

    int itemId;

    char description[100];

    struct ChecklistItem *next;

};


void createChecklist();

void insertChecklistItem(int itemId, char description[]);

void deleteChecklistItem(int itemId);

void displayChecklist();


struct ChecklistItem *head = NULL;


int main() {

    int choice, itemId;

    char description[100];


    while (1) {

        printf("\nQuality Control Checklist:\n");

        printf("1. Create Quality Control Checklist\n");

        printf("2. Insert New Checklist Item\n");
printf("3. Delete Completed or Outdated Checklist Item\n");
printf("4. Display Current Quality Control Checklist\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        createChecklist();
        break;
    case 2:
        printf("Enter Item ID: ");
        scanf("%d", &itemId);
        printf("Enter Item Description: ");
        scanf(" %[^\n]s", description);
        insertChecklistItem(itemId, description);
        break;
    case 3:
        printf("Enter Item ID to Delete: ");
        scanf("%d", &itemId);
        deleteChecklistItem(itemId);
        break;
    case 4:
        displayChecklist();
        break;
```

```
            case 5:
                printf("Exiting!\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}

void createChecklist() {
    if (head != NULL) {
        printf("Checklist already created!\n");
    } else {
        head = NULL;
        printf("Quality Control Checklist created successfully!\n");
    }
}

void insertChecklistItem(int itemId, char description[]) {
    struct ChecklistItem *newItem = (struct ChecklistItem *)malloc(sizeof(struct ChecklistItem));
    newItem->itemId = itemId;
    strcpy(newItem->description, description);
    newItem->next = NULL;
    if (head == NULL) {
        head = newItem;
    } else {
        struct ChecklistItem *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("New checklist item with ID %d ('%s') added successfully.\n", itemId, description);
}

void deleteChecklistItem(int itemId) {
    struct ChecklistItem *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No items in the checklist!\n");
        return;
    }

    if (temp->itemId == itemId) {
        head = temp->next;
        free(temp);
        printf("Checklist item with ID %d has been deleted successfully.\n", itemId);
    } else {
        while (temp != NULL && temp->itemId != itemId) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Checklist item with ID %d does not exist!\n", itemId);
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Checklist item with ID %d has been deleted successfully.\n", itemId);
        }
    }
}

void displayChecklist() {
    struct ChecklistItem *temp = head;

    if (temp == NULL) {
        printf("No items in the checklist!\n");
        return;
    }

    printf("Current Quality Control Checklist:\n");
    while (temp != NULL) {
        printf("Item ID: %d, Description: %s\n", temp->itemId, temp->description);
```

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

Problem 10: Manufacturing Project Timeline

Description: Develop a linked list to manage the timeline of a manufacturing project.

Operations:

1. Create a project timeline.
2. Insert a new project milestone.
3. Delete a completed milestone.
4. Display the current project timeline.

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

struct Project {
        int projectid;
        char description[100];
        struct Project *next;
};
struct Project *head=NULL;

void createTimeline();
void insertNew(int projectid, char description[]);
void deleteMilestone(int id);
void displayTimeline(void);

int main() {
```

```
int choice,projectid;

char description[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 Current Project Timeline\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);


        switch (choice) {

        case 1:

                createTimeline();

                break;


        case 2:

                printf("Enter new Project Milestone ID: ");

                scanf("%d",&projectid);

                printf("Enter milestone Description: ");

                scanf(" %[^\n]s",description);

                insertNew(projectid,description);

                break;
                case 3:

                        printf("Enter Milestone ID to Delete: ");

                        scanf("%d",&projectid);

                        deleteMilestone(projectid);

                        break;


                case 4:

                        displayTimeline();

                        break;


                case 5:

                        printf("Exiting..\n");

                        break;


                default:

                        printf("Invalid Choice!\n");


                }


        }

        return 0;

}


        void createTimeline() {

                if(head!=NULL) {
```

```
        printf("\nTimeline Already sreated.\n");
    }
    else {
        head=NULL;
        printf("Timeline Created successfully!\n");
    }
}

void insertNew(int projectid, char description[]) {
    struct Project *new=(struct Project *)malloc(sizeof(struct Project));
    if(new ==NULL) {
        printf("Memory Allocation failed!");
        return;
    }
    new->projectid=projectid;
    strcpy(new->description,description);
    new->next=NULL;

    if(head==NULL) {
        head=new;
    }
    else {
        struct Project *temp=head;
        while(temp->next!=NULL) {
            temp=temp->next;
```

```
		}
		temp->next=new;
	}
	printf("New Milestone Added successfully!\n");
}

void deleteMilestone(int id) {
	struct Project *temp=head, *prev=NULL;
	if(head==NULL) {
		printf("No Milestones!\n");
		return;
	}
	if(temp->projectid==id) {
		head=temp->next;
		free(temp);
		printf("Project %d deleted successfully!\n",id);
	}
	else {
		while(temp!=NULL && temp->projectid !=id) {
			prev=temp;
			temp=temp->next;
		}
		if(temp==NULL) {
			printf("Mile with %d ID doesnt exist!\n",id);
		}
```

```
        else {
            prev->next=temp->next;
            free(temp);
            printf("Milestone with ID %d has been deleted.\n",temp->projectid);
        }
    }
}


void displayTimeline()
{
    struct Project *temp=head;
    if(temp==NULL) {
        printf("No milestone exists.\n");
        return;
    }
    while(temp!=NULL) {
        printf("project ID:\t%d\nDescription:\t%s \n",temp->projectid,temp->description);
        temp=temp->next;
    }
}
```

Problem 11: Warehouse Storage Management

Description: Implement a linked list to manage the storage of goods in a warehouse.

Operations:

1. Create a storage list.
2. Insert a new storage entry.

3. Delete a storage entry when goods are shipped.
4. Display the current warehouse storage.

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

struct Storage {
    int productId;
    char productName[100];
    int quantity;
    struct Storage* next;
};

void createStorageList();
void insertStorageEntry(int productId, char productName[], int quantity);
void deleteStorageEntry(int productId);
void displayStorage();

struct Storage *head = NULL;

int main() {
    int choice, productId, quantity;
    char productName[100];

    while(1) {
        printf("\nWarehouse Storage Management:\n");
        printf("1. Create Storage List\n2. Insert New Storage Entry\n");
        printf("3. Delete Storage Entry (Ship Goods)\n4. Display Warehouse Storage\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                createStorageList();
                break;

            case 2:
                printf("Enter Product ID: ");
                scanf("%d", &productId);
                printf("Enter Product Name: ");
                scanf(" %[^\n]s", productName);
                printf("Enter Quantity: ");
                scanf("%d", &quantity);
                insertStorageEntry(productId, productName, quantity);
                break;

            case 3:
                printf("Enter Product ID to Delete (Ship Goods): ");
                scanf("%d", &productId);
                deleteStorageEntry(productId);
                break;
            case 4:
                displayStorage();
                break;

            case 5:
                printf("Exiting!\n");
                exit(0);

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

    return 0;
}


void createStorageList() {
    if (head != NULL) {
        printf("Storage list already created.\n");
    } else {
        head = NULL;
        printf("Storage list created successfully.\n");
    }
```

```
}



void insertStorageEntry(int productId, char productName[], int quantity) {
    struct Storage* newEntry = (struct Storage*) malloc(sizeof(struct Storage));
    newEntry->productId = productId;
    strcpy(newEntry->productName, productName);
    newEntry->quantity = quantity;
    newEntry->next = NULL;

    if (head == NULL) {
        head = newEntry;
    } else {
        struct Storage* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEntry;
    }
    printf("New storage entry added: %d - %s (Quantity: %d)\n", productId,
productName, quantity);
}


void deleteStorageEntry(int productId) {
    struct Storage* temp = head;
    struct Storage* prev = NULL;
    if (head == NULL) {
        printf("No storage entries available.\n");
        return;
    }

    if (temp != NULL && temp->productId == productId) {
        head = temp->next;
        free(temp);
        printf("Storage entry with Product ID %d deleted (Goods shipped).\n", productId);
        return;
    }

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

    if (temp == NULL) {
        printf("Product ID %d not found in storage.\n", productId);
        return;
    }


    prev->next = temp->next;
    free(temp);
    printf("Storage entry with Product ID %d deleted (Goods shipped).\n", productId);
}



void displayStorage() {
    struct Storage* temp = head;

    if (temp == NULL) {
        printf("No items in warehouse storage.\n");
        return;
    }

    printf("Warehouse Storage:\n");
    while (temp != NULL) {
        printf("Product ID: %d\n", temp->productId);
        printf("Product Name: %s\n", temp->productName);
        printf("Quantity: %d\n", temp->quantity);
        printf("------------------------------\n");
        temp = temp->next;
    }
}
```

Problem 12: Machine Parts Inventory

Description: Use a linked list to track machine parts inventory.

Operations:

1. Create a parts inventory list.
2. Insert a new part.
3. Delete a part that is used up or obsolete.
4. Display the current parts inventory.

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

struct Part {
    int partid;
    char name[100];
    int quantity;
    struct Part *next;
};

struct Part *head = NULL;

void createInventory();
void insertNewPart(int partid, char name[], int quantity);
void deletePart(int partid);
void displayInventory(void);

int main() {
    int choice, partid, quantity;
    char name[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 (Used/Obsolete)\n");
printf("4. Display Current Parts Inventory\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        createInventory();
        break;

    case 2:
        printf("Enter Part ID: ");
        scanf("%d", &partid);
        printf("Enter Part Name: ");
        scanf(" %[^\n]s", name);
        printf("Enter Quantity: ");
        scanf("%d", &quantity);
        insertNewPart(partid, name, quantity);
        break;

    case 3:
```

```
                printf("Enter Part ID to Delete: ");
                scanf("%d", &partid);
                deletePart(partid);
                break;

            case 4:
                displayInventory();
                break;

            case 5:
                printf("Exiting..\n");
                return 0;

            default:
                printf("Invalid Choice!\n");
        }
    }

    return 0;
}

void createInventory() {
    if (head != NULL) {
        printf("\nParts Inventory already exists.\n");
    } else {
        head = NULL;
        printf("Parts Inventory created successfully!\n");
    }
}

void insertNewPart(int partid, char name[], int quantity) {
    struct Part *newPart = (struct Part *)malloc(sizeof(struct Part));
    if (newPart == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    newPart->partid = partid;
    strcpy(newPart->name, name);
    newPart->quantity = quantity;
    newPart->next = NULL;

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

    printf("New Part added successfully!\n");
}

void deletePart(int partid) {
    struct Part *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No parts in the inventory!\n");
        return;
    }

    // If the part to be deleted is the first node
    if (temp->partid == partid) {
        head = temp->next;
        free(temp);
        printf("Part with ID %d deleted successfully!\n", partid);
        return;
    }

    // Search for the part to delete
    while (temp != NULL && temp->partid != partid) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Part with ID %d doesn't exist!\n", partid);
        return;
    }

    // Unlink the node and delete it
    prev->next = temp->next;
    free(temp);
    printf("Part with ID %d deleted successfully!\n", partid);
}

void displayInventory() {
    struct Part *temp = head;
    if (temp == NULL) {
        printf("No parts in the inventory.\n");
        return;
    }

    while (temp != NULL) {
        printf("Part ID: %d\nPart Name: %s\nQuantity: %d\n", temp->partid, temp->name, temp->quantity);
        temp = temp->next;
    }
}
```

Problem 13: Packaging Line Schedule

Description: Manage the schedule of packaging tasks using a linked list.

Operations:

1. Create a packaging task schedule.
2. Insert a new packaging task.
3. Delete a completed packaging task.
4. Display the current packaging schedule.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct Task {

    int taskid;

    char description[100];

    struct Task *next;

};



struct Task *head = NULL;



void createSchedule();

void insertNewTask(int taskid, char description[]);

void deleteTask(int taskid);

void displaySchedule(void);



int main() {

    int choice, taskid;

    char description[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 Schedule\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            createSchedule();
            break;

        case 2:
            printf("Enter Task ID: ");
            scanf("%d", &taskid);
            printf("Enter Task Description: ");
            scanf(" %[^\n]s", description);
            insertNewTask(taskid, description);
            break;

        case 3:
                printf("Enter Task ID to Delete: ");
                scanf("%d", &taskid);
                deleteTask(taskid);
                break;

            case 4:
                displaySchedule();
                break;

            case 5:
                printf("Exiting..\n");
                return 0;

            default:
                printf("Invalid Choice!\n");
        }
    }

    return 0;
}

void createSchedule() {
    if (head != NULL) {
        printf("\nPackaging Task Schedule already exists.\n");
    } else {
        head = NULL;
        printf("Packaging Task Schedule created successfully!\n");
    }
}

void insertNewTask(int taskid, char description[]) {
    struct Task *newTask = (struct Task *)malloc(sizeof(struct Task));
    if (newTask == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    newTask->taskid = taskid;
    strcpy(newTask->description, description);
    newTask->next = NULL;

    if (head == NULL) {
        head = newTask;
    } else {
        struct Task *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("New Packaging Task added successfully!\n");
}

void deleteTask(int taskid) {
    struct Task *temp = head, *prev = NULL;

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

    // If the task to be deleted is the first node
    if (temp->taskid == taskid) {
        head = temp->next;
        free(temp);
        printf("Task with ID %d deleted successfully!\n", taskid);
        return;
    }

    // Search for the task to delete
    while (temp != NULL && temp->taskid != taskid) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Task with ID %d doesn't exist!\n", taskid);
        return;
    }

    // Unlink the node and delete it
    prev->next = temp->next;
    free(temp);
    printf("Task with ID %d deleted successfully!\n", taskid);
}

void displaySchedule() {
    struct Task *temp = head;
    if (temp == NULL) {
        printf("No tasks in the schedule.\n");
        return;
    }

    while (temp != NULL) {
        printf("Task ID: %d\nDescription: %s\n", temp->taskid, temp->description);
        temp = temp->next;
    }
}
```

Problem 14: Production Defect Tracking

Description: Implement a linked list to track defects in the production process.

Operations:

1. Create a defect tracking list.
2. Insert a new defect report.
3. Delete a resolved defect.
4. Display all current defects.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct Defect {

   int defectID;

   char description[100];

   struct Defect *next;

};



struct Defect *head = NULL;



void createDefectList();

void insertDefect(int defectID, char description[]);

void deleteDefect(int defectID);

void displayDefects(void);



int main() {

   int choice, defectID;

   char description[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:
            createDefectList();
            break;

        case 2:
            printf("Enter Defect ID: ");
            scanf("%d", &defectID);
            printf("Enter Defect Description: ");
            scanf(" %[^\n]s", description);
            insertDefect(defectID, description);
            break;

        case 3:
            printf("Enter Defect ID to Resolve: ");
```

```
                scanf("%d", &defectID);
                deleteDefect(defectID);
                break;

            case 4:
                displayDefects();
                break;

            case 5:
                printf("Exiting..\n");
                return 0;

            default:
                printf("Invalid Choice!\n");
        }
    }

    return 0;
}

void createDefectList() {
    if (head != NULL) {
        printf("\nDefect Tracking List already created.\n");
    } else {
        head = NULL;
        printf("Defect Tracking List created successfully!\n");
    }
}

void insertDefect(int defectID, char description[]) {
    struct Defect *newDefect = (struct Defect *)malloc(sizeof(struct Defect));
    if (newDefect == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    newDefect->defectID = defectID;
    strcpy(newDefect->description, description);
    newDefect->next = NULL;

    if (head == NULL) {
        head = newDefect;
    } else {
        struct Defect *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newDefect;
    }
    printf("New Defect Report added successfully!\n");
}

void deleteDefect(int defectID) {
    struct Defect *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No defects reported!\n");
        return;
    }

    // If the defect to be resolved is the first node
    if (temp->defectID == defectID) {
        head = temp->next;
        free(temp);
        printf("Defect with ID %d resolved successfully!\n", defectID);
        return;
    }

    // Search for the defect to resolve
    while (temp != NULL && temp->defectID != defectID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Defect with ID %d doesn't exist!\n", defectID);
        return;
    }

    // Unlink the node and resolve it
    prev->next = temp->next;
    free(temp);
    printf("Defect with ID %d resolved successfully!\n", defectID);
}

void displayDefects() {
    struct Defect *temp = head;
    if (temp == NULL) {
        printf("No defects reported.\n");
        return;
    }

    while (temp != NULL) {
        printf("Defect ID: %d\nDescription: %s\n", temp->defectID, temp->description);
        temp = temp->next;
    }
}
```

Problem 15: Finished Goods Dispatch System

Description: Use a linked list to manage the dispatch schedule of finished goods.

Operations:

1. Create a dispatch schedule.
2. Insert a new dispatch entry.
3. Delete a dispatched or canceled entry.
4. Display the current dispatch schedule.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>



struct Dispatch {

    int dispatchID;

    char productName[100];

    int quantity;

    char destination[100];

    struct Dispatch *next;

};



struct Dispatch *head = NULL;



void createDispatchSchedule();

void insertDispatch(int dispatchID, char productName[], int quantity, char
destination[]);

void deleteDispatch(int dispatchID);

void displayDispatchSchedule(void);



int main() {

    int choice, dispatchID, quantity;
```

```
char productName[100], destination[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 or Canceled Entry\n");
    printf("4. Display Current Dispatch Schedule\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            createDispatchSchedule();
            break;

        case 2:
            printf("Enter Dispatch ID: ");
            scanf("%d", &dispatchID);
            printf("Enter Product Name: ");
            scanf(" %[^\n]s", productName);
            printf("Enter Quantity: ");
            scanf("%d", &quantity);
            printf("Enter Destination: ");
```

```
                scanf(" %[^\n]s", destination);
                insertDispatch(dispatchID, productName, quantity, destination);
                break;

            case 3:
                printf("Enter Dispatch ID to Delete: ");
                scanf("%d", &dispatchID);
                deleteDispatch(dispatchID);
                break;

            case 4:
                displayDispatchSchedule();
                break;

            case 5:
                printf("Exiting..\n");
                return 0;

            default:
                printf("Invalid Choice!\n");
        }
    }

    return 0;
}
```

```
void createDispatchSchedule() {
    if (head != NULL) {
        printf("\nDispatch Schedule already created.\n");
    } else {
        head = NULL;
        printf("Dispatch Schedule created successfully!\n");
    }
}


void insertDispatch(int dispatchID, char productName[], int quantity, char destination[]) {
    struct Dispatch *newDispatch = (struct Dispatch *)malloc(sizeof(struct Dispatch));
    if (newDispatch == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    newDispatch->dispatchID = dispatchID;
    strcpy(newDispatch->productName, productName);
    newDispatch->quantity = quantity;
    strcpy(newDispatch->destination, destination);
    newDispatch->next = NULL;

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

    printf("New Dispatch Entry added successfully!\n");
}

void deleteDispatch(int dispatchID) {
    struct Dispatch *temp = head, *prev = NULL;

    if (head == NULL) {
        printf("No dispatch entries found!\n");
        return;
    }

    // If the dispatch to be deleted is the first node
    if (temp->dispatchID == dispatchID) {
        head = temp->next;
        free(temp);
        printf("Dispatch entry with ID %d deleted successfully!\n", dispatchID);
        return;
    }

    // Search for the dispatch entry to delete
    while (temp != NULL && temp->dispatchID != dispatchID) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Dispatch entry with ID %d doesn't exist!\n", dispatchID);
        return;
    }

    // Unlink the node and delete the dispatch
    prev->next = temp->next;
    free(temp);
    printf("Dispatch entry with ID %d deleted successfully!\n", dispatchID);
}

void displayDispatchSchedule() {
    struct Dispatch *temp = head;
    if (temp == NULL) {
        printf("No dispatch entries found.\n");
        return;
    }
    while (temp != NULL) {
        printf("Dispatch ID: %d\nProduct Name: %s\nQuantity: %d\nDestination: %s\n",
            temp->dispatchID, temp->productName, temp->quantity, temp->destination);
        temp = temp->next;
    }
}
```

Problem 1: Team Roster Management

Description: Implement a linked list to manage the roster of players in a sports team.**Operations:**

1. Create a team roster.
2. Insert a new player.
3. Delete a player who leaves the team.
4. Display the current team roster.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



// Define the structure for a player in the roster

struct Player {

    int playerID;

    char playerName[100];

    char position[50];

    struct Player *next;

};
```

```
// Global pointer to the head of the linked list
struct Player *head = NULL;

// Function prototypes
void createRoster();
void insertPlayer(int playerID, char playerName[], char position[]);
void deletePlayer(int playerID);
void displayRoster();

int main() {
    int choice, playerID;
    char playerName[100], position[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 Current Team Roster\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
case 1:
    createRoster();
    break;

case 2:
    printf("Enter Player ID: ");
    scanf("%d", &playerID);
    printf("Enter Player Name: ");
    scanf(" %[^\n]s", playerName);
    printf("Enter Player Position: ");
    scanf(" %[^\n]s", position);
    insertPlayer(playerID, playerName, position);
    break;

case 3:
    printf("Enter Player ID to Delete: ");
    scanf("%d", &playerID);
    deletePlayer(playerID);
    break;

case 4:
    displayRoster();
    break;

case 5:
```

```
                printf("Exiting..\n");
                return 0;

            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}

// Function to create a team roster
void createRoster() {
    if (head != NULL) {
        printf("Roster already exists.\n");
    } else {
        head = NULL;
        printf("Team roster created successfully!\n");
    }
}

// Function to insert a new player into the roster
void insertPlayer(int playerID, char playerName[], char position[]) {
    struct Player *newPlayer = (struct Player *)malloc(sizeof(struct Player));
    if (newPlayer == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    newPlayer->playerID = playerID;
    strcpy(newPlayer->playerName, playerName);
    strcpy(newPlayer->position, position);
    newPlayer->next = NULL;

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

    printf("Player added successfully!\n");
}

// Function to delete a player from the roster
void deletePlayer(int playerID) {
    struct Player *temp = head, *prev = NULL;
```

```
if (head == NULL) {

    printf("No players in the roster.\n");

    return;

}


// If the player to be deleted is the first node

if (temp != NULL && temp->playerID == playerID) {

    head = temp->next;

    free(temp);

    printf("Player with ID %d deleted successfully!\n", playerID);

    return;

}


// Search for the player to delete

while (temp != NULL && temp->playerID != playerID) {

    prev = temp;

    temp = temp->next;

}


if (temp == NULL) {

    printf("Player with ID %d not found.\n", playerID);

    return;

}
```

```
    // Unlink the node and delete it
    prev->next = temp->next;
    free(temp);
    printf("Player with ID %d deleted successfully!\n", playerID);
}

// Function to display the current team roster
void displayRoster() {
    struct Player *temp = head;

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

    printf("Current Team Roster:\n");
    while (temp != NULL) {
        printf("Player ID: %d\nPlayer Name: %s\nPosition: %s\n",
               temp->playerID, temp->playerName, temp->position);
        temp = temp->next;
    }
}
```

Problem 2: Tournament Match Scheduling

Description: Use a linked list to schedule matches in a tournament.**Operations:**

1. Create a match schedule.

2. Insert a new match.
3. Delete a completed or canceled match.
4. Display the current match schedule.

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

struct Match{
    int matchID;
    char team1Name[50];
    char team2Name [50];
    char date[10];
    struct Match *next;
};

struct Match *head=NULL;

void createSchedule();
void insertMatch(int matchID,char team1[],char team2[],char date[]);
void deleteMatch(int matchID);
void displaySchedule();

int main(){
    int matchID;
    char team1[50],team2[50],date[10];
```

```
int choice;

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 Current 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 ID: ");

        scanf("%d",&matchID);

        printf("Enter team1 Name: ");

        scanf(" %[^\n]s",team1);

        printf("Enter team2 Name: ");

        scanf(" %[^\n]s",team2);

        printf("Enter Match Date: ");

        scanf(" %[^\n]s",date);
        insertMatch(matchID,team1,team2,date);

        break;



        case 3:

        printf("Enter match ID to delete : ");

        scanf("%d",&matchID);

        deleteMatch(matchID);

        break;



        case 4:

        displaySchedule();

        break;



        case 5:

        printf("Exiting..");

        exit(0);



        default:

        printf("Invalid Choice");



    }

}



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 matchID,char team1[],char team2[],char date[]){
    struct Match *newMatch = (struct Match *)malloc(sizeof(struct Match));
    if(newMatch==NULL){
        printf("Memory Allocation Failed!\n");
        return;
    }
    newMatch->matchID=matchID;
    strcpy(newMatch->team1Name,team1);
    strcpy(newMatch->team2Name,team2);
    strcpy(newMatch->date,date);
    newMatch->next=NULL;

    if(head==NULL){
        head=newMatch;
    }
    else{
        struct Match *temp=head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=newMatch;
    }
    printf("Match with ID %d is inserted!\n",matchID);
}

void deleteMatch(int matchID){
    struct Match *temp=head,*prev=NULL;
    if(temp==NULL){
        printf("No Matches in the schedule!\n");
        return;
    }
    if(temp->matchID==matchID){
        head=temp->next;
        free(temp);
        printf("Match with ID %d is deleted!\n",matchID);
        return;
    }
    else{
        while(temp!=NULL && temp->matchID!=matchID){
            prev=temp;

            temp=temp->next;

        }

        prev->next=temp->next;

        free(temp);

        printf("Match with ID %d is deleted succesfully!\n",matchID);

    }

}

void displaySchedule(){

    struct Match *temp=head;

    if(temp==NULL){

        printf("No match scheduled!\n");

        return;

    }

    while(temp!=NULL){

        printf("Match ID:\t%d, Team1 Name:\t%s, team2Name:\t%s,date:\t%s\n",temp->matchID,temp->team1Name,temp->team2Name,temp->date);

        temp=temp->next;

    }



}
```

Problem 3: Athlete Training Log

Description: Develop a linked list to log training sessions for athletes.**Operations:**

1. Create a training log.
2. Insert a new training session.
3. Delete a completed or canceled session.
4. Display the training log.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



// Define the structure for a training session

struct TrainingSession {

    int sessionID;

    char athleteName[50];

    char trainingType[50];

    char date[10];

    struct TrainingSession *next;

};



// Global pointer to the head of the linked list

struct TrainingSession *head = NULL;



// Function prototypes

void createTrainingLog();

void insertSession(int sessionID, char athleteName[], char trainingType[], char date[]);

void deleteSession(int sessionID);

void displayTrainingLog();
```

```
int main() {
    int sessionID;
    char athleteName[50], trainingType[50], date[10];
    int choice;

    while (1) {
        printf("\nAthlete Training Log:\n");
        printf("1. Create Training Log\n");
        printf("2. Insert New Training Session\n");
        printf("3. Delete Completed or Canceled Session\n");
        printf("4. Display Training Log\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createTrainingLog();
                break;

            case 2:
                printf("Enter Session ID: ");
                scanf("%d", &sessionID);
                printf("Enter Athlete Name: ");
                scanf(" %[^\n]s", athleteName);
            printf("Enter Training Type: ");
            scanf(" %[^\n]s", trainingType);
            printf("Enter Training Date: ");
            scanf(" %[^\n]s", date);
            insertSession(sessionID, athleteName, trainingType, date);
            break;

        case 3:
            printf("Enter Session ID to delete: ");
            scanf("%d", &sessionID);
            deleteSession(sessionID);
            break;

        case 4:
            displayTrainingLog();
            break;

        case 5:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice!\n");
    }
}
```

```
    return 0;
}


// Function to create a training log
void createTrainingLog() {
    if (head != NULL) {
        printf("Training Log already exists!\n");
    } else {
        head = NULL;
        printf("Training Log created successfully!\n");
    }
}


// Function to insert a new training session
void insertSession(int sessionID, char athleteName[], char trainingType[], char date[]) {
    struct TrainingSession *newSession = (struct TrainingSession *)malloc(sizeof(struct TrainingSession));
    if (newSession == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newSession->sessionID = sessionID;
    strcpy(newSession->athleteName, athleteName);
    strcpy(newSession->trainingType, trainingType);
    strcpy(newSession->date, date);
    newSession->next = NULL;

    if (head == NULL) {
        head = newSession;
    } else {
        struct TrainingSession *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newSession;
    }
    printf("Training session with ID %d added successfully!\n", sessionID);
}

// Function to delete a training session
void deleteSession(int sessionID) {
    struct TrainingSession *temp = head, *prev = NULL;

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

    if (temp->sessionID == sessionID) {
        head = temp->next;
        free(temp);
        printf("Training session with ID %d deleted successfully!\n", sessionID);
        return;
    }

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

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

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

// Function to display the training log
void displayTrainingLog() {
    struct TrainingSession *temp = head;
    if (temp == NULL) {
        printf("No training sessions in the log!\n");
        return;
    }

    printf("Current Training Log:\n");
    while (temp != NULL) {
        printf("Session ID: %d, Athlete Name: %s, Training Type: %s, Date: %s\n",
               temp->sessionID, temp->athleteName, temp->trainingType, temp->date);
        temp = temp->next;
    }
}
```

Problem 4: Sports Equipment Inventory

Description: Use a linked list to manage the inventory of sports equipment.**Operations:**

1. Create an equipment inventory list.
2. Insert a new equipment item.
3. Delete an item that is no longer usable.
4. Display the current equipment inventory.

Problem 5: Player Performance Tracking

Description: Implement a linked list to track player performance over the season.**Operations:**

1. Create a performance record list.
2. Insert a new performance entry.
3. Delete an outdated or erroneous entry.
4. Display all performance records.

Problem 6: Event Registration System

Description: Use a linked list to manage athlete registrations for sports events.**Operations:**

1. Create a registration list.
2. Insert a new registration.
3. Delete a canceled registration.

4. Display all current registrations.

Problem 7: Sports League Standings

Description: Develop a linked list to manage the standings of teams in a sports league.**Operations:**

1. Create a league standings list.
2. Insert a new team.
3. Delete a team that withdraws.
4. Display the current league standings.

Problem 8: Match Result Recording

Description: Implement a linked list to record results of matches.**Operations:**

1. Create a match result list.
2. Insert a new match result.
3. Delete an incorrect or outdated result.
4. Display all recorded match results.

Problem 9: Player Injury Tracker

Description: Use a linked list to track injuries of players.**Operations:**

1. Create an injury tracker list.
2. Insert a new injury report.
3. Delete a resolved or erroneous injury report.
4. Display all current injury reports.

Problem 10: Sports Facility Booking System

Description: Manage bookings for sports facilities using a linked list.**Operations:**

1. Create a booking list.
2. Insert a new booking.
3. Delete a canceled or completed booking.
4. Display all current bookings.

Problem 11: Coaching Staff Management

Description: Use a linked list to manage the coaching staff of a sports team.**Operations:**

1. Create a coaching staff list.
2. Insert a new coach.
3. Delete a coach who leaves the team.
4. Display the current coaching staff.

Problem 12: Fan Club Membership Management

Description: Implement a linked list to manage memberships in a sports team's fan club.**Operations:**

1. Create a membership list.
2. Insert a new member.
3. Delete a member who cancels their membership.
4. Display all current members.

Problem 13: Sports Event Scheduling

Description: Use a linked list to manage the schedule of sports events.**Operations:**

1. Create an event schedule.
2. Insert a new event.
3. Delete a completed or canceled event.
4. Display the current event schedule.

Problem 14: Player Transfer Records

Description: Maintain a linked list to track player transfers between teams.**Operations:**

1. Create a transfer record list.
2. Insert a new transfer record.
3. Delete an outdated or erroneous transfer record.
4. Display all current transfer records.

Problem 15: Championship Points Tracker

Description: Implement a linked list to track championship points for teams.**Operations:**

1. Create a points tracker list.
2. Insert a new points entry.
3. Delete an incorrect or outdated points entry.
4. Display all current points standings.
5.