Deleting a node in linkedlist

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int data;
   struct Node *next;
}*first = NULL;
void create(int [], int);
void display(struct Node *);
void Insert(struct Node *,int ,int );
int DeleteNode(struct Node *, int );
int main()
{
   int A[] = {1,2,3,4,5};
   int x;
   create(A,5);
   display(first);
   Insert(first,4,6);
   printf("\n");
   display(first);
   printf("\n");
   x = DeleteNode(first, 2);
   display(first);
   printf("\n");
   printf("deletednode = %d \n",x);
   x = DeleteNode(first, 1);
   display(first);
```

```
    printf("\n");
    printf("deletednode = %d \n",x);
    x = DeleteNode(first, 4);
    display(first);
    printf("\n");
    printf("deletednode = %d \n",x);
    return 0;
}
void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}
void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
    }
```

```
}
void Insert(struct Node *p,int index ,int x){
   struct Node *temp;
   int i;
   temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = x;  //x =6
   if(index ==0){
      temp->next = first;
      first = temp;
   }
   else{
      for(i=0;i<(index-1);i++){
         p = p->next;
      }
      temp->next = p->next;
      p->next =temp;
   }
}

int DeleteNode(struct Node *p, int pos){
   struct Node *q = NULL;
   int num, i;
   if(pos == 1){
      q = first;
      num = first->data; //extracting the value from the first node
      first = first->next; //moving the pointer first point to the next Node
      free(q); //deleting the first node
      return num;
```

```
    }else{
        for (i = 0; i < pos-1; i++){
            q = p;
            p = p->next;
        }
        q->next = p->next;
        num = p->data;
        free(p);
        return num;
    }
}
```

/*

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 Inventory {
   char materialName[100];
   int quantity;
   struct Inventory *next;
} *first = NULL;



void createInventoryList(char *names[], int quantities[], int n);
void displayInventory(struct Inventory *p);
void insertRawMaterial(struct Inventory *p, int index, char *materialName, int quantity);
int deleteRawMaterial(struct Inventory *p, int index);

int main() {
   char *materials[] = {"Material1", "Material2", "Material3", "Material4", "Material5"};
   int quantities[] = {10, 20, 30, 40, 50};
   int n = 5;
   int x;

   createInventoryList(materials, quantities, n);
   printf("/n");
   displayInventory(first);

   insertRawMaterial(first, 4, "Material6", 60);
   printf("\n");
   displayInventory(first);
```

```
    printf("\n");
    x = deleteRawMaterial(first, 2);
    displayInventory(first);
    printf("\n");
    printf("Deleted material quantity = %d \n",x);

    // x = deleteRawMaterial(first, 1);
    // displayInventory(first);
    // printf("\n");
    // printf("Deleted material quantity = %d \n",x);

    // x = deleteRawMaterial(first, 4);
    // displayInventory(first);
    // printf("\n");
    // printf("Deleted material quantity = %d \n",x);

    return 0;
}


void createInventoryList(char *names[], int quantities[], int n) {
    int i;
    struct Inventory *temp, *last;
    first = (struct Inventory*)malloc(sizeof(struct Inventory));
    strcpy(first->materialName, names[0]);
    first->quantity = quantities[0];
    first->next = NULL;
```

```
    last = first;
    for (i = 1; i < n; i++) {
        temp = (struct Inventory*)malloc(sizeof(struct Inventory));
        strcpy(temp->materialName, names[i]);
        temp->quantity = quantities[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}


void displayInventory(struct Inventory *p) {
    while (p != NULL) {
        printf("Material: %s, Quantity: %d -> ", p->materialName, p->quantity);
        p = p->next;
    }
    printf("NULL\n");
}


void insertRawMaterial(struct Inventory *p, int index, char *materialName, int quantity) {
    struct Inventory *temp;
    int i;
    temp = (struct Inventory*)malloc(sizeof(struct Inventory));
    strcpy(temp->materialName, materialName);
    temp->quantity = quantity;
```

```
    if (index == 0) {
        temp->next = first;
        first = temp;
    } else {
        for (i = 0; i < (index - 1); i++) {
            p = p->next;
        }
        temp->next = p->next;
        p->next = temp;
    }
    printf("Raw material %s with quantity %d added.\n", materialName, quantity);
}


int deleteRawMaterial(struct Inventory *p, int index) {
    struct Inventory *q = NULL;
    int quantity, i;

    if (index == 0) {
        q = first;
        quantity = first->quantity;
        first = first->next;
        free(q);
        return quantity;
    } else {
        for (i = 0; i < (index - 1); i++) {
            q = p;
            p = p->next;
```

```
        }
        q->next = p->next;
        quantity = p->quantity;
        free(p);
        return quantity;
    }
}
```

```
/*

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 Task {
    char taskName[100];
    struct Task *next;
```

```
} *first = NULL;



void createTaskQueue();
void insertTask(char *taskName);
int deleteTask();
void displayTaskQueue();

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

    while (1) {
        printf("\nProduction Line Queue Management System\n");
        printf("1. Create Task Queue\n2. Insert New Task\n3. Delete Completed Task\n4. Display Current Task Queue\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createTaskQueue();
                break;
            case 2:
                printf("Enter task name: ");
                scanf("%s", taskName);
                insertTask(taskName);
```

```
            break;
        case 3:
            deleted = deleteTask();
            if (deleted != -1)
                printf("Task %d deleted from the queue.\n", deleted);
            break;
        case 4:
            displayTaskQueue();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createTaskQueue() {
   first = NULL;
   printf("New task queue created.\n");
}



void insertTask(char *taskName) {
```

```
    struct Task *newTask = (struct Task*)malloc(sizeof(struct Task));
    strcpy(newTask->taskName, taskName);
    newTask->next = NULL;

    if (first == NULL) {
        first = newTask;
    } else {
        struct Task *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("Task %s added to the queue.\n", taskName);
}


int deleteTask() {
    struct Task *temp = first;
    if (first == NULL) {
        printf("The task queue is empty.\n");
        return -1;
    } else {
        first = first->next;
        free(temp);
        return 1;
    }
}
```

```
void displayTaskQueue() {
    struct Task *temp = first;
    if (temp == NULL) {
        printf("The task queue is empty.\n");
        return;
    }
    printf("Current Task Queue:\n");
    while (temp != NULL) {
        printf("Task: %s -> ", temp->taskName);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
/*

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 MaintenanceTask {

   char machineName[100];

   char taskDescription[200];

   char maintenanceDate[20];

   struct MaintenanceTask *next;

} *first = NULL;



void createMaintenanceSchedule();
```

```
void insertMaintenanceTask(char *machineName, char *taskDescription, char
*maintenanceDate);

int deleteMaintenanceTask();

void displayMaintenanceSchedule();


int main() {
   int choice;
   char machineName[100], taskDescription[200], maintenanceDate[20];
   int deleted;


   while (1) {
      printf("\nMachine Maintenance Schedule Management System\n");
      printf("1. Create Maintenance Schedule\n2. Insert New Maintenance Task\n3. Delete
Completed Maintenance Task\n4. Display Maintenance Schedule\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);


      switch (choice) {
         case 1:
            createMaintenanceSchedule();
            break;
         case 2:
            printf("Enter machine name: ");
            scanf("%s", machineName);
            printf("Enter task description: ");
            scanf(" %s", taskDescription);
            printf("Enter maintenance date (DD/MM/YYYY): ");
            scanf("%s", maintenanceDate);
            insertMaintenanceTask(machineName, taskDescription, maintenanceDate);
```

```
            break;
        case 3:
            deleted = deleteMaintenanceTask();
            if (deleted != -1)
                printf("Completed maintenance task deleted.\n");
            break;
        case 4:
            displayMaintenanceSchedule();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createMaintenanceSchedule() {
   first = NULL;
   printf("New maintenance schedule created.\n");
}

void insertMaintenanceTask(char *machineName, char *taskDescription, char
*maintenanceDate) {
```

```
    struct MaintenanceTask *newTask = (struct MaintenanceTask*)malloc(sizeof(struct
MaintenanceTask));
    strcpy(newTask->machineName, machineName);
    strcpy(newTask->taskDescription, taskDescription);
    strcpy(newTask->maintenanceDate, maintenanceDate);
    newTask->next = NULL;

    if (first == NULL) {
        first = newTask;
    } else {
        struct MaintenanceTask *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("Maintenance task for machine %s added to the schedule.\n", machineName);
}



int deleteMaintenanceTask() {
    struct MaintenanceTask *temp = first;
    if (first == NULL) {
        printf("The maintenance schedule is empty.\n");
        return -1;
    } else {
        first = first->next;
        free(temp);
```

```
        return 1;
    }
}



void displayMaintenanceSchedule() {
    struct MaintenanceTask *temp = first;
    if (temp == NULL) {
        printf("The maintenance schedule is empty.\n");
        return;
    }
    printf("Maintenance Schedule:\n");
    while (temp != NULL) {
        printf("Machine: %s, Task: %s, Date: %s\n", temp->machineName, temp-
>taskDescription, temp->maintenanceDate);
        temp = temp->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 Shift {
    char employeeName[100];
    char shiftDate[20];
    char shiftTime[10];
    struct Shift *next;
} *first = NULL;


void createShiftSchedule();
void insertShift(char *employeeName, char *shiftDate, char *shiftTime);
int deleteShift(int index);
void displayShiftSchedule();

int main() {
    int choice, index;
    char employeeName[100], shiftDate[20], shiftTime[10];
    int deleted;

    while (1) {
        printf("\nEmployee Shift Management System\n");
```

```
    printf("1. Create Shift Schedule\n2. Insert New Shift\n3. Delete Completed or Canceled Shift\n4. Display Current Shift Schedule\n5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            createShiftSchedule();
            break;
        case 2:
            printf("Enter employee name: ");
            scanf("%s", employeeName);
            printf("Enter shift date (DD/MM/YYYY): ");
            scanf("%s", shiftDate);
            printf("Enter shift time (HH:MM): ");
            scanf("%s", shiftTime);
            insertShift(employeeName, shiftDate, shiftTime);
            break;
        case 3:
            printf("Enter shift index to delete: ");
            scanf("%d", &index);
            deleted = deleteShift(index);
            if (deleted != -1)
                printf("Shift at index %d deleted.\n", index);
            break;
        case 4:
            displayShiftSchedule();
            break;
```

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

    return 0;
}

// Function to create a new shift schedule
void createShiftSchedule() {
    first = NULL;
    printf("New shift schedule created.\n");
}

// Function to insert a new shift
void insertShift(char *employeeName, char *shiftDate, char *shiftTime) {
    struct Shift *newShift = (struct Shift*)malloc(sizeof(struct Shift));
    strcpy(newShift->employeeName, employeeName);
    strcpy(newShift->shiftDate, shiftDate);
    strcpy(newShift->shiftTime, shiftTime);
    newShift->next = NULL;

    if (first == NULL) {
        first = newShift;
    } else {
```

```
        struct Shift *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newShift;
    }
    printf("Shift for employee %s on %s at %s added to the schedule.\n", employeeName,
shiftDate, shiftTime);
}

// Function to delete shift from the schedule
int deleteShift(int index) {
    struct Shift *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
```

```
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}

// Function to display the schedule
void displayShiftSchedule() {
    struct Shift *temp = first;
    if (temp == NULL) {
        printf("The shift schedule is empty.\n");
        return;
    }
    printf("Current Shift Schedule:\n");
    while (temp != NULL) {
        printf("Employee: %s, Date: %s, Time: %s\n", temp->employeeName, temp->shiftDate,
temp->shiftTime);
        temp = temp->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 Order {

    char customerName[100];

    char productName[100];

    int quantity;

    struct Order *next;

} *first = NULL;



void createOrderList();

void insertOrder(char *customerName, char *productName, int quantity);
```

```
int deleteOrder(int index);
void displayOrders();

int main() {
   int choice, index, quantity;
   char customerName[100], productName[100];
   int deleted;

   while (1) {

      printf("1. Create Order List\n2. Insert New Customer Order\n3. Delete Completed or Canceled Order\n4. Display All Current Orders\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            createOrderList();
            break;
         case 2:
            printf("Enter customer name: ");
            scanf("%s", customerName);
            printf("Enter product name: ");
            scanf("%s", productName);
            printf("Enter quantity: ");
            scanf("%d", &quantity);
            insertOrder(customerName, productName, quantity);
            break;
```

```
            case 3:
                printf("Enter order index to delete: ");
                scanf("%d", &index);
                deleted = deleteOrder(index);
                if (deleted != -1)
                    printf("Order at index %d deleted.\n", index);
                break;
            case 4:
                displayOrders();
                break;
            case 5:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}


void createOrderList() {
    first = NULL;
    printf("New order list created.\n");
}
```

```
void insertOrder(char *customerName, char *productName, int quantity) {
    struct Order *newOrder = (struct Order*)malloc(sizeof(struct Order));
    strcpy(newOrder->customerName, customerName);
    strcpy(newOrder->productName, productName);
    newOrder->quantity = quantity;
    newOrder->next = NULL;

    if (first == NULL) {
        first = newOrder;
    } else {
        struct Order *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newOrder;
    }
    printf("Order for customer %s added to the list.\n", customerName);
}

int deleteOrder(int index) {
    struct Order *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
```

```
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}



void displayOrders() {
    struct Order *temp = first;
    if (temp == NULL) {
        printf("No orders available.\n");
        return;
    }
    printf("Current Orders:\n");
    while (temp != NULL) {
        printf("Customer: %s, Product: %s, Quantity: %d\n", temp->customerName, temp-
>productName, temp->quantity);
        temp = temp->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 Tool {

   char toolName[100];

   char toolID[20];

   char lastUsedDate[20];

   struct Tool *next;

} *first = NULL;



void createToolList();

void insertTool(char *toolName, char *toolID, char *lastUsedDate);
```

```
int deleteTool(int index);
void displayTools();

int main() {
    int choice, index;
    char toolName[100], toolID[20], lastUsedDate[20];
    int deleted;

    while (1) {

        printf("1. Create Tool Tracking List\n2. Insert New Tool Entry\n3. Delete Tool No Longer in Use\n4. Display All Tools Currently Tracked\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createToolList();
                break;
            case 2:
                printf("Enter tool name: ");
                scanf("%s", toolName);
                printf("Enter tool ID: ");
                scanf("%s", toolID);
                printf("Enter last used date (DD/MM/YYYY): ");
                scanf("%s", lastUsedDate);
                insertTool(toolName, toolID, lastUsedDate);
                break;
```

```
            case 3:
                printf("Enter tool index to delete: ");
                scanf("%d", &index);
                deleted = deleteTool(index);
                if (deleted != -1)
                    printf("Tool at index %d deleted.\n", index);
                break;
            case 4:
                displayTools();
                break;
            case 5:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Function to create a new tool tracking list
void createToolList() {
    first = NULL;
    printf("New tool tracking list created.\n");
}

// Function to insert a new tool entry
```

```
void insertTool(char *toolName, char *toolID, char *lastUsedDate) {
    struct Tool *newTool = (struct Tool*)malloc(sizeof(struct Tool));
    strcpy(newTool->toolName, toolName);
    strcpy(newTool->toolID, toolID);
    strcpy(newTool->lastUsedDate, lastUsedDate);
    newTool->next = NULL;

    if (first == NULL) {
        first = newTool;
    } else {
        struct Tool *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTool;
    }
    printf("Tool %s added to the list.\n", toolName);
}

// Function to delete a tool
int deleteTool(int index) {
    struct Tool *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
```

```
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}

// Function to display all tools
void displayTools() {
    struct Tool *temp = first;
    if (temp == NULL) {
        printf("No tools are currently tracked.\n");
        return;
    }
    printf("Currently Tracked Tools:\n");
    while (temp != NULL) {
        printf("Tool: %s, ID: %s, Last Used Date: %s\n", temp->toolName, temp->toolID, temp->lastUsedDate);
        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 AssemblyStage {
    char stageName[100];
    int stageNumber;
    struct AssemblyStage *next;
} *first = NULL;


void createAssemblyLine();
void insertStage(char *stageName, int stageNumber);
int deleteStage(int stageNumber);
```

```
void displayAssemblyStages();

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

    while (1) {

        printf("1. Create Assembly Line Stage List\n2. Insert New Stage\n3. Delete Completed Stage\n4. Display Current Assembly Stages\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createAssemblyLine();
                break;
            case 2:
                printf("Enter stage name: ");
                scanf("%s", stageName);
                printf("Enter stage number: ");
                scanf("%d", &stageNumber);
                insertStage(stageName, stageNumber);
                break;
            case 3:
                printf("Enter stage number to delete: ");
                scanf("%d", &stageNumber);
                deleted = deleteStage(stageNumber);
```

```
            if (deleted != -1)
                printf("Stage %d deleted.\n", stageNumber);
            break;
        case 4:
            displayAssemblyStages();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new assembly line stage list
void createAssemblyLine() {
   first = NULL;
   printf("New assembly line stage list created.\n");
}

// Function to insert a new stage into the list
void insertStage(char *stageName, int stageNumber) {
   struct AssemblyStage *newStage = (struct AssemblyStage*)malloc(sizeof(struct
AssemblyStage));
   strcpy(newStage->stageName, stageName);
```

```
    newStage->stageNumber = stageNumber;
    newStage->next = NULL;

    if (first == NULL) {
        first = newStage;
    } else {
        struct AssemblyStage *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newStage;
    }
    printf("Stage %d: %s added to the list.\n", stageNumber, stageName);
}

// Function to delete a completed stage from the list
int deleteStage(int stageNumber) {
    struct AssemblyStage *temp = first, *prev = NULL;

    if (temp != NULL && temp->stageNumber == stageNumber) {
        first = temp->next;
        free(temp);
        return stageNumber;
    } else {
        while (temp != NULL && temp->stageNumber != stageNumber) {
            prev = temp;
            temp = temp->next;
        }
```

```
        if (temp == NULL) {
            printf("Stage %d not found.\n", stageNumber);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return stageNumber;
    }
}

// Function to display assembly stages
void displayAssemblyStages() {
    struct AssemblyStage *temp = first;
    if (temp == NULL) {
        printf("No assembly stages currently tracked.\n");
        return;
    }
    printf("Current Assembly Stages:\n");
    while (temp != NULL) {
        printf("Stage %d: %s\n", temp->stageNumber, 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.

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


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


void createChecklist();
void insertChecklistItem(char *itemName, int itemID);
int deleteChecklistItem(int itemID);
void displayChecklist();

int main() {
```

```
    int choice, itemID, deleted;
    char itemName[100];

    while (1) {

        printf("1. Create Checklist\n2. Insert New Checklist Item\n3. Delete Completed or Outdated Checklist Item\n4. Display Current Checklist\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createChecklist();
                break;
            case 2:
                printf("Enter item name: ");
                scanf("%s", itemName);
                printf("Enter item ID: ");
                scanf("%d", &itemID);
                insertChecklistItem(itemName, itemID);
                break;
            case 3:
                printf("Enter item ID to delete: ");
                scanf("%d", &itemID);
                deleted = deleteChecklistItem(itemID);
                if (deleted != -1)
                    printf("Item %d deleted.\n", itemID);
                break;
```

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

    return 0;
}

// Function to create a new checklist
void createChecklist() {
    first = NULL;
    printf("New checklist created.\n");
}

// Function to insert a new checklist item into the list
void insertChecklistItem(char *itemName, int itemID) {
    struct ChecklistItem *newItem = (struct ChecklistItem*)malloc(sizeof(struct
ChecklistItem));
    strcpy(newItem->itemName, itemName);
    newItem->itemID = itemID;
    newItem->next = NULL;
```

```
    if (first == NULL) {
        first = newItem;
    } else {
        struct ChecklistItem *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("Checklist item %s with ID %d added to the list.\n", itemName, itemID);
}

// Function to delete checklist item
int deleteChecklistItem(int itemID) {
    struct ChecklistItem *temp = first, *prev = NULL;

    if (temp != NULL && temp->itemID == itemID) {
        first = temp->next;
        free(temp);
        return itemID;
    } else {
        while (temp != NULL && temp->itemID != itemID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Item with ID %d not found.\n", itemID);
            return -1;
```

```
        }
        prev->next = temp->next;
        free(temp);
        return itemID;
    }
}

// Function to display quality control checklist
void displayChecklist() {
    struct ChecklistItem *temp = first;
    if (temp == NULL) {
        printf("No items in the checklist.\n");
        return;
    }
    printf("Current Quality Control Checklist:\n");
    while (temp != NULL) {
        printf("Item ID: %d, Name: %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 ChecklistItem {
   char itemName[100];
   int itemID;
   struct ChecklistItem *next;
} *first = NULL;



void createChecklist();
void insertChecklistItem(char *itemName, int itemID);
int deleteChecklistItem(int itemID);
void displayChecklist();
```

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

    while (1) {

        printf("1. Create Checklist\n2. Insert New Checklist Item\n3. Delete Completed or Outdated Checklist Item\n4. Display Current Checklist\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createChecklist();
                break;
            case 2:
                printf("Enter item name: ");
                scanf("%s", itemName);
                printf("Enter item ID: ");
                scanf("%d", &itemID);
                insertChecklistItem(itemName, itemID);
                break;
            case 3:
                printf("Enter item ID to delete: ");
                scanf("%d", &itemID);
                deleted = deleteChecklistItem(itemID);
                if (deleted != -1)
```

```
                printf("Item %d deleted.\n", itemID);
            break;
        case 4:
            displayChecklist();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new checklist
void createChecklist() {
   first = NULL;
   printf("New checklist created.\n");
}

// Function to insert a new checklist item into the list
void insertChecklistItem(char *itemName, int itemID) {
   struct ChecklistItem *newItem = (struct ChecklistItem*)malloc(sizeof(struct
ChecklistItem));
   strcpy(newItem->itemName, itemName);
   newItem->itemID = itemID;
```

```
    newItem->next = NULL;

    if (first == NULL) {
        first = newItem;
    } else {
        struct ChecklistItem *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("Checklist item %s with ID %d added to the list.\n", itemName, itemID);
}

// Function to delete checklist item
int deleteChecklistItem(int itemID) {
    struct ChecklistItem *temp = first, *prev = NULL;

    if (temp != NULL && temp->itemID == itemID) {
        first = temp->next;
        free(temp);
        return itemID;
    } else {
        while (temp != NULL && temp->itemID != itemID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
```

```
            printf("Item with ID %d not found.\n", itemID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return itemID;
    }
}

// Function to display quality control checklist
void displayChecklist() {
    struct ChecklistItem *temp = first;
    if (temp == NULL) {
        printf("No items in the checklist.\n");
        return;
    }
    printf("Current Quality Control Checklist:\n");
    while (temp != NULL) {
        printf("Item ID: %d, Name: %s\n", temp->itemID, temp->itemName);
        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 Milestone {
   char milestoneName[100];
   char dueDate[20];
   struct Milestone *next;
} *first = NULL;


void createProjectTimeline();
void insertMilestone(char *milestoneName, char *dueDate);
int deleteMilestone(int index);
void displayTimeline();
```

```
int main() {
    int choice, index, deleted;
    char milestoneName[100], dueDate[20];

    while (1) {
        printf("\nManufacturing Project Timeline Management System\n");
        printf("1. Create Project Timeline\n2. Insert New Project Milestone\n3. Delete
Completed Milestone\n4. Display Current Project Timeline\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createProjectTimeline();
                break;
            case 2:
                printf("Enter milestone name: ");
                scanf("%s", milestoneName);
                printf("Enter due date (DD/MM/YYYY): ");
                scanf("%s", dueDate);
                insertMilestone(milestoneName, dueDate);
                break;
            case 3:
                printf("Enter milestone index to delete: ");
                scanf("%d", &index);
                deleted = deleteMilestone(index);
                if (deleted != -1)
                    printf("Milestone at index %d deleted.\n", index);
```

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

   return 0;
}

// Function to create a new project timelinw
void createProjectTimeline() {
   first = NULL;
   printf("New project timeline created.\n");
}

// Function to insert a new project milestone into the timeline
void insertMilestone(char *milestoneName, char *dueDate) {
   struct Milestone *newMilestone = (struct Milestone*)malloc(sizeof(struct Milestone));
   strcpy(newMilestone->milestoneName, milestoneName);
   strcpy(newMilestone->dueDate, dueDate);
   newMilestone->next = NULL;
```

```
    if (first == NULL) {
        first = newMilestone;
    } else {
        struct Milestone *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newMilestone;
    }
    printf("Milestone %s added to the timeline.\n", milestoneName);
}

// Function to delete a milestone
int deleteMilestone(int index) {
    struct Milestone *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
```

```
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}

// Function to display the timeline
void displayTimeline() {
    struct Milestone *temp = first;
    if (temp == NULL) {
        printf("No milestones in the timeline.\n");
        return;
    }
    printf("Current Project Timeline:\n");
    while (temp != NULL) {
        printf("Milestone: %s, Due Date: %s\n", temp->milestoneName, temp->dueDate);
        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 StorageEntry {
    char itemName[100];
    int quantity;
    char storageDate[20];
    struct StorageEntry *next;
} *first = NULL;



void createStorageList();
void insertStorageEntry(char *itemName, int quantity, char *storageDate);
int deleteStorageEntry(int index);
void displayStorage();
```

```
int main() {
    int choice, index, quantity, deleted;
    char itemName[100], storageDate[20];

    while (1) {
        printf("\nWarehouse Storage Management System\n");
        printf("1. Create Storage List\n2. Insert New Storage Entry\n3. Delete Storage Entry When Goods are Shipped\n4. Display Current Warehouse Storage\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createStorageList();
                break;
            case 2:
                printf("Enter item name: ");
                scanf("%s", itemName);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                printf("Enter storage date (DD/MM/YYYY): ");
                scanf("%s", storageDate);
                insertStorageEntry(itemName, quantity, storageDate);
                break;
            case 3:
                printf("Enter storage entry index to delete: ");
                scanf("%d", &index);
```

```
            deleted = deleteStorageEntry(index);
            if (deleted != -1)
                printf("Storage entry at index %d deleted.\n", index);
            break;
        case 4:
            displayStorage();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new storage list
void createStorageList() {
   first = NULL;
   printf("New storage list created.\n");
}

// Function to insert a new storage entry into the list
void insertStorageEntry(char *itemName, int quantity, char *storageDate) {
   struct StorageEntry *newEntry = (struct StorageEntry*)malloc(sizeof(struct StorageEntry));
   strcpy(newEntry->itemName, itemName);
```

```
    newEntry->quantity = quantity;
    strcpy(newEntry->storageDate, storageDate);
    newEntry->next = NULL;

    if (first == NULL) {
        first = newEntry;
    } else {
        struct StorageEntry *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEntry;
    }
    printf("Storage entry for item %s added to the list.\n", itemName);
}

// Function to delete a storage
    struct StorageEntry *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
```

```
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}

// Function to display warehouse storage
void displayStorage() {
    struct StorageEntry *temp = first;
    if (temp == NULL) {
        printf("No items in storage.\n");
        return;
    }
    printf("Current Warehouse Storage:\n");
    while (temp != NULL) {
        printf("Item: %s, Quantity: %d, Storage Date: %s\n", temp->itemName, temp->quantity,
temp->storageDate);
        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 Part {
   char partName[100];
   int partID;
   int quantity;
   struct Part *next;
} *first = NULL;


void createInventoryList();
void insertPart(char *partName, int partID, int quantity);
int deletePart(int partID);
void displayInventory();

int main() {
```

```
    int choice, partID, quantity, deleted;
    char partName[100];

    while (1) {

        printf("1. Create Parts Inventory List\n2. Insert New Part\n3. Delete Part That is Used Up or Obsolete\n4. Display Current Parts Inventory\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createInventoryList();
                break;
            case 2:
                printf("Enter part name: ");
                scanf("%s", partName);
                printf("Enter part ID: ");
                scanf("%d", &partID);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                insertPart(partName, partID, quantity);
                break;
            case 3:
                printf("Enter part ID to delete: ");
                scanf("%d", &partID);
                deleted = deletePart(partID);
                if (deleted != -1)
```

```
                printf("Part with ID %d deleted.\n", partID);
            break;
        case 4:
            displayInventory();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new parts inventory list
void createInventoryList() {
   first = NULL;
   printf("New parts inventory list created.\n");
}

// Function to insert a new part into the inventory list
void insertPart(char *partName, int partID, int quantity) {
   struct Part *newPart = (struct Part*)malloc(sizeof(struct Part));
   strcpy(newPart->partName, partName);
   newPart->partID = partID;
   newPart->quantity = quantity;
```

```
    newPart->next = NULL;

    if (first == NULL) {
        first = newPart;
    } else {
        struct Part *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPart;
    }
    printf("Part %s with ID %d added to the inventory.\n", partName, partID);
}

// Function to delete a part that is used up or obsolete
int deletePart(int partID) {
    struct Part *temp = first, *prev = NULL;

    if (temp != NULL && temp->partID == partID) {
        first = temp->next;
        free(temp);
        return partID;
    } else {
        while (temp != NULL && temp->partID != partID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
```

```
            printf("Part with ID %d not found.\n", partID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return partID;
    }
}

// Function to display the current parts inventory
void displayInventory() {
    struct Part *temp = first;
    if (temp == NULL) {
        printf("No parts in the inventory.\n");
        return;
    }
    printf("Current Parts Inventory:\n");
    while (temp != NULL) {
        printf("Part ID: %d, Name: %s, Quantity: %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 PackagingTask {
   char taskName[100];
   char dueDate[20];
   struct PackagingTask *next;
} *first = NULL;


void createSchedule();
void insertTask(char *taskName, char *dueDate);
int deleteTask(int index);
void displaySchedule();
```

```
int main() {
    int choice, index, deleted;
    char taskName[100], dueDate[20];

    while (1) {

        printf("1. Create Packaging Task Schedule\n2. Insert New Packaging Task\n3. Delete Completed Packaging Task\n4. Display Current Packaging Schedule\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createSchedule();
                break;
            case 2:
                printf("Enter task name: ");
                scanf("%s", taskName);
                printf("Enter due date (DD/MM/YYYY): ");
                scanf("%s", dueDate);
                insertTask(taskName, dueDate);
                break;
            case 3:
                printf("Enter task index to delete: ");
                scanf("%d", &index);
                deleted = deleteTask(index);
                if (deleted != -1)
                    printf("Task at index %d deleted.\n", index);
```

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

   return 0;
}



void createSchedule() {
   first = NULL;
   printf("New packaging task schedule created.\n");
}



void insertTask(char *taskName, char *dueDate) {
   struct PackagingTask *newTask = (struct PackagingTask*)malloc(sizeof(struct
PackagingTask));
   strcpy(newTask->taskName, taskName);
   strcpy(newTask->dueDate, dueDate);
   newTask->next = NULL;
```

```
    if (first == NULL) {
        first = newTask;
    } else {
        struct PackagingTask *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("Task %s added to the schedule.\n", taskName);
}



int deleteTask(int index) {
    struct PackagingTask *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
```

```
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}



void displaySchedule() {
    struct PackagingTask *temp = first;
    if (temp == NULL) {
        printf("No tasks in the schedule.\n");
        return;
    }
    printf("Current Packaging Schedule:\n");
    while (temp != NULL) {
        printf("Task: %s, Due Date: %s\n", temp->taskName, temp->dueDate);
        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.

*/


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


struct Defect {
   char defectDescription[200];
   int defectID;
   char reportedDate[20];
   struct Defect *next;
} *first = NULL;

void createDefectList();
void insertDefect(char *defectDescription, int defectID, char *reportedDate);
int deleteDefect(int defectID);
void displayDefects();
```

```
int main() {
    int choice, defectID, deleted;
    char defectDescription[200], reportedDate[20];

    while (1) {

        printf("1. Create Defect Tracking List\n2. Insert New Defect Report\n3. Delete Resolved Defect\n4. Display All Current Defects\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createDefectList();
                break;
            case 2:
                printf("Enter defect description: ");
                scanf(" %[^\n]s", defectDescription);
                printf("Enter defect ID: ");
                scanf("%d", &defectID);
                printf("Enter reported date (DD/MM/YYYY): ");
                scanf("%s", reportedDate);
                insertDefect(defectDescription, defectID, reportedDate);
                break;
            case 3:
                printf("Enter defect ID to delete: ");
                scanf("%d", &defectID);
```

```
            deleted = deleteDefect(defectID);
            if (deleted != -1)
                printf("Defect with ID %d deleted.\n", defectID);
            break;
        case 4:
            displayDefects();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new defect tracking list
void createDefectList() {
   first = NULL;
   printf("New defect tracking list created.\n");
}

// Function to insert a new defect report
void insertDefect(char *defectDescription, int defectID, char *reportedDate) {
   struct Defect *newDefect = (struct Defect*)malloc(sizeof(struct Defect));
   strcpy(newDefect->defectDescription, defectDescription);
```

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

    if (first == NULL) {
        first = newDefect;
    } else {
        struct Defect *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newDefect;
    }
    printf("Defect %d: %s added to the list.\n", defectID, defectDescription);
}

// Function to delete a resolved defect
int deleteDefect(int defectID) {
    struct Defect *temp = first, *prev = NULL;

    if (temp != NULL && temp->defectID == defectID) {
        first = temp->next;
        free(temp);
        return defectID;
    } else {
        while (temp != NULL && temp->defectID != defectID) {
            prev = temp;
            temp = temp->next;
```

```
        }
        if (temp == NULL) {
            printf("Defect with ID %d not found.\n", defectID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return defectID;
    }
}

// Function to display all current defects
void displayDefects() {
    struct Defect *temp = first;
    if (temp == NULL) {
        printf("No defects reported.\n");
        return;
    }
    printf("Current Defects:\n");
    while (temp != NULL) {
        printf("Defect ID: %d, Description: %s, Reported Date: %s\n", temp->defectID, temp->defectDescription, temp->reportedDate);
        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 DispatchEntry {
    char productName[100];
    int quantity;
    char dispatchDate[20];
    struct DispatchEntry *next;
} *first = NULL;


void createDispatchSchedule();
void insertDispatchEntry(char *productName, int quantity, char *dispatchDate);
int deleteDispatchEntry(int index);
void displayDispatchSchedule();
```

```
int main() {
    int choice, index, quantity, deleted;
    char productName[100], dispatchDate[20];

    while (1) {

        printf("1. Create Dispatch Schedule\n2. Insert New Dispatch Entry\n3. Delete Dispatched or Canceled Entry\n4. Display Current Dispatch Schedule\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createDispatchSchedule();
                break;
            case 2:
                printf("Enter product name: ");
                scanf("%s", productName);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                printf("Enter dispatch date (DD/MM/YYYY): ");
                scanf("%s", dispatchDate);
                insertDispatchEntry(productName, quantity, dispatchDate);
                break;
            case 3:
                printf("Enter dispatch entry index to delete: ");
                scanf("%d", &index);
                deleted = deleteDispatchEntry(index);
```

```
            if (deleted != -1)
                printf("Dispatch entry at index %d deleted.\n", index);
            break;
        case 4:
            displayDispatchSchedule();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new dispatch schedule
void createDispatchSchedule() {
   first = NULL;
   printf("New dispatch schedule created.\n");
}

// Function to insert a new dispatch entry
void insertDispatchEntry(char *productName, int quantity, char *dispatchDate) {
   struct DispatchEntry *newEntry = (struct DispatchEntry*)malloc(sizeof(struct
DispatchEntry));
   strcpy(newEntry->productName, productName);
```

```
    newEntry->quantity = quantity;
    strcpy(newEntry->dispatchDate, dispatchDate);
    newEntry->next = NULL;

    if (first == NULL) {
        first = newEntry;
    } else {
        struct DispatchEntry *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEntry;
    }
    printf("Dispatch entry for product %s added to the schedule.\n", productName);
}

// Function to delete a dispatched or canceled entry
int deleteDispatchEntry(int index) {
    struct DispatchEntry *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
```

```
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}

// Function to display the current dispatch schedule
void displayDispatchSchedule() {
    struct DispatchEntry *temp = first;
    if (temp == NULL) {
        printf("No dispatch entries in the schedule.\n");
        return;
    }
    printf("Current Dispatch Schedule:\n");
    while (temp != NULL) {
        printf("Product: %s, Quantity: %d, Dispatch Date: %s\n", temp->productName, temp->quantity, temp->dispatchDate);
        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 Player {
   char playerName[100];
   int playerID;
   struct Player *next;
} *first = NULL;


void createTeamRoster();
void insertPlayer(char *playerName, int playerID);
int deletePlayer(int playerID);
void displayRoster();
```

```
int main() {
    int choice, playerID, deleted;
    char playerName[100];

    while (1) {

        printf("1. Create Team Roster\n2. Insert New Player\n3. Delete Player Who Leaves the Team\n4. Display Current Team Roster\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createTeamRoster();
                break;
            case 2:
                printf("Enter player name: ");
                scanf("%s", playerName);
                printf("Enter player ID: ");
                scanf("%d", &playerID);
                insertPlayer(playerName, playerID);
                break;
            case 3:
                printf("Enter player ID to delete: ");
                scanf("%d", &playerID);
                deleted = deletePlayer(playerID);
                if (deleted != -1)
                    printf("Player with ID %d deleted.\n", playerID);
```

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

   return 0;
}

// Function to create a new team roster
void createTeamRoster() {
   first = NULL;
   printf("New team roster created.\n");
}



void insertPlayer(char *playerName, int playerID) {
   struct Player *newPlayer = (struct Player*)malloc(sizeof(struct Player));
   strcpy(newPlayer->playerName, playerName);
   newPlayer->playerID = playerID;
   newPlayer->next = NULL;
```

```
    if (first == NULL) {
        first = newPlayer;
    } else {
        struct Player *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPlayer;
    }
    printf("Player %s with ID %d added to the roster.\n", playerName, playerID);
}

// Function to delete a player who leaves the team
int deletePlayer(int playerID) {
    struct Player *temp = first, *prev = NULL;

    if (temp != NULL && temp->playerID == playerID) {
        first = temp->next;
        free(temp);
        return playerID;
    } else {
        while (temp != NULL && temp->playerID != playerID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Player with ID %d not found.\n", playerID);
            return -1;
```

```
        }
        prev->next = temp->next;
        free(temp);
        return playerID;
    }
}

// Function to display the current team roster
void displayRoster() {
    struct Player *temp = first;
    if (temp == NULL) {
        printf("No players in the roster.\n");
        return;
    }
    printf("Current Team Roster:\n");
    while (temp != NULL) {
        printf("Player ID: %d, Name: %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 Match {
    char team1[50];
    char team2[50];
    char matchDate[20];
    char matchTime[10];
    int matchID;
    struct Match *next;
} *first = NULL;



void createMatchSchedule();
```

```
void insertMatch(char *team1, char *team2, char *matchDate, char *matchTime, int matchID);

int deleteMatch(int matchID);

void displaySchedule();


int main() {
   int choice, matchID, deleted;
   char team1[50], team2[50], matchDate[20], matchTime[10];


   while (1) {


      printf("1. Create Match Schedule\n2. Insert New Match\n3. Delete Completed or Canceled Match\n4. Display Current Match Schedule\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);


      switch (choice) {
         case 1:
            createMatchSchedule();
            break;
         case 2:
            printf("Enter team 1 name: ");
            scanf("%s", team1);
            printf("Enter team 2 name: ");
            scanf("%s", team2);
            printf("Enter match date (DD/MM/YYYY): ");
            scanf("%s", matchDate);
            printf("Enter match time (HH:MM): ");
            scanf("%s", matchTime);
```

```
            printf("Enter match ID: ");
            scanf("%d", &matchID);
            insertMatch(team1, team2, matchDate, matchTime, matchID);
            break;
        case 3:
            printf("Enter match ID to delete: ");
            scanf("%d", &matchID);
            deleted = deleteMatch(matchID);
            if (deleted != -1)
                printf("Match with ID %d deleted.\n", matchID);
            break;
        case 4:
            displaySchedule();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new match schedule
void createMatchSchedule() {
   first = NULL;
```

```
    printf("New match schedule created.\n");
}

// Function to insert a new match
void insertMatch(char *team1, char *team2, char *matchDate, char *matchTime, int matchID) {
    struct Match *newMatch = (struct Match*)malloc(sizeof(struct Match));
    strcpy(newMatch->team1, team1);
    strcpy(newMatch->team2, team2);
    strcpy(newMatch->matchDate, matchDate);
    strcpy(newMatch->matchTime, matchTime);
    newMatch->matchID = matchID;
    newMatch->next = NULL;

    if (first == NULL) {
        first = newMatch;
    } else {
        struct Match *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newMatch;
    }
    printf("Match %d: %s vs %s on %s at %s added to the schedule.\n", matchID, team1, team2, matchDate, matchTime);
}

// Function to delete a completed or canceled match
int deleteMatch(int matchID) {
```

```
    struct Match *temp = first, *prev = NULL;

    if (temp != NULL && temp->matchID == matchID) {
        first = temp->next;
        free(temp);
        return matchID;
    } else {
        while (temp != NULL && temp->matchID != matchID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Match with ID %d not found.\n", matchID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return matchID;
    }
}

// Function to display the current match schedule
void displaySchedule() {
    struct Match *temp = first;
    if (temp == NULL) {
        printf("No matches scheduled.\n");
        return;
    }
```

```
    printf("Current Match Schedule:\n");
    while (temp != NULL) {
        printf("Match ID: %d, Teams: %s vs %s, Date: %s, Time: %s\n", temp->matchID, temp->team1, temp->team2, temp->matchDate, temp->matchTime);
        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 {
    char sessionName[100];
    char athleteName[100];
```

```
    char trainingDate[20];
    char duration[10];
    struct TrainingSession *next;
} *first = NULL;

// Function prototypes
void createTrainingLog();
void insertTrainingSession(char *sessionName, char *athleteName, char *trainingDate, char *duration);
int deleteTrainingSession(int index);
void displayTrainingLog();

int main() {
    int choice, index, deleted;
    char sessionName[100], athleteName[100], trainingDate[20], duration[10];

    while (1) {
        printf("\nAthlete Training Log\n");
        printf("1. Create Training Log\n2. Insert New Training Session\n3. Delete Completed or Canceled Session\n4. Display Training Log\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createTrainingLog();
                break;
            case 2:
                printf("Enter session name: ");
```

```
            scanf("%s", sessionName);
            printf("Enter athlete name: ");
            scanf("%s", athleteName);
            printf("Enter training date (DD/MM/YYYY): ");
            scanf("%s", trainingDate);
            printf("Enter duration (HH:MM): ");
            scanf("%s", duration);
            insertTrainingSession(sessionName, athleteName, trainingDate, duration);
            break;
        case 3:
            printf("Enter session index to delete: ");
            scanf("%d", &index);
            deleted = deleteTrainingSession(index);
            if (deleted != -1)
                printf("Training session at index %d deleted.\n", index);
            break;
        case 4:
            displayTrainingLog();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

return 0;
```

```
}

// Function to create a new training log
void createTrainingLog() {
    first = NULL;
    printf("New training log created.\n");
}

// Function to insert a new training session
void insertTrainingSession(char *sessionName, char *athleteName, char *trainingDate, char *duration) {
    struct TrainingSession *newSession = (struct TrainingSession*)malloc(sizeof(struct TrainingSession));
    strcpy(newSession->sessionName, sessionName);
    strcpy(newSession->athleteName, athleteName);
    strcpy(newSession->trainingDate, trainingDate);
    strcpy(newSession->duration, duration);
    newSession->next = NULL;

    if (first == NULL) {
        first = newSession;
    } else {
        struct TrainingSession *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newSession;
    }
```

```
    printf("Training session %s for athlete %s added to the log.\n", sessionName,
athleteName);
}

// Function to delete a completed or canceled training session
int deleteTrainingSession(int index) {
    struct TrainingSession *temp = first, *prev = NULL;
    int i;

    if (index == 0) {
        first = temp->next;
        free(temp);
        return 1;
    } else {
        for (i = 0; temp != NULL && i < index - 1; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index.\n");
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return 1;
    }
}
```

```
// Function to display the training log
void displayTrainingLog() {
    struct TrainingSession *temp = first;
    if (temp == NULL) {
        printf("No training sessions logged.\n");
        return;
    }
    printf("Current Training Log:\n");
    while (temp != NULL) {
        printf("Session: %s, Athlete: %s, Date: %s, Duration: %s\n", temp->sessionName, temp->athleteName, temp->trainingDate, temp->duration);
        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 {
   char itemName[100];
   int itemID;
   int quantity;
   struct Equipment *next;
} *first = NULL;


void createInventoryList();
void insertEquipmentItem(char *itemName, int itemID, int quantity);
int deleteEquipmentItem(int itemID);
void displayInventory();

int main() {
```

```
    int choice, itemID, quantity, deleted;
    char itemName[100];

    while (1) {

        printf("1. Create Equipment Inventory List\n2. Insert New Equipment Item\n3. Delete Item That is No Longer Usable\n4. Display Current Equipment Inventory\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createInventoryList();
                break;
            case 2:
                printf("Enter item name: ");
                scanf("%s", itemName);
                printf("Enter item ID: ");
                scanf("%d", &itemID);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                insertEquipmentItem(itemName, itemID, quantity);
                break;
            case 3:
                printf("Enter item ID to delete: ");
                scanf("%d", &itemID);
                deleted = deleteEquipmentItem(itemID);
                if (deleted != -1)
```

```
                printf("Item with ID %d deleted.\n", itemID);
            break;
        case 4:
            displayInventory();
            break;
        case 5:
            printf("Exiting \n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new equipment inventory list
void createInventoryList() {
   first = NULL;
   printf("New equipment inventory list created.\n");
}

// Function to insert a new equipment item
void insertEquipmentItem(char *itemName, int itemID, int quantity) {
   struct Equipment *newItem = (struct Equipment*)malloc(sizeof(struct Equipment));
   strcpy(newItem->itemName, itemName);
   newItem->itemID = itemID;
   newItem->quantity = quantity;
```

```
    newItem->next = NULL;

    if (first == NULL) {
        first = newItem;
    } else {
        struct Equipment *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("Equipment item %s with ID %d added to the inventory.\n", itemName, itemID);
}

// Function to delete an equipment item that is no longer usable
int deleteEquipmentItem(int itemID) {
    struct Equipment *temp = first, *prev = NULL;

    if (temp != NULL && temp->itemID == itemID) {
        first = temp->next;
        free(temp);
        return itemID;
    } else {
        while (temp != NULL && temp->itemID != itemID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
```

```
            printf("Item with ID %d not found.\n", itemID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return itemID;
    }
}

// Function to display the current equipment inventory
void displayInventory() {
    struct Equipment *temp = first;
    if (temp == NULL) {
        printf("No equipment items in the inventory.\n");
        return;
    }
    printf("Current Equipment Inventory:\n");
    while (temp != NULL) {
        printf("Item ID: %d, Name: %s, Quantity: %d\n", temp->itemID, temp->itemName,
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 {
   char playerName[100];
   int playerID;
   int matchesPlayed;
   int goalsScored;
   struct Performance *next;
} *first = NULL;



void createPerformanceList();
void insertPerformanceEntry(char *playerName, int playerID, int matchesPlayed, int
goalsScored);
int deletePerformanceEntry(int playerID);
```

```
void displayPerformanceRecords();

int main() {
    int choice, playerID, matchesPlayed, goalsScored, deleted;
    char playerName[100];

    while (1) {

        printf("1. Create Performance Record List\n2. Insert New Performance Entry\n3. Delete Outdated or Erroneous Entry\n4. Display All Performance Records\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createPerformanceList();
                break;
            case 2:
                printf("Enter player name: ");
                scanf("%s", playerName);
                printf("Enter player ID: ");
                scanf("%d", &playerID);
                printf("Enter matches played: ");
                scanf("%d", &matchesPlayed);
                printf("Enter goals scored: ");
                scanf("%d", &goalsScored);
                insertPerformanceEntry(playerName, playerID, matchesPlayed, goalsScored);
                break;
```

```
            case 3:
                printf("Enter player ID to delete: ");
                scanf("%d", &playerID);
                deleted = deletePerformanceEntry(playerID);
                if (deleted != -1)
                    printf("Performance record for player with ID %d deleted.\n", playerID);
                break;
            case 4:
                displayPerformanceRecords();
                break;
            case 5:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Function to create a new performance record list
void createPerformanceList() {
    first = NULL;
    printf("New performance record list created.\n");
}

// Function to insert a new performance entry
```

```
void insertPerformanceEntry(char *playerName, int playerID, int matchesPlayed, int
goalsScored) {
    struct Performance *newEntry = (struct Performance*)malloc(sizeof(struct Performance));
    strcpy(newEntry->playerName, playerName);
    newEntry->playerID = playerID;
    newEntry->matchesPlayed = matchesPlayed;
    newEntry->goalsScored = goalsScored;
    newEntry->next = NULL;

    if (first == NULL) {
        first = newEntry;
    } else {
        struct Performance *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEntry;
    }
    printf("Performance entry for player %s added to the list.\n", playerName);
}

// Function to delete an outdated or erroneous performance entry
int deletePerformanceEntry(int playerID) {
    struct Performance *temp = first, *prev = NULL;

    if (temp != NULL && temp->playerID == playerID) {
        first = temp->next;
        free(temp);
```

```
        return playerID;
    } else {
        while (temp != NULL && temp->playerID != playerID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Performance record for player with ID %d not found.\n", playerID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return playerID;
    }
}

// Function to display all performance records
void displayPerformanceRecords() {
    struct Performance *temp = first;
    if (temp == NULL) {
        printf("No performance records available.\n");
        return;
    }
    printf("Current Performance Records:\n");
    while (temp != NULL) {
        printf("Player ID: %d, Name: %s, Matches Played: %d, Goals Scored: %d\n", temp-
>playerID, temp->playerName, temp->matchesPlayed, temp->goalsScored);
        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 {
    char athleteName[100];
    char eventName[100];
    int registrationID;
    struct Registration *next;
} *first = NULL;


void createRegistrationList();
void insertRegistration(char *athleteName, char *eventName, int registrationID);
```

```
int deleteRegistration(int registrationID);
void displayRegistrations();

int main() {
    int choice, registrationID, deleted;
    char athleteName[100], eventName[100];

    while (1) {
        printf("\nEvent Registration System\n");
        printf("1. Create Registration List\n2. Insert New Registration\n3. Delete Canceled Registration\n4. Display All Current Registrations\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createRegistrationList();
                break;
            case 2:
                printf("Enter athlete name: ");
                scanf("%s", athleteName);
                printf("Enter event name: ");
                scanf("%s", eventName);
                printf("Enter registration ID: ");
                scanf("%d", &registrationID);
                insertRegistration(athleteName, eventName, registrationID);
                break;
            case 3:
```

```
            printf("Enter registration ID to delete: ");
            scanf("%d", &registrationID);
            deleted = deleteRegistration(registrationID);
            if (deleted != -1)
                printf("Registration with ID %d deleted.\n", registrationID);
            break;
        case 4:
            displayRegistrations();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new registration list
   first = NULL;
   printf("New registration list created.\n");
}

// Function to insert a new registration
void insertRegistration(char *athleteName, char *eventName, int registrationID) {
```

```
    struct Registration *newRegistration = (struct Registration*)malloc(sizeof(struct
Registration));
    strcpy(newRegistration->athleteName, athleteName);
    strcpy(newRegistration->eventName, eventName);
    newRegistration->registrationID = registrationID;
    newRegistration->next = NULL;

    if (first == NULL) {
        first = newRegistration;
    } else {
        struct Registration *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newRegistration;
    }
    printf("Registration for athlete %s in event %s added to the list.\n", athleteName,
eventName);
}

// Function to delete a canceled registration
int deleteRegistration(int registrationID) {
    struct Registration *temp = first, *prev = NULL;

    if (temp != NULL && temp->registrationID == registrationID) {
        first = temp->next;
        free(temp);
        return registrationID;
    } else {
```

```
        while (temp != NULL && temp->registrationID != registrationID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Registration with ID %d not found.\n", registrationID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return registrationID;
    }
}

// Function to display all current registrations
void displayRegistrations() {
    struct Registration *temp = first;
    if (temp == NULL) {
        printf("No registrations available.\n");
        return;
    }
    printf("Current Registrations:\n");
    while (temp != NULL) {
        printf("Registration ID: %d, Athlete: %s, Event: %s\n", temp->registrationID, temp-
>athleteName, temp->eventName);
        temp = temp->next;
    }
}
```

```
/*
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[100];
    int teamID;
    int points;
    struct Team *next;
} *first = NULL;


void createStandingsList();
void insertTeam(char *teamName, int teamID, int points);
int deleteTeam(int teamID);
void displayStandings();
```

```
int main() {
    int choice, teamID, points, deleted;
    char teamName[100];

    while (1) {

        printf("1. Create League Standings List\n2. Insert New Team\n3. Delete Team That Withdraws\n4. Display Current League Standings\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createStandingsList();
                break;
            case 2:
                printf("Enter team name: ");
                scanf("%s", teamName);
                printf("Enter team ID: ");
                scanf("%d", &teamID);
                printf("Enter points: ");
                scanf("%d", &points);
                insertTeam(teamName, teamID, points);
                break;
            case 3:
                printf("Enter team ID to delete: ");
                scanf("%d", &teamID);
```

```
            deleted = deleteTeam(teamID);
            if (deleted != -1)
                printf("Team with ID %d deleted.\n", teamID);
            break;
        case 4:
            displayStandings();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new league standings list
void createStandingsList() {
   first = NULL;
   printf("New league standings list created.\n");
}

// Function to insert a new team
void insertTeam(char *teamName, int teamID, int points) {
   struct Team *newTeam = (struct Team*)malloc(sizeof(struct Team));
   strcpy(newTeam->teamName, teamName);
```

```
    newTeam->teamID = teamID;
    newTeam->points = points;
    newTeam->next = NULL;

    if (first == NULL) {
        first = newTeam;
    } else {
        struct Team *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTeam;
    }
    printf("Team %s with ID %d and %d points added to the standings list.\n", teamName,
teamID, points);
}

// Function to delete a team that withdraw
int deleteTeam(int teamID) {
    struct Team *temp = first, *prev = NULL;

    if (temp != NULL && temp->teamID == teamID) {
        first = temp->next;
        free(temp);
        return teamID;
    } else {
        while (temp != NULL && temp->teamID != teamID) {
            prev = temp;
```

```
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Team with ID %d not found.\n", teamID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return teamID;
    }
}

// Function to display the current league standings
void displayStandings() {
    struct Team *temp = first;
    if (temp == NULL) {
        printf("No teams in the standings.\n");
        return;
    }
    printf("Current League Standings:\n");
    while (temp != NULL) {
        printf("Team ID: %d, Name: %s, Points: %d\n", temp->teamID, temp->teamName, temp->points);
        temp = temp->next;
    }
}
```

```
/*
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 MatchResult {
    char team1[50];
    char team2[50];
    int team1Score;
    int team2Score;
    char matchDate[20];
    int matchID;
    struct MatchResult *next;
} *first = NULL;

void createResultList();
void insertMatchResult(char *team1, char *team2, int team1Score, int team2Score, char
*matchDate, int matchID);
```

```
int deleteMatchResult(int matchID);
void displayMatchResults();

int main() {
    int choice, matchID, team1Score, team2Score, deleted;
    char team1[50], team2[50], matchDate[20];

    while (1) {

        printf("1. Create Match Result List\n2. Insert New Match Result\n3. Delete Incorrect or Outdated Result\n4. Display All Recorded Match Results\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createResultList();
                break;
            case 2:
                printf("Enter team 1 name: ");
                scanf("%s", team1);
                printf("Enter team 2 name: ");
                scanf("%s", team2);
                printf("Enter team 1 score: ");
                scanf("%d", &team1Score);
                printf("Enter team 2 score: ");
                scanf("%d", &team2Score);
                printf("Enter match date (DD/MM/YYYY): ");
```

```
            scanf("%s", matchDate);
            printf("Enter match ID: ");
            scanf("%d", &matchID);
            insertMatchResult(team1, team2, team1Score, team2Score, matchDate, matchID);
            break;
        case 3:
            printf("Enter match ID to delete: ");
            scanf("%d", &matchID);
            deleted = deleteMatchResult(matchID);
            if (deleted != -1)
                printf("Match result with ID %d deleted.\n", matchID);
            break;
        case 4:
            displayMatchResults();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new match result list
void createResultList() {
```

```
    first = NULL;
    printf("New match result list created.\n");
}

// Function to insert a new match result
void insertMatchResult(char *team1, char *team2, int team1Score, int team2Score, char
*matchDate, int matchID) {
    struct MatchResult *newResult = (struct MatchResult*)malloc(sizeof(struct MatchResult));
    strcpy(newResult->team1, team1);
    strcpy(newResult->team2, team2);
    newResult->team1Score = team1Score;
    newResult->team2Score = team2Score;
    strcpy(newResult->matchDate, matchDate);
    newResult->matchID = matchID;
    newResult->next = NULL;

    if (first == NULL) {
        first = newResult;
    } else {
        struct MatchResult *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newResult;
    }
    printf("Match result for %s vs %s added to the list.\n", team1, team2);
}
```

```
// Function to delete an incorrect or outdated match result
int deleteMatchResult(int matchID) {
    struct MatchResult *temp = first, *prev = NULL;

    if (temp != NULL && temp->matchID == matchID) {
        first = temp->next;
        free(temp);
        return matchID;
    } else {
        while (temp != NULL && temp->matchID != matchID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Match result with ID %d not found.\n", matchID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return matchID;
    }
}

// Function to display all recorded match results
void displayMatchResults() {
    struct MatchResult *temp = first;
    if (temp == NULL) {
        printf("No match results recorded.\n");
```

```
        return;
    }
    printf("Recorded Match Results:\n");
    while (temp != NULL) {
        printf("Match ID: %d, Teams: %s vs %s, Score: %d - %d, Date: %s\n",
            temp->matchID, temp->team1, temp->team2, temp->team1Score, temp->team2Score, temp->matchDate);
        temp = temp->next;
    }
}
```

```
/*
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 InjuryReport {
    char playerName[100];
    char injuryType[100];
    char reportDate[20];
    int reportID;
    struct InjuryReport *next;
} *first = NULL;



void createInjuryTracker();
void insertInjuryReport(char *playerName, char *injuryType, char *reportDate, int reportID);
int deleteInjuryReport(int reportID);
void displayInjuryReports();

int main() {
    int choice, reportID, deleted;
    char playerName[100], injuryType[100], reportDate[20];

    while (1) {
        printf("\nPlayer Injury Tracker System\n");
        printf("1. Create Injury Tracker List\n2. Insert New Injury Report\n3. Delete Resolved or
Erroneous Injury Report\n4. Display All Current Injury Reports\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            break;
        case 2:
            printf("Enter player name: ");
            scanf("%s", playerName);
            printf("Enter injury type: ");
            scanf("%s", injuryType);
            printf("Enter report date (DD/MM/YYYY): ");
            scanf("%s", reportDate);
            printf("Enter report ID: ");
            scanf("%d", &reportID);
            insertInjuryReport(playerName, injuryType, reportDate, reportID);
            break;
        case 3:
            printf("Enter report ID to delete: ");
            scanf("%d", &reportID);
            deleted = deleteInjuryReport(reportID);
            if (deleted != -1)
                printf("Injury report with ID %d deleted.\n", reportID);
            break;
        case 4:
            displayInjuryReports();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
```

```
    }

    return 0;
}

// Function to create a new injury tracker list
void createInjuryTracker() {
    first = NULL;
    printf("New injury tracker list created.\n");
}

// Function to insert a new injury report
void insertInjuryReport(char *playerName, char *injuryType, char *reportDate, int reportID)
{
    struct InjuryReport *newReport = (struct InjuryReport*)malloc(sizeof(struct
InjuryReport));
    strcpy(newReport->playerName, playerName);
    strcpy(newReport->injuryType, injuryType);
    strcpy(newReport->reportDate, reportDate);
    newReport->reportID = reportID;
    newReport->next = NULL;

    if (first == NULL) {
        first = newReport;
    } else {
        struct InjuryReport *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
```

```
        temp->next = newReport;
    }
    printf("Injury report for player %s added to the list.\n", playerName);
}

// Function to delete a resolved or erroneous injury report
int deleteInjuryReport(int reportID) {
    struct InjuryReport *temp = first, *prev = NULL;

    if (temp != NULL && temp->reportID == reportID) {
        first = temp->next;
        free(temp);
        return reportID;
    } else {
        while (temp != NULL && temp->reportID != reportID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Injury report with ID %d not found.\n", reportID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return reportID;
    }
}
```

```
// Function to display all current injury reports
void displayInjuryReports() {
    struct InjuryReport *temp = first;
    if (temp == NULL) {
        printf("No injury reports available.\n");
        return;
    }
    printf("Current Injury Reports:\n");
    while (temp != NULL) {
        printf("Report ID: %d, Player: %s, Injury Type: %s, Date: %s\n",
            temp->reportID, temp->playerName, temp->injuryType, temp->reportDate);
        temp = temp->next;
    }
}
```

```
/*
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[100];
   char userName[100];
   char bookingDate[20];
   char bookingTime[10];
   int bookingID;
   struct Booking *next;
} *first = NULL;


void createBookingList();
void insertBooking(char *facilityName, char *userName, char *bookingDate, char
*bookingTime, int bookingID);
int deleteBooking(int bookingID);
```

```
void displayBookings();

int main() {
    int choice, bookingID, deleted;
    char facilityName[100], userName[100], bookingDate[20], bookingTime[10];

    while (1) {

        printf("1. Create Booking List\n2. Insert New Booking\n3. Delete Canceled or Completed Booking\n4. Display All Current Bookings\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createBookingList();
                break;
            case 2:
                printf("Enter facility name: ");
                scanf("%s", facilityName);
                printf("Enter user name: ");
                scanf("%s", userName);
                printf("Enter booking date (DD/MM/YYYY): ");
                scanf("%s", bookingDate);
                printf("Enter booking time (HH:MM): ");
                scanf("%s", bookingTime);
                printf("Enter booking ID: ");
                scanf("%d", &bookingID);
```

```
            insertBooking(facilityName, userName, bookingDate, bookingTime, bookingID);
            break;
        case 3:
            printf("Enter booking ID to delete: ");
            scanf("%d", &bookingID);
            deleted = deleteBooking(bookingID);
            if (deleted != -1)
                printf("Booking with ID %d deleted.\n", bookingID);
            break;
        case 4:
            displayBookings();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new booking list
void createBookingList() {
   first = NULL;
   printf("New booking list created.\n");
}
```

```
// Function to insert a new booking
void insertBooking(char *facilityName, char *userName, char *bookingDate, char *bookingTime, int bookingID) {
    struct Booking *newBooking = (struct Booking*)malloc(sizeof(struct Booking));
    strcpy(newBooking->facilityName, facilityName);
    strcpy(newBooking->userName, userName);
    strcpy(newBooking->bookingDate, bookingDate);
    strcpy(newBooking->bookingTime, bookingTime);
    newBooking->bookingID = bookingID;
    newBooking->next = NULL;

    if (first == NULL) {
        first = newBooking;
    } else {
        struct Booking *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newBooking;
    }
    printf("Booking for facility %s by user %s added to the list.\n", facilityName, userName);
}

// Function to delete a canceled or completed booking
int deleteBooking(int bookingID) {
    struct Booking *temp = first, *prev = NULL;
```

```
    if (temp != NULL && temp->bookingID == bookingID) {
        first = temp->next;
        free(temp);
        return bookingID;
    } else {
        while (temp != NULL && temp->bookingID != bookingID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Booking with ID %d not found.\n", bookingID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return bookingID;
    }
}

// Function to display all current bookings
void displayBookings() {
    struct Booking *temp = first;
    if (temp == NULL) {
        printf("No bookings available.\n");
        return;
    }
    printf("Current Bookings:\n");
    while (temp != NULL) {
```

```
        printf("Booking ID: %d, Facility: %s, User: %s, Date: %s, Time: %s\n",
            temp->bookingID, temp->facilityName, temp->userName, temp->bookingDate, temp->bookingTime);
        temp = temp->next;
    }
}

/*
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[100];
    int coachID;
    char position[100];
    struct Coach *next;
} *first = NULL;
```

```
void createCoachingStaff();
void insertCoach(char *coachName, int coachID, char *position);
int deleteCoach(int coachID);
void displayCoachingStaff();

int main() {
   int choice, coachID, deleted;
   char coachName[100], position[100];

   while (1) {

      printf("1. Create Coaching Staff List\n2. Insert New Coach\n3. Delete Coach Who Leaves the Team\n4. Display Current Coaching Staff\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            createCoachingStaff();
            break;
         case 2:
            printf("Enter coach name: ");
            scanf("%s", coachName);
            printf("Enter coach ID: ");
            scanf("%d", &coachID);
            printf("Enter position: ");
            scanf("%s", position);
```

```
            insertCoach(coachName, coachID, position);
            break;
        case 3:
            printf("Enter coach ID to delete: ");
            scanf("%d", &coachID);
            deleted = deleteCoach(coachID);
            if (deleted != -1)
                printf("Coach with ID %d deleted.\n", coachID);
            break;
        case 4:
            displayCoachingStaff();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new coaching staff list
void createCoachingStaff() {
   first = NULL;
   printf("New coaching staff list created.\n");
}
```

```
// Function to insert a new coach
void insertCoach(char *coachName, int coachID, char *position) {
    struct Coach *newCoach = (struct Coach*)malloc(sizeof(struct Coach));
    strcpy(newCoach->coachName, coachName);
    newCoach->coachID = coachID;
    strcpy(newCoach->position, position);
    newCoach->next = NULL;

    if (first == NULL) {
        first = newCoach;
    } else {
        struct Coach *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newCoach;
    }
    printf("Coach %s with ID %d and position %s added to the list.\n", coachName, coachID,
position);
}

// Function to delete a coach who leaves the team
int deleteCoach(int coachID) {
    struct Coach *temp = first, *prev = NULL;

    if (temp != NULL && temp->coachID == coachID) {
        first = temp->next;
```

```
        free(temp);
        return coachID;
    } else {
        while (temp != NULL && temp->coachID != coachID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Coach with ID %d not found.\n", coachID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return coachID;
    }
}

// Function to display the current coaching staff
void displayCoachingStaff() {
    struct Coach *temp = first;
    if (temp == NULL) {
        printf("No coaches in the staff.\n");
        return;
    }
    printf("Current Coaching Staff:\n");
    while (temp != NULL) {
        printf("Coach ID: %d, Name: %s, Position: %s\n", temp->coachID, temp->coachName,
temp->position);
```

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

/*
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[100];
    int memberID;
    char joinDate[20];
    struct Member *next;
} *first = NULL;


void createMembershipList();
```

```
void insertMember(char *memberName, int memberID, char *joinDate);
int deleteMember(int memberID);
void displayMembers();

int main() {
   int choice, memberID, deleted;
   char memberName[100], joinDate[20];

   while (1) {

      printf("1. Create Membership List\n2. Insert New Member\n3. Delete Member Who Cancels Membership\n4. Display All Current Members\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            createMembershipList();
            break;
         case 2:
            printf("Enter member name: ");
            scanf("%s", memberName);
            printf("Enter member ID: ");
            scanf("%d", &memberID);
            printf("Enter join date (DD/MM/YYYY): ");
            scanf("%s", joinDate);
            insertMember(memberName, memberID, joinDate);
            break;
```

```
            case 3:
                printf("Enter member ID to delete: ");
                scanf("%d", &memberID);
                deleted = deleteMember(memberID);
                if (deleted != -1)
                    printf("Member with ID %d deleted.\n", memberID);
                break;
            case 4:
                displayMembers();
                break;
            case 5:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Function to create a new membership list
void createMembershipList() {
    first = NULL;
    printf("New membership list created.\n");
}

// Function to insert a new member
```

```
void insertMember(char *memberName, int memberID, char *joinDate) {
    struct Member *newMember = (struct Member*)malloc(sizeof(struct Member));
    strcpy(newMember->memberName, memberName);
    newMember->memberID = memberID;
    strcpy(newMember->joinDate, joinDate);
    newMember->next = NULL;

    if (first == NULL) {
        first = newMember;
    } else {
        struct Member *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newMember;
    }
    printf("Member %s with ID %d added to the list.\n", memberName, memberID);
}

// Function to delete a member who cancels their membership
int deleteMember(int memberID) {
    struct Member *temp = first, *prev = NULL;

    if (temp != NULL && temp->memberID == memberID) {
        first = temp->next;
        free(temp);
        return memberID;
    } else {
```

```
        while (temp != NULL && temp->memberID != memberID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Member with ID %d not found.\n", memberID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return memberID;
    }
}

// Function to display all current members
void displayMembers() {
    struct Member *temp = first;
    if (temp == NULL) {
        printf("No members in the list.\n");
        return;
    }
    printf("Current Members:\n");
    while (temp != NULL) {
        printf("Member ID: %d, Name: %s, Join Date: %s\n", temp->memberID, temp-
>memberName, temp->joinDate);
        temp = temp->next;
    }
}
```

```
/*
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[100];
    char eventDate[20];
    char eventTime[10];
    int eventID;
    struct Event *next;
} *first = NULL;


void createEventSchedule();
void insertEvent(char *eventName, char *eventDate, char *eventTime, int eventID);
int deleteEvent(int eventID);
void displayEvents();
```

```
int main() {
    int choice, eventID, deleted;
    char eventName[100], eventDate[20], eventTime[10];

    while (1) {

        printf("1. Create Event Schedule\n2. Insert New Event\n3. Delete Completed or Canceled Event\n4. Display Current Event Schedule\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createEventSchedule();
                break;
            case 2:
                printf("Enter event name: ");
                scanf("%s", eventName);
                printf("Enter event date (DD/MM/YYYY): ");
                scanf("%s", eventDate);
                printf("Enter event time (HH:MM): ");
                scanf("%s", eventTime);
                printf("Enter event ID: ");
                scanf("%d", &eventID);
                insertEvent(eventName, eventDate, eventTime, eventID);
                break;
            case 3:
```

```
            printf("Enter event ID to delete: ");
            scanf("%d", &eventID);
            deleted = deleteEvent(eventID);
            if (deleted != -1)
                printf("Event with ID %d deleted.\n", eventID);
            break;
        case 4:
            displayEvents();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new event schedule
void createEventSchedule() {
   first = NULL;
   printf("New event schedule created.\n");
}

// Function to insert a new event
void insertEvent(char *eventName, char *eventDate, char *eventTime, int eventID) {
```

```
    struct Event *newEvent = (struct Event*)malloc(sizeof(struct Event));
    strcpy(newEvent->eventName, eventName);
    strcpy(newEvent->eventDate, eventDate);
    strcpy(newEvent->eventTime, eventTime);
    newEvent->eventID = eventID;
    newEvent->next = NULL;

    if (first == NULL) {
        first = newEvent;
    } else {
        struct Event *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEvent;
    }
    printf("Event %s with ID %d added to the schedule.\n", eventName, eventID);
}

// Function to delete a completed or canceled event
int deleteEvent(int eventID) {
    struct Event *temp = first, *prev = NULL;

    if (temp != NULL && temp->eventID == eventID) {
        first = temp->next;
        free(temp);
        return eventID;
    } else {
```

```
        while (temp != NULL && temp->eventID != eventID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Event with ID %d not found.\n", eventID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return eventID;
    }
}

// Function to display the current event schedule
void displayEvents() {
    struct Event *temp = first;
    if (temp == NULL) {
        printf("No events scheduled.\n");
        return;
    }
    printf("Current Event Schedule:\n");
    while (temp != NULL) {
        printf("Event ID: %d, Name: %s, Date: %s, Time: %s\n",
            temp->eventID, temp->eventName, temp->eventDate, temp->eventTime);
        temp = temp->next;
    }
}
```

```
/*
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 TransferRecord {
    char playerName[100];
    char fromTeam[100];
    char toTeam[100];
    char transferDate[20];
    int transferID;
    struct TransferRecord *next;
} *first = NULL;



void createTransferRecordList();
void insertTransferRecord(char *playerName, char *fromTeam, char *toTeam, char
*transferDate, int transferID);
```

```
int deleteTransferRecord(int transferID);
void displayTransferRecords();

int main() {
    int choice, transferID, deleted;
    char playerName[100], fromTeam[100], toTeam[100], transferDate[20];

    while (1) {

        printf("1. Create Transfer Record List\n2. Insert New Transfer Record\n3. Delete Outdated or Erroneous Transfer Record\n4. Display All Current Transfer Records\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createTransferRecordList();
                break;
            case 2:
                printf("Enter player name: ");
                scanf("%s", playerName);
                printf("Enter from team: ");
                scanf("%s", fromTeam);
                printf("Enter to team: ");
                scanf("%s", toTeam);
                printf("Enter transfer date (DD/MM/YYYY): ");
                scanf("%s", transferDate);
                printf("Enter transfer ID: ");
```

```
            scanf("%d", &transferID);
            insertTransferRecord(playerName, fromTeam, toTeam, transferDate, transferID);
            break;
        case 3:
            printf("Enter transfer ID to delete: ");
            scanf("%d", &transferID);
            deleted = deleteTransferRecord(transferID);
            if (deleted != -1)
                printf("Transfer record with ID %d deleted.\n", transferID);
            break;
        case 4:
            displayTransferRecords();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to create a new transfer record list
void createTransferRecordList() {
   first = NULL;
   printf("New transfer record list created.\n");
```

```
}

// Function to insert a new transfer record
void insertTransferRecord(char *playerName, char *fromTeam, char *toTeam, char *transferDate, int transferID) {
    struct TransferRecord *newRecord = (struct TransferRecord*)malloc(sizeof(struct TransferRecord));
    strcpy(newRecord->playerName, playerName);
    strcpy(newRecord->fromTeam, fromTeam);
    strcpy(newRecord->toTeam, toTeam);
    strcpy(newRecord->transferDate, transferDate);
    newRecord->transferID = transferID;
    newRecord->next = NULL;

    if (first == NULL) {
        first = newRecord;
    } else {
        struct TransferRecord *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newRecord;
    }
    printf("Transfer record for player %s from %s to %s added to the list.\n", playerName, fromTeam, toTeam);
}

// Function to delete an outdated or erroneous transfer record
int deleteTransferRecord(int transferID) {
```

```
    struct TransferRecord *temp = first, *prev = NULL;

    if (temp != NULL && temp->transferID == transferID) {
        first = temp->next;
        free(temp);
        return transferID;
    } else {
        while (temp != NULL && temp->transferID != transferID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Transfer record with ID %d not found.\n", transferID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return transferID;
    }
}

// Function to display all current transfer records
void displayTransferRecords() {
    struct TransferRecord *temp = first;
    if (temp == NULL) {
        printf("No transfer records available.\n");
        return;
    }
```

```
    printf("Current Transfer Records:\n");
    while (temp != NULL) {
        printf("Transfer ID: %d, Player: %s, From: %s, To: %s, Date: %s\n",
            temp->transferID, temp->playerName, temp->fromTeam, temp->toTeam, temp->transferDate);
        temp = temp->next;
    }
}

/*
roblem 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 PointsEntry {
    char teamName[100];
```

```
    int points;
    int entryID;
    struct PointsEntry *next;
} *first = NULL;



void createPointsTracker();
void insertPointsEntry(char *teamName, int points, int entryID);
int deletePointsEntry(int entryID);
void displayPointsStandings();

int main() {
    int choice, entryID, points, deleted;
    char teamName[100];

    while (1) {

        printf("1. Create Points Tracker List\n2. Insert New Points Entry\n3. Delete Incorrect or
Outdated Points Entry\n4. Display All Current Points Standings\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createPointsTracker();
                break;
            case 2:
                printf("Enter team name: ");
```

```
            scanf("%s", teamName);
            printf("Enter points: ");
            scanf("%d", &points);
            printf("Enter entry ID: ");
            scanf("%d", &entryID);
            insertPointsEntry(teamName, points, entryID);
            break;
        case 3:
            printf("Enter entry ID to delete: ");
            scanf("%d", &entryID);
            deleted = deletePointsEntry(entryID);
            if (deleted != -1)
                printf("Points entry with ID %d deleted.\n", entryID);
            break;
        case 4:
            displayPointsStandings();
            break;
        case 5:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

```
// Function to create a new points tracker list
void createPointsTracker() {
    first = NULL;
    printf("New points tracker list created.\n");
}

// Function to insert a new points entry
void insertPointsEntry(char *teamName, int points, int entryID) {
    struct PointsEntry *newEntry = (struct PointsEntry*)malloc(sizeof(struct PointsEntry));
    strcpy(newEntry->teamName, teamName);
    newEntry->points = points;
    newEntry->entryID = entryID;
    newEntry->next = NULL;

    if (first == NULL) {
        first = newEntry;
    } else {
        struct PointsEntry *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newEntry;
    }
    printf("Points entry for team %s with ID %d added to the list.\n", teamName, entryID);
}

// Function to delete an incorrect or outdated points entry
int deletePointsEntry(int entryID) {
```

```
    struct PointsEntry *temp = first, *prev = NULL;

    if (temp != NULL && temp->entryID == entryID) {
        first = temp->next;
        free(temp);
        return entryID;
    } else {
        while (temp != NULL && temp->entryID != entryID) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Points entry with ID %d not found.\n", entryID);
            return -1;
        }
        prev->next = temp->next;
        free(temp);
        return entryID;
    }
}

// Function to display all current points standings
void displayPointsStandings() {
    struct PointsEntry *temp = first;
    if (temp == NULL) {
        printf("No points entries available.\n");
        return;
    }
```

```
    printf("Current Points Standings:\n");
    while (temp != NULL) {
        printf("Entry ID: %d, Team: %s, Points: %d\n", temp->entryID, temp->teamName, temp->points);
        temp = temp->next;
    }
}
```