Problem 1: Inventory Management System

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

Operations:

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

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

struct Node
{
   char material[50];
   int quantity;
   struct Node *next;
} *first = NULL;

// Function prototypes
void createInventory(char materials[][50], int quantities[], int n);
void displayInventory(struct Node *p);
void insertMaterial(struct Node *p, int index, char material[], int quantity);
int deleteMaterial(struct Node *p, int pos);
```

```
int main()
{
    int option, quantity, index, pos;
    char material[50];
    char materials[][50] = {"Steel", "Plastic", "Wood"};
    int quantities[] = {150, 250, 100};
    createInventory(materials, quantities, 3);
    do
    {
        printf("\n--- Inventory Management System ---\n");
        printf("1. Create an inventory list\n");
        printf("2. Insert a new raw material\n");
        printf("3. Delete a raw material from the inventory\n");
        printf("4. Display current inventory\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: createInventory(materials, quantities, 3);
                    printf("Inventory list created successfully.\n");
                    break;
            case 2: printf("Enter position to insert: ");
                    scanf("%d", &index);
                    printf("Enter raw material name: ");
                    scanf("%s", material);
                    printf("Enter quantity: ");
```

```
                scanf("%d", &quantity);
                insertMaterial(first, index, material, quantity);
                printf("Material inserted successfully\n");
                break;
            case 3: printf("Enter position to delete: ");
                scanf("%d", &pos);
                if (deleteMaterial(first, pos))
                   printf("Material deleted successfully.\n");
                else
                   printf("Invalid position.\n");
                break;
            case 4: printf("Current Inventory:\n");
                displayInventory(first);
                break;
            case 5: printf("Exit the system\n");
                break;
            default:printf("Invalid option\n");
        }
    } while( option != 5);
    return 0;
}

void createInventory(char materials[][50], int quantities[], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
```

```
    strcpy(first->material, materials[0]);
    first->quantity = quantities[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        strcpy(temp->material, materials[i]);
        temp->quantity = quantities[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayInventory(struct Node *p)
{
    if (!p)
    {
        printf("Inventory is empty\n");
        return;
    }
    while (p)
    {
        printf("%s -> ", p->material);
        p = p->next;
    }
```

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

void insertMaterial(struct Node *p, int index, char material[], int quantity)
{
    struct Node *temp;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    strcpy(temp->material, material);
    temp->quantity = quantity;
    if (index == 0)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 0; i < index - 1 && p; i++)
            p = p->next;
        if (p)
        {
            temp->next = p->next;
            p->next = temp;
        }
        else
        {
            printf("Invalid position\n");
```

```
            free(temp);
        }
    }
}

int deleteMaterial(struct Node *p, int pos)
{
    struct Node *q = NULL;
    int i;
    if (pos == 1)
    {
        if (first)
        {
            q = first;
            first = first->next;
            free(q);
            return 1;
        }
        return 0;
    }
    for (i = 0; i < pos - 2 && p; i++)
        p = p->next;
    if (p && p->next)
    {
        q = p->next;
        p->next = q->next;
        free(q);
```

```
        return 1;
    }
    return 0;
}
```

Problem 2: Production Line Queue

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

Operations:

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

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

struct Task
{
    char name[50];
    struct Task *next;
} *first = NULL, *last = NULL;

// Function prototypes
void createTaskQueue(char tasks[][50], int n);
void insertQueue(char taskName[]);
```

```
void deleteQueue();
void displayQueue();

int main()
{
   int option;
   char taskName[50];
   char tasks[][50] = {"Task1", "Task2", "Task3"};
   do
   {
      printf("\n--- Production Line Queue ---\n");
      printf("1. Create a production task queue\n");
      printf("2. Insert a new task into the queue\n");
      printf("3. Delete a completed task\n");
      printf("4. Display the current task queue\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createTaskQueue(tasks, 3);
                 printf("Task queue created successfully\n");
                 break;
         case 2: printf("Enter task name: ");
                 scanf("%s", taskName);
                 insertQueue(taskName);
                 printf("Task added to the queue\n");
```

```
                break;
            case 3: deleteQueue();
                break;
            case 4: printf("Current Task Queue:\n");
                displayQueue();
                break;
            case 5: printf("Exit the queue\n");
                break;
            default: printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}

void createTaskQueue(char tasks[][50], int n)
{
    for (int i = 0; i < n; i++)
        insertQueue(tasks[i]);
}

void insertQueue(char taskName[])
{
    struct Task *temp = (struct Task *)malloc(sizeof(struct Task));
    strcpy(temp->name, taskName);
    temp->next = NULL;
    if (first == NULL)
        first = last = temp;
```

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

void deleteQueue()
{
    struct Task *temp;
    if (first == NULL)
    {
        printf("Queue is empty\n");
        return;
    }
    temp = first;
    first = first->next;
    if (first == NULL)
        last = NULL;
    printf("Completed task: %s\n", temp->name);
    free(temp);
}

void displayQueue()
{
    struct Task *temp = first;
    if (temp == NULL)
```

```
    {
        printf("Queue is empty\n");
        return;
    }
    while (temp)
    {
        printf("%s -> ", temp->name);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

Problem 3: Machine Maintenance Schedule

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

Operations:

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

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

struct Node
{
```

```
    char machine[50];
    char task[100];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createSchedule(char machines[][50], char tasks[][100], int n);
void displaySchedule(struct Node *p);
void insertTask(char machine[], char task[], int position);
void deleteTask(int position);

int main()
{
    int option, position;
    char machine[50], task[100];
     char machines[][50] = {"Lathe Machine", "Drilling Machine", "Milling Machine"};
    char tasks[][100] = {"Oil lubrication and cleaning", "Replace drill bits",
        "Calibrate spindle speed" };

    createSchedule(machines, tasks, 3);

    do
    {
        printf("\n--- Machine Maintenance Schedule ---\n");
        printf("1. Create a maintenance schedule\n");
        printf("2. Insert a new maintenance task\n");
        printf("3. Delete a completed maintenance task\n");
```

```
printf("4. Display maintenance schedule\n");
printf("5. Exit\n");
printf("Enter the option: ");
scanf("%d", &option);
switch (option)
{
   case 1: createSchedule(machines, tasks, 3);
           printf("Maintenance schedule created successfully\n");
           break;
   case 2: printf("Enter machine name: ");
           scanf(" %[^\n]", machine);
           printf("Enter maintenance task: ");
           scanf(" %[^\n]", task);
           printf("Enter position to insert the task: ");
           scanf("%d", &position);
           insertTask(machine, task, position);
           printf("Task added successfully\n");
           break;
   case 3: printf("Enter position of the task to delete: ");
           scanf("%d", &position);
           deleteTask(position);
           break;
   case 4: printf("Current maintenance schedule:\n");
           displaySchedule(first);
           break;
   case 5: printf("Exit\n");
           break;
```

```
            default:printf("Invalid option\n");
        }
    }while (option != 5);
    return 0;
}

void createSchedule(char machines[][50], char tasks[][100], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    strcpy(first->machine, machines[0]);
    strcpy(first->task, tasks[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        strcpy(temp->machine, machines[i]);
        strcpy(temp->task, tasks[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displaySchedule(struct Node *p)
```

```
{
    int i = 1;
    if (!p)
    {
        printf("Maintenance schedule is empty\n");
        return;
    }
    while (p)
    {
        printf("%d. Machine: %s | Task: %s\n", i++, p->machine, p->task);
        p = p->next;
    }
}

void insertTask(char machine[], char task[], int position)
{
    struct Node *temp, *current = first;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    strcpy(temp->machine, machine);
    strcpy(temp->task, task);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
```

```
    else
    {
        for (i = 1; i < position - 1 && current; i++)
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}

void deleteTask(int position)
{
    struct Node *temp = first, *prev = NULL;
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
          printf("Deleted task: Machine: %s | Task: %s\n", temp->machine, temp->task);
```

```
            free(temp);
        }
        else
            printf("No tasks to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }

    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted task: Machine: %s | Task: %s\n", temp->machine, temp->task);
        free(temp);
    }
    else
        printf("Invalid position\n");
}
```

Problem 4: Employee Shift Management

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

Operations:

1. Create a shift schedule.

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

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

struct Node
{
    char employeeName[50];
    char shift[20];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createShiftSchedule(char employees[][50], char shifts[][20], int n);
void displayShiftSchedule(struct Node *p);
void insertShift(char employeeName[], char shift[], int position);
void deleteShift(int position);

int main()
{
    int option, position;
    char employeeName[50], shift[20];
    char employees[][50] = {"ABC", "DEF", "GHI"};
    char shifts[][20] = {"Morning", "Afternoon", "Night"};
```

```
createShiftSchedule(employees, shifts, 3);
do
{
   printf("\n--- Employee Shift Management ---\n");
   printf("1. Create shift schedule\n");
   printf("2. Insert a new shift\n");
   printf("3. Delete a completed or canceled shift\n");
   printf("4. Display the current shift schedule\n");
   printf("5. Exit\n");
   printf("Enter the option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: createShiftSchedule(employees, shifts, 3);
            printf("Shift schedule created successfully.\n");
            break;
      case 2: printf("Enter employee name: ");
            scanf(" %[^\n]", employeeName);
            printf("Enter shift (Morning/Afternoon/Night): ");
            scanf(" %[^\n]", shift);
            printf("Enter position to insert the shift: ");
            scanf("%d", &position);
            insertShift(employeeName, shift, position);
            printf("Shift added successfully.\n");
            break;
      case 3: printf("Enter position of the shift to delete: ");
            scanf("%d", &position);
```

```
                deleteShift(position);
                break;
            case 4: printf("Current Shift Schedule:\n");
                displayShiftSchedule(first);
                break;
            case 5: printf("Exit\n");
                break;
            default:printf("Invalid option\n");
        }
    } while( option != 5);
    return 0;
}

void createShiftSchedule(char employees[][50], char shifts[][20], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    strcpy(first->employeeName, employees[0]);
    strcpy(first->shift, shifts[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        strcpy(temp->employeeName, employees[i]);
        strcpy(temp->shift, shifts[i]);
```

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

void displayShiftSchedule(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("Shift schedule is empty\n");
        return;
    }
    while (p)
    {
         printf("%d. Employee: %s | Shift: %s\n", i++, p->employeeName, p->shift);
        p = p->next;
    }
}

void insertShift(char employeeName[], char shift[], int position)
{
    struct Node *temp, *current = first;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    strcpy(temp->employeeName, employeeName);
```

```
    strcpy(temp->shift, shift);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 1; i < position - 1 && current; i++)
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}

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

```
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
                printf("Deleted shift: Employee: %s | Shift: %s\n", temp->employeeName, temp->shift);
            free(temp);
        }
        else
            printf("No shifts to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
            printf("Deleted shift: Employee: %s | Shift: %s\n", temp->employeeName, temp->shift);
        free(temp);
    }
    else
        printf("Invalid position\n");
```

```
}
```

Problem 5: Order Processing System

Description: Implement a linked list to track customer orders.

Operations:

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

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

struct Node
{
   int orderId;
   char customerName[50];
   char orderDetails[100];
   struct Node *next;
} *first = NULL;

// Function prototypes
void createOrderList(int orderIds[], char customers[][50], char details[][100],
int n);
void displayOrders(struct Node *p);
```

```
void insertOrder(int orderId, char customerName[], char orderDetails[], int position);
void deleteOrder(int position);


int main()
{
   int option, position, orderId;
   char customerName[50], orderDetails[100];
   int orderIds[] = {1, 2, 3};
   char customers[][50] = {"ABC", "DEF", "GHI"};
    char details[][100] = {"2 Laptops, 1 Mouse", "1 Smartphone, 1 Charger", "5 Books"};
   createOrderList(orderIds, customers, details, 3);
   do
   {
      printf("\n--- Order Processing System ---\n");
      printf("1. Create an order list\n");
      printf("2. Insert a new customer order\n");
      printf("3. Delete a completed or canceled order\n");
      printf("4. Display all current orders\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createOrderList(orderIds, customers, details, 3);
               printf("Order list created successfully\n");
               break;
```

```
            case 2: printf("Enter order ID: ");
                    scanf("%d", &orderId);
                    printf("Enter customer name: ");
                    scanf(" %[^\n]", customerName);
                    printf("Enter order details: ");
                    scanf(" %[^\n]", orderDetails);
                    printf("Enter position to insert the order: ");
                    scanf("%d", &position);
                    insertOrder(orderId, customerName, orderDetails, position);
                    printf("Order added successfully\n");
                    break;
            case 3: printf("Enter position of the order to delete: ");
                    scanf("%d", &position);
                    deleteOrder(position);
                    break;
            case 4: printf("Current Orders:\n");
                    displayOrders(first);
                    break;
            case 5: printf("Exit the system\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}

void createOrderList(int orderIds[], char customers[][50], char details[][100],
int n)
```

```
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    first->orderId = orderIds[0];
    strcpy(first->customerName, customers[0]);
    strcpy(first->orderDetails, details[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->orderId = orderIds[i];
        strcpy(temp->customerName, customers[i]);
        strcpy(temp->orderDetails, details[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayOrders(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("Order list is empty\n");
```

```
        return;
    }
    while (p)
    {
        printf("%d. Order ID: %d | Customer: %s | Details: %s\n", i++, p->orderId, p->customerName, p->orderDetails);
        p = p->next;
    }
}

void insertOrder(int orderId, char customerName[], char orderDetails[], int position)
{
    struct Node *temp, *current = first;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->orderId = orderId;
    strcpy(temp->customerName, customerName);
    strcpy(temp->orderDetails, orderDetails);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 1; i < position - 1 && current; i++)
```

```
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}

void deleteOrder(int position)
{
    struct Node *temp = first, *prev = NULL;
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
             printf("Deleted order: Order ID: %d | Customer: %s | Details: %s\n",
temp->orderId, temp->customerName, temp->orderDetails);
            free(temp);
        }
        else
```

```
            printf("No orders to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted order: Order ID: %d | Customer: %s | Details: %s\n",
temp->orderId, temp->customerName, temp->orderDetails);
        free(temp);
    }
    else
        printf("Invalid position\n");
}
```

Problem 6: Tool Tracking System

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

Operations:

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

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

struct Node
{
    int toolId;
    char toolName[50];
    char status[20];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createToolList(int toolIds[], char toolNames[][50], char statuses[][20], int n);
void displayTools(struct Node *p);
void insertTool(int toolId, char toolName[], char status[], int position);
void deleteTool(int position);

int main()
{
    int option, position, toolId;
    char toolName[50], status[20];
    int toolIds[] = {101, 102, 103};
    char toolNames[][50] = {"Hammer", "Screwdriver"};
    char statuses[][20] = {"In Use", "Available"};
    createToolList(toolIds, toolNames, statuses, 2);
    do
```

```
{
    printf("\n--- Tool Tracking System ---\n");
    printf("1. Create a tool tracking list\n");
    printf("2. Insert a new tool entry\n");
    printf("3. Delete a tool that is no longer in use\n");
    printf("4. Display all tools currently tracked\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);
    switch (option)
    {
        case 1: createToolList(toolIds, toolNames, statuses, 3);
                printf("Tool tracking list created successfully\n");
                break;
        case 2: printf("Enter tool ID: ");
                scanf("%d", &toolId);
                printf("Enter tool name: ");
                scanf(" %[^\n]", toolName);
                printf("Enter tool status (In Use/Available): ");
                scanf(" %[^\n]", status);
                printf("Enter position to insert the tool: ");
                scanf("%d", &position);
                insertTool(toolId, toolName, status, position);
                printf("Tool added successfully\n");
                break;
        case 3: printf("Enter position of the tool to delete: ");
                scanf("%d", &position);
```

```
                deleteTool(position);
                break;
            case 4: printf("Current Tools:\n");
                displayTools(first);
                break;
            case 5: printf("Exit the system\n");
                break;
            default: printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}

void createToolList(int toolIds[], char toolNames[][50], char statuses[][20], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    first->toolId = toolIds[0];
    strcpy(first->toolName, toolNames[0]);
    strcpy(first->status, statuses[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->toolId = toolIds[i];
```

```
        strcpy(temp->toolName, toolNames[i]);
        strcpy(temp->status, statuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayTools(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("Tool tracking list is empty\n");
        return;
    }
    while (p)
    {
        printf("%d. Tool ID: %d | Name: %s | Status: %s\n", i++, p->toolId, p->toolName, p->status);
        p = p->next;
    }
}

void insertTool(int toolId, char toolName[], char status[], int position)
{
    struct Node *temp, *current = first;
    int i;
```

```
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->toolId = toolId;
    strcpy(temp->toolName, toolName);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 1; i < position - 1 && current; i++)
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}
```

```
void deleteTool(int position)
{
   struct Node *temp = first, *prev = NULL;
   int i;
   if (position == 1)
   {
      if (first)
      {
         first = first->next;
         printf("Deleted tool: Tool ID: %d | Name: %s | Status: %s\n", temp->toolId, temp->toolName, temp->status);
         free(temp);
      }
      else
         printf("No tools to delete\n");
      return;
   }
   for (i = 1; i < position && temp; i++)
   {
      prev = temp;
      temp = temp->next;
   }
   if (temp)
   {
      prev->next = temp->next;
      printf("Deleted tool: Tool ID: %d | Name: %s | Status: %s\n", temp->toolId, temp->toolName, temp->status);
      free(temp);
```

```
    }
    else
        printf("Invalid position\n");
}
```

Problem 7: Product Assembly Line

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

Operations:

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

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

struct Node
{
    int stageId;
    char stageName[50];
    char status[20];
    struct Node *next;
} *first = NULL;

// Function prototypes
```

```
void createAssemblyLine(int stageIds[], char stageNames[][50], char statuses[][20], int n);
void displayAssemblyStages(struct Node *p);
void insertStage(int stageId, char stageName[], char status[], int position);
void deleteStage(int position);

int main()
{
   int option, position, stageId;
   char stageName[50], status[20];
   int stageIds[] = {1, 2, 3};
   char stageNames[][50] = {"FGH", "IKL", "MNB"};
   char statuses[][20] = {"In Progress", "Pending", "Pending"};
   createAssemblyLine(stageIds, stageNames, statuses, 3);
   do
   {
      printf("\n--- Product Assembly Line Management ---\n");
      printf("1. Create an assembly line stage list\n");
      printf("2. Insert a new stage\n");
      printf("3. Delete a completed stage\n");
      printf("4. Display the current assembly stages\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createAssemblyLine(stageIds, stageNames, statuses, 3);
                 printf("Assembly line stages created successfully\n");
```

```
            break;
        case 2: printf("Enter stage ID: ");
            scanf("%d", &stageId);
            printf("Enter stage name: ");
            scanf(" %[^\n]", stageName);
            printf("Enter stage status (In Progress/Completed/Pending): ");
            scanf(" %[^\n]", status);
            printf("Enter position to insert the stage: ");
            scanf("%d", &position);
            insertStage(stageId, stageName, status, position);
            printf("Stage added successfully\n");
            break;
        case 3: printf("Enter position of the stage to delete: ");
            scanf("%d", &position);
            deleteStage(position);
            break;
        case 4: printf("Current Assembly Stages:\n");
            displayAssemblyStages(first);
            break;
        case 5: printf("Exit the system\n");
            break;
        default: printf("Invalid option\n");
      }
   } while( option != 5);
   return 0;
}
```

```
void createAssemblyLine(int stageIds[], char stageNames[][50], char statuses[][20], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->stageId = stageIds[0];
   strcpy(first->stageName, stageNames[0]);
   strcpy(first->status, statuses[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Node *)malloc(sizeof(struct Node));
      temp->stageId = stageIds[i];
      strcpy(temp->stageName, stageNames[i]);
      strcpy(temp->status, statuses[i]);
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void displayAssemblyStages(struct Node *p)
{
   int i = 1;
   if (!p)
   {
```

```
        printf("Assembly line is empty\n");
        return;
    }
    while (p)
    {
        printf("%d. Stage ID: %d | Name: %s | Status: %s\n", i++, p->stageId, p->stageName, p->status);
        p = p->next;
    }
}

void insertStage(int stageId, char stageName[], char status[], int position)
{
    struct Node *temp, *current = first;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->stageId = stageId;
    strcpy(temp->stageName, stageName);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 1; i < position - 1 && current; i++)
```

```
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}

void deleteStage(int position)
{
    struct Node *temp = first, *prev = NULL;
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
            printf("Deleted stage: Stage ID: %d | Name: %s | Status: %s\n", temp->stageId, temp->stageName, temp->status);
            free(temp);
        }
        else
```

```
            printf("No stages to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted stage: Stage ID: %d | Name: %s | Status: %s\n", temp->stageId, temp->stageName, temp->status);
        free(temp);
    }
    else
        printf("Invalid position\n");
}
```

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 Node
{
    int itemId;
    char itemName[100];
    char status[20];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createChecklist(int itemIds[], char itemNames[][100], char statuses[][20], int n);
void displayChecklist(struct Node *p);
void insertItem(int itemId, char itemName[], char status[], int position);
void deleteItem(int position);

int main()
{
    int option, position, itemId;
    char itemName[100], status[20];
    int itemIds[] = {1, 2, 3};
    char itemNames[][100] = {"ABC", "DEF", "GHI"};
    char statuses[][20] = {"Pending", "Completed", "Pending"};
    createChecklist(itemIds, itemNames, statuses, 3);
    do
```

```
{
    printf("\n--- Quality Control Checklist Management ---\n");
    printf("1. Create a quality control checklist\n");
    printf("2. Insert a new checklist item\n");
    printf("3. Delete a completed or outdated checklist item\n");
    printf("4. Display the current quality control checklist\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);
    switch (option)
    {
        case 1: createChecklist(itemIds, itemNames, statuses, 3);
                printf("Quality control checklist created successfully\n");
                break;
        case 2: printf("Enter item ID: ");
                scanf("%d", &itemId);
                printf("Enter item name: ");
                scanf(" %[^\n]", itemName);
                printf("Enter item status (Completed/Pending/Outdated): ");
                scanf(" %[^\n]", status);
                printf("Enter position to insert the item: ");
                scanf("%d", &position);
                insertItem(itemId, itemName, status, position);
                printf("Item added successfully\n");
                break;
        case 3: printf("Enter position of the item to delete: ");
                scanf("%d", &position);
```

```
            deleteItem(position);
            break;
        case 4: printf("Current Quality Control Checklist:\n");
            displayChecklist(first);
            break;
        case 5: printf("Exit the checklist\n");
            break;
        default: printf("Invalid option\n");
      }
   } while(option != 5);
   return 0;
}

void createChecklist(int itemIds[], char itemNames[][100], char statuses[][20], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->itemId = itemIds[0];
   strcpy(first->itemName, itemNames[0]);
   strcpy(first->status, statuses[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Node *)malloc(sizeof(struct Node));
      temp->itemId = itemIds[i];
```

```
        strcpy(temp->itemName, itemNames[i]);
        strcpy(temp->status, statuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayChecklist(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("Quality control checklist is empty\n");
        return;
    }
    while (p)
    {
        printf("%d. Item ID: %d | Name: %s | Status: %s\n", i++, p->itemId, p->itemName, p->status);
        p = p->next;
    }
}

void insertItem(int itemId, char itemName[], char status[], int position)
{
    struct Node *temp, *current = first;
    int i;
```

```
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->itemId = itemId;
    strcpy(temp->itemName, itemName);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
    {
        for (i = 1; i < position - 1 && current; i++)
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}
```

```
void deleteItem(int position)
{
   struct Node *temp = first, *prev = NULL;
   int i;
   if (position == 1)
   {
      if (first)
      {
         first = first->next;
          printf("Deleted item: Item ID: %d | Name: %s | Status: %s\n", temp->itemId, temp->itemName, temp->status);
         free(temp);
      }
      else
         printf("No items to delete\n");
      return;
   }
   for (i = 1; i < position && temp; i++)
   {
      prev = temp;
      temp = temp->next;
   }
   if (temp)
   {
      prev->next = temp->next;
       printf("Deleted item: Item ID: %d | Name: %s | Status: %s\n", temp->itemId, temp->itemName, temp->status);
      free(temp);
```

```
    }
    else
        printf("Invalid position\n");
}
```

Problem 9: Supplier Management System

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

Operations:

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

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

struct Node
{
    int supplierId;
    char supplierName[100];
    char status[20];
    struct Node *next;
} *first = NULL;

// Function prototypes
```

```
void createSupplierList(int supplierIds[], char supplierNames[][100], char statuses[][20], int n);
void displaySuppliers(struct Node *p);
void insertSupplier(int supplierId, char supplierName[], char status[], int position);
void deleteSupplier(int position);


int main()
{
   int option, position, supplierId;
   char supplierName[100], status[20];
   int supplierIds[] = {1, 2, 3};
   char supplierNames[][100] = {"ABC Corp", "XYZ Ltd", "MNO Ltd"};
   char statuses[][20] = {"Active", "Inactive", "Active"};
   createSupplierList(supplierIds, supplierNames, statuses, 3);
   do
   {
      printf("\n--- Supplier Management System ---\n");
      printf("1. Create a supplier list\n");
      printf("2. Insert a new supplier\n");
      printf("3. Delete an inactive or outdated supplier\n");
      printf("4. Display all current suppliers\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createSupplierList(supplierIds, supplierNames, statuses, 3);
```

```
            printf("Supplier list created successfully\n");
            break;
        case 2: printf("Enter supplier ID: ");
            scanf("%d", &supplierId);
            printf("Enter supplier name: ");
            scanf(" %[^\n]", supplierName);
            printf("Enter supplier status (Active/Inactive): ");
            scanf(" %[^\n]", status);
            printf("Enter position to insert the supplier: ");
            scanf("%d", &position);
            insertSupplier(supplierId, supplierName, status, position);
            printf("Supplier added successfully\n");
            break;
        case 3: printf("Enter position of the supplier to delete: ");
            scanf("%d", &position);
            deleteSupplier(position);
            break;
        case 4: printf("Current Suppliers:\n");
            displaySuppliers(first);
            break;
        case 5: printf("Exit the system\n");
            break;
        default:printf("Invalid option\n");
      }
   } while(option != 5);
   return 0;
}
```

```
void createSupplierList(int supplierIds[], char supplierNames[][100], char statuses[][20], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->supplierId = supplierIds[0];
   strcpy(first->supplierName, supplierNames[0]);
   strcpy(first->status, statuses[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Node *)malloc(sizeof(struct Node));
      temp->supplierId = supplierIds[i];
      strcpy(temp->supplierName, supplierNames[i]);
      strcpy(temp->status, statuses[i]);
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void displaySuppliers(struct Node *p)
{
   int i = 1;
   if (!p)
```

```
    {
        printf("Supplier list is empty\n");
        return;
    }
    while (p)
    {
            printf("%d. Supplier ID: %d | Name: %s | Status: %s\n", i++, p->supplierId, p->supplierName, p->status);
        p = p->next;
    }
}

void insertSupplier(int supplierId, char supplierName[], char status[], int position)
{
    struct Node *temp, *current = first;
    int i;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->supplierId = supplierId;
    strcpy(temp->supplierName, supplierName);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (position == 1)
    {
        temp->next = first;
        first = temp;
    }
    else
```

```
    {
        for (i = 1; i < position - 1 && current; i++)
            current = current->next;
        if (current)
        {
            temp->next = current->next;
            current->next = temp;
        }
        else
        {
            printf("Invalid position\n");
            free(temp);
        }
    }
}

void deleteSupplier(int position)
{
    struct Node *temp = first, *prev = NULL;
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
            printf("Deleted supplier: Supplier ID: %d | Name: %s | Status: %s\n",
temp->supplierId, temp->supplierName, temp->status);
            free(temp);
```

```
        }
        else
            printf("No suppliers to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted supplier: Supplier ID: %d | Name: %s | Status: %s\n",
temp->supplierId, temp->supplierName, temp->status);
        free(temp);
    }
    else
        printf("Invalid position\n");
}
```

Problem 10: Manufacturing Project Timeline

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

Operations:

1. Create a project timeline.
2. Insert a new project milestone.

3. Delete a completed milestone.
4. Display the current project timeline.

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

struct Node
{
   int milestoneId;
   char milestoneName[100];
   char status[20];
   struct Node *next;
} *first = NULL;

// Function prototypes
void createTimeline(int milestoneIds[], char milestoneNames[][100], char statuses[][20], int n);
void displayTimeline(struct Node *p);
void insertMilestone(int milestoneId, char milestoneName[], char status[], int position);
void deleteMilestone(int position);

int main()
{
   int option, position, milestoneId;
   char milestoneName[100], status[20];
   int milestoneIds[] = {1, 2, 3};
```

```
char milestoneNames[][100] = {"Design", "Assembly", "Testing"};
char statuses[][20] = {"Pending", "Completed", "Pending"};
createTimeline(milestoneIds, milestoneNames, statuses, 3);
do
{
   printf("\nManufacturing Project Timeline\n");
   printf("1. Create a project timeline\n");
   printf("2. Insert a new project milestone\n");
   printf("3. Delete a completed milestone\n");
   printf("4. Display the current project timeline\n");
   printf("5. Exit\n");
   printf("Enter the option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: createTimeline(milestoneIds, milestoneNames, statuses, 3);
            printf("Project timeline created successfully.\n");
            break;
      case 2: printf("Enter milestone ID: ");
            scanf("%d", &milestoneId);
            printf("Enter milestone name: ");
            scanf(" %[^\n]", milestoneName);
            printf("Enter milestone status (Completed/Pending): ");
            scanf(" %[^\n]", status);
            printf("Enter position to insert the milestone: ");
            scanf("%d", &position);
            insertMilestone(milestoneId, milestoneName, status, position);
```

```
            printf("Milestone added successfully.\n");
            break;
        case 3: printf("Enter position of the milestone to delete: ");
            scanf("%d", &position);
            deleteMilestone(position);
            break;
        case 4: printf("Current Project Timeline:\n");
            displayTimeline(first);
            break;
        case 5: printf("Exit the timeline\n");
            break;
        default: printf("Invalid option\n");
      }
   } while(option != 5);
   return 0;
}

void createTimeline(int milestoneIds[], char milestoneNames[][100], char statuses[][20], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->milestoneId = milestoneIds[0];
   strcpy(first->milestoneName, milestoneNames[0]);
   strcpy(first->status, statuses[0]);
   first->next = NULL;
   last = first;
```

```
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->milestoneId = milestoneIds[i];
        strcpy(temp->milestoneName, milestoneNames[i]);
        strcpy(temp->status, statuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayTimeline(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("Project timeline is empty\n");
        return;
    }
    while (p)
    {
          printf("%d. Milestone ID: %d | Name: %s | Status: %s\n", i++, p->milestoneId, p->milestoneName, p->status);
        p = p->next;
    }
}
```

```
void insertMilestone(int milestoneId, char milestoneName[], char status[], int
position)
{
   struct Node *temp, *current = first;
   int i;
   temp = (struct Node *)malloc(sizeof(struct Node));
   temp->milestoneId = milestoneId;
   strcpy(temp->milestoneName, milestoneName);
   strcpy(temp->status, status);
   temp->next = NULL;
   if (position == 1)
   {
      temp->next = first;
      first = temp;
   }
   else
   {
      for (i = 1; i < position - 1 && current; i++)
         current = current->next;
      if (current)
      {
         temp->next = current->next;
         current->next = temp;
      }
      else
      {
         printf("Invalid position\n");
         free(temp);
```

```
        }
    }
}

void deleteMilestone(int position)
{
    struct Node *temp = first, *prev = NULL;
    int i;
    if (position == 1)
    {
        if (first)
        {
            first = first->next;
              printf("Deleted milestone: Milestone ID: %d | Name: %s | Status: %s\n", temp->milestoneId, temp->milestoneName, temp->status);
            free(temp);
        }
        else
            printf("No milestones to delete\n");
        return;
    }
    for (i = 1; i < position && temp; i++)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
```

```
        prev->next = temp->next;
        printf("Deleted milestone: Milestone ID: %d | Name: %s | Status: %s\n",
temp->milestoneId, temp->milestoneName, temp->status);
        free(temp);
    }
    else
        printf("Invalid position\n");
}
```

Problem 11: Warehouse Storage Management

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

Operations:

1. Create a storage list.
2. Insert a new storage entry.
3. Delete a storage entry when goods are shipped.
4. Display the current warehouse storage.

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

struct Node
{
    int itemId;
    char itemName[100];
    int quantity;
```

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

// Function prototypes
void createStorageList(int itemIds[], char itemNames[][100], int quantities[], int n);
void displayStorage(struct Node *p);
void insertStorage(int itemId, char itemName[], int quantity);
void deleteStorage(int itemId);

int main()
{
    int option, itemId, quantity;
    char itemName[100];
    int itemIds[] = {1, 2, 3};
    char itemNames[][100] = {"ABC", "DEF", "MNO"};
    int quantities[] = {100, 200, 120};
    createStorageList(itemIds, itemNames, quantities, 3);
    do
    {
        printf("\n--- Warehouse Storage Management ---\n");
        printf("1. Create a storage list\n");
        printf("2. Insert a new storage entry\n");
        printf("3. Delete a storage entry when goods are shipped\n");
        printf("4. Display the current warehouse storage\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
```

```
    switch (option)
    {
        case 1: createStorageList(itemIds, itemNames, quantities, 3);
                printf("Storage list created successfully\n");
                break;
        case 2: printf("Enter item ID: ");
                scanf("%d", &itemId);
                printf("Enter item name: ");
                scanf(" %[^\n]", itemName);
                printf("Enter item quantity: ");
                scanf("%d", &quantity);
                insertStorage(itemId, itemName, quantity);
                printf("Item added to storage\n");
                break;
        case 3: printf("Enter item ID to delete: ");
                scanf("%d", &itemId);
                deleteStorage(itemId);
                break;
        case 4: printf("Current Warehouse Storage:\n");
                displayStorage(first);
                break;
        case 5: printf("Exit the management\n");
                break;
        default: printf("Invalid option\n");
    }
} while( option != 5);
return 0;
```

```
}

void createStorageList(int itemIds[], char itemNames[][100], int quantities[],
int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->itemId = itemIds[0];
   strcpy(first->itemName, itemNames[0]);
   first->quantity = quantities[0];
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Node *)malloc(sizeof(struct Node));
      temp->itemId = itemIds[i];
      strcpy(temp->itemName, itemNames[i]);
      temp->quantity = quantities[i];
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void displayStorage(struct Node *p)
{
   int i = 1;
```

```
    if (!p)
    {
        printf("No items in warehouse\n");
        return;
    }
    while (p)
    {
        printf("%d. Item ID: %d | Name: %s | Quantity: %d\n", i++, p->itemId,
p->itemName, p->quantity);
        p = p->next;
    }
}

void insertStorage(int itemId, char itemName[], int quantity)
{
    struct Node *temp, *current = first;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->itemId = itemId;
    strcpy(temp->itemName, itemName);
    temp->quantity = quantity;
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
```

```
    }
}

void deleteStorage(int itemId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->itemId == itemId)
    {
        first = first->next;
        printf("Deleted item from storage: Item ID: %d | Name: %s | Quantity: %d\n", temp->itemId, temp->itemName, temp->quantity);
        free(temp);
        return;
    }
    while (temp && temp->itemId != itemId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted item from storage: Item ID: %d | Name: %s | Quantity: %d\n", temp->itemId, temp->itemName, temp->quantity);
        free(temp);
    }
    else
        printf("Item not found\n");
```

```
}
```

Problem 12: Machine Parts Inventory

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

Operations:

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

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

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

// Function prototypes
void createInventoryList(int partIds[], char partNames[][100], int quantities[],
int n);
void displayInventory(struct Node *p);
void insertPart(int partId, char partName[], int quantity);
```

```
void deletePart(int partId);

int main()
{
   int option, partId, quantity;
   char partName[100];
   int partIds[] = {1, 2, 3};
   char partNames[][100] = {"Bolt", "Screw", "Nut"};
   int quantities[] = {500, 900, 650};
   createInventoryList(partIds, partNames, quantities, 3);
   do
   {
      printf("\n--- Machine Parts Inventory Management ---\n");
      printf("1. Create a parts inventory list\n");
      printf("2. Insert a new part\n");
      printf("3. Delete a part that is used up or obsolete\n");
      printf("4. Display the current parts inventory\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createInventoryList(partIds, partNames, quantities, 3);
               printf("Parts inventory list created successfully\n");
               break;
         case 2: printf("Enter part ID: ");
               scanf("%d", &partId);
```

```
            printf("Enter part name: ");
            scanf(" %[^\n]", partName);
            printf("Enter part quantity: ");
            scanf("%d", &quantity);
            insertPart(partId, partName, quantity);
            printf("Part added to inventory\n");
            break;
        case 3: printf("Enter part ID to delete: ");
            scanf("%d", &partId);
            deletePart(partId);
            break;
        case 4: printf("Current Machine Parts Inventory:\n");
            displayInventory(first);
            break;
        case 5: printf("Exit the program\n");
            break;
        default: printf("Invalid option\n");
      }
   } while( option != 5);
   return 0;
}

void createInventoryList(int partIds[], char partNames[][100], int quantities[],
int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
```

```
    first->partId = partIds[0];
    strcpy(first->partName, partNames[0]);
    first->quantity = quantities[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->partId = partIds[i];
        strcpy(temp->partName, partNames[i]);
        temp->quantity = quantities[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayInventory(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("No parts in inventory\n");
        return;
    }
    while (p)
    {
```

```
        printf("%d. Part ID: %d | Name: %s | Quantity: %d\n", i++, p->partId, p->partName, p->quantity);
        p = p->next;
    }
}

void insertPart(int partId, char partName[], int quantity)
{
    struct Node *temp, *current = first;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->partId = partId;
    strcpy(temp->partName, partName);
    temp->quantity = quantity;
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deletePart(int partId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->partId == partId)
```

```
    {
        first = first->next;
        printf("Deleted part from inventory: Part ID: %d | Name: %s | Quantity: %d\n", temp->partId, temp->partName, temp->quantity);
        free(temp);
        return;
    }
    while (temp && temp->partId != partId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted part from inventory: Part ID: %d | Name: %s | Quantity: %d\n", temp->partId, temp->partName, temp->quantity);
        free(temp);
    }
    else
        printf("Part not found\n");
}
```

Problem 13: Packaging Line Schedule

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

Operations:

1. Create a packaging task schedule.

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

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

struct Node
{
    int taskId;
    char taskName[100];
    char taskStatus[50];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createScheduleList(int taskIds[], char taskNames[][100], char taskStatuses[][50], int n);
void displaySchedule(struct Node *p);
void insertTask(int taskId, char taskName[], char taskStatus[]);
void deleteTask(int taskId);

int main()
{
    int option, taskId;
    char taskName[100], taskStatus[50];
    int taskIds[] = {1, 2, 3};
```

```
    char taskNames[][100] = {"Pack Boxes", "Label Products", "Seal Packages"};
    char taskStatuses[][50] = {"Pending", "Pending", "Pending"};
    createScheduleList(taskIds, taskNames, taskStatuses, 3);
    do
    {
        printf("\n--- Packaging Line Schedule Management ---\n");
        printf("1. Create a packaging task schedule\n");
        printf("2. Insert a new packaging task\n");
        printf("3. Delete a completed packaging task\n");
        printf("4. Display the current packaging schedule\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: createScheduleList(taskIds, taskNames, taskStatuses, 3);
                    printf("Packaging task schedule created successfully\n");
                    break;
            case 2: printf("Enter task ID: ");
                    scanf("%d", &taskId);
                    printf("Enter task name: ");
                    scanf(" %[^\n]", taskName);
                    printf("Enter task status: ");
                    scanf(" %[^\n]", taskStatus);
                    insertTask(taskId, taskName, taskStatus);
                    printf("Task added to the schedule.\n");
                    break;
```

```
            case 3: printf("Enter task ID to delete: ");
                    scanf("%d", &taskId);
                    deleteTask(taskId);
                    break;
            case 4: printf("Current Packaging Task Schedule:\n");
                    displaySchedule(first);
                    break;
            case 5: printf("Exit the schedule\n");
                    break;
            default:printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}


void createScheduleList(int taskIds[], char taskNames[][100], char taskStatuses[][50], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    first->taskId = taskIds[0];
    strcpy(first->taskName, taskNames[0]);
    strcpy(first->taskStatus, taskStatuses[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
```

```
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->taskId = taskIds[i];
        strcpy(temp->taskName, taskNames[i]);
        strcpy(temp->taskStatus, taskStatuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displaySchedule(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("No tasks in the schedule\n");
        return;
    }
    while (p)
    {
         printf("%d. Task ID: %d | Name: %s | Status: %s\n", i++, p->taskId, p->taskName, p->taskStatus);
        p = p->next;
    }
}

void insertTask(int taskId, char taskName[], char taskStatus[])
{
```

```
    struct Node *temp, *current = first;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->taskId = taskId;
    strcpy(temp->taskName, taskName);
    strcpy(temp->taskStatus, taskStatus);
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteTask(int taskId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->taskId == taskId)
    {
        first = first->next;
         printf("Deleted task from schedule: Task ID: %d | Name: %s | Status: %s\n", temp->taskId, temp->taskName, temp->taskStatus);
        free(temp);
        return;
    }
    while (temp && temp->taskId != taskId)
```

```
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted task from schedule: Task ID: %d | Name: %s | Status: %s\n", temp->taskId, temp->taskName, temp->taskStatus);
        free(temp);
    }
    else
        printf("Task not found\n");
}
```

Problem 14: Production Defect Tracking

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

Operations:

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

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

```
struct Node
{
   int defectId;
   char defectDescription[200];
   char severity[50];
   char status[50];
   struct Node *next;
} *first = NULL;

// Function prototypes
void createDefectList(int defectIds[], char defectDescriptions[][200], char severities[][50], char statuses[][50], int n);
void displayDefectList(struct Node *p);
void insertDefect(int defectId, char defectDescription[], char severity[], char status[]);
void deleteDefect(int defectId);

int main()
{
   int option, defectId;
   char defectDescription[200], severity[50], status[50];
   int defectIds[] = {101, 102, 103};
   char defectDescriptions[][200] = {"ABC", "DEF", "GHI"};
   char severities[][50] = {"High", "Medium", "High"};
   char statuses[][50] = {"Open", "Open", "Resolved"};
   createDefectList(defectIds, defectDescriptions, severities, statuses, 3);
   do
   {
```

```
        printf("\n--- Production Defect Tracking System ---\n");
        printf("1. Create a defect tracking list\n");
        printf("2. Insert a new defect report\n");
        printf("3. Delete a resolved defect\n");
        printf("4. Display all current defects\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: createDefectList(defectIds, defectDescriptions, severities, statuses, 3);
                    printf("Defect tracking list created successfully\n");
                    break;
           case 2: printf("Enter defect ID: ");
                    scanf("%d", &defectId);
                    printf("Enter defect description: ");
                    scanf(" %[^\n]", defectDescription);
                    printf("Enter severity: ");
                    scanf(" %[^\n]", severity);
                    printf("Enter status: ");
                    scanf(" %[^\n]", status);
                    insertDefect(defectId, defectDescription, severity, status);
                    printf("Defect added to the tracking list\n");
                    break;
           case 3: printf("Enter defect ID to delete: ");
                    scanf("%d", &defectId);
                    deleteDefect(defectId);
```

```
            break;
        case 4: printf("Current Defect Tracking List:\n");
            displayDefectList(first);
            break;
        case 5: printf("Exit\n");
            break;
        default: printf("Invalid option\n");
      }
   } while(option != 5);
   return 0;
}

void createDefectList(int defectIds[], char defectDescriptions[][200], char severities[][50], char statuses[][50], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->defectId = defectIds[0];
   strcpy(first->defectDescription, defectDescriptions[0]);
   strcpy(first->severity, severities[0]);
   strcpy(first->status, statuses[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Node *)malloc(sizeof(struct Node));
      temp->defectId = defectIds[i];
```

```
        strcpy(temp->defectDescription, defectDescriptions[i]);
        strcpy(temp->severity, severities[i]);
        strcpy(temp->status, statuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayDefectList(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("No defects to display\n");
        return;
    }
    while (p)
    {
       printf("%d. Defect ID: %d | Description: %s | Severity: %s | Status: %s\n",
i++, p->defectId, p->defectDescription, p->severity, p->status);
        p = p->next;
    }
}

void insertDefect(int defectId, char defectDescription[], char severity[], char
status[])
{
```

```
    struct Node *temp, *current = first;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->defectId = defectId;
    strcpy(temp->defectDescription, defectDescription);
    strcpy(temp->severity, severity);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteDefect(int defectId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->defectId == defectId)
    {
        first = first->next;
        printf("Deleted defect: Defect ID: %d | Description: %s | Severity: %s | Status: %s\n", temp->defectId, temp->defectDescription, temp->severity, temp->status);
        free(temp);
        return;
```

```
    }
    while (temp && temp->defectId != defectId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted defect: Defect ID: %d | Description: %s | Severity: %s | Status: %s\n", temp->defectId, temp->defectDescription, temp->severity, temp->status);
        free(temp);
    }
    else
        printf("Defect not found\n");
}
```

Problem 15: Finished Goods Dispatch System

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

Operations:

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

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

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

struct Node
{
    int dispatchId;
    char productName[100];
    int quantity;
    char dispatchDate[20];
    char status[50];
    struct Node *next;
} *first = NULL;

// Function prototypes
void createDispatchSchedule(int dispatchIds[], char productNames[][100], int quantities[], char dispatchDates[][20], char statuses[][50], int n);
void displayDispatchSchedule(struct Node *p);
void insertDispatchEntry(int dispatchId, char productName[], int quantity, char dispatchDate[], char status[]);
void deleteDispatchEntry(int dispatchId);

int main()
{
    int option, dispatchId, quantity;
    char productName[100], dispatchDate[20], status[50];
    int dispatchIds[] = {1001, 1002, 1003};
    char productNames[][100] = {"Product A", "Product B", "Product C"};
    int quantities[] = {100, 300, 200};
```

```
    char dispatchDates[][20] = {"2025-01-07", "2025-01-04", "2025-01-12"};
    char statuses[][50] = {"Pending", "Dispatched", "Dispatched"};
          createDispatchSchedule(dispatchIds, productNames, quantities,
dispatchDates, statuses, 3);
    do
    {
       printf("\n--- Finished Goods Dispatch System ---\n");
       printf("1. Create a dispatch schedule\n");
       printf("2. Insert a new dispatch entry\n");
       printf("3. Delete a dispatched or canceled entry\n");
       printf("4. Display the current dispatch schedule\n");
       printf("5. Exit\n");
       printf("Enter the option: ");
       scanf("%d", &option);
       switch (option)
       {
         case 1: createDispatchSchedule(dispatchIds, productNames, quantities,
dispatchDates, statuses, 3);
                 printf("Dispatch schedule created successfully\n");
                 break;
          case 2: printf("Enter dispatch ID: ");
                 scanf("%d", &dispatchId);
                 printf("Enter product name: ");
                 scanf(" %[^\n]", productName);
                 printf("Enter quantity: ");
                 scanf("%d", &quantity);
                 printf("Enter dispatch date (YYYY-MM-DD): ");
                 scanf(" %[^\n]", dispatchDate);
```

```
            printf("Enter status: ");
            scanf(" %[^\n]", status);
                insertDispatchEntry(dispatchId, productName, quantity, dispatchDate, status);
            printf("Dispatch entry added successfully\n");
            break;
        case 3: printf("Enter dispatch ID to delete: ");
            scanf("%d", &dispatchId);
            deleteDispatchEntry(dispatchId);
            break;
        case 4: printf("Current Dispatch Schedule:\n");
            displayDispatchSchedule(first);
            break;
        case 5: printf("Exit the system\n");
            break;
        default: printf("Invalid option\n");
      }
   } while(option != 5);
   return 0;
}

void createDispatchSchedule(int dispatchIds[], char productNames[][100], int quantities[], char dispatchDates[][20], char statuses[][50], int n)
{
   int i;
   struct Node *temp, *last;
   first = (struct Node *)malloc(sizeof(struct Node));
   first->dispatchId = dispatchIds[0];
```

```
    strcpy(first->productName, productNames[0]);
    first->quantity = quantities[0];
    strcpy(first->dispatchDate, dispatchDates[0]);
    strcpy(first->status, statuses[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->dispatchId = dispatchIds[i];
        strcpy(temp->productName, productNames[i]);
        temp->quantity = quantities[i];
        strcpy(temp->dispatchDate, dispatchDates[i]);
        strcpy(temp->status, statuses[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayDispatchSchedule(struct Node *p)
{
    int i = 1;
    if (!p)
    {
        printf("No dispatch entries to display\n");
        return;
```

```
    }
    while (p)
    {
        printf("%d. Dispatch ID: %d | Product: %s | Quantity: %d | Dispatch Date: %s | Status: %s\n",
               i++, p->dispatchId, p->productName, p->quantity, p->dispatchDate, p->status);
        p = p->next;
    }
}

void insertDispatchEntry(int dispatchId, char productName[], int quantity, char dispatchDate[], char status[])
{
    struct Node *temp, *current = first;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->dispatchId = dispatchId;
    strcpy(temp->productName, productName);
    temp->quantity = quantity;
    strcpy(temp->dispatchDate, dispatchDate);
    strcpy(temp->status, status);
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
```

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

void deleteDispatchEntry(int dispatchId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->dispatchId == dispatchId)
    {
        first = first->next;
        printf("Deleted dispatch entry: Dispatch ID: %d | Product: %s | Quantity: %d | Dispatch Date: %s | Status: %s\n",
                temp->dispatchId, temp->productName, temp->quantity, temp->dispatchDate, temp->status);
        free(temp);
        return;
    }
    while (temp && temp->dispatchId != dispatchId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted dispatch entry: Dispatch ID: %d | Product: %s | Quantity: %d | Dispatch Date: %s | Status: %s\n",
                temp->dispatchId, temp->productName, temp->quantity, temp->dispatchDate, temp->status);
```

```
        free(temp);
    }
    else
        printf("Dispatch entry not found\n");
}
```

Problem 1: Team Roster Management

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

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

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

struct Node
{
    int playerId;
    char playerName[100];
    int age;
    char position[50];
    struct Node *next;
} *first = NULL;
```

```
// Function prototypes
void createTeamRoster(int playerIds[], char playerNames[][100], int ages[], char positions[][50], int n);
void displayTeamRoster(struct Node *p);
void insertNewPlayer(int playerId, char playerName[], int age, char position[]);
void deletePlayer(int playerId);


int main()
{
   int option, playerId, age;
   char playerName[100], position[50];
   int playerIds[] = {1, 2, 3};
   char playerNames[][100] = {"WXY", "PQR", "XYZ"};
   int ages[] = {28, 23, 29};
   char positions[][50] = {"Forward", "Midfielder", "Defender"};
   createTeamRoster(playerIds, playerNames, ages, positions, 3);
   do
   {
      printf("\n--- Team Roster Management ---\n");
      printf("1. Create team roster\n");
      printf("2. Insert a new player\n");
      printf("3. Delete a player who leaves the team\n");
      printf("4. Display the current team roster\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
```

```
    {
        case 1: createTeamRoster(playerIds, playerNames, ages, positions, 3);
                printf("Team roster created successfully\n");
                break;
        case 2: printf("Enter player ID: ");
                scanf("%d", &playerId);
                printf("Enter player name: ");
                scanf(" %[^\n]", playerName);
                printf("Enter player age: ");
                scanf("%d", &age);
                printf("Enter player position: ");
                scanf(" %[^\n]", position);
                insertNewPlayer(playerId, playerName, age, position);
                printf("New player added successfully\n");
                break;
        case 3: printf("Enter player ID to delete: ");
                scanf("%d", &playerId);
                deletePlayer(playerId);
                break;
        case 4: printf("Current Team Roster:\n");
                displayTeamRoster(first);
                break;
        case 5: printf("Exit\n");
                break;
        default: printf("Invalid option\n");
    }
} while( option != 5);
```

```
    return 0;
}

void createTeamRoster(int playerIds[], char playerNames[][100], int ages[], char positions[][50], int n)
{
    int i;
    struct Node *temp, *last;
    first = (struct Node *)malloc(sizeof(struct Node));
    first->playerId = playerIds[0];
    strcpy(first->playerName, playerNames[0]);
    first->age = ages[0];
    strcpy(first->position, positions[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Node *)malloc(sizeof(struct Node));
        temp->playerId = playerIds[i];
        strcpy(temp->playerName, playerNames[i]);
        temp->age = ages[i];
        strcpy(temp->position, positions[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}
```

```
void displayTeamRoster(struct Node *p)
{
   int i = 1;
   if (!p)
   {
      printf("No players in the team roster\n");
      return;
   }
   while (p)
   {
      printf("%d. Player ID: %d | Name: %s | Age: %d | Position: %s\n",
            i++, p->playerId, p->playerName, p->age, p->position);
      p = p->next;
   }
}

void insertNewPlayer(int playerId, char playerName[], int age, char position[])
{
   struct Node *temp, *current = first;
   temp = (struct Node *)malloc(sizeof(struct Node));
   temp->playerId = playerId;
   strcpy(temp->playerName, playerName);
   temp->age = age;
   strcpy(temp->position, position);
   temp->next = NULL;
   if (!first)
      first = temp;
```

```
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deletePlayer(int playerId)
{
    struct Node *temp = first, *prev = NULL;
    if (first && first->playerId == playerId)
    {
        first = first->next;
        printf("Deleted player: Player ID: %d | Name: %s | Age: %d | Position:
%s\n",
                temp->playerId, temp->playerName, temp->age, temp->position);
        free(temp);
        return;
    }
    while (temp && temp->playerId != playerId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
```

```
        printf("Deleted player: Player ID: %d | Name: %s | Age: %d | Position: %s\n",
               temp->playerId, temp->playerName, temp->age, temp->position);
        free(temp);
    }
    else
        printf("Player not found\n");
}
```

Problem 2: Tournament Match Scheduling

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

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

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

struct Match
{
    int matchId;
    char team1[100];
    char team2[100];
    char date[20];
    struct Match *next;
```

```
} *first = NULL;

// Function prototypes
void createMatchSchedule(int matchIds[], char team1[][100], char team2[][100], char dates[][20], int n);
void displayMatchSchedule(struct Match *p);
void insertNewMatch(int matchId, char team1[], char team2[], char date[]);
void deleteMatch(int matchId);

int main()
{
   int option, matchId;
   char team1[100], team2[100], date[20];
   int matchIds[] = {1, 2, 3};
   char team1Names[][100] = {"Team A", "Team B", "Team C"};
   char team2Names[][100] = {"Team D", "Team E", "Team F"};
   char matchDates[][20] = {"2025-01-20", "2025-01-22", "2025-01-25"};
    createMatchSchedule(matchIds, team1Names, team2Names, matchDates, 3);
   do
   {
      printf("\n--- Tournament Match Scheduling ---\n");
      printf("1. Create match schedule\n");
      printf("2. Insert a new match\n");
      printf("3. Delete a completed or canceled match\n");
      printf("4. Display the current match schedule\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
```

```
    scanf("%d", &option);
    switch (option)
    {
        case 1: createMatchSchedule(matchIds, team1Names, team2Names,
matchDates, 3);
                printf("Match schedule created successfully\n");
                break;
        case 2: printf("Enter match ID: ");
                scanf("%d", &matchId);
                printf("Enter team 1 name: ");
                scanf(" %[^\n]", team1);
                printf("Enter team 2 name: ");
                scanf(" %[^\n]", team2);
                printf("Enter match date: ");
                scanf(" %[^\n]", date);
                insertNewMatch(matchId, team1, team2, date);
                printf("New match added successfully\n");
                break;
        case 3: printf("Enter match ID to delete: ");
                scanf("%d", &matchId);
                deleteMatch(matchId);
                break;
        case 4: printf("Current Match Schedule:\n");
                displayMatchSchedule(first);
                break;
        case 5: printf("Exit\n");
                break;
        default: printf("Invalid option\n");
```

```
        }
    } while(option != 5);
    return 0;
}

void createMatchSchedule(int matchIds[], char team1[][100], char team2[][100], char dates[][20], int n)
{
    int i;
    struct Match *temp, *last;
    first = (struct Match *)malloc(sizeof(struct Match));
    first->matchId = matchIds[0];
    strcpy(first->team1, team1[0]);
    strcpy(first->team2, team2[0]);
    strcpy(first->date, dates[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Match *)malloc(sizeof(struct Match));
        temp->matchId = matchIds[i];
        strcpy(temp->team1, team1[i]);
        strcpy(temp->team2, team2[i]);
        strcpy(temp->date, dates[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
```

```
}

void displayMatchSchedule(struct Match *p)
{
   if (!p)
   {
      printf("No matches scheduled\n");
      return;
   }
   while (p)
   {
      printf("Match ID: %d | %s vs %s | Date: %s\n",
            p->matchId, p->team1, p->team2, p->date);
      p = p->next;
   }
}

void insertNewMatch(int matchId, char team1[], char team2[], char date[])
{
   struct Match *temp, *current = first;
   temp = (struct Match *)malloc(sizeof(struct Match));
   temp->matchId = matchId;
   strcpy(temp->team1, team1);
   strcpy(temp->team2, team2);
   strcpy(temp->date, date);
   temp->next = NULL;
   if (!first)
```

```
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteMatch(int matchId)
{
    struct Match *temp = first, *prev = NULL;
    if (first && first->matchId == matchId)
    {
        first = first->next;
        printf("Deleted match: Match ID: %d | %s vs %s | Date: %s\n",
               temp->matchId, temp->team1, temp->team2, temp->date);
        free(temp);
        return;
    }
    while (temp && temp->matchId != matchId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
```

```
        prev->next = temp->next;
        printf("Deleted match: Match ID: %d | %s vs %s | Date: %s\n",
              temp->matchId, temp->team1, temp->team2, temp->date);
        free(temp);
    }
    else
        printf("Match not found\n");
}
```

Problem 3: Athlete Training Log

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

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

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

struct TrainingSession
{
    int sessionId;
    char athleteName[100];
    char trainingType[100];
    char date[20];
```

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

// Function prototypes
void createTrainingLog(int sessionIds[], char athleteNames[][100], char trainingTypes[][100], char dates[][20], int n);
void displayTrainingLog(struct TrainingSession *p);
void insertNewTrainingSession(int sessionId, char athleteName[], char trainingType[], char date[]);
void deleteTrainingSession(int sessionId);

int main()
{
    int option, sessionId;
    char athleteName[100], trainingType[100], date[20];
    int sessionIds[] = {1, 2, 3};
    char athleteNames[][100] = {"KLP", "MKL", "ABC"};
      char trainingTypes[][100] = {"Sprinting", "Long Jump", "Strength Training"};
    char trainingDates[][20] = {"2025-01-15", "2025-01-18", "2025-01-22"};
    createTrainingLog(sessionIds, athleteNames, trainingTypes, trainingDates, 3);
    do
    {
        printf("\n--- Athlete Training Log ---\n");
        printf("1. Create training log\n");
        printf("2. Insert a new training session\n");
        printf("3. Delete a completed or canceled training session\n");
        printf("4. Display the training log\n");
```

```
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: createTrainingLog(sessionIds, athleteNames, trainingTypes,
trainingDates, 3);
                    printf("Training log created successfully\n");
                    break;
            case 2: printf("Enter session ID: ");
                    scanf("%d", &sessionId);
                    printf("Enter athlete name: ");
                    scanf(" %[^\n]", athleteName);
                    printf("Enter training type: ");
                    scanf(" %[^\n]", trainingType);
                    printf("Enter training date: ");
                    scanf(" %[^\n]", date);
                    insertNewTrainingSession(sessionId, athleteName, trainingType,
date);
                    printf("New training session added successfully.\n");
                    break;
            case 3: printf("Enter session ID to delete: ");
                    scanf("%d", &sessionId);
                    deleteTrainingSession(sessionId);
                    break;
            case 4: printf("Current Training Log:\n");
                    displayTrainingLog(first);
                    break;
```

```
            case 5: printf("Exit\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}

void createTrainingLog(int sessionIds[], char athleteNames[][100], char trainingTypes[][100], char dates[][20], int n)
{
    int i;
    struct TrainingSession *temp, *last;
    first = (struct TrainingSession *)malloc(sizeof(struct TrainingSession));
    first->sessionId = sessionIds[0];
    strcpy(first->athleteName, athleteNames[0]);
    strcpy(first->trainingType, trainingTypes[0]);
    strcpy(first->date, dates[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct TrainingSession *)malloc(sizeof(struct TrainingSession));
        temp->sessionId = sessionIds[i];
        strcpy(temp->athleteName, athleteNames[i]);
        strcpy(temp->trainingType, trainingTypes[i]);
        strcpy(temp->date, dates[i]);
        temp->next = NULL;
```

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

void displayTrainingLog(struct TrainingSession *p)
{
    if (!p)
    {
        printf("No training sessions recorded\n");
        return;
    }
    while (p)
    {
        printf("Session ID: %d | Athlete: %s | Training: %s | Date: %s\n",
                p->sessionId, p->athleteName, p->trainingType, p->date);
        p = p->next;
    }
}

void insertNewTrainingSession(int sessionId, char athleteName[], char
trainingType[], char date[])
{
    struct TrainingSession *temp, *current = first;
    temp = (struct TrainingSession *)malloc(sizeof(struct TrainingSession));
    temp->sessionId = sessionId;
    strcpy(temp->athleteName, athleteName);
    strcpy(temp->trainingType, trainingType);
```

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

void deleteTrainingSession(int sessionId)
{
    struct TrainingSession *temp = first, *prev = NULL;
    if (first && first->sessionId == sessionId)
    {
        first = first->next;
         printf("Deleted session: Session ID: %d | Athlete: %s | Training: %s | Date: %s\n",
                temp->sessionId, temp->athleteName, temp->trainingType, temp->date);
        free(temp);
        return;
    }
    while (temp && temp->sessionId != sessionId)
    {
        prev = temp;
```

```
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted session: Session ID: %d | Athlete: %s | Training: %s | Date: %s\n",
                temp->sessionId, temp->athleteName, temp->trainingType, temp->date);
        free(temp);
    }
    else
        printf("Session not found\n");
}
```

Problem 4: Sports Equipment Inventory

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

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

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

```
struct Equipment
{
    int equipmentId;
    char name[100];
    char condition[20];
    struct Equipment *next;
} *first = NULL;

// Function prototypes
void createInventory(int equipmentIds[], char names[][100], char conditions[][20], int n);
void displayInventory(struct Equipment *p);
void insertNewEquipment(int equipmentId, char name[], char condition[]);
void deleteEquipment(int equipmentId);

int main()
{
    int option, equipmentId;
    char name[100], condition[20];
    int equipmentIds[] = {1, 2, 3};
    char names[][100] = {"Basketball", "Football", "Tennis Racket"};
    char conditions[][20] = {"Good", "Damaged", "Good"};
    createInventory(equipmentIds, names, conditions, 3);
    do
    {
        printf("\n--- Sports Equipment Inventory ---\n");
        printf("1. Create equipment inventory\n");
        printf("2. Insert a new equipment item\n");
```

```
printf("3. Delete an item that is no longer usable\n");
printf("4. Display the current equipment inventory\n");
printf("5. Exit\n");
printf("Enter the option: ");
scanf("%d", &option);
switch (option)
{
   case 1: createInventory(equipmentIds, names, conditions, 3);
           printf("Inventory created successfully\n");
           break;
   case 2: printf("Enter equipment ID: ");
           scanf("%d", &equipmentId);
           printf("Enter equipment name: ");
           scanf(" %[^\n]", name);
           printf("Enter equipment condition: ");
           scanf(" %[^\n]", condition);
           insertNewEquipment(equipmentId, name, condition);
           printf("New equipment item added successfully\n");
           break;
   case 3: printf("Enter equipment ID to delete: ");
           scanf("%d", &equipmentId);
           deleteEquipment(equipmentId);
           break;
   case 4: printf("Current Equipment Inventory:\n");
           displayInventory(first);
           break;
   case 5: printf("Exit\n");
```

```
                break;
            default:printf("Invalid option\n");
        }
    } while( option != 5);
    return 0;
}

void createInventory(int equipmentIds[], char names[][100], char conditions[][20], int n)
{
    int i;
    struct Equipment *temp, *last;
    first = (struct Equipment *)malloc(sizeof(struct Equipment));
    first->equipmentId = equipmentIds[0];
    strcpy(first->name, names[0]);
    strcpy(first->condition, conditions[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Equipment *)malloc(sizeof(struct Equipment));
        temp->equipmentId = equipmentIds[i];
        strcpy(temp->name, names[i]);
        strcpy(temp->condition, conditions[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
```

```
}

void displayInventory(struct Equipment *p)
{
   if (!p)
   {
      printf("No equipment in inventory\n");
      return;
   }
   while (p)
   {
      printf("Equipment ID: %d | Name: %s | Condition: %s\n",
            p->equipmentId, p->name, p->condition);
      p = p->next;
   }
}

void insertNewEquipment(int equipmentId, char name[], char condition[])
{
   struct Equipment *temp, *current = first;
   temp = (struct Equipment *)malloc(sizeof(struct Equipment));
   temp->equipmentId = equipmentId;
   strcpy(temp->name, name);
   strcpy(temp->condition, condition);
   temp->next = NULL;
   if (!first)
      first = temp;
```

```
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteEquipment(int equipmentId)
{
    struct Equipment *temp = first, *prev = NULL;
    if (first && first->equipmentId == equipmentId)
    {
        first = first->next;
        printf("Deleted equipment: ID: %d | Name: %s | Condition: %s\n",
               temp->equipmentId, temp->name, temp->condition);
        free(temp);
        return;
    }
    while (temp && temp->equipmentId != equipmentId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
```

```
        printf("Deleted equipment: ID: %d | Name: %s | Condition: %s\n",
               temp->equipmentId, temp->name, temp->condition);
        free(temp);
    }
    else
        printf("Equipment not found\n");
}
```

Problem 5: Player Performance Tracking

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

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

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

struct Performance
{
    int playerId;
    char playerName[100];
    int matchesPlayed;
    int totalRuns;
    int totalWickets;
```

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

// Function prototypes
void createPerformanceList(int playerIds[], char playerNames[][100], int matches[], int runs[], int wickets[], int n);
void displayPerformanceRecords(struct Performance *p);
void insertPerformanceEntry(int playerId, char playerName[], int matchesPlayed, int totalRuns, int totalWickets);
void deletePerformanceEntry(int playerId);

int main()
{
    int option, playerId, matchesPlayed, totalRuns, totalWickets;
    char playerName[100];
    int playerIds[] = {1, 2, 3};
    char playerNames[][100] = {"JHI", "ABC", "BCD"};
    int matches[] = {10, 8, 12};
    int runs[] = {450, 390, 550};
    int wickets[] = {15, 12, 20};
    createPerformanceList(playerIds, playerNames, matches, runs, wickets, 3);
    do
    {
        printf("\n--- Player Performance Tracking System ---\n");
        printf("1. Create a performance record list\n");
        printf("2. Insert a new performance entry\n");
        printf("3. Delete an outdated or erroneous entry\n");
        printf("4. Display all performance records\n");
```

```
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
           case 1: createPerformanceList(playerIds, playerNames, matches, runs,
wickets, 3);
                   printf("Performance record list created successfully\n");
                   break;
           case 2: printf("Enter player ID: ");
                   scanf("%d", &playerId);
                   printf("Enter player name: ");
                   scanf(" %[^\n]", playerName);
                   printf("Enter matches played: ");
                   scanf("%d", &matchesPlayed);
                   printf("Enter total runs: ");
                   scanf("%d", &totalRuns);
                   printf("Enter total wickets: ");
                   scanf("%d", &totalWickets);
                     insertPerformanceEntry(playerId, playerName, matchesPlayed,
totalRuns, totalWickets);
                   printf("New performance entry added successfully\n");
                   break;
           case 3: printf("Enter player ID to delete: ");
                   scanf("%d", &playerId);
                   deletePerformanceEntry(playerId);
                   break;
           case 4: printf("Player Performance Records:\n");
```

```
                displayPerformanceRecords(first);
                break;
            case 5: printf("Exit\n");
                break;
            default:printf("Invalid option\n");
        }
    } while(option != 5);
    return 0;
}

void createPerformanceList(int playerIds[], char playerNames[][100], int matches[], int runs[], int wickets[], int n)
{
    int i;
    struct Performance *temp, *last;
    first = (struct Performance *)malloc(sizeof(struct Performance));
    first->playerId = playerIds[0];
    strcpy(first->playerName, playerNames[0]);
    first->matchesPlayed = matches[0];
    first->totalRuns = runs[0];
    first->totalWickets = wickets[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Performance *)malloc(sizeof(struct Performance));
        temp->playerId = playerIds[i];
        strcpy(temp->playerName, playerNames[i]);
```

```
        temp->matchesPlayed = matches[i];
        temp->totalRuns = runs[i];
        temp->totalWickets = wickets[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayPerformanceRecords(struct Performance *p)
{
    if (!p)
    {
        printf("No performance records found\n");
        return;
    }
    while (p)
    {
         printf("Player ID: %d | Name: %s | Matches: %d | Runs: %d | Wickets: %d\n",
                p->playerId, p->playerName, p->matchesPlayed, p->totalRuns, p->totalWickets);
        p = p->next;
    }
}

void insertPerformanceEntry(int playerId, char playerName[], int matchesPlayed, int totalRuns, int totalWickets)
```

```
{
    struct Performance *temp, *current = first;
    temp = (struct Performance *)malloc(sizeof(struct Performance));
    temp->playerId = playerId;
    strcpy(temp->playerName, playerName);
    temp->matchesPlayed = matchesPlayed;
    temp->totalRuns = totalRuns;
    temp->totalWickets = totalWickets;
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deletePerformanceEntry(int playerId)
{
    struct Performance *temp = first, *prev = NULL;
    if (first && first->playerId == playerId)
    {
        first = first->next;
        printf("Deleted performance entry: Player ID: %d | Name: %s\n",
                temp->playerId, temp->playerName);
```

```
        free(temp);
        return;
    }
    while (temp && temp->playerId != playerId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted performance entry: Player ID: %d | Name: %s\n",
              temp->playerId, temp->playerName);
        free(temp);
    }
    else
        printf("Performance entry not found\n");
}
```

Problem 6: Event Registration System

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

1. Create a registration list.
2. Insert a new registration.
3. Delete a canceled registration.
4. Display all current registrations.

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

struct Registration
{
    int athleteId;
    char athleteName[100];
    char eventName[100];
    struct Registration *next;
} *first = NULL;

// Function prototypes
void createRegistrationList(int athleteIds[], char athleteNames[][100], char eventNames[][100], int n);
void displayRegistrations(struct Registration *p);
void insertRegistration(int athleteId, char athleteName[], char eventName[]);
void deleteRegistration(int athleteId);

int main()
{
    int option, athleteId;
    char athleteName[100], eventName[100];
    int athleteIds[] = {1, 2, 3};
    char athleteNames[][100] = {"ABC", "JKL", "ERT"};
    char eventNames[][100] = {"Sprint", "Long Jump", "High Jump"};
    createRegistrationList(athleteIds, athleteNames, eventNames, 3);
    do
```

```
    {
        printf("\n--- Event Registration System ---\n");
        printf("1. Create a registration list\n");
        printf("2. Insert a new registration\n");
        printf("3. Delete a canceled registration\n");
        printf("4. Display all current registrations\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: createRegistrationList(athleteIds, athleteNames, eventNames,
3);
                    printf("Registration list created successfully\n");
                    break;
            case 2:printf("Enter athlete ID: ");
                    scanf("%d", &athleteId);
                    printf("Enter athlete name: ");
                    scanf(" %[^\n]", athleteName);
                    printf("Enter event name: ");
                    scanf(" %[^\n]", eventName);
                    insertRegistration(athleteId, athleteName, eventName);
                    printf("New registration added successfully\n");
                    break;
            case 3: printf("Enter athlete ID to delete: ");
                    scanf("%d", &athleteId);
                    deleteRegistration(athleteId);
                    break;
```

```
            case 4: printf("Current Registrations:\n");
                    displayRegistrations(first);
                    break;
            case 5: printf("Exit the system\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while( option != 5);
    return 0;
}

void createRegistrationList(int athleteIds[], char athleteNames[][100], char eventNames[][100], int n)
{
    int i;
    struct Registration *temp, *last;
    first = (struct Registration *)malloc(sizeof(struct Registration));
    first->athleteId = athleteIds[0];
    strcpy(first->athleteName, athleteNames[0]);
    strcpy(first->eventName, eventNames[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Registration *)malloc(sizeof(struct Registration));
        temp->athleteId = athleteIds[i];
        strcpy(temp->athleteName, athleteNames[i]);
        strcpy(temp->eventName, eventNames[i]);
```

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

void displayRegistrations(struct Registration *p)
{
    if (!p)
    {
        printf("No registrations found\n");
        return;
    }
    while (p)
    {
        printf("Athlete ID: %d | Name: %s | Event: %s\n",
                p->athleteId, p->athleteName, p->eventName);
        p = p->next;
    }
}

void insertRegistration(int athleteId, char athleteName[], char eventName[])
{
    struct Registration *temp, *current = first;
    temp = (struct Registration *)malloc(sizeof(struct Registration));
    temp->athleteId = athleteId;
    strcpy(temp->athleteName, athleteName);
```

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

void deleteRegistration(int athleteId)
{
    struct Registration *temp = first, *prev = NULL;
    if (first && first->athleteId == athleteId) {
        first = first->next;
        printf("Deleted registration: Athlete ID: %d | Name: %s\n",
               temp->athleteId, temp->athleteName);
        free(temp);
        return;
    }
    while (temp && temp->athleteId != athleteId)
    {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted registration: Athlete ID: %d | Name: %s\n",
              temp->athleteId, temp->athleteName);
        free(temp);
    }
    else
        printf("Registration not found\n");
}
```

Problem 7: Sports League Standings

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

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

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

struct Team
{
    int teamId;
    char teamName[100];
```

```
    int points;
    struct Team *next;
} *first = NULL;

// Function prototypes
void createLeagueStandings(int teamIds[], char teamNames[][100], int points[], int n);
void displayLeagueStandings(struct Team *p);
void insertTeam(int teamId, char teamName[], int points);
void deleteTeam(int teamId);

int main()
{
    int option, teamId;
    char teamName[100];
    int teamIds[] = {1, 2, 3};
    char teamNames[][100] = {"Red", "Blue", "Green"};
    int points[] = {25, 30, 20};
    createLeagueStandings(teamIds, teamNames, points, 3);
    do
    {
        printf("\n--- Sports League Standings ---\n");
        printf("1. Create a league standings list\n");
        printf("2. Insert a new team\n");
        printf("3. Delete a team that withdraws\n");
        printf("4. Display the current league standings\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
```

```
    scanf("%d", &option);
    switch (option)
    {
        case 1: createLeagueStandings(teamIds, teamNames, points, 3);
                printf("League standings list created successfully\n");
                break;
        case 2: printf("Enter team ID: ");
                scanf("%d", &teamId);
                printf("Enter team name: ");
                scanf(" %[^\n]", teamName);
                printf("Enter team points: ");
                scanf("%d", &points);
                insertTeam(teamId, teamName, points);
                printf("New team added successfully\n");
                break;
        case 3: printf("Enter team ID to delete: ");
                scanf("%d", &teamId);
                deleteTeam(teamId);
                break;
        case 4: printf("Current League Standings:\n");
                displayLeagueStandings(first);
                break;
        case 5: printf("Exit\n");
                break;
        default:printf("Invalid option\n");
    }
} while(option != 5);
```

```
    return 0;
}

void createLeagueStandings(int teamIds[], char teamNames[][100], int points[], int n)
{
    int i;
    struct Team *temp, *last;
    first = (struct Team *)malloc(sizeof(struct Team));
    first->teamId = teamIds[0];
    strcpy(first->teamName, teamNames[0]);
    first->points = points[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Team *)malloc(sizeof(struct Team));
        temp->teamId = teamIds[i];
        strcpy(temp->teamName, teamNames[i]);
        temp->points = points[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayLeagueStandings(struct Team *p)
{
```

```
    if (!p)
    {
        printf("No teams found in the standings\n");
        return;
    }
    printf("Team ID\tTeam Name\tPoints\n");
    while (p)
    {
        printf("%d\t%s\t\t%d\n", p->teamId, p->teamName, p->points);
        p = p->next;
    }
}

void insertTeam(int teamId, char teamName[], int points)
{
    struct Team *temp, *current = first;
    temp = (struct Team *)malloc(sizeof(struct Team));
    temp->teamId = teamId;
    strcpy(temp->teamName, teamName);
    temp->points = points;
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
```

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

void deleteTeam(int teamId)
{
    struct Team *temp = first, *prev = NULL;
    if (first && first->teamId == teamId)
    {
        first = first->next;
        printf("Deleted team: Team ID: %d | Name: %s\n", temp->teamId, temp->teamName);
        free(temp);
        return;
    }
    while (temp && temp->teamId != teamId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted team: Team ID: %d | Name: %s\n", temp->teamId, temp->teamName);
        free(temp);
    }
    else
```

```
        printf("Team not found\n");
    }
```

Problem 8: Match Result Recording

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

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

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

struct Match
{
   int matchId;
   char team1[50];
   char team2[50];
   char result[50];
   struct Match *next;
} *first = NULL;

// Function prototypes
void createMatchResultList(int matchIds[], char team1[][50], char team2[][50], char results[][50], int n);
void insertMatchResult(int matchId, char team1[], char team2[], char result[]);
```

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

int main()
{
   int option, matchId;
   char team1[50], team2[50], result[50];
   do
   {
      printf("\n--- Match Results Recording ---\n");
      printf("1. Create a match result list\n");
      printf("2. Insert a new match result\n");
      printf("3. Delete an incorrect or outdated result\n");
      printf("4. Display all recorded match results\n");
      printf("5. Exit\n");
      printf("Enter your option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: int matchIds[] = {1, 2, 3};
                 char team1[][50] = {"Team A", "Team C", "Team E"};
                 char team2[][50] = {"Team B", "Team D", "Team F"};
                 char results[][50] = {"Team A wins", "Draw", "Team F wins"};
                 createMatchResultList(matchIds, team1, team2, results, 3);
                 printf("Match result list created successfully\n");
                 break;
         case 2: printf("Enter Match ID: ");
```

```
            scanf("%d", &matchId);
            printf("Enter Team 1: ");
            scanf(" %[^\n]", team1);
            printf("Enter Team 2: ");
            scanf(" %[^\n]", team2);
            printf("Enter Result: ");
            scanf(" %[^\n]", result);
            insertMatchResult(matchId, team1, team2, result);
            printf("New match result added successfully\n");
            break;
        case 3: printf("Enter Match ID to delete: ");
            scanf("%d", &matchId);
            deleteMatchResult(matchId);
            break;
        case 4: printf("All Recorded Match Results:\n");
            displayMatchResults();
            break;
        case 5: printf("Exit\n");
            break;
        default:
           printf("Invalid option\n");
      }
   } while (option != 5);
   return 0;
}
```

```
void createMatchResultList(int matchIds[], char team1[][50], char team2[][50], char results[][50], int n)
```

```
{
    struct Match *temp, *last;
    first = NULL;
    for (int i = 0; i < n; i++)
    {
        temp = (struct Match *)malloc(sizeof(struct Match));
        temp->matchId = matchIds[i];
        strcpy(temp->team1, team1[i]);
        strcpy(temp->team2, team2[i]);
        strcpy(temp->result, results[i]);
        temp->next = NULL;
        if (!first)
            first = temp;
        else
            last->next = temp;
        last = temp;
    }
}

void insertMatchResult(int matchId, char team1[], char team2[], char result[])
{
    struct Match *temp, *current = first;
    temp = (struct Match *)malloc(sizeof(struct Match));
    temp->matchId = matchId;
    strcpy(temp->team1, team1);
    strcpy(temp->team2, team2);
    strcpy(temp->result, result);
```

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

void deleteMatchResult(int matchId)
{
    struct Match *temp = first, *prev = NULL;
    if (!first)
    {
        printf("No match results to delete\n");
        return;
    }
    if (first->matchId == matchId)
    {
        first = first->next;
        printf("Deleted match result: Match ID: %d\n", temp->matchId);
        free(temp);
        return;
    }
    while (temp && temp->matchId != matchId)
```

```
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted match result: Match ID: %d\n", temp->matchId);
        free(temp);
    }
    else
        printf("Match result with ID %d not found\n", matchId);
}

void displayMatchResults()
{
    struct Match *temp = first;
    if (!temp)
    {
        printf("No match results recorded.\n");
        return;
    }
    printf("Match ID\tTeam 1\t\tTeam 2\t\tResult\n");
    while (temp)
    {
       printf("%d\t\t%-10s\t%-10s\t%s\n", temp->matchId, temp->team1, temp->team2, temp->result);
        temp = temp->next;
```

```
    }
}
```

Problem 9: Player Injury Tracker

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

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

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

struct InjuryReport
{
    int playerId;
    char playerName[100];
    char injuryDescription[200];
    char injuryDate[20];
    struct InjuryReport *next;
} *first = NULL;

// Function prototypes
void createInjuryList(int playerIds[], char playerNames[][100], char injuries[][200], char dates[][20], int n);
void displayInjuryReports(struct InjuryReport *p);
```

```
void insertInjuryReport(int playerId, char playerName[], char injuryDescription[], char injuryDate[]);
void deleteInjuryReport(int playerId);

int main()
{
   int option, playerId;
   char playerName[100], injuryDescription[200], injuryDate[20];
   int playerIds[] = {1, 2, 3};
   char playerNames[][100] = {"JHI", "ABC", "BCD"};
   char injuries[][200] = {"Knee injury", "Shoulder sprain", "Ankle fracture"};
   char dates[][20] = {"2025-01-01", "2025-01-10", "2025-01-15"};
   createInjuryList(playerIds, playerNames, injuries, dates, 3);
   do
   {
      printf("\n--- Player Injury Tracker ---\n");
      printf("1. Create an injury tracker list\n");
      printf("2. Insert a new injury report\n");
      printf("3. Delete a resolved or erroneous injury report\n");
      printf("4. Display all current injury reports\n");
      printf("5. Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: createInjuryList(playerIds, playerNames, injuries, dates, 3);
                 printf("Injury tracker list created successfully\n");
                 break;
```

```
            case 2: printf("Enter player ID: ");
                    scanf("%d", &playerId);
                    printf("Enter player name: ");
                    scanf(" %[^\n]", playerName);
                    printf("Enter injury description: ");
                    scanf(" %[^\n]", injuryDescription);
                    printf("Enter injury date (YYYY-MM-DD): ");
                    scanf("%s", injuryDescription);
                        insertInjuryReport(playerId, playerName, injuryDescription,
injuryDate);
                    printf("New injury report added successfully\n");
                    break;
            case 3: printf("Enter player ID to delete: ");
                    scanf("%d", &playerId);
                    deleteInjuryReport(playerId);
                    break;
            case 4: printf("Player Injury Reports:\n");
                    displayInjuryReports(first);
                    break;
            case 5: printf("Exit\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    return 0;
}
```

```
void createInjuryList(int playerIds[], char playerNames[][100], char injuries[][200], char dates[][20], int n)
{
   int i;
   struct InjuryReport *temp, *last;
   first = (struct InjuryReport *)malloc(sizeof(struct InjuryReport));
   first->playerId = playerIds[0];
   strcpy(first->playerName, playerNames[0]);
   strcpy(first->injuryDescription, injuries[0]);
   strcpy(first->injuryDate, dates[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct InjuryReport *)malloc(sizeof(struct InjuryReport));
      temp->playerId = playerIds[i];
      strcpy(temp->playerName, playerNames[i]);
      strcpy(temp->injuryDescription, injuries[i]);
      strcpy(temp->injuryDate, dates[i]);
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void displayInjuryReports(struct InjuryReport *p)
{
   if (!p)
```

```
    {
        printf("No injury reports found\n");
        return;
    }
    while (p)
    {
        printf("Player ID: %d | Name: %s | Injury: %s | Date: %s\n",
                p->playerId, p->playerName, p->injuryDescription, p->injuryDate);
        p = p->next;
    }
}

void insertInjuryReport(int playerId, char playerName[], char injuryDescription[], char injuryDate[])
{
    struct InjuryReport *temp, *current = first;
    temp = (struct InjuryReport *)malloc(sizeof(struct InjuryReport));
    temp->playerId = playerId;
    strcpy(temp->playerName, playerName);
    strcpy(temp->injuryDescription, injuryDescription);
    strcpy(temp->injuryDate, injuryDate);
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
```

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

void deleteInjuryReport(int playerId)
{
    struct InjuryReport *temp = first, *prev = NULL;
    if (first && first->playerId == playerId)
    {
        first = first->next;
        printf("Deleted injury report: Player ID: %d | Name: %s\n",
               temp->playerId, temp->playerName);
        free(temp);
        return;
    }
    while (temp && temp->playerId != playerId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted injury report: Player ID: %d | Name: %s\n",
               temp->playerId, temp->playerName);
        free(temp);
    }
```

```
    else
        printf("Injury report not found\n");
}
```

Problem 10: Sports Facility Booking System

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

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

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

struct Booking
{
    int bookingId;
    char customerName[100];
    char facilityName[100];
    char bookingDate[20];
    char bookingTime[20];
    struct Booking *next;
} *first = NULL;

// Function prototypes
```

```
void createBookingList(int bookingIds[], char customerNames[][100], char facilityNames[][100], char bookingDates[][20], char bookingTimes[][20], int n);

void displayBookings(struct Booking *p);

void insertBooking(int bookingId, char customerName[], char facilityName[], char bookingDate[], char bookingTime[]);

void deleteBooking(int bookingId);


int main()

{

   int option, bookingId;

        char customerName[100], facilityName[100], bookingDate[20], bookingTime[20];

   int bookingIds[] = {101, 102, 103};

   char customerNames[][100] = {"JKL", "ABC", "BCD"};

   char facilityNames[][100] = {"Tennis Court", "Football Field", "Basketball Court"};

   char bookingDates[][20] = {"2025-01-20", "2025-01-21", "2025-01-22"};

   char bookingTimes[][20] = {"10:00 AM", "02:00 PM", "05:00 PM"};

         createBookingList(bookingIds, customerNames, facilityNames, bookingDates, bookingTimes, 3);

   do

   {

      printf("\n--- Sports Facility Booking System ---\n");

      printf("1. Create a booking list\n");

      printf("2. Insert a new booking\n");

      printf("3. Delete a canceled or completed booking\n");

      printf("4. Display all current bookings\n");

      printf("5. Exit\n");
```

```
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
          case 1: createBookingList(bookingIds, customerNames, facilityNames,
bookingDates, bookingTimes, 3);
                  printf("Booking list created successfully\n");
                  break;
           case 2: printf("Enter booking ID: ");
                  scanf("%d", &bookingId);
                  printf("Enter customer name: ");
                  scanf(" %[^\n]", customerName);
                  printf("Enter facility name: ");
                  scanf(" %[^\n]", facilityName);
                  printf("Enter booking date (YYYY-MM-DD): ");
                  scanf("%s", bookingDate);
                  printf("Enter booking time (HH:MM AM/PM): ");
                  scanf("%s", bookingTime);
                        insertBooking(bookingId, customerName, facilityName,
bookingDate, bookingTime);
                  printf("New booking added successfully\n");
                  break;
           case 3: printf("Enter booking ID to delete: ");
                  scanf("%d", &bookingId);
                  deleteBooking(bookingId);
                  break;
           case 4: printf("Current Bookings:\n");
                  displayBookings(first);
```

```
            break;
        case 5: printf("Exit\n");
            break;
        default:printf("Invalid option\n");
      }
   } while (option != 5);
   return 0;
}

void createBookingList(int bookingIds[], char customerNames[][100], char facilityNames[][100], char bookingDates[][20], char bookingTimes[][20], int n)
{
   int i;
   struct Booking *temp, *last;
   first = (struct Booking *)malloc(sizeof(struct Booking));
   first->bookingId = bookingIds[0];
   strcpy(first->customerName, customerNames[0]);
   strcpy(first->facilityName, facilityNames[0]);
   strcpy(first->bookingDate, bookingDates[0]);
   strcpy(first->bookingTime, bookingTimes[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      temp = (struct Booking *)malloc(sizeof(struct Booking));
      temp->bookingId = bookingIds[i];
      strcpy(temp->customerName, customerNames[i]);
```

```
        strcpy(temp->facilityName, facilityNames[i]);
        strcpy(temp->bookingDate, bookingDates[i]);
        strcpy(temp->bookingTime, bookingTimes[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayBookings(struct Booking *p)
{
    if (!p)
    {
        printf("No bookings found\n");
        return;
    }
    while (p)
    {
        printf("Booking ID: %d | Customer: %s | Facility: %s | Date: %s | Time: %s\n",
                        p->bookingId, p->customerName, p->facilityName, p->bookingDate, p->bookingTime);
        p = p->next;
    }
}

void insertBooking(int bookingId, char customerName[], char facilityName[], char bookingDate[], char bookingTime[])
```

```
{
   struct Booking *temp, *current = first;
   temp = (struct Booking *)malloc(sizeof(struct Booking));
   temp->bookingId = bookingId;
   strcpy(temp->customerName, customerName);
   strcpy(temp->facilityName, facilityName);
   strcpy(temp->bookingDate, bookingDate);
   strcpy(temp->bookingTime, bookingTime);
   temp->next = NULL;
   if (!first)
      first = temp;
   else
   {
      while (current->next)
         current = current->next;
      current->next = temp;
   }
}

void deleteBooking(int bookingId)
{
   struct Booking *temp = first, *prev = NULL;
   if (first && first->bookingId == bookingId)
   {
      first = first->next;
        printf("Deleted booking: Booking ID: %d | Customer: %s | Facility:
%s\n",
            temp->bookingId, temp->customerName, temp->facilityName);
```

```
        free(temp);
        return;
    }
    while (temp && temp->bookingId != bookingId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
          printf("Deleted booking: Booking ID: %d | Customer: %s | Facility: %s\n",
                temp->bookingId, temp->customerName, temp->facilityName);
        free(temp);
    }
    else
        printf("Booking ID not found\n");
}
```

Problem 11: Coaching Staff Management

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

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

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

struct Coach
{
    int coachId;
    char coachName[100];
    char role[50];
    int yearsOfExperience;
    struct Coach *next;
} *first = NULL;

// Function prototypes
void createCoachingStaffList(int coachIds[], char coachNames[][100], char roles[][50], int experiences[], int n);
void displayCoachingStaff(struct Coach *p);
void insertCoach(int coachId, char coachName[], char role[], int yearsOfExperience);
void deleteCoach(int coachId);

int main()
{
    int option, coachId;
    char coachName[100], role[50];
    int yearsOfExperience;
    int coachIds[] = {1, 2, 3};
    char coachNames[][100] = {"John Smith", "Alice Brown", "Bob Lee"};
```

```
    char roles[][50] = {"Head Coach", "Assistant Coach", "Fitness Coach"};
    int experiences[] = {10, 5, 7};
    createCoachingStaffList(coachIds, coachNames, roles, experiences, 3);
    do
    {
       printf("\n--- Coaching Staff Management ---\n");
       printf("1. Create a coaching staff list\n");
       printf("2. Insert a new coach\n");
       printf("3. Delete a coach who leaves the team\n");
       printf("4. Display the current coaching staff\n");
       printf("5. Exit\n");
       printf("Enter the option: ");
       scanf("%d", &option);
       switch (option)
       {
          case 1: createCoachingStaffList(coachIds, coachNames, roles,
experiences, 3);
                  printf("Coaching staff list created successfully\n");
                  break;
          case 2: printf("Enter coach ID: ");
                  scanf("%d", &coachId);
                  printf("Enter coach name: ");
                  scanf(" %[^\n]", coachName);
                  printf("Enter coach role: ");
                  scanf(" %[^\n]", role);
                  printf("Enter years of experience: ");
                  scanf("%d", &yearsOfExperience);
                  insertCoach(coachId, coachName, role, yearsOfExperience);
```

```
            printf("New coach added successfully\n");
            break;
        case 3: printf("Enter coach ID to delete: ");
            scanf("%d", &coachId);
            deleteCoach(coachId);
            break;
        case 4: printf("Current Coaching Staff:\n");
            displayCoachingStaff(first);
            break;
        case 5: printf("Exit\n");
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 5);
   return 0;
}

void createCoachingStaffList(int coachIds[], char coachNames[][100], char roles[][50], int experiences[], int n)
{
   int i;
   struct Coach *temp, *last;
   first = (struct Coach *)malloc(sizeof(struct Coach));
   first->coachId = coachIds[0];
   strcpy(first->coachName, coachNames[0]);
   strcpy(first->role, roles[0]);
   first->yearsOfExperience = experiences[0];
   first->next = NULL;
```

```
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct Coach *)malloc(sizeof(struct Coach));
        temp->coachId = coachIds[i];
        strcpy(temp->coachName, coachNames[i]);
        strcpy(temp->role, roles[i]);
        temp->yearsOfExperience = experiences[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayCoachingStaff(struct Coach *p)
{
    if (!p)
    {
        printf("No coaching staff found\n");
        return;
    }
    while (p)
    {
        printf("Coach ID: %d | Name: %s | Role: %s | Years of Experience: %d\n",
               p->coachId, p->coachName, p->role, p->yearsOfExperience);
        p = p->next;
    }
```

```
}

void insertCoach(int coachId, char coachName[], char role[], int yearsOfExperience)
{
   struct Coach *temp, *current = first;
   temp = (struct Coach *)malloc(sizeof(struct Coach));
   temp->coachId = coachId;
   strcpy(temp->coachName, coachName);
   strcpy(temp->role, role);
   temp->yearsOfExperience = yearsOfExperience;
   temp->next = NULL;
   if (!first)
      first = temp;
   else
   {
      while (current->next)
         current = current->next;
      current->next = temp;
   }
}

void deleteCoach(int coachId)
{
   struct Coach *temp = first, *prev = NULL;
   if (first && first->coachId == coachId)
   {
      first = first->next;
```

```
        printf("Deleted coach: Coach ID: %d | Name: %s | Role: %s\n",
              temp->coachId, temp->coachName, temp->role);
        free(temp);
        return;
    }
    while (temp && temp->coachId != coachId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted coach: Coach ID: %d | Name: %s | Role: %s\n",
              temp->coachId, temp->coachName, temp->role);
        free(temp);
    }
    else
        printf("Coach ID not found\n");
}
```

Problem 12: Fan Club Membership Management

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

1. Create a membership list.
2. Insert a new member.
3. Delete a member who cancels their membership.

4. Display all current members.

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

struct Member
{
    int memberId;
    char memberName[100];
    int membershipDuration;
    struct Member *next;
} *first = NULL;

// Function prototypes
void createMembershipList(int memberIds[], char memberNames[][100], int durations[], int n);
void displayMembers(struct Member *p);
void insertMember(int memberId, char memberName[], int membershipDuration);
void deleteMember(int memberId);

int main()
{
    int option, memberId;
    char memberName[100];
    int membershipDuration;
    int memberIds[] = {1, 2, 3};
```

```
    char memberNames[][100] = {"JKL", "ABV", "BNM"};
    int durations[] = {1, 2, 3};
    createMembershipList(memberIds, memberNames, durations, 3);
    do
    {
       printf("\n--- Fan Club Membership Management ---\n");
       printf("1. Create a membership list\n");
       printf("2. Insert a new member\n");
       printf("3. Delete a member who cancels membership\n");
       printf("4. Display all current members\n");
       printf("5. Exit\n");
       printf("Enter the option: ");
       scanf("%d", &option);
       switch (option)
       {
          case 1: createMembershipList(memberIds, memberNames, durations,
3);
                  printf("Membership list created successfully\n");
                  break;
          case 2: printf("Enter member ID: ");
                  scanf("%d", &memberId);
                  printf("Enter member name: ");
                  scanf(" %[^\n]", memberName);
                  printf("Enter membership duration (years): ");
                  scanf("%d", &membershipDuration);
                  insertMember(memberId, memberName, membershipDuration);
                  printf("New member added successfully\n");
                  break;
```

```
            case 3: printf("Enter member ID to delete: ");
                    scanf("%d", &memberId);
                    deleteMember(memberId);
                    break;
            case 4: printf("Current Members:\n");
                    displayMembers(first);
                    break;
            case 5: printf("Exit\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    return 0;
}

void createMembershipList(int memberIds[], char memberNames[][100], int
durations[], int n)
{
    int i;
    struct Member *temp, *last;
    first = (struct Member *)malloc(sizeof(struct Member));
    first->memberId = memberIds[0];
    strcpy(first->memberName, memberNames[0]);
    first->membershipDuration = durations[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
```

```
        temp = (struct Member *)malloc(sizeof(struct Member));
        temp->memberId = memberIds[i];
        strcpy(temp->memberName, memberNames[i]);
        temp->membershipDuration = durations[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayMembers(struct Member *p)
{
    if (!p)
    {
        printf("No members found\n");
        return;
    }
    while (p)
    {
       printf("Member ID: %d | Name: %s | Membership Duration: %d years\n",
              p->memberId, p->memberName, p->membershipDuration);
        p = p->next;
    }
}

void insertMember(int memberId, char memberName[], int
membershipDuration)
{
```

```
    struct Member *temp, *current = first;
    temp = (struct Member *)malloc(sizeof(struct Member));
    temp->memberId = memberId;
    strcpy(temp->memberName, memberName);
    temp->membershipDuration = membershipDuration;
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteMember(int memberId)
{
    struct Member *temp = first, *prev = NULL;
    if (first && first->memberId == memberId)
    {
        first = first->next;
        printf("Deleted member: Member ID: %d | Name: %s\n",
                temp->memberId, temp->memberName);
        free(temp);
        return;
    }
```

```
    while (temp && temp->memberId != memberId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted member: Member ID: %d | Name: %s\n",
               temp->memberId, temp->memberName);
        free(temp);
    }
    else
        printf("Member ID not found\n");
}
```

Problem 13: Sports Event Scheduling

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

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

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

```
struct Event
{
   int eventId;
   char eventName[100];
   char eventDate[20];
   char eventLocation[100];
   struct Event *next;
} *first = NULL;

// Function prototypes
void createEventSchedule(int eventIds[], char eventNames[][100], char eventDates[][20], char eventLocations[][100], int n);
void displayEventSchedule(struct Event *p);
void insertEvent(int eventId, char eventName[], char eventDate[], char eventLocation[]);
void deleteEvent(int eventId);

int main()
{
   int option, eventId;
   char eventName[100], eventDate[20], eventLocation[100];
   int eventIds[] = {1, 2, 3};
    char eventNames[][100] = {"Football Match", "Basketball Tournament", "Tennis Championship"};
   char eventDates[][20] = {"2025-02-20", "2025-03-10", "2025-04-15"};
   char eventLocations[][100] = {"Stadium A", "Arena B", "Court C"};
    createEventSchedule(eventIds, eventNames, eventDates, eventLocations, 3);
```

```
    do
    {
        printf("\n--- Sports Event Scheduling ---\n");
        printf("1. Create an event schedule\n");
        printf("2. Insert a new event\n");
        printf("3. Delete a completed or canceled event\n");
        printf("4. Display the current event schedule\n");
        printf("5. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
                case 1: createEventSchedule(eventIds, eventNames, eventDates,
eventLocations, 3);
                    printf("Event schedule created successfully\n");
                    break;
            case 2: printf("Enter event ID: ");
                    scanf("%d", &eventId);
                    printf("Enter event name: ");
                    scanf(" %[^\n]", eventName);
                    printf("Enter event date (YYYY-MM-DD): ");
                    scanf(" %[^\n]", eventDate);
                    printf("Enter event location: ");
                    scanf(" %[^\n]", eventLocation);
                    insertEvent(eventId, eventName, eventDate, eventLocation);
                    printf("New event added successfully\n");
                    break;
            case 3: printf("Enter event ID to delete: ");
```

```
            scanf("%d", &eventId);
            deleteEvent(eventId);
            break;
        case 4: printf("Current Event Schedule:\n");
            displayEventSchedule(first);
            break;
        case 5: printf("Exit\n");
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 5);
   return 0;
}

void createEventSchedule(int eventIds[], char eventNames[][100], char eventDates[][20], char eventLocations[][100], int n)
{
   int i;
   struct Event *temp, *last;
   first = (struct Event *)malloc(sizeof(struct Event));
   first->eventId = eventIds[0];
   strcpy(first->eventName, eventNames[0]);
   strcpy(first->eventDate, eventDates[0]);
   strcpy(first->eventLocation, eventLocations[0]);
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
```

```
        temp = (struct Event *)malloc(sizeof(struct Event));
        temp->eventId = eventIds[i];
        strcpy(temp->eventName, eventNames[i]);
        strcpy(temp->eventDate, eventDates[i]);
        strcpy(temp->eventLocation, eventLocations[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}

void displayEventSchedule(struct Event *p)
{
    if (!p)
    {
        printf("No events found\n");
        return;
    }
    while (p)
    {
        printf("Event ID: %d | Name: %s | Date: %s | Location: %s\n",
                p->eventId, p->eventName, p->eventDate, p->eventLocation);
        p = p->next;
    }
}

void insertEvent(int eventId, char eventName[], char eventDate[], char
eventLocation[])
```

```
{
   struct Event *temp, *current = first;
   temp = (struct Event *)malloc(sizeof(struct Event));
   temp->eventId = eventId;
   strcpy(temp->eventName, eventName);
   strcpy(temp->eventDate, eventDate);
   strcpy(temp->eventLocation, eventLocation);
   temp->next = NULL;
   if (!first)
      first = temp;
   else
   {
      while (current->next)
         current = current->next;
      current->next = temp;
   }
}

void deleteEvent(int eventId)
{
   struct Event *temp = first, *prev = NULL;
   if (first && first->eventId == eventId)
   {
      first = first->next;
       printf("Deleted event: Event ID: %d | Name: %s | Date: %s | Location: %s\n",
                  temp->eventId, temp->eventName, temp->eventDate, temp->eventLocation);
```

```
        free(temp);
        return;
    }
    while (temp && temp->eventId != eventId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
         printf("Deleted event: Event ID: %d | Name: %s | Date: %s | Location: %s\n",
                    temp->eventId, temp->eventName, temp->eventDate, temp->eventLocation);
        free(temp);
    }
    else
        printf("Event ID not found\n");
}
```

Problem 14: Player Transfer Records

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

1. Create a transfer record list.
2. Insert a new transfer record.
3. Delete an outdated or erroneous transfer record.

4. Display all current transfer records.

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

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

// Function prototypes
void createTransferRecordList(int transferIds[], int playerIds[], char playerNames[][100], char fromTeams[][100], char toTeams[][100], char transferDates[][20], int n);
void displayTransferRecords(struct TransferRecord *p);
void insertTransferRecord(int transferId, int playerId, char playerName[], char fromTeam[], char toTeam[], char transferDate[]);
void deleteTransferRecord(int transferId);

int main()
{
```

```
    int option, transferId;
    int playerId;
    char playerName[100], fromTeam[100], toTeam[100], transferDate[20];

    int transferIds[] = {1, 2, 3};
    int playerIds[] = {101, 102, 103};
    char playerNames[][100] = {"JKL", "ABC", "BCD"};
    char fromTeams[][100] = {"Team A", "Team B", "Team C"};
    char toTeams[][100] = {"Team D", "Team E", "Team F"};
    char transferDates[][20] = {"2025-01-10", "2025-02-15", "2025-03-20"};
   createTransferRecordList(transferIds, playerIds, playerNames, fromTeams,
toTeams, transferDates, 3);
    do
    {
       printf("\n--- Player Transfer Records ---\n");
       printf("1. Create a transfer record list\n");
       printf("2. Insert a new transfer record\n");
       printf("3. Delete an outdated or erroneous transfer record\n");
       printf("4. Display all current transfer records\n");
       printf("5. Exit\n");
       printf("Enter the option: ");
       scanf("%d", &option);
       switch (option)
       {
          case 1: createTransferRecordList(transferIds, playerIds, playerNames,
fromTeams, toTeams, transferDates, 3);
                  printf("Transfer record list created successfully\n");
                  break;
```

```
            case 2: printf("Enter transfer ID: ");
                    scanf("%d", &transferId);
                    printf("Enter player ID: ");
                    scanf("%d", &playerId);
                    printf("Enter player name: ");
                    scanf(" %[^\n]", playerName);
                    printf("Enter from team: ");
                    scanf(" %[^\n]", fromTeam);
                    printf("Enter to team: ");
                    scanf(" %[^\n]", toTeam);
                    printf("Enter transfer date (YYYY-MM-DD): ");
                    scanf(" %[^\n]", transferDate);
                            insertTransferRecord(transferId, playerId, playerName,
fromTeam, toTeam, transferDate);
                    printf("New transfer record added successfully\n");
                    break;
            case 3: printf("Enter transfer ID to delete: ");
                    scanf("%d", &transferId);
                    deleteTransferRecord(transferId);
                    break;
            case 4: printf("Current Transfer Records:\n");
                    displayTransferRecords(first);
                    break;
            case 5: printf("Exit\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
```

```
    return 0;
}

void createTransferRecordList(int transferIds[], int playerIds[], char playerNames[][100], char fromTeams[][100], char toTeams[][100], char transferDates[][20], int n)
{
    int i;
    struct TransferRecord *temp, *last;
    first = (struct TransferRecord *)malloc(sizeof(struct TransferRecord));
    first->transferId = transferIds[0];
    first->playerId = playerIds[0];
    strcpy(first->playerName, playerNames[0]);
    strcpy(first->fromTeam, fromTeams[0]);
    strcpy(first->toTeam, toTeams[0]);
    strcpy(first->transferDate, transferDates[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct TransferRecord *)malloc(sizeof(struct TransferRecord));
        temp->transferId = transferIds[i];
        temp->playerId = playerIds[i];
        strcpy(temp->playerName, playerNames[i]);
        strcpy(temp->fromTeam, fromTeams[i]);
        strcpy(temp->toTeam, toTeams[i]);
        strcpy(temp->transferDate, transferDates[i]);
        temp->next = NULL;
```

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

void displayTransferRecords(struct TransferRecord *p)
{
    if (!p)
    {
        printf("No transfer records found\n");
        return;
    }
    while (p)
    {
        printf("Transfer ID: %d | Player ID: %d | Player: %s | From: %s | To: %s | Date: %s\n",
                p->transferId, p->playerId, p->playerName, p->fromTeam, p->toTeam, p->transferDate);
        p = p->next;
    }
}

void insertTransferRecord(int transferId, int playerId, char playerName[], char fromTeam[], char toTeam[], char transferDate[])
{
    struct TransferRecord *temp, *current = first;
    temp = (struct TransferRecord *)malloc(sizeof(struct TransferRecord));
    temp->transferId = transferId;
```

```
    temp->playerId = playerId;
    strcpy(temp->playerName, playerName);
    strcpy(temp->fromTeam, fromTeam);
    strcpy(temp->toTeam, toTeam);
    strcpy(temp->transferDate, transferDate);
    temp->next = NULL;
    if (!first)
        first = temp;
    else
    {
        while (current->next)
            current = current->next;
        current->next = temp;
    }
}

void deleteTransferRecord(int transferId)
{
    struct TransferRecord *temp = first, *prev = NULL;
    if (first && first->transferId == transferId)
    {
        first = first->next;
        printf("Deleted transfer record: Transfer ID: %d | Player: %s | From: %s
| To: %s | Date: %s\n",
                temp->transferId, temp->playerName, temp->fromTeam, temp-
>toTeam, temp->transferDate);
        free(temp);
        return;
```

```
    }
    while (temp && temp->transferId != transferId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
    {
        prev->next = temp->next;
        printf("Deleted transfer record: Transfer ID: %d | Player: %s | From: %s
| To: %s | Date: %s\n",
                temp->transferId, temp->playerName, temp->fromTeam, temp-
>toTeam, temp->transferDate);
        free(temp);
    }
    else
        printf("Transfer record not found\n");
}
```

Problem 15: Championship Points Tracker

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

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

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

struct PointsRecord
{
   int teamId;
   char teamName[100];
   int points;
   struct PointsRecord *next;
} *first = NULL;

// Function prototypes
void createPointsTrackerList(int teamIds[], char teamNames[][100], int points[], int n);
void displayPointsStandings(struct PointsRecord *p);
void insertPointsEntry(int teamId, char teamName[], int points);
void deletePointsEntry(int teamId);

int main()
{
   int option, teamId, points;
   char teamName[100];
   int teamIds[] = {1, 2, 3};
   char teamNames[][100] = {"Team A", "Team B", "Team C"};
   int pointsArray[] = {10, 15, 8};
   createPointsTrackerList(teamIds, teamNames, pointsArray, 3);
   do
```

```
{
    printf("\n--- Championship Points Tracker ---\n");
    printf("1. Create a points tracker list\n");
    printf("2. Insert a new points entry\n");
    printf("3. Delete an incorrect or outdated points entry\n");
    printf("4. Display all current points standings\n");
    printf("5. Exit\n");
    printf("Enter the option: ");
    scanf("%d", &option);
    switch (option)
    {
        case 1: createPointsTrackerList(teamIds, teamNames, pointsArray, 3);
                printf("Points tracker list created successfully\n");
                break;
        case 2: printf("Enter team ID: ");
                scanf("%d", &teamId);
                printf("Enter team name: ");
                scanf(" %[^\n]", teamName);
                printf("Enter team points: ");
                scanf("%d", &points);
                insertPointsEntry(teamId, teamName, points);
                printf("New points entry added successfully\n");
                break;
        case 3: printf("Enter team ID to delete: ");
                scanf("%d", &teamId);
                deletePointsEntry(teamId);
                break;
```

```
            case 4: printf("Current Points Standings:\n");
                    displayPointsStandings(first);
                    break;
            case 5: printf("Exit\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 5);
    return 0;
}

void createPointsTrackerList(int teamIds[], char teamNames[][100], int points[], int n)
{
    int i;
    struct PointsRecord *temp, *last;
    first = (struct PointsRecord *)malloc(sizeof(struct PointsRecord));
    first->teamId = teamIds[0];
    strcpy(first->teamName, teamNames[0]);
    first->points = points[0];
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++)
    {
        temp = (struct PointsRecord *)malloc(sizeof(struct PointsRecord));
        temp->teamId = teamIds[i];
        strcpy(temp->teamName, teamNames[i]);
        temp->points = points[i];
```

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

void displayPointsStandings(struct PointsRecord *p)
{
    if (!p)
    {
        printf("No points records found\n");
        return;
    }
    while (p)
    {
            printf("Team ID: %d | Team: %s | Points: %d\n", p->teamId, p->teamName, p->points);
        p = p->next;
    }
}

void insertPointsEntry(int teamId, char teamName[], int points)
{
    struct PointsRecord *temp, *current = first;
    temp = (struct PointsRecord *)malloc(sizeof(struct PointsRecord));
    temp->teamId = teamId;
    strcpy(temp->teamName, teamName);
    temp->points = points;
```

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

void deletePointsEntry(int teamId)
{
    struct PointsRecord *temp = first, *prev = NULL;
    if (first && first->teamId == teamId)
    {
        first = first->next;
         printf("Deleted points entry: Team ID: %d | Team: %s | Points: %d\n",
temp->teamId, temp->teamName, temp->points);
        free(temp);
        return;
    }
    while (temp && temp->teamId != teamId)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp)
```

```
    {
        prev->next = temp->next;
         printf("Deleted points entry: Team ID: %d | Team: %s | Points: %d\n",
temp->teamId, temp->teamName, temp->points);
        free(temp);
    }
    else
        printf("Points entry not found\n");
}
```