Day 17 programs

Problem 1: Inventory Management System

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

Operations:

Create an inventory list.

Insert a new raw material.

Delete a raw material from the inventory.

Display the current inventory.

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

// Define a structure for raw materials
typedef struct RawMaterial {
    char name[50];
    int quantity;
    struct RawMaterial* next;
} RawMaterial;

// Function to create a new raw material node
RawMaterial* createRawMaterial(char* name, int quantity) {
    RawMaterial* newMaterial = (RawMaterial*)malloc(sizeof(RawMaterial));
    strcpy(newMaterial->name, name);
    newMaterial->quantity = quantity;
    newMaterial->next = NULL;
    return newMaterial;
}
```

```
// Function to insert a new raw material into the inventory
void insertRawMaterial(RawMaterial** first, RawMaterial** last, char* name, int quantity) {
    RawMaterial* newMaterial = createRawMaterial(name, quantity);
    if (*first == NULL) {  // If the list is empty
        *first = *last = newMaterial;
    } else {
        (*last)->next = newMaterial;
        *last = newMaterial;
    }
    printf("Added: %s (Quantity: %d)\n", name, quantity);
}

// Function to delete a raw material from the inventory
void deleteRawMaterial(RawMaterial** first, RawMaterial** last, char* name) {
    RawMaterial *temp = *first, *prev = NULL;

    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {  // Found the material
            if (prev == NULL) {  // Deleting the first node
                *first = temp->next;
                if (*first == NULL) {  // List is now empty
                    *last = NULL;
                }
            } else {
                prev->next = temp->next;
                if (temp == *last) {  // Deleting the last node
                    *last = prev;
```

```
            }
        }
        free(temp);
        printf("Deleted: %s\n", name);
        return;
    }
    prev = temp;
    temp = temp->next;
  }
  printf("Material not found: %s\n", name);
}

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

  printf("Current Inventory:\n");
  RawMaterial* temp = first;
  while (temp != NULL) {
    printf(" - %s (Quantity: %d)\n", temp->name, temp->quantity);
    temp = temp->next;
  }
}

// Main function
int main() {
```

```
RawMaterial *first = NULL, *last = NULL;
int choice;
char name[50];
int quantity;

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

    switch (choice) {
        case 1:
            printf("Enter material name: ");
            scanf("%s", name);
            printf("Enter quantity: ");
            scanf("%d", &quantity);
            insertRawMaterial(&first, &last, name, quantity);
            break;

        case 2:
            printf("Enter material name to delete: ");
            scanf("%s", name);
            deleteRawMaterial(&first, &last, name);
            break;
```

```
            case 3:
                displayInventory(first);
                break;

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

            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Problem 2: Production Line Queue

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

Operations:

Create a production task queue.

Insert a new task into the queue.

Delete a completed task.

Display the current task queue.

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

// Define a structure for production tasks
```

```
typedef struct Task {
    char description[100];
    struct Task* next;
} Task;

// Function to create a new task node
Task* createTask(char* description) {
    Task* newTask = (Task*)malloc(sizeof(Task));
    strcpy(newTask->description, description);
    newTask->next = NULL;
    return newTask;
}

// Function to insert a new task into the queue
void enqueueTask(Task** first, Task** last, char* description) {
    Task* newTask = createTask(description);
    if (*last != NULL) {  // If the queue is not empty
        (*last)->next = newTask;
    } else {  // If the queue is empty
        *first = newTask;
    }
    *last = newTask;  // Update the tail
    printf("Added Task: %s\n", description);
}

// Function to delete a completed task from the queue
void dequeueTask(Task** first, Task** last) {
    if (*first == NULL) {
        printf("No tasks to complete. Queue is empty.\n");
```

```
        return;
    }

    Task* temp = *first;
    *first = (*first)->next;  // Move the head to the next task
    if (*first == NULL) {  // If the queue becomes empty
        *last = NULL;
    }
    printf("Completed Task: %s\n", temp->description);
    free(temp);
}

// Function to display the current task queue
void displayQueue(Task* first) {
    if (first == NULL) {
        printf("The task queue is empty.\n");
        return;
    }

    printf("Current Task Queue:\n");
    Task* temp = first;
    while (temp != NULL) {
        printf(" - %s\n", temp->description);
        temp = temp->next;
    }
}

// Main function
int main() {
```

```
Task *first = NULL, *last = NULL;
int choice;
char description[100];

do {
    printf("\nProduction Line Queue Management\n");
    printf("1. Add New Task\n");
    printf("2. Complete Task\n");
    printf("3. Display Task Queue\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter task description (single word): ");
            scanf("%s", description);  // Reads a single word as input
            enqueueTask(&first, &last, description);
            break;

        case 2:
            dequeueTask(&first, &last);
            break;

        case 3:
            displayQueue(first);
            break;

        case 4:
```

```
            printf("Exiting system.\n");
            break;

        default:
            printf("Invalid choice. Try again.\n");
    }
} while (choice != 4);

return 0;
}
```

Problem 3: Machine Maintenance Schedule

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

Operations:

Create a maintenance schedule.

Insert a new maintenance task.

Delete a completed maintenance task.

Display the maintenance schedule.

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

// Define a structure for maintenance tasks
typedef struct MaintenanceTask {
    char machineName[50];
    char taskDescription[100];
    struct MaintenanceTask* next;
} MaintenanceTask;
```

```
// Function to create a new maintenance task node
MaintenanceTask* createTask(char* machineName, char* taskDescription) {
    MaintenanceTask* newTask =
(MaintenanceTask*)malloc(sizeof(MaintenanceTask));
    strcpy(newTask->machineName, machineName);
    strcpy(newTask->taskDescription, taskDescription);
    newTask->next = NULL;
    return newTask;
}

// Function to insert a new maintenance task into the schedule
void insertTask(MaintenanceTask** first, MaintenanceTask** last, char*
machineName, char* taskDescription) {
    MaintenanceTask* newTask = createTask(machineName, taskDescription);
    if (*last != NULL) {  // If the list is not empty
        (*last)->next = newTask;
    } else {  // If the list is empty
        *first = newTask;
    }
    *last = newTask;  // Update the tail
    printf("Added Task: [%s] %s\n", machineName, taskDescription);
}

// Function to delete a completed maintenance task from the schedule
void deleteTask(MaintenanceTask** first, MaintenanceTask** last, char*
machineName) {
    MaintenanceTask *temp = *first, *prev = NULL;

    while (temp != NULL) {
```

```
        if (strcmp(temp->machineName, machineName) == 0) {  // Found the task
            if (prev == NULL) {  // Deleting the first node
                *first = temp->next;
                if (*first == NULL) {  // If the list becomes empty
                    *last = NULL;
                }
            } else {
                prev->next = temp->next;
                if (temp == *last) {  // Deleting the last node
                    *last = prev;
                }
            }
            printf("Completed Task: [%s] %s\n", temp->machineName, temp->taskDescription);
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    printf("Task for machine '%s' not found.\n", machineName);
}

// Function to display the maintenance schedule
void displaySchedule(MaintenanceTask* first) {
    if (first == NULL) {
        printf("The maintenance schedule is empty.\n");
        return;
    }
```

```
    printf("Current Maintenance Schedule:\n");
    MaintenanceTask* temp = first;
    while (temp != NULL) {
        printf(" - Machine: %s | Task: %s\n", temp->machineName, temp-
>taskDescription);
        temp = temp->next;
    }
}

// Main function
int main() {
    MaintenanceTask *first = NULL, *last = NULL;
    int choice;
    char machineName[50];
    char taskDescription[100];

    do {
        printf("\nMachine Maintenance Schedule\n");
        printf("1. Add Maintenance Task\n");
        printf("2. Complete Maintenance Task\n");
        printf("3. Display Maintenance Schedule\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter machine name: ");
```

```
            scanf("%s", machineName);
            printf("Enter task description: ");
            scanf(" %[^\n]", taskDescription);  // Reads entire line including spaces
            insertTask(&first, &last, machineName, taskDescription);
            break;

        case 2:
            printf("Enter machine name to mark task as complete: ");
            scanf("%s", machineName);
            deleteTask(&first, &last, machineName);
            break;

        case 3:
            displaySchedule(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 4: Employee Shift Management

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

Operations:

Create a shift schedule.

Insert a new shift.

Delete a completed or canceled shift.

Display the current shift schedule.

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

// Define a structure for employee shift
typedef struct Shift {
    char employeeName[50];
    char shiftTime[20];
    struct Shift* next;
} Shift;

// Function to create a new shift node
Shift* createShift(char* employeeName, char* shiftTime) {
    Shift* newShift = (Shift*)malloc(sizeof(Shift));
    strcpy(newShift->employeeName, employeeName);
    strcpy(newShift->shiftTime, shiftTime);
    newShift->next = NULL;
    return newShift;
}

// Function to insert a new shift into the schedule
void insertShift(Shift** first, Shift** last, char* employeeName, char* shiftTime) {
```

```
    Shift* newShift = createShift(employeeName, shiftTime);
    if (*last != NULL) {  // If the list is not empty
        (*last)->next = newShift;
    } else {  // If the list is empty
        *first = newShift;
    }
    *last = newShift;  // Update the tail
    printf("Added Shift: %s | %s\n", employeeName, shiftTime);
}

// Function to delete a completed or canceled shift
void deleteShift(Shift** first, Shift** last, char* employeeName) {
    Shift *temp = *first, *prev = NULL;

    while (temp != NULL) {
        if (strcmp(temp->employeeName, employeeName) == 0) {  // Found the shift
            if (prev == NULL) {  // Deleting the first node
                *first = temp->next;
                if (*first == NULL) {  // If the list becomes empty
                    *last = NULL;
                }
            } else {
                prev->next = temp->next;
                if (temp == *last) {  // Deleting the last node
                    *last = prev;
                }
            }
            printf("Completed/Cancelled Shift: %s | %s\n", temp->employeeName, temp-
>shiftTime);
```

```
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    printf("Shift for employee '%s' not found.\n", employeeName);
}

// Function to display the current shift schedule
void displaySchedule(Shift* first) {
    if (first == NULL) {
        printf("The shift schedule is empty.\n");
        return;
    }

    printf("Current Shift Schedule:\n");
    Shift* temp = first;
    while (temp != NULL) {
        printf(" - Employee: %s | Shift Time: %s\n", temp->employeeName, temp->shiftTime);
        temp = temp->next;
    }
}

// Main function
int main() {
    Shift *first = NULL, *last = NULL;
    int choice;
```

```
char employeeName[50];
char shiftTime[20];

do {
    printf("\nEmployee Shift Management\n");
    printf("1. Add Shift\n");
    printf("2. Complete/Cancel Shift\n");
    printf("3. Display Shift Schedule\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter employee name: ");
            scanf("%s", employeeName);
            printf("Enter shift time: ");
            scanf("%s", shiftTime);
            insertShift(&first, &last, employeeName, shiftTime);
            break;

        case 2:
            printf("Enter employee name to cancel/complete shift: ");
            scanf("%s", employeeName);
            deleteShift(&first, &last, employeeName);
            break;

        case 3:
            displaySchedule(first);
```

```
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 5: Order Processing System

Description: Implement a linked list to track customer orders.

Operations:

Create an order list.

Insert a new customer order.

Delete a completed or canceled order.

Display all current orders.

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

// Define a structure for customer orders
typedef struct Order {
   int orderID;
```

```
    char customerName[50];
    char orderDetails[100];
    struct Order* next;
} Order;

// Function to create a new order node
Order* createOrder(int orderID, char* customerName, char* orderDetails) {
    Order* newOrder = (Order*)malloc(sizeof(Order));
    newOrder->orderID = orderID;
    strcpy(newOrder->customerName, customerName);
    strcpy(newOrder->orderDetails, orderDetails);
    newOrder->next = NULL;
    return newOrder;
}

// Function to insert a new customer order into the list
void insertOrder(Order** first, Order** last, int orderID, char* customerName, char*
orderDetails) {
    Order* newOrder = createOrder(orderID, customerName, orderDetails);
    if (*last != NULL) {  // If the list is not empty
        (*last)->next = newOrder;
    } else {  // If the list is empty
        *first = newOrder;
    }
    *last = newOrder;  // Update the tail
    printf("Added Order: ID %d | Customer: %s | Details: %s\n", orderID,
customerName, orderDetails);
}

// Function to delete a completed or canceled order from the list
```

```
void deleteOrder(Order** first, Order** last, int orderID) {
    Order *temp = *first, *prev = NULL;

    while (temp != NULL) {
        if (temp->orderID == orderID) {  // Found the order
            if (prev == NULL) {  // Deleting the first node
                *first = temp->next;
                if (*first == NULL) {  // If the list becomes empty
                    *last = NULL;
                }
            } else {
                prev->next = temp->next;
                if (temp == *last) {  // Deleting the last node
                    *last = prev;
                }
            }
            printf("Completed/Cancelled Order: ID %d | Customer: %s | Details: %s\n",
temp->orderID, temp->customerName, temp->orderDetails);
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    printf("Order ID %d not found.\n", orderID);
}

// Function to display all current orders
void displayOrders(Order* first) {
```

```
    if (first == NULL) {
        printf("No current orders.\n");
        return;
    }

    printf("Current Orders:\n");
    Order* temp = first;
    while (temp != NULL) {
        printf(" - Order ID: %d | Customer: %s | Order: %s\n", temp->orderID, temp->customerName, temp->orderDetails);
        temp = temp->next;
    }
}

// Main function
int main() {
    Order *first = NULL, *last = NULL;
    int choice, orderID;
    char customerName[50];
    char orderDetails[100];

    do {
        printf("\nOrder Processing System\n");
        printf("1. Add New Order\n");
        printf("2. Complete/Cancel Order\n");
        printf("3. Display All Orders\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("Enter order ID: ");
        scanf("%d", &orderID);
        printf("Enter customer name: ");
        scanf("%s", customerName);
        printf("Enter order details: ");
        scanf(" %[^\n]", orderDetails);  // Reads entire line
        insertOrder(&first, &last, orderID, customerName, orderDetails);
        break;

    case 2:
        printf("Enter order ID to cancel/complete: ");
        scanf("%d", &orderID);
        deleteOrder(&first, &last, orderID);
        break;

    case 3:
        displayOrders(first);
        break;

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

    default:
        printf("Invalid choice. Try again.\n");
}
```

```
    } while (choice != 4);

    return 0;
}
```

Problem 6: Tool Tracking System

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

Operations:

Create a tool tracking list.

Insert a new tool entry.

Delete a tool that is no longer in use.

Display all tools currently tracked.

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

// Define a structure for tool entry
typedef struct Tool {
    int toolID;
    char toolName[50];
    int quantity;
    struct Tool* next;
} Tool;

// Function to create a new tool entry
Tool* createTool(int toolID, char* toolName, int quantity) {
    Tool* newTool = (Tool*)malloc(sizeof(Tool));
    newTool->toolID = toolID;
```

```
    strcpy(newTool->toolName, toolName);
    newTool->quantity = quantity;
    newTool->next = NULL;
    return newTool;
}

// Function to insert a new tool entry into the tracking list
void insertTool(Tool** first, int toolID, char* toolName, int quantity) {
    Tool* newTool = createTool(toolID, toolName, quantity);
    if (*first == NULL) {  // If the list is empty
        *first = newTool;
    } else {
        Tool* temp = *first;
        // Traverse to the end of the list
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTool;  // Insert the new tool at the end
    }
    printf("Added Tool: ID %d | Name: %s | Quantity: %d\n", toolID, toolName,
quantity);
}

// Function to delete a tool that is no longer in use
void deleteTool(Tool** first, int toolID) {
    Tool *temp = *first, *prev = NULL;

    // If the tool to delete is the first node
    if (temp != NULL && temp->toolID == toolID) {
```

```
        *first = temp->next;
        free(temp);
        printf("Removed Tool with ID %d\n", toolID);
        return;
    }

    // Traverse the list to find the tool to delete
    while (temp != NULL && temp->toolID != toolID) {
        prev = temp;
        temp = temp->next;
    }

    // If the tool was not found
    if (temp == NULL) {
        printf("Tool with ID %d not found.\n", toolID);
        return;
    }

    // Unlink the node from the list
    prev->next = temp->next;
    free(temp);
    printf("Removed Tool with ID %d\n", toolID);
}

// Function to display all currently tracked tools
void displayTools(Tool* first) {
    if (first == NULL) {
        printf("No tools are currently tracked.\n");
        return;
```

```
    }

    printf("Currently Tracked Tools:\n");
    Tool* temp = first;
    while (temp != NULL) {
        printf(" - Tool ID: %d | Name: %s | Quantity: %d\n", temp->toolID, temp->toolName, temp->quantity);
        temp = temp->next;
    }
}

// Main function
int main() {
    Tool* first = NULL;
    int choice, toolID, quantity;
    char toolName[50];

    do {
        printf("\nTool Tracking System\n");
        printf("1. Add New Tool\n");
        printf("2. Remove Tool\n");
        printf("3. Display All Tracked Tools\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter tool ID: ");
```

```
            scanf("%d", &toolID);
            printf("Enter tool name: ");
            scanf("%s", toolName);
            printf("Enter tool quantity: ");
            scanf("%d", &quantity);
            insertTool(&first, toolID, toolName, quantity);
            break;

        case 2:
            printf("Enter tool ID to remove: ");
            scanf("%d", &toolID);
            deleteTool(&first, toolID);
            break;

        case 3:
            displayTools(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 7: Product Assembly Line

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

Operations:

Create an assembly line stage list.

Insert a new stage.

Delete a completed stage.

Display the current assembly stages.

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

// Define a structure for assembly stages
typedef struct Stage {
    int stageID;
    char stageName[50];
    struct Stage* next;
} Stage;

// Function to create a new assembly stage
Stage* createStage(int stageID, char* stageName) {
    Stage* newStage = (Stage*)malloc(sizeof(Stage));
    newStage->stageID = stageID;
    strcpy(newStage->stageName, stageName);
    newStage->next = NULL;
    return newStage;
}
```

```
// Function to insert a new stage into the assembly line
void insertStage(Stage** first, Stage** last, int stageID, char* stageName) {
    Stage* newStage = createStage(stageID, stageName);
    if (*last != NULL) {  // If the list is not empty
        (*last)->next = newStage;
    } else {  // If the list is empty
        *first = newStage;
    }
    *last = newStage;  // Update the last pointer
    printf("Added Stage: ID %d | Name: %s\n", stageID, stageName);
}

// Function to delete a completed stage from the assembly line
void deleteStage(Stage** first, Stage** last, int stageID) {
    Stage *temp = *first, *prev = NULL;

    // If the stage to delete is the first node
    if (temp != NULL && temp->stageID == stageID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {  // If the list becomes empty, update last
            *last = NULL;
        }
        printf("Removed Stage with ID %d\n", stageID);
        return;
    }

    // Traverse the list to find the stage to delete
    while (temp != NULL && temp->stageID != stageID) {
```

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

    // If the stage was not found
    if (temp == NULL) {
        printf("Stage with ID %d not found.\n", stageID);
        return;
    }

    // Unlink the node from the list
    prev->next = temp->next;
    if (temp == *last) {  // If it's the last node
        *last = prev;
    }
    free(temp);
    printf("Removed Stage with ID %d\n", stageID);
}

// Function to display all currently tracked stages
void displayStages(Stage* first) {
    if (first == NULL) {
        printf("No stages are currently in the assembly line.\n");
        return;
    }

    printf("Current Assembly Stages:\n");
    Stage* temp = first;
    while (temp != NULL) {
```

```
        printf(" - Stage ID: %d | Name: %s\n", temp->stageID, temp->stageName);
        temp = temp->next;
    }
}

// Main function
int main() {
    Stage* first = NULL, *last = NULL;
    int choice, stageID;
    char stageName[50];

    do {
        printf("\nProduct Assembly Line Management\n");
        printf("1. Add New Assembly Stage\n");
        printf("2. Remove Completed Stage\n");
        printf("3. Display Current Assembly Stages\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter stage ID: ");
                scanf("%d", &stageID);
                printf("Enter stage name: ");
                scanf("%s", stageName);
                insertStage(&first, &last, stageID, stageName);
                break;
```

```
            case 2:
                printf("Enter stage ID to remove: ");
                scanf("%d", &stageID);
                deleteStage(&first, &last, stageID);
                break;

            case 3:
                displayStages(first);
                break;

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

            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Problem 8: Quality Control Checklist

Description: Implement a linked list to manage a quality control checklist.

Operations:

Create a quality control checklist.

Insert a new checklist item.

Delete a completed or outdated checklist item.

Display the current quality control checklist.

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

// Define a structure for checklist items
typedef struct ChecklistItem {
    int itemID;
    char itemName[50];
    struct ChecklistItem* next;
} ChecklistItem;

// Function to create a new checklist item
ChecklistItem* createItem(int itemID, char* itemName) {
    ChecklistItem* newItem = (ChecklistItem*)malloc(sizeof(ChecklistItem));
    newItem->itemID = itemID;
    strcpy(newItem->itemName, itemName);
    newItem->next = NULL;
    return newItem;
}

// Function to insert a new checklist item into the quality control checklist
void insertItem(ChecklistItem** first, ChecklistItem** last, int itemID, char* itemName)
{
    ChecklistItem* newItem = createItem(itemID, itemName);
    if (*last != NULL) {  // If the list is not empty
        (*last)->next = newItem;
    } else {  // If the list is empty
        *first = newItem;
```

```
    }
    *last = newItem;  // Update the last pointer
    printf("Added Item: ID %d | Name: %s\n", itemID, itemName);
}

// Function to delete a completed or outdated checklist item
void deleteItem(ChecklistItem** first, ChecklistItem** last, int itemID) {
    ChecklistItem *temp = *first, *prev = NULL;

    // If the item to delete is the first node
    if (temp != NULL && temp->itemID == itemID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {  // If the list becomes empty, update last
            *last = NULL;
        }
        printf("Removed Item with ID %d\n", itemID);
        return;
    }

    // Traverse the list to find the item to delete
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }

    // If the item was not found
    if (temp == NULL) {
        printf("Item with ID %d not found.\n", itemID);
```

```
        return;
    }

    // Unlink the node from the list
    prev->next = temp->next;
    if (temp == *last) {  // If it's the last node
        *last = prev;
    }
    free(temp);
    printf("Removed Item with ID %d\n", itemID);
}

// Function to display the current quality control checklist
void displayChecklist(ChecklistItem* first) {
    if (first == NULL) {
        printf("No items in the checklist.\n");
        return;
    }

    printf("Current Quality Control Checklist:\n");
    ChecklistItem* temp = first;
    while (temp != NULL) {
        printf(" - Item ID: %d | Name: %s\n", temp->itemID, temp->itemName);
        temp = temp->next;
    }
}

// Main function
int main() {
```

```
ChecklistItem* first = NULL, *last = NULL;
int choice, itemID;
char itemName[50];

do {
    printf("\nQuality Control Checklist Management\n");
    printf("1. Add New Checklist Item\n");
    printf("2. Remove Completed or Outdated Item\n");
    printf("3. Display Current Quality Control Checklist\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter item ID: ");
            scanf("%d", &itemID);
            printf("Enter item name: ");
            scanf("%s", itemName);
            insertItem(&first, &last, itemID, itemName);
            break;

        case 2:
            printf("Enter item ID to remove: ");
            scanf("%d", &itemID);
            deleteItem(&first, &last, itemID);
            break;

        case 3:
```

```
            displayChecklist(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 9: Supplier Management System

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

Operations:

Create a supplier list.

Insert a new supplier.

Delete an inactive or outdated supplier.

Display all current suppliers.

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

typedef struct Supplier {
   int supplierID;
```

```
    char supplierName[50];
    char supplierContact[50];
    struct Supplier* next;
} Supplier;

Supplier* createSupplier(int supplierID, char* supplierName, char* supplierContact) {
    Supplier* newSupplier = (Supplier*)malloc(sizeof(Supplier));
    newSupplier->supplierID = supplierID;
    strcpy(newSupplier->supplierName, supplierName);
    strcpy(newSupplier->supplierContact, supplierContact);
    newSupplier->next = NULL;
    return newSupplier;
}

void insertSupplier(Supplier** first, Supplier** last, int supplierID, char*
supplierName, char* supplierContact) {
    Supplier* newSupplier = createSupplier(supplierID, supplierName,
supplierContact);
    if (*last != NULL) {
        (*last)->next = newSupplier;
    } else {
        *first = newSupplier;
    }
    *last = newSupplier;
    printf("Added Supplier: ID %d | Name: %s | Contact: %s\n", supplierID,
supplierName, supplierContact);
}

void deleteSupplier(Supplier** first, Supplier** last, int supplierID) {
    Supplier *temp = *first, *prev = NULL;
```

```
    if (temp != NULL && temp->supplierID == supplierID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Removed Supplier with ID %d\n", supplierID);
        return;
    }
    while (temp != NULL && temp->supplierID != supplierID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Supplier with ID %d not found.\n", supplierID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Removed Supplier with ID %d\n", supplierID);
}

void displaySuppliers(Supplier* first) {
    if (first == NULL) {
        printf("No suppliers in the list.\n");
        return;
```

```
    }
    printf("Current Supplier List:\n");
    Supplier* temp = first;
    while (temp != NULL) {
        printf(" - Supplier ID: %d | Name: %s | Contact: %s\n", temp->supplierID, temp->supplierName, temp->supplierContact);
        temp = temp->next;
    }
}

int main() {
    Supplier* first = NULL, *last = NULL;
    int choice, supplierID;
    char supplierName[50], supplierContact[50];

    do {
        printf("\nSupplier Management System\n");
        printf("1. Add New Supplier\n");
        printf("2. Remove Inactive or Outdated Supplier\n");
        printf("3. Display Current Supplier List\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter supplier ID: ");
                scanf("%d", &supplierID);
                printf("Enter supplier name: ");
```

```
            scanf("%s", supplierName);
            printf("Enter supplier contact: ");
            scanf("%s", supplierContact);
            insertSupplier(&first, &last, supplierID, supplierName, supplierContact);
            break;

        case 2:
            printf("Enter supplier ID to remove: ");
            scanf("%d", &supplierID);
            deleteSupplier(&first, &last, supplierID);
            break;

        case 3:
            displaySuppliers(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
    }
  } while (choice != 4);

  return 0;
}
```

Problem 10: Manufacturing Project Timeline

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

Operations:

Create a project timeline.

Insert a new project milestone.

Delete a completed milestone.

Display the current project timeline.

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

typedef struct Milestone {
    int milestoneID;
    char milestoneName[50];
    char milestoneDate[20];
    struct Milestone* next;
} Milestone;

Milestone* createMilestone(int milestoneID, char* milestoneName, char*
milestoneDate) {
    Milestone* newMilestone = (Milestone*)malloc(sizeof(Milestone));
    newMilestone->milestoneID = milestoneID;
    strcpy(newMilestone->milestoneName, milestoneName);
    strcpy(newMilestone->milestoneDate, milestoneDate);
    newMilestone->next = NULL;
    return newMilestone;
}

void insertMilestone(Milestone** first, Milestone** last, int milestoneID, char*
milestoneName, char* milestoneDate) {
```

```
    Milestone* newMilestone = createMilestone(milestoneID, milestoneName,
milestoneDate);
    if (*last != NULL) {
        (*last)->next = newMilestone;
    } else {
        *first = newMilestone;
    }
    *last = newMilestone;
    printf("Added Milestone: ID %d | Name: %s | Date: %s\n", milestoneID,
milestoneName, milestoneDate);
}

void deleteMilestone(Milestone** first, Milestone** last, int milestoneID) {
    Milestone *temp = *first, *prev = NULL;
    if (temp != NULL && temp->milestoneID == milestoneID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Removed Milestone with ID %d\n", milestoneID);
        return;
    }
    while (temp != NULL && temp->milestoneID != milestoneID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Milestone with ID %d not found.\n", milestoneID);
        return;
```

```
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Removed Milestone with ID %d\n", milestoneID);
}

void displayMilestones(Milestone* first) {
    if (first == NULL) {
        printf("No milestones in the project timeline.\n");
        return;
    }
    printf("Current Project Timeline:\n");
    Milestone* temp = first;
    while (temp != NULL) {
        printf(" - Milestone ID: %d | Name: %s | Date: %s\n", temp->milestoneID, temp->milestoneName, temp->milestoneDate);
        temp = temp->next;
    }
}

int main() {
    Milestone* first = NULL, *last = NULL;
    int choice, milestoneID;
    char milestoneName[50], milestoneDate[20];

    do {
```

```
printf("\nManufacturing Project Timeline\n");
printf("1. Add New Milestone\n");
printf("2. Remove Completed Milestone\n");
printf("3. Display Current Project Timeline\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter milestone ID: ");
        scanf("%d", &milestoneID);
        printf("Enter milestone name: ");
        scanf("%s", milestoneName);
        printf("Enter milestone date (YYYY-MM-DD): ");
        scanf("%s", milestoneDate);
        insertMilestone(&first, &last, milestoneID, milestoneName, milestoneDate);
        break;

    case 2:
        printf("Enter milestone ID to remove: ");
        scanf("%d", &milestoneID);
        deleteMilestone(&first, &last, milestoneID);
        break;

    case 3:
        displayMilestones(first);
        break;
```

```
            case 4:
                printf("Exiting system.\n");
                break;

            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Problem 11: Warehouse Storage Management

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

Operations:

Create a storage list.

Insert a new storage entry.

Delete a storage entry when goods are shipped.

Display the current warehouse storage.

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

typedef struct Storage {
    int itemID;
    char itemName[50];
    int quantity;
    struct Storage* next;
```

```
} Storage;

Storage* createStorageEntry(int itemID, char* itemName, int quantity) {
    Storage* newEntry = (Storage*)malloc(sizeof(Storage));
    newEntry->itemID = itemID;
    strcpy(newEntry->itemName, itemName);
    newEntry->quantity = quantity;
    newEntry->next = NULL;
    return newEntry;
}

void insertStorageEntry(Storage** first, Storage** last, int itemID, char* itemName,
int quantity) {
    Storage* newEntry = createStorageEntry(itemID, itemName, quantity);
    if (*last != NULL) {
        (*last)->next = newEntry;
    } else {
        *first = newEntry;
    }
    *last = newEntry;
    printf("Added Storage Entry: Item ID %d | Name: %s | Quantity: %d\n", itemID,
itemName, quantity);
}

void deleteStorageEntry(Storage** first, Storage** last, int itemID) {
    Storage *temp = *first, *prev = NULL;
    if (temp != NULL && temp->itemID == itemID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
```

```
            *last = NULL;
        }
        printf("Removed Storage Entry for Item ID %d\n", itemID);
        return;
    }
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Storage Entry for Item ID %d not found.\n", itemID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Removed Storage Entry for Item ID %d\n", itemID);
}

void displayStorage(Storage* first) {
    if (first == NULL) {
        printf("Warehouse storage is empty.\n");
        return;
    }
    printf("Current Warehouse Storage:\n");
    Storage* temp = first;
    while (temp != NULL) {
```

```
        printf(" - Item ID: %d | Name: %s | Quantity: %d\n", temp->itemID, temp->itemName, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    Storage* first = NULL, *last = NULL;
    int choice, itemID, quantity;
    char itemName[50];

    do {
        printf("\nWarehouse Storage Management\n");
        printf("1. Add New Storage Entry\n");
        printf("2. Remove Storage Entry when Goods are Shipped\n");
        printf("3. Display Current Warehouse Storage\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter item ID: ");
                scanf("%d", &itemID);
                printf("Enter item name: ");
                scanf("%s", itemName);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                insertStorageEntry(&first, &last, itemID, itemName, quantity);
```

```
            break;

        case 2:
            printf("Enter item ID to remove: ");
            scanf("%d", &itemID);
            deleteStorageEntry(&first, &last, itemID);
            break;

        case 3:
            displayStorage(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 12: Machine Parts Inventory

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

Operations:

Create a parts inventory list.

Insert a new part.

Delete a part that is used up or obsolete.

Display the current parts inventory.

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

typedef struct Part {
    int partID;
    char partName[50];
    int quantity;
    struct Part* next;
} Part;

Part* createPart(int partID, char* partName, int quantity) {
    Part* newPart = (Part*)malloc(sizeof(Part));
    newPart->partID = partID;
    strcpy(newPart->partName, partName);
    newPart->quantity = quantity;
    newPart->next = NULL;
    return newPart;
}

void insertPart(Part** first, Part** last, int partID, char* partName, int quantity) {
    Part* newPart = createPart(partID, partName, quantity);
    if (*last != NULL) {
        (*last)->next = newPart;
    } else {
        *first = newPart;
```

```
    }
    *last = newPart;
    printf("Added Part: Part ID %d | Name: %s | Quantity: %d\n", partID, partName,
quantity);
}

void deletePart(Part** first, Part** last, int partID) {
    Part *temp = *first, *prev = NULL;
    if (temp != NULL && temp->partID == partID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Removed Part with ID %d\n", partID);
        return;
    }
    while (temp != NULL && temp->partID != partID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Part with ID %d not found.\n", partID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
```

```
    free(temp);
    printf("Removed Part with ID %d\n", partID);
}

void displayParts(Part* first) {
    if (first == NULL) {
        printf("Machine parts inventory is empty.\n");
        return;
    }
    printf("Current Parts Inventory:\n");
    Part* temp = first;
    while (temp != NULL) {
        printf(" - Part ID: %d | Name: %s | Quantity: %d\n", temp->partID, temp->partName, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    Part* first = NULL, *last = NULL;
    int choice, partID, quantity;
    char partName[50];

    do {
        printf("\nMachine Parts Inventory Management\n");
        printf("1. Add New Part\n");
        printf("2. Remove Part (Used/Obsolete)\n");
        printf("3. Display Current Parts Inventory\n");
        printf("4. Exit\n");
```

```
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter part ID: ");
        scanf("%d", &partID);
        printf("Enter part name: ");
        scanf("%s", partName);
        printf("Enter quantity: ");
        scanf("%d", &quantity);
        insertPart(&first, &last, partID, partName, quantity);
        break;

    case 2:
        printf("Enter part ID to remove: ");
        scanf("%d", &partID);
        deletePart(&first, &last, partID);
        break;

    case 3:
        displayParts(first);
        break;

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

    default:
```

```
            printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Problem 13: Packaging Line Schedule

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

Operations:

Create a packaging task schedule.

Insert a new packaging task.

Delete a completed packaging task.

Display the current packaging schedule.

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

typedef struct Task {
    int taskID;
    char taskName[50];
    int quantity;
    struct Task* next;
} Task;

Task* createTask(int taskID, char* taskName, int quantity) {
    Task* newTask = (Task*)malloc(sizeof(Task));
    newTask->taskID = taskID;
```

```
    strcpy(newTask->taskName, taskName);
    newTask->quantity = quantity;
    newTask->next = NULL;
    return newTask;
}

void insertTask(Task** first, Task** last, int taskID, char* taskName, int quantity) {
    Task* newTask = createTask(taskID, taskName, quantity);
    if (*last != NULL) {
        (*last)->next = newTask;
    } else {
        *first = newTask;
    }
    *last = newTask;
    printf("Added Packaging Task: Task ID %d | Name: %s | Quantity: %d\n", taskID,
taskName, quantity);
}

void deleteTask(Task** first, Task** last, int taskID) {
    Task *temp = *first, *prev = NULL;
    if (temp != NULL && temp->taskID == taskID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Removed Task with ID %d\n", taskID);
        return;
    }
```

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

void displayTasks(Task* first) {
    if (first == NULL) {
        printf("Packaging schedule is empty.\n");
        return;
    }
    printf("Current Packaging Task Schedule:\n");
    Task* temp = first;
    while (temp != NULL) {
        printf(" - Task ID: %d | Name: %s | Quantity: %d\n", temp->taskID, temp-
>taskName, temp->quantity);
        temp = temp->next;
    }
}
```

```
int main() {
    Task* first = NULL, *last = NULL;
    int choice, taskID, quantity;
    char taskName[50];

    do {
        printf("\nPackaging Line Schedule Management\n");
        printf("1. Add New Packaging Task\n");
        printf("2. Remove Completed Packaging Task\n");
        printf("3. Display Current Packaging Task Schedule\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter task ID: ");
                scanf("%d", &taskID);
                printf("Enter task name: ");
                scanf("%s", taskName);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                insertTask(&first, &last, taskID, taskName, quantity);
                break;

            case 2:
                printf("Enter task ID to remove: ");
                scanf("%d", &taskID);
```

```
            deleteTask(&first, &last, taskID);
            break;

        case 3:
            displayTasks(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

Problem 14: Production Defect Tracking

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

Operations:

Create a defect tracking list.

Insert a new defect report.

Delete a resolved defect.

Display all current defects.

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

```
#include <string.h>

typedef struct Defect {
    int defectID;
    char defectDescription[100];
    char status[20];
    struct Defect* next;
} Defect;

Defect* createDefect(int defectID, char* defectDescription, char* status) {
    Defect* newDefect = (Defect*)malloc(sizeof(Defect));
    newDefect->defectID = defectID;
    strcpy(newDefect->defectDescription, defectDescription);
    strcpy(newDefect->status, status);
    newDefect->next = NULL;
    return newDefect;
}

void insertDefect(Defect** first, Defect** last, int defectID, char* defectDescription,
char* status) {
    Defect* newDefect = createDefect(defectID, defectDescription, status);
    if (*last != NULL) {
        (*last)->next = newDefect;
    } else {
        *first = newDefect;
    }
    *last = newDefect;
    printf("Inserted Defect Report: ID %d | Description: %s | Status: %s\n", defectID,
defectDescription, status);
}
```

```
void deleteDefect(Defect** first, Defect** last, int defectID) {
    Defect *temp = *first, *prev = NULL;
    if (temp != NULL && temp->defectID == defectID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Resolved Defect with ID %d\n", defectID);
        return;
    }
    while (temp != NULL && temp->defectID != defectID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Defect with ID %d not found.\n", defectID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Resolved Defect with ID %d\n", defectID);
}

void displayDefects(Defect* first) {
```

```
    if (first == NULL) {
        printf("No defects reported.\n");
        return;
    }
    printf("Current Defects in Production Process:\n");
    Defect* temp = first;
    while (temp != NULL) {
        printf(" - Defect ID: %d | Description: %s | Status: %s\n", temp->defectID, temp->defectDescription, temp->status);
        temp = temp->next;
    }
}

int main() {
    Defect* first = NULL, *last = NULL;
    int choice, defectID;
    char defectDescription[100], status[20];

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

        switch (choice) {
            case 1:
```

```
            printf("Enter defect ID: ");
            scanf("%d", &defectID);
            getchar(); // Clear newline left in the input buffer
            printf("Enter defect description: ");
            fgets(defectDescription, sizeof(defectDescription), stdin);
            defectDescription[strcspn(defectDescription, "\n")] = '\0'; // Remove newline
character
            printf("Enter status (e.g., Open, Resolved, In Progress): ");
            scanf("%s", status);
            insertDefect(&first, &last, defectID, defectDescription, status);
            break;

        case 2:
            printf("Enter defect ID to resolve: ");
            scanf("%d", &defectID);
            deleteDefect(&first, &last, defectID);
            break;

        case 3:
            displayDefects(first);
            break;

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

        default:
            printf("Invalid choice. Try again.\n");
    }
```

```
    } while (choice != 4);

    return 0;
}
```

Problem 15: Finished Goods Dispatch System

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

Operations:

Create a dispatch schedule.

Insert a new dispatch entry.

Delete a dispatched or canceled entry.

Display the current dispatch schedule.

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

typedef struct Dispatch {
    int dispatchID;
    char productName[50];
    int quantity;
    char status[20];
    struct Dispatch* next;
} Dispatch;

Dispatch* createDispatch(int dispatchID, char* productName, int quantity, char*
status) {
    Dispatch* newDispatch = (Dispatch*)malloc(sizeof(Dispatch));
    newDispatch->dispatchID = dispatchID;
```

```
    strcpy(newDispatch->productName, productName);
    newDispatch->quantity = quantity;
    strcpy(newDispatch->status, status);
    newDispatch->next = NULL;
    return newDispatch;
}

void insertDispatch(Dispatch** first, Dispatch** last, int dispatchID, char*
productName, int quantity, char* status) {
    Dispatch* newDispatch = createDispatch(dispatchID, productName, quantity,
status);
    if (*last != NULL) {
        (*last)->next = newDispatch;
    } else {
        *first = newDispatch;
    }
    *last = newDispatch;
    printf("Inserted Dispatch Entry: ID %d | Product: %s | Quantity: %d | Status: %s\n",
dispatchID, productName, quantity, status);
}

void deleteDispatch(Dispatch** first, Dispatch** last, int dispatchID) {
    Dispatch *temp = *first, *prev = NULL;
    if (temp != NULL && temp->dispatchID == dispatchID) {
        *first = temp->next;
        free(temp);
        if (*first == NULL) {
            *last = NULL;
        }
        printf("Deleted Dispatch Entry with ID %d\n", dispatchID);
```

```
        return;
    }
    while (temp != NULL && temp->dispatchID != dispatchID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Dispatch Entry with ID %d not found.\n", dispatchID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Deleted Dispatch Entry with ID %d\n", dispatchID);
}

void displayDispatches(Dispatch* first) {
    if (first == NULL) {
        printf("No dispatch entries found.\n");
        return;
    }
    printf("Current Dispatch Schedule:\n");
    Dispatch* temp = first;
    while (temp != NULL) {
        printf(" - Dispatch ID: %d | Product: %s | Quantity: %d | Status: %s\n", temp-
>dispatchID, temp->productName, temp->quantity, temp->status);
        temp = temp->next;
```

```
    }
}

int main() {
    Dispatch* first = NULL, *last = NULL;
    int choice, dispatchID, quantity;
    char productName[50], status[20];

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

        switch (choice) {
            case 1:
                printf("Enter dispatch ID: ");
                scanf("%d", &dispatchID);
                printf("Enter product name: ");
                scanf("%s", productName);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                printf("Enter status (e.g., Dispatched, Canceled): ");
                scanf("%s", status);
                insertDispatch(&first, &last, dispatchID, productName, quantity, status);
                break;
```

```
            case 2:
                printf("Enter dispatch ID to delete: ");
                scanf("%d", &dispatchID);
                deleteDispatch(&first, &last, dispatchID);
                break;

            case 3:
                displayDispatches(first);
                break;

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

            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Set 2 programs

Problem 1: Team Roster Management

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

Create a team roster.

Insert a new player.

Delete a player who leaves the team.

Display the current team roster.

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

typedef struct Player {
    char name[50];
    int age;
    struct Player* next;
} Player;

void insertPlayer(Player** first, Player** last, char* name, int age) {
    Player* newPlayer = (Player*)malloc(sizeof(Player));
    strcpy(newPlayer->name, name);
    newPlayer->age = age;
    newPlayer->next = NULL;
    if (*last != NULL) {
        (*last)->next = newPlayer;
    } else {
        *first = newPlayer;
    }
    *last = newPlayer;
    printf("Player %s added.\n", name);
}

void deletePlayer(Player** first, Player** last, char* name) {
```

```
    Player *temp = *first, *prev = NULL;
    if (temp != NULL && strcmp(temp->name, name) == 0) {
        *first = temp->next;
        free(temp);
        printf("Player %s removed.\n", name);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && strcmp(temp->name, name) != 0) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Player %s not found.\n", name);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Player %s removed.\n", name);
}

void displayRoster(Player* first) {
    if (first == NULL) {
        printf("No players in the roster.\n");
```

```
        return;
    }
    Player* temp = first;
    printf("Team Roster:\n");
    while (temp != NULL) {
        printf("%s, Age: %d\n", temp->name, temp->age);
        temp = temp->next;
    }
}

int main() {
    Player* first = NULL, *last = NULL;
    int choice;
    char name[50];
    int age;

    do {
        printf("\nTeam Roster Management\n");
        printf("1. Add Player\n");
        printf("2. Remove Player\n");
        printf("3. Display Roster\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter player name: ");
                scanf("%s", name);
```

```
            printf("Enter player age: ");
            scanf("%d", &age);
            insertPlayer(&first, &last, name, age);
            break;
        case 2:
            printf("Enter player name to remove: ");
            scanf("%s", name);
            deletePlayer(&first, &last, name);
            break;
        case 3:
            displayRoster(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while(choice != 4);

   return 0;
}
```

Problem 2: Tournament Match Scheduling

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

Create a match schedule.

Insert a new match.

Delete a completed or canceled match.

Display the current match schedule.

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

typedef struct Match {
    int matchID;
    char teams[100];
    char date[20];
    struct Match* next;
} Match;

void insertMatch(Match** first, Match** last, int matchID, char* teams, char* date) {
    Match* newMatch = (Match*)malloc(sizeof(Match));
    newMatch->matchID = matchID;
    strcpy(newMatch->teams, teams);
    strcpy(newMatch->date, date);
    newMatch->next = NULL;
    if (*last != NULL) {
        (*last)->next = newMatch;
    } else {
        *first = newMatch;
    }
    *last = newMatch;
    printf("Match scheduled: %s on %s.\n", teams, date);
}

void deleteMatch(Match** first, Match** last, int matchID) {
    Match *temp = *first, *prev = NULL;
```

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

void displayMatches(Match* first) {
    if (first == NULL) {
        printf("No matches scheduled.\n");
        return;
```

```
    }
    Match* temp = first;
    printf("Tournament Match Schedule:\n");
    while (temp != NULL) {
        printf("Match ID: %d | Teams: %s | Date: %s\n", temp->matchID, temp->teams,
temp->date);
        temp = temp->next;
    }
}

int main() {
    Match* first = NULL, *last = NULL;
    int choice;
    int matchID;
    char teams[100], date[20];

    do {
        printf("\nTournament Match Scheduling\n");
        printf("1. Schedule Match\n");
        printf("2. Remove Match\n");
        printf("3. Display Match Schedule\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter match ID: ");
                scanf("%d", &matchID);
```

```
            printf("Enter teams: ");
            scanf(" %[^\n]", teams); // using scanf to read multi-word input
            printf("Enter match date: ");
            scanf(" %[^\n]", date);
            insertMatch(&first, &last, matchID, teams, date);
            break;
        case 2:
            printf("Enter match ID to remove: ");
            scanf("%d", &matchID);
            deleteMatch(&first, &last, matchID);
            break;
        case 3:
            displayMatches(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while(choice != 4);

  return 0;
}
```

Problem 3: Athlete Training Log

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

Create a training log.

Insert a new training session.

Delete a completed or canceled session.

Display the training log.

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

typedef struct TrainingSession {
    int sessionID;
    char date[20];
    char details[100];
    struct TrainingSession* next;
} TrainingSession;

void insertTrainingSession(TrainingSession** first, TrainingSession** last, int
sessionID, char* date, char* details) {
    TrainingSession* newSession =
(TrainingSession*)malloc(sizeof(TrainingSession));
    newSession->sessionID = sessionID;
    strcpy(newSession->date, date);
    strcpy(newSession->details, details);
    newSession->next = NULL;
    if (*last != NULL) {
        (*last)->next = newSession;
    } else {
        *first = newSession;
    }
    *last = newSession;
    printf("Training session added: %s on %s.\n", details, date);
```

```
}

void deleteTrainingSession(TrainingSession** first, TrainingSession** last, int
sessionID) {
    TrainingSession *temp = *first, *prev = NULL;
    if (temp != NULL && temp->sessionID == sessionID) {
        *first = temp->next;
        free(temp);
        printf("Training session with ID %d removed.\n", sessionID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->sessionID != sessionID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Training session ID %d not found.\n", sessionID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Training session with ID %d removed.\n", sessionID);
}
```

```
void displayTrainingSessions(TrainingSession* first) {
    if (first == NULL) {
        printf("No training sessions logged.\n");
        return;
    }
    TrainingSession* temp = first;
    printf("Training Log:\n");
    while (temp != NULL) {
        printf("Session ID: %d | Date: %s | Details: %s\n", temp->sessionID, temp->date, temp->details);
        temp = temp->next;
    }
}

int main() {
    TrainingSession* first = NULL, *last = NULL;
    int choice;
    int sessionID;
    char date[20], details[100];

    do {
        printf("\nAthlete Training Log\n");
        printf("1. Log Training Session\n");
        printf("2. Remove Training Session\n");
        printf("3. Display Training Log\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch(choice) {
            case 1:
                printf("Enter session ID: ");
                scanf("%d", &sessionID);
                printf("Enter training date: ");
                scanf(" %[^\n]", date);
                printf("Enter training details: ");
                scanf(" %[^\n]", details);
                insertTrainingSession(&first, &last, sessionID, date, details);
                break;
            case 2:
                printf("Enter session ID to remove: ");
                scanf("%d", &sessionID);
                deleteTrainingSession(&first, &last, sessionID);
                break;
            case 3:
                displayTrainingSessions(first);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 4: Sports Equipment Inventory

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

Create an equipment inventory list.

Insert a new equipment item.

Delete an item that is no longer usable.

Display the current equipment inventory.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


typedef struct Equipment {

    char name[50];

    int quantity;

    struct Equipment* next;

} Equipment;


void insertEquipment(Equipment** first, Equipment** last, char* name, int quantity) {

    Equipment* newEquipment = (Equipment*)malloc(sizeof(Equipment));

    strcpy(newEquipment->name, name);

    newEquipment->quantity = quantity;

    newEquipment->next = NULL;

    if (*last != NULL) {

        (*last)->next = newEquipment;

    } else {

        *first = newEquipment;

    }
```

```
    *last = newEquipment;
    printf("Equipment %s added with quantity %d.\n", name, quantity);
}

void deleteEquipment(Equipment** first, Equipment** last, char* name) {
    Equipment *temp = *first, *prev = NULL;
    if (temp != NULL && strcmp(temp->name, name) == 0) {
        *first = temp->next;
        free(temp);
        printf("Equipment %s removed.\n", name);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && strcmp(temp->name, name) != 0) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Equipment %s not found.\n", name);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Equipment %s removed.\n", name);
```

```
}

void displayEquipment(Equipment* first) {
    if (first == NULL) {
        printf("No equipment in inventory.\n");
        return;
    }
    Equipment* temp = first;
    printf("Equipment Inventory:\n");
    while (temp != NULL) {
        printf("%s (Quantity: %d)\n", temp->name, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    Equipment* first = NULL, *last = NULL;
    int choice;
    char name[50];
    int quantity;

    do {
        printf("\nSports Equipment Inventory\n");
        printf("1. Add Equipment\n");
        printf("2. Remove Equipment\n");
        printf("3. Display Inventory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch(choice) {
            case 1:
                printf("Enter equipment name: ");
                scanf("%s", name);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                insertEquipment(&first, &last, name, quantity);
                break;
            case 2:
                printf("Enter equipment name to remove: ");
                scanf("%s", name);
                deleteEquipment(&first, &last, name);
                break;
            case 3:
                displayEquipment(first);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 5: Player Performance Tracking

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

Create a performance record list.

Insert a new performance entry.

Delete an outdated or erroneous entry.

Display all performance records.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


typedef struct Performance {

    int playerID;

    char performanceDetails[100];

    struct Performance* next;

} Performance;


void insertPerformance(Performance** first, Performance** last, int playerID, char* performanceDetails) {

    Performance* newPerformance = (Performance*)malloc(sizeof(Performance));

    newPerformance->playerID = playerID;

    strcpy(newPerformance->performanceDetails, performanceDetails);

    newPerformance->next = NULL;

    if (*last != NULL) {

        (*last)->next = newPerformance;

    } else {

        *first = newPerformance;

    }

    *last = newPerformance;

    printf("Performance record added for player ID %d.\n", playerID);
```

```
}

void deletePerformance(Performance** first, Performance** last, int playerID) {
    Performance *temp = *first, *prev = NULL;
    if (temp != NULL && temp->playerID == playerID) {
        *first = temp->next;
        free(temp);
        printf("Performance record for player ID %d removed.\n", playerID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->playerID != playerID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Performance record for player ID %d not found.\n", playerID);
        return;
    }
    prev->next
```

Problem 6: Event Registration System

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

Create a registration list.

Insert a new registration.

Delete a canceled registration.

Display all current registrations.

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

typedef struct Registration {
    int regID;
    char athleteName[50];
    char eventName[50];
    struct Registration* next;
} Registration;

void insertRegistration(Registration** first, Registration** last, int regID, char*
athleteName, char* eventName) {
    Registration* newRegistration = (Registration*)malloc(sizeof(Registration));
    newRegistration->regID = regID;
    strcpy(newRegistration->athleteName, athleteName);
    strcpy(newRegistration->eventName, eventName);
    newRegistration->next = NULL;
    if (*last != NULL) {
        (*last)->next = newRegistration;
    } else {
        *first = newRegistration;
    }
    *last = newRegistration;
    printf("Registration added for athlete %s in event %s.\n", athleteName,
eventName);
}
```

```
void deleteRegistration(Registration** first, Registration** last, int regID) {
    Registration *temp = *first, *prev = NULL;
    if (temp != NULL && temp->regID == regID) {
        *first = temp->next;
        free(temp);
        printf("Registration ID %d removed.\n", regID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->regID != regID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Registration ID %d not found.\n", regID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Registration ID %d removed.\n", regID);
}

void displayRegistrations(Registration* first) {
    if (first == NULL) {
```

```
        printf("No registrations found.\n");
        return;
    }
    Registration* temp = first;
    printf("Event Registrations:\n");
    while (temp != NULL) {
        printf("Reg ID: %d | Athlete: %s | Event: %s\n", temp->regID, temp->athleteName, temp->eventName);
        temp = temp->next;
    }
}

int main() {
    Registration* first = NULL, *last = NULL;
    int choice;
    int regID;
    char athleteName[50], eventName[50];

    do {
        printf("\nEvent Registration System\n");
        printf("1. Add Registration\n");
        printf("2. Remove Registration\n");
        printf("3. Display Registrations\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
```

```
            printf("Enter registration ID: ");
            scanf("%d", &regID);
            printf("Enter athlete name: ");
            scanf(" %[^\n]", athleteName);
            printf("Enter event name: ");
            scanf(" %[^\n]", eventName);
            insertRegistration(&first, &last, regID, athleteName, eventName);
            break;
        case 2:
            printf("Enter registration ID to remove: ");
            scanf("%d", &regID);
            deleteRegistration(&first, &last, regID);
            break;
        case 3:
            displayRegistrations(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while(choice != 4);

  return 0;
}
```

Problem 7: Sports League Standings

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

Create a league standings list.

Insert a new team.

Delete a team that withdraws.

Display the current league standings.

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

typedef struct Team {
    char name[50];
    int points;
    struct Team* next;
} Team;

void insertTeam(Team** first, Team** last, char* name, int points) {
    Team* newTeam = (Team*)malloc(sizeof(Team));
    strcpy(newTeam->name, name);
    newTeam->points = points;
    newTeam->next = NULL;
    if (*last != NULL) {
        (*last)->next = newTeam;
    } else {
        *first = newTeam;
    }
    *last = newTeam;
    printf("Team %s added with %d points.\n", name, points);
}
```

```
void deleteTeam(Team** first, Team** last, char* name) {
    Team *temp = *first, *prev = NULL;
    if (temp != NULL && strcmp(temp->name, name) == 0) {
        *first = temp->next;
        free(temp);
        printf("Team %s removed.\n", name);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && strcmp(temp->name, name) != 0) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Team %s not found.\n", name);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Team %s removed.\n", name);
}

void displayStandings(Team* first) {
```

```
    if (first == NULL) {
        printf("No teams in the league.\n");
        return;
    }
    Team* temp = first;
    printf("League Standings:\n");
    while (temp != NULL) {
        printf("Team: %s | Points: %d\n", temp->name, temp->points);
        temp = temp->next;
    }
}

int main() {
    Team* first = NULL, *last = NULL;
    int choice;
    char name[50];
    int points;

    do {
        printf("\nSports League Standings\n");
        printf("1. Add Team\n");
        printf("2. Remove Team\n");
        printf("3. Display Standings\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
```

```
            printf("Enter team name: ");
            scanf(" %[^\n]", name);
            printf("Enter team points: ");
            scanf("%d", &points);
            insertTeam(&first, &last, name, points);
            break;
        case 2:
            printf("Enter team name to remove: ");
            scanf(" %[^\n]", name);
            deleteTeam(&first, &last, name);
            break;
        case 3:
            displayStandings(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while(choice != 4);

   return 0;
}
```

Problem 8: Match Result Recording

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

Create a match result list.

Insert a new match result.

Delete an incorrect or outdated result.

Display all recorded match results.

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

typedef struct MatchResult {
    int matchID;
    char team1[50];
    char team2[50];
    char result[100];
    struct MatchResult* next;
} MatchResult;

void insertMatchResult(MatchResult** first, MatchResult** last, int matchID, char*
team1, char* team2, char* result) {
    MatchResult* newMatch = (MatchResult*)malloc(sizeof(MatchResult));
    newMatch->matchID = matchID;
    strcpy(newMatch->team1, team1);
    strcpy(newMatch->team2, team2);
    strcpy(newMatch->result, result);
    newMatch->next = NULL;
    if (*last != NULL) {
        (*last)->next = newMatch;
    } else {
        *first = newMatch;
    }
```

```
    *last = newMatch;
    printf("Match result added for match ID %d.\n", matchID);
}

void deleteMatchResult(MatchResult** first, MatchResult** last, int matchID) {
    MatchResult *temp = *first, *prev = NULL;
    if (temp != NULL && temp->matchID == matchID) {
        *first = temp->next;
        free(temp);
        printf("Match result for match ID %d removed.\n", matchID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->matchID != matchID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Match result for match ID %d not found.\n", matchID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Match result for match ID %d removed.\n", matchID);
```

```
}

void displayMatchResults(MatchResult* first) {
    if (first == NULL) {
        printf("No match results available.\n");
        return;
    }
    MatchResult* temp = first;
    printf("Match Results:\n");
    while (temp != NULL) {
        printf("Match ID: %d | Teams: %s vs %s | Result: %s\n", temp->matchID, temp->team1, temp->team2, temp->result);
        temp = temp->next;
    }
}

int main() {
    MatchResult* first = NULL, *last = NULL;
    int choice;
    int matchID;
    char team1[50], team2[50], result[100];

    do {
        printf("\nMatch Result Recording\n");
        printf("1. Add Match Result\n");
        printf("2. Remove Match Result\n");
        printf("3. Display Match Results\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

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

    switch(choice) {
        case 1:
            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 result: ");
            scanf(" %[^\n]", result);
            insertMatchResult(&first, &last, matchID, team1, team2, result);
            break;
        case 2:
            printf("Enter match ID to remove: ");
            scanf("%d", &matchID);
            deleteMatchResult(&first, &last, matchID);
            break;
        case 3:
            displayMatchResults(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while(choice != 4);
```

```
    return 0;
}
```

Problem 9: Player Injury Tracker

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

Create an injury tracker list.

Insert a new injury report.

Delete a resolved or erroneous injury report.

Display all current injury reports.

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

typedef struct Injury {
    int playerID;
    char injuryDetails[100];
    struct Injury* next;
} Injury;

void insertInjury(Injury** first, Injury** last, int playerID, char* injuryDetails) {
    Injury* newInjury = (Injury*)malloc(sizeof(Injury));
    newInjury->playerID = playerID;
    strcpy(newInjury->injuryDetails, injuryDetails);
    newInjury->next = NULL;
    if (*last != NULL) {
        (*last)->next = newInjury;
    } else {
```

```
        *first = newInjury;
    }
    *last = newInjury;
    printf("Injury report added for player ID %d.\n", playerID);
}

void deleteInjury(Injury** first, Injury** last, int playerID) {
    Injury *temp = *first, *prev = NULL;
    if (temp != NULL && temp->playerID == playerID) {
        *first = temp->next;
        free(temp);
        printf("Injury report for player ID %d removed.\n", playerID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->playerID != playerID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Injury report for player ID %d not found.\n", playerID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
```

```
    free(temp);
    printf("Injury report for player ID %d removed.\n", playerID);
}

void displayInjuries(Injury* first) {
    if (first == NULL) {
        printf("No injury reports available.\n");
        return;
    }
    Injury* temp = first;
    printf("Player Injury Reports:\n");
    while (temp != NULL) {
        printf("Player ID: %d | Injury: %s\n", temp->playerID, temp->injuryDetails);
        temp = temp->next;
    }
}

int main() {
    Injury* first = NULL, *last = NULL;
    int choice;
    int playerID;
    char injuryDetails[100];

    do {
        printf("\nPlayer Injury Tracker\n");
        printf("1. Add Injury Report\n");
        printf("2. Remove Injury Report\n");
        printf("3. Display Injury Reports\n");
        printf("4. Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter player ID: ");
                scanf("%d", &playerID);
                printf("Enter injury details: ");
                scanf(" %[^\n]", injuryDetails);
                insertInjury(&first, &last, playerID, injuryDetails);
                break;
            case 2:
                printf("Enter player ID to remove injury report: ");
                scanf("%d", &playerID);
                deleteInjury(&first, &last, playerID);
                break;
            case 3:
                displayInjuries(first);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 10: Sports Facility Booking System

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

Create a booking list.

Insert a new booking.

Delete a canceled or completed booking.

Display all current bookings.

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

typedef struct Booking {
    int bookingID;
    char facilityName[50];
    char customerName[50];
    struct Booking* next;
} Booking;

void insertBooking(Booking** first, Booking** last, int bookingID, char* facilityName,
char* customerName) {
    Booking* newBooking = (Booking*)malloc(sizeof(Booking));
    newBooking->bookingID = bookingID;
    strcpy(newBooking->facilityName, facilityName);
    strcpy(newBooking->customerName, customerName);
    newBooking->next = NULL;
    if (*last != NULL) {
        (*last)->next = newBooking;
    } else {
```

```
        *first = newBooking;
    }
    *last = newBooking;
    printf("Booking added for customer %s in facility %s.\n", customerName,
facilityName);
}

void deleteBooking(Booking** first, Booking** last, int bookingID) {
    Booking *temp = *first, *prev = NULL;
    if (temp != NULL && temp->bookingID == bookingID) {
        *first = temp->next;
        free(temp);
        printf("Booking ID %d removed.\n", bookingID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->bookingID != bookingID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Booking ID %d not found.\n", bookingID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
```

```
    }
    free(temp);
    printf("Booking ID %d removed.\n", bookingID);
}

void displayBookings(Booking* first) {
    if (first == NULL) {
        printf("No bookings available.\n");
        return;
    }
    Booking* temp = first;
    printf("Facility Bookings:\n");
    while (temp != NULL) {
        printf("Booking ID: %d | Facility: %s | Customer: %s\n", temp->bookingID, temp->facilityName, temp->customerName);
        temp = temp->next;
    }
}

int main() {
    Booking* first = NULL, *last = NULL;
    int choice;
    int bookingID;
    char facilityName[50], customerName[50];

    do {
        printf("\nSports Facility Booking System\n");
        printf("1. Add Booking\n");
        printf("2. Remove Booking\n");
```

```
printf("3. Display Bookings\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice) {
    case 1:
        printf("Enter booking ID: ");
        scanf("%d", &bookingID);
        printf("Enter facility name: ");
        scanf(" %[^\n]", facilityName);
        printf("Enter customer name: ");
        scanf(" %[^\n]", customerName);
        insertBooking(&first, &last, bookingID, facilityName, customerName);
        break;
    case 2:
        printf("Enter booking ID to remove: ");
        scanf("%d", &bookingID);
        deleteBooking(&first, &last, bookingID);
        break;
    case 3:
        displayBookings(first);
        break;
    case 4:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
}
```

```
    } while(choice != 4);

    return 0;
}
```

Problem 11: Coaching Staff Management

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

Create a coaching staff list.

Insert a new coach.

Delete a coach who leaves the team.

Display the current coaching staff.

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

typedef struct Coach {
    int coachID;
    char coachName[50];
    char role[50];
    struct Coach* next;
} Coach;

void insertCoach(Coach** first, Coach** last, int coachID, char* coachName, char*
role) {
    Coach* newCoach = (Coach*)malloc(sizeof(Coach));
    newCoach->coachID = coachID;
    strcpy(newCoach->coachName, coachName);
    strcpy(newCoach->role, role);
```

```
    newCoach->next = NULL;
    if (*last != NULL) {
        (*last)->next = newCoach;
    } else {
        *first = newCoach;
    }
    *last = newCoach;
    printf("Coach %s added.\n", coachName);
}

void deleteCoach(Coach** first, Coach** last, int coachID) {
    Coach *temp = *first, *prev = NULL;
    if (temp != NULL && temp->coachID == coachID) {
        *first = temp->next;
        free(temp);
        printf("Coach ID %d removed.\n", coachID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->coachID != coachID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Coach ID %d not found.\n", coachID);
        return;
    }
```

```
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Coach ID %d removed.\n", coachID);
}

void displayCoaches(Coach* first) {
    if (first == NULL) {
        printf("No coaches available.\n");
        return;
    }
    Coach* temp = first;
    printf("Coaching Staff:\n");
    while (temp != NULL) {
        printf("Coach ID: %d | Name: %s | Role: %s\n", temp->coachID, temp-
>coachName, temp->role);
        temp = temp->next;
    }
}

int main() {
    Coach* first = NULL, *last = NULL;
    int choice;
    int coachID;
    char coachName[50], role[50];

    do {
```

```
printf("\nCoaching Staff Management\n");
printf("1. Add Coach\n");
printf("2. Remove Coach\n");
printf("3. Display Coaches\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice) {
    case 1:
        printf("Enter coach ID: ");
        scanf("%d", &coachID);
        printf("Enter coach name: ");
        scanf(" %[^\n]", coachName);
        printf("Enter coach role: ");
        scanf(" %[^\n]", role);
        insertCoach(&first, &last, coachID, coachName, role);
        break;
    case 2:
        printf("Enter coach ID to remove: ");
        scanf("%d", &coachID);
        deleteCoach(&first, &last, coachID);
        break;
    case 3:
        displayCoaches(first);
        break;
    case 4:
        printf("Exiting program.\n");
        break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 12: Fan Club Membership Management

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

Create a membership list.

Insert a new member.

Delete a member who cancels their membership.

Display all current members.

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

typedef struct Member {
    int memberID;
    char memberName[50];
    struct Member* next;
} Member;

void insertMember(Member** first, Member** last, int memberID, char*
memberName) {
    Member* newMember = (Member*)malloc(sizeof(Member));
    newMember->memberID = memberID;
```

```
    strcpy(newMember->memberName, memberName);
    newMember->next = NULL;
    if (*last != NULL) {
        (*last)->next = newMember;
    } else {
        *first = newMember;
    }
    *last = newMember;
    printf("Member %s added to the fan club.\n", memberName);
}

void deleteMember(Member** first, Member** last, int memberID) {
    Member *temp = *first, *prev = NULL;
    if (temp != NULL && temp->memberID == memberID) {
        *first = temp->next;
        free(temp);
        printf("Member ID %d removed from the fan club.\n", memberID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->memberID != memberID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Member ID %d not found.\n", memberID);
        return;
```

```
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Member ID %d removed from the fan club.\n", memberID);
}

void displayMembers(Member* first) {
    if (first == NULL) {
        printf("No fan club members available.\n");
        return;
    }
    Member* temp = first;
    printf("Fan Club Members:\n");
    while (temp != NULL) {
        printf("Member ID: %d | Name: %s\n", temp->memberID, temp-
>memberName);
        temp = temp->next;
    }
}

int main() {
    Member* first = NULL, *last = NULL;
    int choice;
    int memberID;
    char memberName[50];
```

```
do {
    printf("\nFan Club Membership Management\n");
    printf("1. Add Member\n");
    printf("2. Remove Member\n");
    printf("3. Display Members\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            printf("Enter member ID: ");
            scanf("%d", &memberID);
            printf("Enter member name: ");
            scanf(" %[^\n]", memberName);
            insertMember(&first, &last, memberID, memberName);
            break;
        case 2:
            printf("Enter member ID to remove: ");
            scanf("%d", &memberID);
            deleteMember(&first, &last, memberID);
            break;
        case 3:
            displayMembers(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
```

```
            printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 13: Sports Event Scheduling

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

Create an event schedule.

Insert a new event.

Delete a completed or canceled event.

Display the current event schedule.

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

typedef struct Event {
    int eventID;
    char eventName[50];
    char eventDate[50];
    struct Event* next;
} Event;

void insertEvent(Event** first, Event** last, int eventID, char* eventName, char*
eventDate) {
    Event* newEvent = (Event*)malloc(sizeof(Event));
    newEvent->eventID = eventID;
```

```
    strcpy(newEvent->eventName, eventName);
    strcpy(newEvent->eventDate, eventDate);
    newEvent->next = NULL;
    if (*last != NULL) {
        (*last)->next = newEvent;
    } else {
        *first = newEvent;
    }
    *last = newEvent;
    printf("Event %s scheduled on %s.\n", eventName, eventDate);
}

void deleteEvent(Event** first, Event** last, int eventID) {
    Event *temp = *first, *prev = NULL;
    if (temp != NULL && temp->eventID == eventID) {
        *first = temp->next;
        free(temp);
        printf("Event ID %d removed.\n", eventID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->eventID != eventID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Event ID %d not found.\n", eventID);
```

```
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Event ID %d removed.\n", eventID);
}

void displayEvents(Event* first) {
    if (first == NULL) {
        printf("No events scheduled.\n");
        return;
    }
    Event* temp = first;
    printf("Scheduled Events:\n");
    while (temp != NULL) {
        printf("Event ID: %d | Name: %s | Date: %s\n", temp->eventID, temp-
>eventName, temp->eventDate);
        temp = temp->next;
    }
}

int main() {
    Event* first = NULL, *last = NULL;
    int choice;
    int eventID;
    char eventName[50], eventDate[50];
```

```
do {
    printf("\nSports Event Scheduling\n");
    printf("1. Add Event\n");
    printf("2. Remove Event\n");
    printf("3. Display Events\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            printf("Enter event ID: ");
            scanf("%d", &eventID);
            printf("Enter event name: ");
            scanf(" %[^\n]", eventName);
            printf("Enter event date: ");
            scanf(" %[^\n]", eventDate);
            insertEvent(&first, &last, eventID, eventName, eventDate);
            break;
        case 2:
            printf("Enter event ID to remove: ");
            scanf("%d", &eventID);
            deleteEvent(&first, &last, eventID);
            break;
        case 3:
            displayEvents(first);
            break;
        case 4:
```

```
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while(choice != 4);

return 0;
}
```

Problem 14: Player Transfer Records

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

Create a transfer record list.

Insert a new transfer record.

Delete an outdated or erroneous transfer record.

Display all current transfer records.

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

typedef struct Transfer {
    int playerID;
    char fromTeam[50];
    char toTeam[50];
    struct Transfer* next;
} Transfer;
```

```
void insertTransfer(Transfer** first, Transfer** last, int playerID, char* fromTeam,
char* toTeam) {
    Transfer* newTransfer = (Transfer*)malloc(sizeof(Transfer));
    newTransfer->playerID = playerID;
    strcpy(newTransfer->fromTeam, fromTeam);
    strcpy(newTransfer->toTeam, toTeam);
    newTransfer->next = NULL;
    if (*last != NULL) {
        (*last)->next = newTransfer;
    } else {
        *first = newTransfer;
    }
    *last = newTransfer;
    printf("Transfer record added for player ID %d.\n", playerID);
}

void deleteTransfer(Transfer** first, Transfer** last, int playerID) {
    Transfer *temp = *first, *prev = NULL;
    if (temp != NULL && temp->playerID == playerID) {
        *first = temp->next;
        free(temp);
        printf("Transfer record for player ID %d removed.\n", playerID);
        if (*first == NULL) {
            *last = NULL;
        }
        return;
    }
    while (temp != NULL && temp->playerID != playerID) {
        prev = temp;
```

```
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Player ID %d not found.\n", playerID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Transfer record for player ID %d removed.\n", playerID);
}

void displayTransfers(Transfer* first) {
    if (first == NULL) {
        printf("No transfer records available.\n");
        return;
    }
    Transfer* temp = first;
    printf("Player Transfers:\n");
    while (temp != NULL) {
        printf("Player ID: %d | From: %s | To: %s\n", temp->playerID, temp->fromTeam,
temp->toTeam);
        temp = temp->next;
    }
}

int main() {
```

```
Transfer* first = NULL, *last = NULL;
int choice;
int playerID;
char fromTeam[50], toTeam[50];

do {
    printf("\nPlayer Transfer Records\n");
    printf("1. Add Transfer Record\n");
    printf("2. Remove Transfer Record\n");
    printf("3. Display Transfer Records\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            printf("Enter player ID: ");
            scanf("%d", &playerID);
            printf("Enter from team: ");
            scanf(" %[^\n]", fromTeam);
            printf("Enter to team: ");
            scanf(" %[^\n]", toTeam);
            insertTransfer(&first, &last, playerID, fromTeam, toTeam);
            break;
        case 2:
            printf("Enter player ID to remove: ");
            scanf("%d", &playerID);
            deleteTransfer(&first, &last, playerID);
            break;
```

```
            case 3:
                displayTransfers(first);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Problem 15: Championship Points Tracker

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

Create a points tracker list.

Insert a new points entry.

Delete an incorrect or outdated points entry.

Display all current points standings.

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

typedef struct Points {
    int teamID;
    char teamName[50];
```

```
    int points;
    struct Points* next;
} Points;

void insertPoints(Points** first, Points** last, int teamID, char* teamName, int points)
{
    Points* newPoints = (Points*)malloc(sizeof(Points));
    newPoints->teamID = teamID;
    strcpy(newPoints->teamName, teamName);
    newPoints->points = points;
    newPoints->next = NULL;
    if (*last != NULL) {
        (*last)->next = newPoints;
    } else {
        *first = newPoints;
    }
    *last = newPoints;
    printf("Points entry added for team %s.\n", teamName);
}

void deletePoints(Points** first, Points** last, int teamID) {
    Points *temp = *first, *prev = NULL;
    if (temp != NULL && temp->teamID == teamID) {
        *first = temp->next;
        free(temp);
        printf("Points entry for team ID %d removed.\n", teamID);
        if (*first == NULL) {
            *last = NULL;
        }
```

```
        return;
    }
    while (temp != NULL && temp->teamID != teamID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Team ID %d not found.\n", teamID);
        return;
    }
    prev->next = temp->next;
    if (temp == *last) {
        *last = prev;
    }
    free(temp);
    printf("Points entry for team ID %d removed.\n", teamID);
}

void displayPoints(Points* first) {
    if (first == NULL) {
        printf("No points records available.\n");
        return;
    }
    Points* temp = first;
    printf("Championship Points Standings:\n");
    while (temp != NULL) {
        printf("Team ID: %d | Team: %s | Points: %d\n", temp->teamID, temp-
>teamName, temp->points);
        temp = temp->next;
```

```
    }
}

int main() {
    Points* first = NULL, *last = NULL;
    int choice;
    int teamID, points;
    char teamName[50];

    do {
        printf("\nChampionship Points Tracker\n");
        printf("1. Add Points Entry\n");
        printf("2. Remove Points Entry\n");
        printf("3. Display Points Standings\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter team ID: ");
                scanf("%d", &teamID);
                printf("Enter team name: ");
                scanf(" %[^\n]", teamName);
                printf("Enter team points: ");
                scanf("%d", &points);
                insertPoints(&first, &last, teamID, teamName, points);
                break;
            case 2:
```

```
            printf("Enter team ID to remove: ");
            scanf("%d", &teamID);
            deletePoints(&first, &last, teamID);
            break;
        case 3:
            displayPoints(first);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while(choice != 4);

  return 0;
}
```