```
//1.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct RawMaterial {
    int id;
    char name[50];
    int quantity;
    struct RawMaterial* next;
} RawMaterial;

void insertRawMaterial();
void deleteRawMaterial();
void displayInventory();

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

// Function to insert
void insertRawMaterial(RawMaterial** head, int id, const char* name, int quantity) {
    RawMaterial* newMaterial = createRawMaterial(id, name, quantity);
    if (*head == NULL) {
        *head = newMaterial;
```

```
    } else {
        RawMaterial* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newMaterial;
    }
    printf("Raw material %s added to the inventory.\n", name);
}

// Function to delete
void deleteRawMaterial(RawMaterial** head, int id) {
    if (*head == NULL) {
        printf("Inventory is empty.\n");
        return;
    }
    RawMaterial* temp = *head;
    RawMaterial* prev = NULL;
    if (temp != NULL && temp->id == id) {
        *head = temp->next;
        free(temp);
        printf("Raw material with ID %d deleted.\n", id);
        return;
    }

    while (temp != NULL && temp->id != id) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Raw material with ID %d not found in the inventory.\n", id);
```

```
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Raw material with ID %d deleted.\n", id);
}

// Function to display
void displayInventory(RawMaterial* head) {
    if (head == NULL) {
        printf("Inventory is empty.\n");
        return;
    }
    RawMaterial* temp = head;
    printf("Current Inventory:\n");
    printf("ID\tName\tQuantity\n");
    while (temp != NULL) {
        printf("%d\t%s\t%d\n", temp->id, temp->name, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    RawMaterial* inventory = NULL;

    insertRawMaterial(&inventory, 1, "Steel", 100);
    insertRawMaterial(&inventory, 2, "Plastic", 200);
    insertRawMaterial(&inventory, 3, "Copper", 50);


    displayInventory(inventory);
```

```
    deleteRawMaterial(&inventory, 3);
    displayInventory(inventory);

    return 0;
}

//2.

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

// Define the Task structure
typedef struct Task {
    int taskID;
    char taskName[50];
    struct Task* next;
} Task;

void insertTask();
void deleteTask();
void displayQueue();

// Function to create a new task
Task* createTask(int taskID, const char* taskName) {
    Task* newTask = (Task*)malloc(sizeof(Task));
    newTask->taskID = taskID;
    strcpy(newTask->taskName, taskName);
    newTask->next = NULL;
    return newTask;
}
```

```
// Insert a task into the queue
void insertTask(Task** head, int taskID, const char* taskName) {
    Task* newTask = createTask(taskID, taskName);
    if (*head == NULL) {
        *head = newTask;
    } else {
        Task* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("Task %s added to the production line.\n", taskName);
}

// Delete a completed task
void deleteTask(Task** head, int taskID) {
    if (*head == NULL) {
        printf("No tasks in the queue.\n");
        return;
    }
    Task* temp = *head;
    Task* prev = NULL;
    if (temp != NULL && temp->taskID == taskID) {
        *head = temp->next;
        free(temp);
        printf("Task with ID %d completed.\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;
    free(temp);
    printf("Task with ID %d completed.\n", taskID);
}

// Display the current queue
void displayQueue(Task* head) {
    if (head == NULL) {
        printf("No tasks in the production line.\n");
        return;
    }
    Task* temp = head;
    printf("Current Task Queue:\n");
    printf("TaskID\tTaskName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->taskID, temp->taskName);
        temp = temp->next;
    }
}

int main() {
    Task* queue = NULL;

    insertTask(&queue, 1, "Printing");
```

```
    insertTask(&queue, 2, "Painting");
    insertTask(&queue, 3, "Packaging");

    displayQueue(queue);

    deleteTask(&queue, 2);

    displayQueue(queue);

    return 0;
}

//3.

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

typedef struct MaintenanceTask{
    int taskid;
    char taskname[50];
    struct MaintenanceTask *next;
}MaintenanceTask;

void insertMaintenanceTask();
void deleteMaintenanceTask();
void displayMaintenanceschedule();

//function to createTask

MaintenanceTask* createMaintenanceTask(int taskid,const char* taskname){
```

```
    MaintenanceTask* newTask=(MaintenanceTask*)malloc(sizeof(MaintenanceTask));
    newTask->taskid=taskid;
    strcpy(newTask->taskname,taskname);
    newTask->next=NULL;
    return newTask;
}

//function to insert

void insertMaintenanceTask(MaintenanceTask**head,int taskid,const char* taskname){
    MaintenanceTask*newTask=createMaintenanceTask(taskid,taskname);
    if(*head==NULL){
        *head=newTask;
    }
    else{
        MaintenanceTask*temp=*head;
        while(temp->next!=NULL){
            temp = temp->next;
        }
        temp->next=newTask;
    }
    printf("Maintenance task %s added.\n", taskname);
}

//function to delete

void deleteMaintenanceTask(MaintenanceTask**head,int taskid){
    if(*head==NULL){
        printf("No Maintenance Scheduled\n");
        return;
    }
```

```
    MaintenanceTask*temp=*head;
    MaintenanceTask*prev=NULL;
   if(temp!=NULL && temp->taskid==taskid){
      *head=temp->next;
      free(temp);
      printf("Maintenance  task with ID %d completed.\n",  taskid);
      return;
   }
   while(temp  != NULL && temp->taskid  != taskid) {
      prev = temp;
      temp = temp->next;
   }
   if(temp == NULL) {
      printf("Maintenance  task with ID %d not found.\n",  taskid);
      return;
   }
   prev->next = temp->next;
   free(temp);
   printf("Maintenance  task with ID %d completed.\n",  taskid);
}

//function  to display

void displayMaintenanceschedule(MaintenanceTask*  head) {
   if (head == NULL) {
      printf("No  maintenance scheduled.\n");
      return;
   }
   MaintenanceTask*  temp = head;
   printf("Current  Maintenance Schedule:\n");
   printf("TaskID\tTaskName\n");
```

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

int main(){
    MaintenanceTask *schedule= NULL;

    insertMaintenanceTask(&schedule, 1,"Cleaning");
    insertMaintenanceTask(&schedule, 2,"Inserting");
    insertMaintenanceTask(&schedule, 3,"production");

    displayMaintenanceschedule(schedule);
    deleteMaintenanceTask(&schedule,1);
    displayMaintenanceschedule(schedule);

    return 0;

}

//4.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct EmployeeShift {
    int shiftID;
    char employeeName[50];
    struct EmployeeShift* next;
} EmployeeShift;
```

```
void insertShift();
void deleteShift();
void displayShiftSchedule();

// Function to create a new employee shift
EmployeeShift* createShift(int shiftID, const char* employeeName) {
    EmployeeShift* newShift = (EmployeeShift*)malloc(sizeof(EmployeeShift));
    newShift->shiftID = shiftID;
    strcpy(newShift->employeeName, employeeName);
    newShift->next = NULL;
    return newShift;
}

// Insert a new shift
void insertShift(EmployeeShift** head, int shiftID, const char* employeeName) {
    EmployeeShift* newShift = createShift(shiftID, employeeName);
    if (*head == NULL) {
        *head = newShift;
    } else {
        EmployeeShift* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newShift;
    }
    printf("Shift for %s added to the schedule.\n", employeeName);
}

// Delete a completed shift
void deleteShift(EmployeeShift** head, int shiftID) {
```

```
    if (*head == NULL) {
        printf("No shifts scheduled.\n");
        return;
    }
    EmployeeShift* temp = *head;
    EmployeeShift* prev = NULL;
    if (temp != NULL && temp->shiftID == shiftID) {
        *head = temp->next;
        free(temp);
        printf("Shift with ID %d completed.\n", shiftID);
        return;
    }
    while (temp != NULL && temp->shiftID != shiftID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Shift with ID %d not found.\n", shiftID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Shift with ID %d completed.\n", shiftID);
}

// Display the shift schedule
void displayShiftSchedule(EmployeeShift* head) {
    if (head == NULL) {
        printf("No shifts scheduled.\n");
        return;
    }
```

```
    EmployeeShift* temp = head;
    printf("Current Shift Schedule:\n");
    printf("ShiftID\tEmployeeName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->shiftID, temp->employeeName);
        temp = temp->next;
    }
}

int main() {
    EmployeeShift* schedule = NULL;

    insertShift(&schedule, 1, "Sofia");
    insertShift(&schedule, 2, "Mickelen");
    insertShift(&schedule, 3, "Sanjay");

    displayShiftSchedule(schedule);

    deleteShift(&schedule, 2);

    displayShiftSchedule(schedule);

    return 0;
}

//5.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct CustomerOrder {
```

```
    int orderID;
    char customerName[50];
    char productName[50];
    struct CustomerOrder*  next;
} CustomerOrder;

void insertOrder();
void deleteOrder();
void displayOrders();

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

// Insert a new customer order
void insertOrder(CustomerOrder**  head, int orderID, const char* customerName, const char*
productName)  {
    CustomerOrder*  newOrder = createOrder(orderID,  customerName, productName);
    if (*head == NULL) {
        *head = newOrder;
    } else {
        CustomerOrder*  temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
```

```
        temp->next = newOrder;
    }
    printf("Order %d placed by %s for %s.\n", orderID, customerName, productName);
}

// Delete a completed
void deleteOrder(CustomerOrder** head, int orderID) {
    if (*head == NULL) {
        printf("No orders found.\n");
        return;
    }
    CustomerOrder* temp = *head;
    CustomerOrder* prev = NULL;
    if (temp != NULL && temp->orderID == orderID) {
        *head = temp->next;
        free(temp);
        printf("Order with ID %d completed or cancelled\n", orderID);
        return;
    }
    while (temp != NULL && temp->orderID != orderID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Order with ID %d not found.\n", orderID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Order with ID %d completed or canceled.\n", orderID);
}
```

```
// Display all current orders
void displayOrders(CustomerOrder* head) {
    if (head == NULL) {
        printf("No orders in the system.\n");
        return;
    }
    CustomerOrder* temp = head;
    printf("Current Orders:\n");
    printf("OrderID\tCustomerName\tProductName\n");
    while (temp != NULL) {
        printf("%d\t%s\t%s\n", temp->orderID, temp->customerName, temp->productName);
        temp = temp->next;
    }
}

int main() {
    CustomerOrder* orders = NULL;

    insertOrder(&orders, 1, "sofi", "Laptop");
    insertOrder(&orders, 2, "Mickey", "Phone");
    insertOrder(&orders, 3, "christo", "Tablet");

    displayOrders(orders);

    deleteOrder(&orders, 2);

    displayOrders(orders);

    return 0;
}
```

```
//6.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Tool {
    int toolID;
    char toolName[50];
    struct Tool* next;
} Tool;

void insertTool();
void deleteTool();
void displayTools();

// Function to create a new tool entry
Tool* createTool(int toolID, const char* toolName) {
    Tool* newTool = (Tool*)malloc(sizeof(Tool));
    newTool->toolID = toolID;
    strcpy(newTool->toolName, toolName);
    newTool->next = NULL;
    return newTool;
}

// Insert a new tool
void insertTool(Tool** head, int toolID, const char* toolName) {
    Tool* newTool = createTool(toolID, toolName);
    if (*head == NULL) {
        *head = newTool;
    } else {
```

```
        Tool* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTool;
    }
    printf("Tool %s added.\n", toolName);
}

// Delete a tool that is no longer in use
void deleteTool(Tool** head, int toolID) {
    if (*head == NULL) {
        printf("No tools in the system.\n");
        return;
    }
    Tool* temp = *head;
    Tool* prev = NULL;
    if (temp != NULL && temp->toolID == toolID) {
        *head = temp->next;
        free(temp);
        printf("Tool with ID %d removed.\n", toolID);
        return;
    }
    while (temp != NULL && temp->toolID != toolID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Tool with ID %d not found.\n", toolID);
        return;
    }
```

```
    prev->next = temp->next;
    free(temp);
    printf("Tool with ID %d removed.\n", toolID);
}

// Display the tools currently tracked
void displayTools(Tool* head) {
    if (head == NULL) {
        printf("No tools in the system.\n");
        return;
    }
    Tool* temp = head;
    printf("Current Tools:\n");
    printf("ToolID\tToolName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->toolID, temp->toolName);
        temp = temp->next;
    }
}

int main() {
    Tool* tools = NULL;

    insertTool(&tools, 1, "Wrench");
    insertTool(&tools, 2, "Screwdriver");
    insertTool(&tools, 3, "Hammer");

    displayTools(tools);

    deleteTool(&tools, 1);
```

```
    displayTools(tools);

    return 0;
}

//7.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct AssemblyStage {
    int stageID;
    char stageName[50];
    struct AssemblyStage*  next;
} AssemblyStage;

void insertAssemblyStage();
void deleteAssemblyStage();
void displayAssemblyStages();

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

// Insert a new stage in the assembly line
void insertAssemblyStage(AssemblyStage**  head, int stageID, const char* stageName) {
```

```
    AssemblyStage* newStage = createAssemblyStage(stageID, stageName);
    if (*head == NULL) {
        *head = newStage;
    } else {
        AssemblyStage* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newStage;
    }
    printf("Assembly stage %s added to the line.\n", stageName);
}

// Delete a completed stage from the assembly line
void deleteAssemblyStage(AssemblyStage** head, int stageID) {
    if (*head == NULL) {
        printf("No stages.\n");
        return;
    }
    AssemblyStage* temp = *head;
    AssemblyStage* prev = NULL;
    if (temp != NULL && temp->stageID == stageID) {
        *head = temp->next;
        free(temp);
        printf("Assembly stage with ID %d completed.\n", stageID);
        return;
    }
    while (temp != NULL && temp->stageID != stageID) {
        prev = temp;
        temp = temp->next;
    }
```

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

// Display the current assembly stages
void displayAssemblyStages(AssemblyStage* head) {
    if (head == NULL) {
        printf("No stages.\n");
        return;
    }
    AssemblyStage* temp = head;
    printf("Current Assembly Stages:\n");
    printf("StageID\tStageName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->stageID, temp->stageName);
        temp = temp->next;
    }
}

int main() {
    AssemblyStage* stages = NULL;

    insertAssemblyStage(&stages, 1, "Cutting");
    insertAssemblyStage(&stages, 2, "Welding");
    insertAssemblyStage(&stages, 3, "Painting");
```

```
    displayAssemblyStages(stages);

    deleteAssemblyStage(&stages, 3);

    displayAssemblyStages(stages);

    return 0;
}

//8.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct QCItem {
    int itemID;
    char itemName[50];
    struct QCItem* next;
} QCItem;

void insertQCItem();
void deleteQCItem();
void displayQCList();

// Function to create
QCItem* createQCItem(int itemID, const char* itemName) {
    QCItem* newItem = (QCItem*)malloc(sizeof(QCItem));
    newItem->itemID = itemID;
    strcpy(newItem->itemName, itemName);
    newItem->next = NULL;
    return newItem;
```

```
}

// Insert a new checklist item
void insertQCItem(QCItem** head, int itemID, const char* itemName) {
    QCItem* newItem = createQCItem(itemID, itemName);
    if (*head == NULL) {
        *head = newItem;
    } else {
        QCItem* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("QC checklist item %s added.\n", itemName);
}

// Delete a completed or outdated
void deleteQCItem(QCItem** head, int itemID) {
    if (*head == NULL) {
        printf("No QC items in the checklist.\n");
        return;
    }
    QCItem* temp = *head;
    QCItem* prev = NULL;
    if (temp != NULL && temp->itemID == itemID) {
        *head = temp->next;
        free(temp);
        printf("QC item with ID %d removed.\n", itemID);
        return;
    }
```

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

// Display the current checklist
void displayQCList(QCItem* head) {
    if (head == NULL) {
        printf("No QC items in the checklist.\n");
        return;
    }
    QCItem* temp = head;
    printf("Current Quality Control Checklist:\n");
    printf("ItemID\tItemName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->itemID, temp->itemName);
        temp = temp->next;
    }
}

int main() {
    QCItem* checklist = NULL;
```

```
    insertQCItem(&checklist, 1, "Visual");
    insertQCItem(&checklist, 2, "Dimensional");
    insertQCItem(&checklist, 3, "Functionality ");

    displayQCList(checklist);

    deleteQCItem(&checklist, 2);

    displayQCList(checklist);

    return 0;
}

//9.

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

typedef struct Supplier{
    int supplierid;
    char suppliername[50];
    struct Supplier 8next;
}Supplier;



void insertSupplier();
void deleteSupplier();
void displaySupplier();

//function to create
```

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

// Insert a new supplier
void insertSupplier(Supplier** head, int supplierID, const char* supplierName) {
   Supplier* newSupplier = createSupplier(supplierID, supplierName);
   if (*head == NULL) {
      *head = newSupplier;
   } else {
      Supplier* temp = *head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newSupplier;
   }
   printf("Supplier %s added.\n", supplierName);
}

// Delete a supplier
void deleteSupplier(Supplier** head, int supplierID) {
   if (*head == NULL) {
      printf("No suppliers in the system.\n");
      return;
   }
   Supplier* temp = *head;
```

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

// Display all suppliers
void displaySuppliers(Supplier* head) {
    if (head == NULL) {
        printf("No suppliers in the system.\n");
        return;
    }
    Supplier* temp = head;
    printf("Current Suppliers:\n");
    printf("SupplierID\tSupplierName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->supplierID, temp->supplierName);
```

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

int main() {
    Supplier* suppliers = NULL;

    insertSupplier(&suppliers, 1, "Supplier A");
    insertSupplier(&suppliers, 2, "Supplier B");
    insertSupplier(&suppliers, 3, "Supplier C");

    displaySuppliers(suppliers);

    deleteSupplier(&suppliers, 1);

    displaySuppliers(suppliers);

    return 0;
}

//10.

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

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

```
} InventoryItem;

void insertInventoryItem();
void deleteInventoryItem();
void displayInventory();

// Function to create a new inventory item
InventoryItem* createInventoryItem(int itemID, const char* itemName, int quantity) {
    InventoryItem* newItem = (InventoryItem*)malloc(sizeof(InventoryItem));
    newItem->itemID = itemID;
    strcpy(newItem->itemName, itemName);
    newItem->quantity = quantity;
    newItem->next = NULL;
    return newItem;
}

// Insert a new inventory item
void insertInventoryItem(InventoryItem** head, int itemID, const char* itemName, int quantity) {
    InventoryItem* newItem = createInventoryItem(itemID, itemName, quantity);
    if (*head == NULL) {
        *head = newItem;
    } else {
        InventoryItem* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("Item %s added to the inventory.\n", itemName);
}
```

```
// Delete an inventory item
void deleteInventoryItem(InventoryItem** head, int itemID) {
    if (*head == NULL) {
        printf("No items in the inventory.\n");
        return;
    }
    InventoryItem* temp = *head;
    InventoryItem* prev = NULL;
    if (temp != NULL && temp->itemID == itemID) {
        *head = temp->next;
        free(temp);
        printf("Item with ID %d removed from the inventory.\n", itemID);
        return;
    }
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Item with ID %d not found.\n", itemID);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Item with ID %d removed from the inventory.\n", itemID);
}

// Display all inventory items
void displayInventory(InventoryItem* head) {
    if (head == NULL) {
        printf("No items in the inventory.\n");
```

```
        return;
    }
    InventoryItem* temp = head;
    printf("Current Inventory:\n");
    printf("ItemID\tItemName\tQuantity\n");
    while (temp != NULL) {
        printf("%d\t%s\t%d\n", temp->itemID, temp->itemName, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    InventoryItem* inventory = NULL;

    insertInventoryItem(&inventory, 1, "Bolt", 50);
    insertInventoryItem(&inventory, 2, "Nut", 200);
    insertInventoryItem(&inventory, 3, "Washer", 150);

    displayInventory(inventory);

    deleteInventoryItem(&inventory, 3);

    displayInventory(inventory);

    return 0;
}

//11.

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

```
#include <string.h>

typedef struct WarehouseItem {
    int itemID;
    char itemName[50];
    int quantity;
    struct WarehouseItem* next;
} WarehouseItem;

void insertWarehouseItem();
void deleteWarehouseItem();
void displayWarehouse();

// Function to create a new warehouse item
WarehouseItem* createWarehouseItem(int itemID, const char* itemName, int quantity) {
    WarehouseItem* newItem = (WarehouseItem*)malloc(sizeof(WarehouseItem));
    newItem->itemID = itemID;
    strcpy(newItem->itemName, itemName);
    newItem->quantity = quantity;
    newItem->next = NULL;
    return newItem;
}

// Insert a new warehouse item
void insertWarehouseItem(WarehouseItem** head, int itemID, const char* itemName, int quantity) {
    WarehouseItem* newItem = createWarehouseItem(itemID, itemName, quantity);
    if (*head == NULL) {
        *head = newItem;
    } else {
        WarehouseItem* temp = *head;
        while (temp->next != NULL) {
```

```
            temp = temp->next;
        }
        temp->next = newItem;
    }
    printf("Item %s added to warehouse.\n", itemName);
}

// Delete a warehouse item
void deleteWarehouseItem(WarehouseItem** head, int itemID) {
    if (*head == NULL) {
        printf("No items in the warehouse.\n");
        return;
    }
    WarehouseItem* temp = *head;
    WarehouseItem* prev = NULL;
    if (temp != NULL && temp->itemID == itemID) {
        *head = temp->next;
        free(temp);
        printf("Item with ID %d shipped out.\n", itemID);
        return;
    }
    while (temp != NULL && temp->itemID != itemID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Item with ID %d not found.\n", itemID);
        return;
    }
    prev->next = temp->next;
    free(temp);
```

```
    printf("Item with ID %d shipped out.\n", itemID);
}

// Display all warehouse items
void displayWarehouse(WarehouseItem* head) {
    if (head == NULL) {
        printf("No items in the warehouse.\n");
        return;
    }
    WarehouseItem* temp = head;
    printf("Current Warehouse Inventory:\n");
    printf("ItemID\tItemName\tQuantity\n");
    while (temp != NULL) {
        printf("%d\t%s\t%d\n", temp->itemID, temp->itemName, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    WarehouseItem* warehouse = NULL;

    insertWarehouseItem(&warehouse, 1, "Pallet", 100);
    insertWarehouseItem(&warehouse, 2, "Box", 200);
    insertWarehouseItem(&warehouse, 3, "Cage", 50);

    displayWarehouse(warehouse);

    deleteWarehouseItem(&warehouse, 2);

    displayWarehouse(warehouse);
```

```
    return 0;
}

//12.

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

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

void insertPart(Part** head, int partID, const char* partName, int quantity);
void deletePart(Part** head, int partID);
void displayPartsInventory(Part* head);

// Function to create a new machine part
Part* createPart(int partID, const 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;
}

// Insert a new part into the inventory
```

```
void insertPart(Part** head, int partID, const char* partName, int quantity) {
    Part* newPart = createPart(partID, partName, quantity);
    if (*head == NULL) {
        *head = newPart;
    } else {
        Part* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPart;
    }
    printf("Part %s added to inventory.\n", partName);
}

// Delete a part that is used up or obsolete
void deletePart(Part** head, int partID) {
    if (*head == NULL) {
        printf("No parts in the inventory.\n");
        return;
    }
    Part* temp = *head;
    Part* prev = NULL;
    if (temp != NULL && temp->partID == partID) {
        *head = temp->next;
        free(temp);
        printf("Part with ID %d used or obsolete.\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;
    free(temp);
    printf("Part with ID %d used or obsolete.\n", partID);
}

// Display all machine parts in inventory
void displayPartsInventory(Part* head) {
    if (head == NULL) {
        printf("No parts in the inventory.\n");
        return;
    }
    Part* temp = head;
    printf("Current Machine Parts Inventory:\n");
    printf("PartID\tPartName\tQuantity\n");
    while (temp != NULL) {
        printf("%d\t%s\t%d\n", temp->partID, temp->partName, temp->quantity);
        temp = temp->next;
    }
}

int main() {
    Part* partsInventory = NULL;

    insertPart(&partsInventory, 1, "Gear", 150);
    insertPart(&partsInventory, 2, "Belt", 200);
    insertPart(&partsInventory, 3, "Bolt", 100);
```

```
    displayPartsInventory(partsInventory);

    deletePart(&partsInventory,  1);

    displayPartsInventory(partsInventory);

    return 0;
}

//13.

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

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

void insertPackagingTask();
void deletePackagingTask();
void displayPackagingSchedule();

// Function to create a new packaging task
PackagingTask* createPackagingTask(int taskID, const char* taskName) {
    PackagingTask* newTask = (PackagingTask*)malloc(sizeof(PackagingTask));
    newTask->taskID = taskID;
    strcpy(newTask->taskName, taskName);
```

```
    newTask->next = NULL;
    return newTask;
}

// Insert a new packaging task
void insertPackagingTask(PackagingTask** head, int taskID, const char* taskName) {
    PackagingTask* newTask = createPackagingTask(taskID, taskName);
    if (*head == NULL) {
        *head = newTask;
    } else {
        PackagingTask* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTask;
    }
    printf("Packaging task %s scheduled.\n", taskName);
}

// Delete a completed packaging task
void deletePackagingTask(PackagingTask** head, int taskID) {
    if (*head == NULL) {
        printf("No packaging tasks in the schedule.\n");
        return;
    }
    PackagingTask* temp = *head;
    PackagingTask* prev = NULL;
    if (temp != NULL && temp->taskID == taskID) {
        *head = temp->next;
        free(temp);
        printf("Packaging task with ID %d completed.\n", taskID);
```

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

// Display the current packaging schedule
void displayPackagingSchedule(PackagingTask* head) {
    if (head == NULL) {
        printf("No tasks in the packaging schedule.\n");
        return;
    }
    PackagingTask* temp = head;
    printf("Current Packaging Schedule:\n");
    printf("TaskID\tTaskName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->taskID, temp->taskName);
        temp = temp->next;
    }
}

int main() {
```

```
    PackagingTask* packagingSchedule  = NULL;

    insertPackagingTask(&packagingSchedule,  1, "Boxing");
    insertPackagingTask(&packagingSchedule,  2, "Labeling");
    insertPackagingTask(&packagingSchedule,  3, "Sealing");

    displayPackagingSchedule(packagingSchedule);

    deletePackagingTask(&packagingSchedule,  3);

    displayPackagingSchedule(packagingSchedule);

    return 0;
}

//14.

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

typedef struct Defect {
    int defectID;
    char defectName[50];
    struct Defect* next;
} Defect;

void insertDefect();
void deleteDefect();
void displayDefects();
```

```
// Function to create a new defect report
Defect* createDefect(int defectID, const char* defectName) {
    Defect* newDefect = (Defect*)malloc(sizeof(Defect));
    newDefect->defectID = defectID;
    strcpy(newDefect->defectName, defectName);
    newDefect->next = NULL;
    return newDefect;
}

// Insert a new defect report
void insertDefect(Defect** head, int defectID, const char* defectName) {
    Defect* newDefect = createDefect(defectID, defectName);
    if (*head == NULL) {
        *head = newDefect;
    } else {
        Defect* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newDefect;
    }
    printf("Defect %s reported.\n", defectName);
}

// Delete a resolved defect
void deleteDefect(Defect** head, int defectID) {
    if (*head == NULL) {
        printf("No defects in the system.\n");
        return;
    }
    Defect* temp = *head;
```

```
    Defect* prev = NULL;
    if (temp != NULL && temp->defectID == defectID) {
        *head = temp->next;
        free(temp);
        printf("Defect with ID %d resolved.\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;
    free(temp);
    printf("Defect with ID %d resolved.\n", defectID);
}

// Display all current defects
void displayDefects(Defect* head) {
    if (head == NULL) {
        printf("No defects reported.\n");
        return;
    }
    Defect* temp = head;
    printf("Current Defects:\n");
    printf("DefectID\tDefectName\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->defectID, temp->defectName);
```

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

int main() {
    Defect* defects = NULL;

    insertDefect(&defects, 1, "Scratch");
    insertDefect(&defects, 2, "Crack");
    insertDefect(&defects, 3, "Color Mismatch");

    displayDefects(defects);

    deleteDefect(&defects, 2);

    displayDefects(defects);

    return 0;
}

//15.

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

typedef struct Dispatch {
    int dispatchID;
    char dispatchItem[50];
    struct Dispatch* next;
} Dispatch;
```

```
void insertDispatch();
void deleteDispatch();
void displayDispatchSchedule();

// Function to create a new dispatch entry
Dispatch* createDispatch(int dispatchID, const char* dispatchItem) {
    Dispatch* newDispatch = (Dispatch*)malloc(sizeof(Dispatch));
    newDispatch->dispatchID = dispatchID;
    strcpy(newDispatch->dispatchItem, dispatchItem);
    newDispatch->next = NULL;
    return newDispatch;
}

// Insert a new dispatch entry
void insertDispatch(Dispatch** head, int dispatchID, const char* dispatchItem) {
    Dispatch* newDispatch = createDispatch(dispatchID, dispatchItem);
    if (*head == NULL) {
        *head = newDispatch;
    } else {
        Dispatch* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newDispatch;
    }
    printf("Dispatch entry for %s scheduled.\n", dispatchItem);
}

// Delete a dispatched or canceled entry
void deleteDispatch(Dispatch** head, int dispatchID) {
```

```
    if (*head == NULL) {
        printf("No dispatch entries to cancel.\n");
        return;
    }
    Dispatch* temp = *head;
    Dispatch* prev = NULL;
    if (temp != NULL && temp->dispatchID == dispatchID) {
        *head = temp->next;
        free(temp);
        printf("Dispatch entry with ID %d completed.\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;
    free(temp);
    printf("Dispatch entry with ID %d completed.\n", dispatchID);
}

// Display the dispatch schedule
void displayDispatchSchedule(Dispatch* head) {
    if (head == NULL) {
        printf("No dispatch entries in the schedule.\n");
        return;
    }
```

```
    Dispatch* temp = head;
    printf("Current Dispatch Schedule:\n");
    printf("DispatchID\tDispatchItem\n");
    while (temp != NULL) {
        printf("%d\t%s\n", temp->dispatchID, temp->dispatchItem);
        temp = temp->next;
    }
}

int main() {
    Dispatch* dispatchSchedule = NULL;

    insertDispatch(&dispatchSchedule, 1, "Product A");
    insertDispatch(&dispatchSchedule, 2, "Product B");
    insertDispatch(&dispatchSchedule, 3, "Product C");

    displayDispatchSchedule(dispatchSchedule);

    deleteDispatch(&dispatchSchedule, 1);

    displayDispatchSchedule(dispatchSchedule);

    return 0;
}
```

//1.

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

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

void insertPlayer();
void deletePlayer();
void displayRoster();

Player* createPlayer(int id, const char *name) {
    Player* newPlayer = (Player*)malloc(sizeof(Player));
    newPlayer->id = id;
    strcpy(newPlayer->name, name);
    newPlayer->next = NULL;
    return newPlayer;
}

void insertPlayer(Player **head, int id, const char *name) {
    Player* newPlayer = createPlayer(id, name);
    newPlayer->next = *head;
    *head = newPlayer;
    printf("Player %s added to the roster.\n", name);
}

void deletePlayer(Player **head, int id) {
    Player* temp = *head;
```

```
    Player* prev = NULL;

    if (temp != NULL && temp->id == id) {
        *head = temp->next;
        free(temp);
        printf("Player with ID %d removed from the roster.\n", id);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Player with ID %d removed from the roster.\n", id);
}

void displayRoster(Player *head) {
    if (head == NULL) {
        printf("No players in the roster.\n");
        return;
    }
    Player* temp = head;
    printf("Current Team Roster:\n");
```

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

int main() {
    Player* roster = NULL;

    insertPlayer(&roster, 1, "John");
    insertPlayer(&roster, 2, "Sofi");
    insertPlayer(&roster, 3, "Mike");

    displayRoster(roster);

    deletePlayer(&roster, 1);

    displayRoster(roster);

    return 0;
}

//2.

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

typedef struct Match {
    int matchID;
    char team1[50];
```

```
    char team2[50];
    struct Match *next;
} Match;

void insertMatch();
void deleteMatch();
void displayMatchSchedule();

Match* createMatch(int matchID, const char *team1, const char *team2) {
    Match* newMatch = (Match*)malloc(sizeof(Match));
    newMatch->matchID = matchID;
    strcpy(newMatch->team1, team1);
    strcpy(newMatch->team2, team2);
    newMatch->next = NULL;
    return newMatch;
}

void insertMatch(Match **head, int matchID, const char *team1, const char *team2) {
    Match* newMatch = createMatch(matchID, team1, team2);
    newMatch->next = *head;
    *head = newMatch;
    printf("Match between %s and %s scheduled.\n", team1, team2);
}

void deleteMatch(Match **head, int matchID) {
    Match* temp = *head;
    Match* prev = NULL;

    if (temp != NULL && temp->matchID == matchID) {
        *head = temp->next;
        free(temp);
```

```
        printf("Match  with ID %d completed/canceled.\n",  matchID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Match  with ID %d completed/canceled.\n",  matchID);
}

void displayMatchSchedule(Match  *head) {
    if (head == NULL) {
        printf("No  matches scheduled.\n");
        return;
    }
    Match* temp = head;
    printf("Current  Match Schedule:\n");
    while (temp != NULL) {
        printf("Match  ID: %d, %s vs %s\n", temp->matchID, temp->team1, temp->team2);
        temp = temp->next;
    }
}
```

```
int main() {
    Match* schedule = NULL;

    insertMatch(&schedule, 1, "Team A", "Team B");
    insertMatch(&schedule, 2, "Team C", "Team D");

    displayMatchSchedule(schedule);

    deleteMatch(&schedule, 2);

    displayMatchSchedule(schedule);

    return 0;
}

//3.

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

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

void insertTrainingSession();
void deleteTrainingSession();
```

```
void displayTrainingLog();

TrainingSession*  createTrainingSession(int  sessionID, const char *date, const char *details) {
   TrainingSession*  newSession = (TrainingSession*)malloc(sizeof(TrainingSession));
   newSession->sessionID  = sessionID;
   strcpy(newSession->date,  date);
   strcpy(newSession->details,  details);
   newSession->next = NULL;
   return newSession;
}

void insertTrainingSession(TrainingSession  **head, int sessionID, const char *date, const char
*details) {
   TrainingSession*  newSession = createTrainingSession(sessionID,  date, details);
   newSession->next = *head;
   *head = newSession;
   printf("Training  session on %s added.\n", date);
}

void deleteTrainingSession(TrainingSession  **head, int sessionID) {
   TrainingSession*  temp = *head;
   TrainingSession*  prev = NULL;

   if (temp != NULL && temp->sessionID == sessionID) {
      *head = temp->next;
      free(temp);
      printf("Training  session with ID %d removed.\n", sessionID);
      return;
   }

   while (temp != NULL && temp->sessionID  != sessionID) {
```

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

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

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

void displayTrainingLog(TrainingSession *head) {
    if (head == NULL) {
        printf("No training sessions logged.\n");
        return;
    }
    TrainingSession* temp = head;
    printf("Current 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* log = NULL;

    insertTrainingSession(&log, 1, "2025-01-01", "Strength Training");
```

```
    insertTrainingSession(&log, 2, "2025-01-03", "Speed Drills");

    displayTrainingLog(log);

    deleteTrainingSession(&log, 1);

    displayTrainingLog(log);

    return 0;
}



//4.

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

typedef struct Equipment {
    int equipmentID;
    char name[50];
    struct Equipment *next;
} Equipment;

void insertEquipment();
void deleteEquipment();
void displayEquipmentInventory();

Equipment* createEquipment(int equipmentID, const char *name) {
    Equipment* newEquipment = (Equipment*)malloc(sizeof(Equipment));
    newEquipment->equipmentID = equipmentID;
```

```
    strcpy(newEquipment->name, name);
    newEquipment->next = NULL;
    return newEquipment;
}

void insertEquipment(Equipment **head, int equipmentID, const char *name) {
    Equipment* newEquipment = createEquipment(equipmentID, name);
    newEquipment->next = *head;
    *head = newEquipment;
    printf("Equipment %s added to the inventory.\n", name);
}

void deleteEquipment(Equipment **head, int equipmentID) {
    Equipment* temp = *head;
    Equipment* prev = NULL;

    if (temp != NULL && temp->equipmentID == equipmentID) {
        *head = temp->next;
        free(temp);
        printf("Equipment with ID %d removed from the inventory.\n", equipmentID);
        return;
    }

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

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

```
    }

    prev->next = temp->next;
    free(temp);
    printf("Equipment with ID %d removed from the inventory.\n", equipmentID);
}

void displayEquipmentInventory(Equipment *head) {
    if (head == NULL) {
        printf("No equipment in the inventory.\n");
        return;
    }
    Equipment* temp = head;
    printf("Current Equipment Inventory:\n");
    while (temp != NULL) {
        printf("ID: %d, Name: %s\n", temp->equipmentID, temp->name);
        temp = temp->next;
    }
}

int main() {
    Equipment* inventory = NULL;

    insertEquipment(&inventory, 1, "Basketball");
    insertEquipment(&inventory, 2, "Tennis Racket");
    insertEquipment(&inventory, 3, "Football");

    displayEquipmentInventory(inventory);

    deleteEquipment(&inventory, 3);
```

```
    displayEquipmentInventory(inventory);

    return 0;
}



//5.

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

typedef struct Performance {
    int playerID;
    char playerName[50];
    int score;
    struct Performance *next;
} Performance;

void insertPerformance();
void deletePerformance();
void displayPerformanceRecords();

Performance* createPerformance(int playerID, const char *playerName, int score) {
    Performance* newPerformance = (Performance*)malloc(sizeof(Performance));
    newPerformance->playerID = playerID;
    strcpy(newPerformance->playerName, playerName);
    newPerformance->score = score;
    newPerformance->next = NULL;
    return newPerformance;
}
```

```
void insertPerformance(Performance **head, int playerID, const char *playerName, int score) {
    Performance* newPerformance = createPerformance(playerID, playerName, score);
    newPerformance->next = *head;
    *head = newPerformance;
    printf("Performance record for player %s added.\n", playerName);
}

void deletePerformance(Performance **head, int playerID) {
    Performance* temp = *head;
    Performance* prev = NULL;

    if (temp != NULL && temp->playerID == playerID) {
        *head = temp->next;
        free(temp);
        printf("Performance record for player with ID %d deleted.\n", playerID);
        return;
    }

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

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

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

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

void displayPerformanceRecords(Performance *head) {
    if (head == NULL) {
        printf("No performance records available.\n");
        return;
    }
    Performance* temp = head;
    printf("Player Performance Records:\n");
    while (temp != NULL) {
        printf("Player ID: %d, Name: %s, Score: %d\n", temp->playerID, temp->playerName, temp->score);
        temp = temp->next;
    }
}

int main() {
    Performance* records = NULL;

    insertPerformance(&records, 1, "John", 95);
    insertPerformance(&records, 2, "Sofi", 88);

    displayPerformanceRecords(records);

    deletePerformance(&records, 1);

    displayPerformanceRecords(records);

    return 0;
}
```

//6.

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

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

void insertRegistration();
void deleteRegistration();
void displayRegistrations();

Registration* createRegistration(int regID, const char *athleteName) {
    Registration* newReg = (Registration*)malloc(sizeof(Registration));
    newReg->regID = regID;
    strcpy(newReg->athleteName, athleteName);
    newReg->next = NULL;
    return newReg;
}

void insertRegistration(Registration **head, int regID, const char *athleteName) {
    Registration* newReg = createRegistration(regID, athleteName);
    newReg->next = *head;
    *head = newReg;
    printf("Registration for athlete %s added.\n", athleteName);
```

```
}

void deleteRegistration(Registration **head, int regID) {
    Registration* temp = *head;
    Registration* prev = NULL;

    if (temp != NULL && temp->regID == regID) {
        *head = temp->next;
        free(temp);
        printf("Registration with ID %d canceled.\n", regID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Registration with ID %d canceled.\n", regID);
}

void displayRegistrations(Registration *head) {
    if (head == NULL) {
        printf("No registrations available.\n");
```

```
        return;
    }
    Registration*  temp = head;
    printf("Event  Registrations:\n");
    while (temp != NULL) {
        printf("Registration  ID: %d, Athlete: %s\n", temp->regID, temp->athleteName);
        temp = temp->next;
    }
}

int main() {
    Registration*  registrations = NULL;

    insertRegistration(&registrations,  1, "Alice");
    insertRegistration(&registrations,  2, "Bob");

    displayRegistrations(registrations);

    deleteRegistration(&registrations,  1);

    displayRegistrations(registrations);

    return 0;
}

//7.

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

```
typedef struct Team {
    int teamID;
    char teamName[50];
    int points;
    struct Team *next;
} Team;

void insertTeam();
void deleteTeam();
void displayLeagueStandings();

Team* createTeam(int teamID, const char *teamName, int points) {
    Team* newTeam = (Team*)malloc(sizeof(Team));
    newTeam->teamID = teamID;
    strcpy(newTeam->teamName, teamName);
    newTeam->points = points;
    newTeam->next = NULL;
    return newTeam;
}

void insertTeam(Team **head, int teamID, const char *teamName, int points) {
    Team* newTeam = createTeam(teamID, teamName, points);
    newTeam->next = *head;
    *head = newTeam;
    printf("Team %s added to the league.\n", teamName);
}

void deleteTeam(Team **head, int teamID) {
    Team* temp = *head;
    Team* prev = NULL;
```

```
    if (temp != NULL && temp->teamID == teamID) {
        *head = temp->next;
        free(temp);
        printf("Team with ID %d removed from the league.\n", teamID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Team with ID %d removed from the league.\n", teamID);
}

void displayLeagueStandings(Team *head) {
    if (head == NULL) {
        printf("No teams in the league.\n");
        return;
    }
    Team* temp = head;
    printf("League Standings:\n");
    while (temp != NULL) {
```

```
        printf("Team ID: %d, Name: %s, Points: %d\n", temp->teamID, temp->teamName, temp->points);
        temp = temp->next;
    }
}

int main() {
    Team* league = NULL;

    insertTeam(&league, 1, "Lions", 20);
    insertTeam(&league, 2, "Tigers", 15);
    insertTeam(&league, 3, "Bears", 25);

    displayLeagueStandings(league);

    deleteTeam(&league, 2);

    displayLeagueStandings(league);

    return 0;
}

//8.

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

typedef struct MatchResult {
    int matchID;
    char team1[50];
```

```
    char team2[50];
    int score1;
    int score2;
    struct MatchResult *next;
} MatchResult;

void insertMatchResult();
void deleteMatchResult();
void displayMatchResults();

MatchResult* createMatchResult(int matchID, const char *team1, const char *team2, int score1, int score2) {
    MatchResult* newResult = (MatchResult*)malloc(sizeof(MatchResult));
    newResult->matchID = matchID;
    strcpy(newResult->team1, team1);
    strcpy(newResult->team2, team2);
    newResult->score1 = score1;
    newResult->score2 = score2;
    newResult->next = NULL;
    return newResult;
}

void insertMatchResult(MatchResult **head, int matchID, const char *team1, const char *team2, int score1, int score2) {
    MatchResult* newResult = createMatchResult(matchID, team1, team2, score1, score2);
    newResult->next = *head;
    *head = newResult;
    printf("Match result recorded: %s %d-%d %s\n", team1, score1, score2, team2);
}

void deleteMatchResult(MatchResult **head, int matchID) {
    MatchResult* temp = *head;
```

```
    MatchResult* prev = NULL;

    if (temp != NULL && temp->matchID == matchID) {
        *head = temp->next;
        free(temp);
        printf("Match result with ID %d deleted.\n", matchID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Match result with ID %d deleted.\n", matchID);
}

void displayMatchResults(MatchResult *head) {
    if (head == NULL) {
        printf("No match results recorded.\n");
        return;
    }
    MatchResult* temp = head;
    printf("Match Results:\n");
```

```
    while (temp != NULL) {
        printf("Match ID: %d, %s %d-%d %s\n", temp->matchID, temp->team1, temp->score1, temp->score2, temp->team2);
        temp = temp->next;
    }
}

int main() {
    MatchResult* results = NULL;

    insertMatchResult(&results, 1, "Lions", "Tigers", 3, 1);
    insertMatchResult(&results, 2, "Bears", "Tigers", 2, 2);

    displayMatchResults(results);

    deleteMatchResult(&results, 1);

    displayMatchResults(results);

    return 0;
}

//9.

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

typedef struct InjuryReport {
    int playerID;
    char playerName[50];
```

```
    char injuryDetails[100];
    struct InjuryReport *next;
} InjuryReport;

void insertInjury();
void deleteInjury();
void displayInjuryReports();

InjuryReport* createInjuryReport(int playerID, const char *playerName, const char *injuryDetails) {
    InjuryReport* newReport = (InjuryReport*)malloc(sizeof(InjuryReport));
    newReport->playerID = playerID;
    strcpy(newReport->playerName, playerName);
    strcpy(newReport->injuryDetails, injuryDetails);
    newReport->next = NULL;
    return newReport;
}

void insertInjury(InjuryReport **head, int playerID, const char *playerName, const char
*injuryDetails) {
    InjuryReport* newReport = createInjuryReport(playerID, playerName, injuryDetails);
    newReport->next = *head;
    *head = newReport;
    printf("Injury report for player %s added.\n", playerName);
}

void deleteInjury(InjuryReport **head, int playerID) {
    InjuryReport* temp = *head;
    InjuryReport* prev = NULL;

    if (temp != NULL && temp->playerID == playerID) {
        *head = temp->next;
```

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

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

    if (temp == NULL) {
        printf("Injury report for player with ID %d not found.\n", playerID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Injury report for player with ID %d removed.\n", playerID);
}

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

```
    }
}

int main() {
    InjuryReport* reports = NULL;

    insertInjury(&reports, 1, "John", "Ankle Sprain");
    insertInjury(&reports, 2, "Alice", "Hamstring Tear");

    displayInjuryReports(reports);

    deleteInjury(&reports, 1);

    displayInjuryReports(reports);

    return 0;
}

//10.

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

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

```
void insertBooking();
void deleteBooking();
void displayBookings();

Booking* createBooking(int bookingID, const char *facilityName, const char *bookingTime) {
    Booking* newBooking = (Booking*)malloc(sizeof(Booking));
    newBooking->bookingID = bookingID;
    strcpy(newBooking->facilityName, facilityName);
    strcpy(newBooking->bookingTime, bookingTime);
    newBooking->next = NULL;
    return newBooking;
}

void insertBooking(Booking **head, int bookingID, const char *facilityName, const char
*bookingTime) {
    Booking* newBooking = createBooking(bookingID, facilityName, bookingTime);
    newBooking->next = *head;
    *head = newBooking;
    printf("Booking for facility %s added.\n", facilityName);
}

void deleteBooking(Booking **head, int bookingID) {
    Booking* temp = *head;
    Booking* prev = NULL;

    if (temp != NULL && temp->bookingID == bookingID) {
        *head = temp->next;
        free(temp);
        printf("Booking with ID %d canceled.\n", bookingID);
        return;
    }
```

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

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

    prev->next = temp->next;
    free(temp);
    printf("Booking with ID %d canceled.\n", bookingID);
}

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

int main() {
```

```
    Booking* bookings = NULL;

    insertBooking(&bookings, 1, "Tennis Court", "10:00 AM");
    insertBooking(&bookings, 2, "Football Field", "12:00 PM");

    displayBookings(bookings);

    deleteBooking(&bookings, 1);

    displayBookings(bookings);

    return 0;
}

//11.

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

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

void insertCoach();
void deleteCoach();
void displayCoachingStaff();

Coach* createCoach(int coachID, const char *coachName) {
```

```
    Coach* newCoach = (Coach*)malloc(sizeof(Coach));
    newCoach->coachID = coachID;
    strcpy(newCoach->coachName, coachName);
    newCoach->next = NULL;
    return newCoach;
}

void insertCoach(Coach **head, int coachID, const char *coachName) {
    Coach* newCoach = createCoach(coachID, coachName);
    newCoach->next = *head;
    *head = newCoach;
    printf("Coach %s added to the team.\n", coachName);
}

void deleteCoach(Coach **head, int coachID) {
    Coach* temp = *head;
    Coach* prev = NULL;

    if (temp != NULL && temp->coachID == coachID) {
        *head = temp->next;
        free(temp);
        printf("Coach with ID %d removed.\n", coachID);
        return;
    }

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

    if (temp == NULL) {
```

```
        printf("Coach with ID %d not found.\n", coachID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Coach with ID %d removed.\n", coachID);
}

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

int main() {
    Coach* staff = NULL;

    insertCoach(&staff, 1, "John Smith");
    insertCoach(&staff, 2, "Alice Johnson");

    displayCoachingStaff(staff);

    deleteCoach(&staff, 1);
```

```
    displayCoachingStaff(staff);

    return 0;
}

//12.

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

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

void insertFan();
void deleteFan();
void displayFanClub();

Fan* createFan(int memberID, const char *memberName) {
    Fan* newFan = (Fan*)malloc(sizeof(Fan));
    newFan->memberID = memberID;
    strcpy(newFan->memberName, memberName);
    newFan->next = NULL;
    return newFan;
}

void insertFan(Fan **head, int memberID, const char *memberName) {
```

```
    Fan* newFan = createFan(memberID, memberName);
    newFan->next = *head;
    *head = newFan;
    printf("Fan %s added to the club.\n", memberName);
}

void deleteFan(Fan **head, int memberID) {
    Fan* temp = *head;
    Fan* prev = NULL;

    if (temp != NULL && temp->memberID == memberID) {
        *head = temp->next;
        free(temp);
        printf("Fan with ID %d removed from the club.\n", memberID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Fan with ID %d removed from the club.\n", memberID);
}
```

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

int main() {
   Fan* club = NULL;

   insertFan(&club,  1, "Bob");
   insertFan(&club,  2, "Sue");

   displayFanClub(club);

   deleteFan(&club,  1);

   displayFanClub(club);

   return 0;
}

//13.
```

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

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

void insertEvent();
void deleteEvent();
void displayEvents();

Event* createEvent(int eventID, const char *eventName, const char *eventDate) {
    Event* newEvent = (Event*)malloc(sizeof(Event));
    newEvent->eventID = eventID;
    strcpy(newEvent->eventName, eventName);
    strcpy(newEvent->eventDate, eventDate);
    newEvent->next = NULL;
    return newEvent;
}

void insertEvent(Event **head, int eventID, const char *eventName, const char *eventDate) {
    Event* newEvent = createEvent(eventID, eventName, eventDate);
    newEvent->next = *head;
    *head = newEvent;
    printf("Event %s scheduled.\n", eventName);
}
```

```
void deleteEvent(Event **head, int eventID) {
    Event* temp = *head;
    Event* prev = NULL;

    if (temp != NULL && temp->eventID == eventID) {
        *head = temp->next;
        free(temp);
        printf("Event with ID %d canceled.\n", eventID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Event with ID %d canceled.\n", eventID);
}

void displayEvents(Event *head) {
    if (head == NULL) {
        printf("No events scheduled.\n");
        return;
    }
```

```
    Event* temp = head;
    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* events = NULL;

    insertEvent(&events, 1, "Basketball Tournament", "2025-03-01");
    insertEvent(&events, 2, "Soccer Match", "2025-03-05");

    displayEvents(events);

    deleteEvent(&events, 1);

    displayEvents(events);

    return 0;
}

//14.

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

typedef struct TransferRecord {
```

```
    int playerID;
    char playerName[50];
    char fromTeam[50];
    char toTeam[50];
    struct TransferRecord *next;
} TransferRecord;

void insertTransferRecord();
void deleteTransferRecord();
void displayTransferRecords();

TransferRecord* createTransferRecord(int playerID, const char *playerName, const char *fromTeam,
const char *toTeam) {
    TransferRecord* newRecord = (TransferRecord*)malloc(sizeof(TransferRecord));
    newRecord->playerID = playerID;
    strcpy(newRecord->playerName, playerName);
    strcpy(newRecord->fromTeam, fromTeam);
    strcpy(newRecord->toTeam, toTeam);
    newRecord->next = NULL;
    return newRecord;
}

void insertTransferRecord(TransferRecord **head, int playerID, const char *playerName, const char
*fromTeam, const char *toTeam) {
    TransferRecord* newRecord = createTransferRecord(playerID, playerName, fromTeam, toTeam);
    newRecord->next = *head;
    *head = newRecord;
    printf("Player %s transfer recorded from %s to %s.\n", playerName, fromTeam, toTeam);
}

void deleteTransferRecord(TransferRecord **head, int playerID) {
    TransferRecord* temp = *head;
```

```
    TransferRecord* prev = NULL;

    if (temp != NULL && temp->playerID == playerID) {
        *head = temp->next;
        free(temp);
        printf("Transfer record for player with ID %d removed.\n", playerID);
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Transfer record for player with ID %d removed.\n", playerID);
}

void displayTransferRecords(TransferRecord *head) {
    if (head == NULL) {
        printf("No transfer records available.\n");
        return;
    }
    TransferRecord* temp = head;
    printf("Player Transfer Records:\n");
```

```
    while (temp != NULL) {
        printf("Player ID: %d, Name: %s, From Team: %s, To Team: %s\n", temp->playerID, temp->playerName, temp->fromTeam, temp->toTeam);
        temp = temp->next;
    }
}

int main() {
    TransferRecord* records = NULL;

    insertTransferRecord(&records, 1, "John", "Team A", "Team B");
    insertTransferRecord(&records, 2, "Jane", "Team B", "Team C");

    displayTransferRecords(records);

    deleteTransferRecord(&records, 1);

    displayTransferRecords(records);

    return 0;
}

//15.

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

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

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

void insertPointsEntry();
void deletePointsEntry();
void displayPointsEntries();

PointsEntry* createPointsEntry(int teamID, const char *teamName, int points) {
    PointsEntry* newEntry = (PointsEntry*)malloc(sizeof(PointsEntry));
    newEntry->teamID = teamID;
    strcpy(newEntry->teamName, teamName);
    newEntry->points = points;
    newEntry->next = NULL;
    return newEntry;
}

void insertPointsEntry(PointsEntry **head, int teamID, const char *teamName, int points) {
    PointsEntry* newEntry = createPointsEntry(teamID, teamName, points);
    newEntry->next = *head;
    *head = newEntry;
    printf("Points entry for team %s added.\n", teamName);
}

void deletePointsEntry(PointsEntry **head, int teamID) {
    PointsEntry* temp = *head;
    PointsEntry* prev = NULL;

    if (temp != NULL && temp->teamID == teamID) {
        *head = temp->next;
        free(temp);
```

```
        printf("Points entry for team with ID %d removed.\n", teamID);
        return;
    }

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

    if (temp == NULL) {
        printf("Points entry for team with ID %d not found.\n", teamID);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Points entry for team with ID %d removed.\n", teamID);
}

void displayPointsEntries(PointsEntry *head) {
    if (head == NULL) {
        printf("No points entries available.\n");
        return;
    }
    PointsEntry* temp = head;
    printf("Championship Points:\n");
    while (temp != NULL) {
        printf("Team ID: %d, Name: %s, Points: %d\n", temp->teamID, temp->teamName, temp-
>points);
        temp = temp->next;
    }
```

```
}

int main() {
    PointsEntry* standings = NULL;

    insertPointsEntry(&standings, 1, "Lions", 20);
    insertPointsEntry(&standings, 2, "Tigers", 15);

    displayPointsEntries(standings);

    deletePointsEntry(&standings, 1);

    displayPointsEntries(standings);

    return 0;
}
```