```c
/*Develop a system to track real-time inventory levels using structures
for item details and unions for variable attributes (e.g., weight,
volume).
 Use const pointers for immutable item codes and double pointers for
managing dynamic inventory arrays.
Specifications:
Structure: Item details (ID, name, category).
Union: Attributes (weight, volume).
const Pointer: Immutable item codes.
Double Pointers: Dynamic inventory management.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for variable attributes (weight or volume)
union Attribute {
    float weight;
    float volume;
};

// Structure for item details
struct Item {
    char *id;    // Dynamic memory for item ID (modifiable)
    char name[50];
    char category[20];
    union Attribute attr;
    int isWeight;     // 1 = Weight, 0 = Volume
};

// Function to add an item to inventory
void addItem(struct Item **inventory, int *size, const char *id, const
char *name, const char *category, float value, int isWeight) {
    *inventory = (struct Item *)realloc(*inventory, (*size + 1) *
sizeof(struct Item));
    if (*inventory == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
```

```c
    struct Item *newItem = &((*inventory)[*size]);

    // Dynamically allocate memory for item ID
    newItem->id = (char *)malloc(strlen(id) + 1);
    if (newItem->id == NULL) {
        printf("Memory allocation for ID failed!\n");
        return;
    }

    strcpy(newItem->id, id);  // Copy ID to allocated memory
    strcpy(newItem->name, name);
    strcpy(newItem->category, category);
    newItem->isWeight = isWeight;

    if (isWeight) {
        newItem->attr.weight = value;
    } else {
        newItem->attr.volume = value;
    }

    (*size)++;
    printf("Item added successfully!\n");
}

// Function to display inventory
void displayInventory(struct Item *inventory, int size) {
    if (size == 0) {
        printf("Inventory is empty!\n");
        return;
    }

    printf("\nCurrent Inventory:\n");
    for (int i = 0; i < size; i++) {
        printf("ID: %s | Name: %s | Category: %s | ", inventory[i].id,
inventory[i].name, inventory[i].category);
        if (inventory[i].isWeight) {
            printf("Weight: %.2f kg\n", inventory[i].attr.weight);
        } else {
            printf("Volume: %.2f L\n", inventory[i].attr.volume);
        }
```

```
    }
}

// Function to remove an item by ID
void removeItem(struct Item **inventory, int *size, const char *id) {
    if (*size == 0) {
        printf("Inventory is empty!\n");
        return;
    }

    int index = -1;
    for (int i = 0; i < *size; i++) {
        if (strcmp((*inventory)[i].id, id) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Item with ID %s not found!\n", id);
        return;
    }

    free((*inventory)[index].id);  // Free dynamically allocated memory
for ID

    for (int i = index; i < *size - 1; i++) {
        (*inventory)[i] = (*inventory)[i + 1];
    }

    *inventory = (struct Item *)realloc(*inventory, (*size - 1) *
sizeof(struct Item));
    (*size)--;

    printf("Item with ID %s removed successfully!\n", id);
}

// Main function
int main() {
    struct Item *inventory = NULL; // Dynamic inventory array
```

```
    int size = 0; // Number of items

    int choice;
    char id[10], name[50], category[20];
    float value;
    int isWeight;

    do {
        printf("\n1. Add Item\n");
        printf("2. Display Inventory\n");
        printf("3. Remove Item\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Item ID: ");
                scanf("%s", id);
                printf("Enter Name: ");
                scanf("%s", name);
                printf("Enter Category: ");
                scanf("%s", category);
                printf("Enter Attribute (1 for Weight, 0 for Volume): ");
                scanf("%d", &isWeight);
                printf("Enter Value (kg or L): ");
                scanf("%f", &value);
                addItem(&inventory, &size, id, name, category, value,
isWeight);
                break;
            case 2:
                displayInventory(inventory, size);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
                scanf("%s", id);
                removeItem(&inventory, &size, id);
                break;
            case 4:
                printf("Exiting...\n");
```

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

    // Free allocated memory for all items
    for (int i = 0; i < size; i++) {
        free(inventory[i].id);  // Free memory for item ID
    }
    free(inventory); // Free the inventory array

    return 0;
}
```

```
1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice: 3
Enter Item ID to remove: 1
Item with ID 1 removed successfully!

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice: 2

Current Inventory:
ID: 2 | Name: milk | Category: drink | Volume: 200.00 L

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice:
```

```c
/*Create a system to dynamically manage shipping routes using structures
for route data and unions for different modes of transport.
Use const pointers for route IDs and double pointers for managing route
arrays.
Specifications:
Structure: Route details (ID, start, end).
Union: Transport modes (air, sea, land).
const Pointer: Read-only route IDs.
Double Pointers: Dynamic route allocation.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Mode
{
    char air[10];
    char sea[10];
    char land[10];
};
struct Route
{
    const char * id;
    char start[20];
    char end[20];
    int mode;//1.air,2.sea,3.land;
    union Mode m;
};
void add(struct Route **route,int *size,const char *id,const char
*start,const char *end,int mode,const char *type)
{
    *route=(struct Route *)realloc(*route,(*size+1)*sizeof(struct Route));
    struct Route *new=&((*route)[*size]);
    new->id=(char *)malloc((strlen(id)+1)*sizeof(char));
    strcpy(new->id,id);

    strcpy(new->start,start);
    strcpy(new->end,end);

    new->mode=mode;

    if(mode==1)
```

```
    strcpy(new->m.air,type);
    else if(mode==2)
    strcpy(new->m.sea,type);
    else
    strcpy(new->m.land,type);
    (*size)++;

}
void removeroute(struct Route **route,int *size,const char *id)
{
    int index=-1;
    for(int i=0;i<*size;i++)
    {
        if(strcmp((*route)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("ID not found\n");
        return;
    }


    for(int i=index;i<(*size)-1;i++)
    (*route)[i]=(*route)[i+1];
    (*size)--;

}
void display(struct Route *route,int size)
{
    printf("Current route : \n");
    for(int i=0;i<size;i++)
    {
        printf("Id : %s | Start : %s | End :
%s",route[i].id,route[i].start,route[i].end);
        if(route[i].mode==1)
        printf("Mode : %s\n",route[i].m.air);
```

```
        else if(route[i].mode==2)
        printf("Mode : %s\n",route[i].m.sea);
        else if(route[i].mode==3)
        printf("Mode : %s\n",route[i].m.land);
    }
}
void main()
{
    struct Route *route=NULL;
    int size=0;
    int choice;
    char id[10], start[20],end[20],type[10];
    int mode;
    float value;
    int isWeight;

    do {
        printf("\n1. Add Item\n");
        printf("2. Display Inventory\n");
        printf("3. Remove Item\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Item ID: ");
                scanf("%s", id);
                printf("Enter start: ");
                scanf("%s", start);
                printf("Enter end: ");
                scanf("%s", end);
                printf("Enter transport mode(1 for air, 2 for sea, 3 for
land): ");
                scanf("%d", &mode);
                printf("Enter type (air,sea,land): ");
                scanf("%s",type);
                add(&route, &size, id, start, end, mode, type);
                break;
            case 2:
```

```
                display(route, size);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
                scanf("%s", id);
                removeroute(&route, &size, id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    free(route);
}
```

```
3. Remove Item
4. Exit
Enter your choice: 1
Enter Item ID: 2
Enter start: B
Enter end: C
Enter transport mode(1 for air, 2 for sea, 3 for land): 2
Enter type (air,sea,land): sea

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice: 2
Current route :
Id : 1 | Start : A | End : BMode : air
Id : 2 | Start : B | End : CMode : sea

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice:
```

```c
/*Develop a fleet management system using structures for vehicle details
and unions for status (active, maintenance).
Use const pointers for vehicle identifiers and double pointers to manage
vehicle records.
Specifications:
Structure: Vehicle details (ID, type, status).
Union: Status (active, maintenance).
const Pointer: Vehicle IDs.
Double Pointers: Dynamic vehicle list management.
 */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Status
{
```

```
    char active[10];
    char maintenance[10];
};
struct Vehicle
{
    const char *id;
    char type[10];
    int status;//0.active 1.maintenance
    union Status s;
};
void add(struct Vehicle **v,int *count)
{
    printf("Enter id : ");
    (*v)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*v)[*count].id);
    printf("Enter type : ");
    scanf("%s",(*v)[*count].type);
    printf("Enter status : ");
    scanf("%d",&(*v)[*count].status);
    printf("Enter state of vehicle :");
    if((*v)[*count].status==0)
    scanf("%s",(*v)[*count].s.active);
    else
    scanf("%s",(*v)[*count].s.maintenance);
    (*count)++;

}
void removevehicle(struct Vehicle **v,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*v)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count-1;i++)
    (*v)[i]=(*v)[i+1];
```

```
    (*count)--;
}
void display(struct Vehicle *v,int count)
{
    printf("Current vehicles :\n");
    for(int i=0;i<count;i++)
    {
        printf("Vehicle ID : %s | Type : %s | ",v[i].id,v[i].type);
        if(v[i].status==0)
        printf("Status : Active | State : %s\n",v[i].s.active);
        else if(v[i].status==1)
        printf("Status : Maintenance | State : %s\n",v[i].s.active);

    }

}
void main()
{
    struct Vehicle *v=(struct Vehicle *)malloc(10*sizeof(struct Vehicle));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Item\n");
        printf("2. Display Inventory\n");
        printf("3. Remove Item\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&v, &count);
                break;
            case 2:
                display(v,count);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
```

```
                scanf("%s", id);
                removevehicle(&v,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
/*Develop a fleet management system using structures for vehicle details
and unions for status (active, maintenance).
Use const pointers for vehicle identifiers and double pointers to manage
vehicle records.
Specifications:
Structure: Vehicle details (ID, type, status).
Union: Status (active, maintenance).
const Pointer: Vehicle IDs.
Double Pointers: Dynamic vehicle list management.
 */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Status
{
    char active[10];
    char maintenance[10];
};
struct Vehicle
{
    const char *id;
    char type[10];
    int status;//0.active 1.maintenance
    union Status s;
};
void add(struct Vehicle **v,int *count)
{
    printf("Enter id : ");
```

```
    (*v)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*v)[*count].id);
    printf("Enter type : ");
    scanf("%s",(*v)[*count].type);
    printf("Enter status : ");
    scanf("%d",&(*v)[*count].status);
    printf("Enter state of vehicle :");
    if((*v)[*count].status==0)
    scanf("%s",(*v)[*count].s.active);
    else
    scanf("%s",(*v)[*count].s.maintenance);
    (*count)++;

}
void removevehicle(struct Vehicle **v,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*v)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count-1;i++)
    (*v)[i]=(*v)[i+1];
    (*count)--;
}
void display(struct Vehicle *v,int count)
{
    printf("Current vehicles :\n");
    for(int i=0;i<count;i++)
    {
        printf("Vehicle ID : %s | Type : %s | ",v[i].id,v[i].type);
        if(v[i].status==0)
        printf("Status : Active | State : %s\n",v[i].s.active);
        else if(v[i].status==1)
        printf("Status : Maintenance | State : %s\n",v[i].s.active);
```

```
    }

}
void main()
{
    struct Vehicle *v=(struct Vehicle *)malloc(10*sizeof(struct Vehicle));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Item\n");
        printf("2. Display Inventory\n");
        printf("3. Remove Item\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&v, &count);
                break;
            case 2:
                display(v,count);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
                scanf("%s", id);
                removevehicle(&v,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice: 1
Enter id : 2
Enter type : light
Enter status : 1
Enter state of vehicle :underrepair

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice: 2
Current vehicles :
Vehicle ID : 1 | Type : heavy | Status : Active | State : onroad
Vehicle ID : 2 | Type : light | Status : Maintenance | State : underrepair

1. Add Item
2. Display Inventory
3. Remove Item
4. Exit
Enter your choice:
```

```c
/*Implement an order processing system using structures for order details
and unions for payment methods.
Use const pointers for order IDs and double pointers for dynamic order
queues.
Specifications:
Structure: Order details (ID, customer, items).
Union: Payment methods (credit card, cash).
const Pointer: Order IDs.
Double Pointers: Dynamic order queue.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Card
{
    char cno[10];
    float amount;
};
union Payment
{
    struct Card card;
```

```
    float cash;
};
struct Item
{
    char name[10];
};
struct Order
{
    const char *id;
    char customer[20];
    struct Item item[5];
    int pm; //0.cash ,1.card
    union Payment payment;
};
void add(struct Order **o,int *count)
{
    printf("Enter order id :");
    (*o)[*count].id=( char *)malloc(10*sizeof(char));
    scanf("%s",(*o)[*count].id);
    printf("Enter customer name : ");
    scanf("%s",(*o)[*count].customer);
    printf("Enter items :");
    for(int i=0;i<5;i++)
    {
        scanf("%s",(*o)[*count].item[i].name);
    }
    printf("Enter payment method : ");
    scanf(" %d",&(*o)[*count].pm);
    if((*o)[*count].pm==0)
    {
        printf("Enter cash : ");
        scanf("%f",&(*o)[*count].payment.cash);
    }
    else if((*o)[*count].pm==1)
    {
        printf("Enter card number : ");
        scanf("%s",(*o)[*count].payment.card.cno);
        printf("Enter amount : ");
        scanf("%f",&(*o)[*count].payment.card.amount);
    }
```

```
    (*count)++;
}
void removeorder(struct Order **o,int *count,const char *id)
{
    int index =-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*o)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count-1;i++)
    {
        (*o)[i]=(*o)[i+1];
    }
    (*count)--;
}
void display(struct Order *o,int count)
{
    printf("Current order : \n");
    for(int i=0;i<count;i++)
    {
        printf("Order id : %s | Customer name : %s | Items :
",o[i].id,o[i].customer);
        for(int j=0;j<5;j++)
        {
            printf("%s ",o[i].item[j].name);
        }
        printf("|");
        if(o[i].pm==0)
        {
            printf("Payment method : Cash | Amount :
%.2f\n",o[i].payment.cash);
        }
        else if(o[i].pm==1)
        {
            printf("Payment method : Cash |Card no :%s | Amount :
%.2f\n",o[i].payment.card.cno,o[i].payment.card.amount);
```

```
        }

    }
}
void main()
{
    struct Order *o=(struct Order *)malloc(10*sizeof(struct Order));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Order\n");
        printf("2. Display Order\n");
        printf("3. Remove order\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&o, &count);
                break;
            case 2:
                display(o,count);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
                scanf("%s", id);
                removeorder(&o,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
4. Exit
Enter your choice: 1
Enter order id :2
Enter customer name : blue
Enter items :book pen box toy cup
Enter payment method : 1
Enter card number : 10001
Enter amount : 20000

1. Add Order
2. Display Order
3. Remove order
4. Exit
Enter your choice: 2
Current order :
Order id : 1 | Customer name : redd | Items : brick toy car pen box |Payment method : Cash | Amount : 20000.00
Order id : 2 | Customer name : blue | Items : book pen box toy cup |Payment method : Cash |Card no :10001 | Amount : 20000.00

1. Add Order
2. Display Order
3. Remove order
4. Exit
```

```c
/*Create a system to manage real-time traffic data for logistics using
structures for traffic nodes and unions for traffic conditions.
Use const pointers for node identifiers and double pointers for dynamic
traffic data storage.
Specifications:
Structure: Traffic node details (ID, location).
Union: Traffic conditions (clear, congested).
const Pointer: Node IDs.
Double Pointers: Dynamic traffic data management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define a union for tracking events
union TrackingEvent {
    char dispatched[20];  // Event when the shipment is dispatched
    char delivered[20];   // Event when the shipment is delivered
};

// Define a structure for shipment details
struct Shipment {
    const char *trackingNumber;  // Const pointer to protect the tracking
number
    char origin[30];             // Origin address
    char destination[30];        // Destination address
    union TrackingEvent event;   // Tracking event (dispatched or
delivered)
};
```

```c
// Function to add a new shipment
void addShipment(struct Shipment **shipments, int *count) {
    // Reallocate memory for the shipment list
    struct Shipment *temp = realloc(*shipments, (*count + 1) *
sizeof(struct Shipment));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *shipments = temp;

    struct Shipment *newShipment = &(*shipments)[*count];

    // Allocate memory for the tracking number
    newShipment->trackingNumber = (char *)malloc(15 * sizeof(char));  //
Allocate memory for tracking number
    if (!newShipment->trackingNumber) {
        printf("Memory allocation failed for tracking number!\n");
        return;
    }

    // Get shipment details from user
    printf("Enter tracking number: ");
    scanf("%s", newShipment->trackingNumber);
    printf("Enter origin: ");
    scanf("%s", newShipment->origin);
    printf("Enter destination: ");
    scanf("%s", newShipment->destination);

    // Ask for tracking event
    printf("Enter tracking event (1 for dispatched, 2 for delivered): ");
    int eventChoice;
    scanf("%d", &eventChoice);

    if (eventChoice == 1) {
        strcpy(newShipment->event.dispatched, "Dispatched");
    } else if (eventChoice == 2) {
        strcpy(newShipment->event.delivered, "Delivered");
    } else {
        printf("Invalid event choice!\n");
```

```
    }

    (*count)++;
}

// Function to remove a shipment
void removeShipment(struct Shipment **shipments, int *count, const char
*trackingNumber) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*shipments)[i].trackingNumber, trackingNumber) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Tracking number not found!\n");
        return;
    }

    // Free memory for tracking number
    free((*shipments)[index].trackingNumber);

    // Shift remaining shipments to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*shipments)[i] = (*shipments)[i + 1];
    }

    // Reallocate memory to shrink the shipment list
    struct Shipment *temp = realloc(*shipments, (*count - 1) *
sizeof(struct Shipment));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *shipments = temp;
    (*count)--;
}
```

```c
// Function to display all shipments
void displayShipments(struct Shipment *shipments, int count) {
    if (count == 0) {
        printf("No shipments available.\n");
        return;
    }

    printf("Current Shipments:\n");
    for (int i = 0; i < count; i++) {
        printf("Tracking Number: %s | Origin: %s | Destination: %s |
Event: ", shipments[i].trackingNumber, shipments[i].origin,
shipments[i].destination);

        if (strlen(shipments[i].event.dispatched) > 0) {
            printf("%s\n", shipments[i].event.dispatched);
        } else if (strlen(shipments[i].event.delivered) > 0) {
            printf("%s\n", shipments[i].event.delivered);
        }
    }
}

int main() {
    struct Shipment *shipments = (struct Shipment *)malloc(10 *
sizeof(struct Shipment));  // Initial allocation for 10 shipments
    int count = 0;  // To keep track of the number of shipments
    int choice;
    char trackingNumber[15];

    do {
        printf("\n1. Add Shipment\n");
        printf("2. Display Shipments\n");
        printf("3. Remove Shipment\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addShipment(&shipments, &count);
```

```
                break;
            case 2:
                displayShipments(shipments, count);
                break;
            case 3:
                printf("Enter tracking number to remove: ");
                scanf("%s", trackingNumber);
                removeShipment(&shipments, &count, trackingNumber);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(shipments[i].trackingNumber);
    }
    free(shipments);

    return 0;
}
```

```
Enter tracking event (1 for dispatched, 2 for delivered): 2

1. Add Shipment
2. Display Shipments
3. Remove Shipment
4. Exit
Enter your choice: 2
Current Shipments:
Tracking Number: 1001 | Origin: A | Destination: B | Event: Dispatched
Tracking Number: 2 | Origin: C | Destination: D | Event: Delivered

1. Add Shipment
2. Display Shipments
3. Remove Shipment
4. Exit
```

```
/*Design a warehouse slot allocation system using structures for slot
details and unions for item types.
```

```
 Use const pointers for slot identifiers and double pointers for dynamic
slot management.
Specifications:
Structure: Slot details (ID, location, size).
Union: Item types (perishable, non-perishable).
const Pointer: Slot IDs.
Double Pointers: Dynamic slot allocation.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define a union for traffic conditions
union TrafficCondition {
    char clear[20];      // Condition when the traffic is clear
    char congested[20];  // Condition when the traffic is congested
};

// Define a structure for traffic node details
struct TrafficNode {
    const char *nodeID;  // Const pointer to protect the node ID
    char location[50];   // Location of the traffic node
    union TrafficCondition condition;  // Traffic condition (clear or
congested)
};

// Function to add a new traffic node
void addTrafficNode(struct TrafficNode **nodes, int *count) {
    // Reallocate memory for the node list
    struct TrafficNode *temp = realloc(*nodes, (*count + 1) *
sizeof(struct TrafficNode));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *nodes = temp;

    struct TrafficNode *newNode = &(*nodes)[*count];

    // Allocate memory for the node ID (const pointer)
```

```
    newNode->nodeID = (char *)malloc(10 * sizeof(char));  // Allocate
memory for node ID
    if (!newNode->nodeID) {
        printf("Memory allocation failed for node ID!\n");
        return;
    }

    // Get traffic node details from the user
    printf("Enter node ID: ");
    scanf("%s", newNode->nodeID);
    printf("Enter location: ");
    scanf("%s", newNode->location);

    // Ask for traffic condition
    printf("Enter traffic condition (1 for clear, 2 for congested): ");
    int conditionChoice;
    scanf("%d", &conditionChoice);

    if (conditionChoice == 1) {
        strcpy(newNode->condition.clear, "Clear");
    } else if (conditionChoice == 2) {
        strcpy(newNode->condition.congested, "Congested");
    } else {
        printf("Invalid condition choice!\n");
    }

    (*count)++;
}

// Function to remove a traffic node
void removeTrafficNode(struct TrafficNode **nodes, int *count, const char
*nodeID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*nodes)[i].nodeID, nodeID) == 0) {
            index = i;
            break;
        }
    }
```

```c
    if (index == -1) {
        printf("Node ID not found!\n");
        return;
    }

    // Free memory for node ID
    free((*nodes)[index].nodeID);

    // Shift remaining nodes to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*nodes)[i] = (*nodes)[i + 1];
    }

    // Reallocate memory to shrink the node list
    struct TrafficNode *temp = realloc(*nodes, (*count - 1) *
sizeof(struct TrafficNode));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *nodes = temp;
    (*count)--;
}

// Function to display all traffic nodes and their conditions
void displayTrafficNodes(struct TrafficNode *nodes, int count) {
    if (count == 0) {
        printf("No traffic data available.\n");
        return;
    }

    printf("Current Traffic Nodes:\n");
    for (int i = 0; i < count; i++) {
        printf("Node ID: %s | Location: %s | Condition: ",
nodes[i].nodeID, nodes[i].location);
        if (strlen(nodes[i].condition.clear) > 0) {
            printf("%s\n", nodes[i].condition.clear);
        } else if (strlen(nodes[i].condition.congested) > 0) {
            printf("%s\n", nodes[i].condition.congested);
```

```
        }
    }
}

int main() {
    struct TrafficNode *nodes = (struct TrafficNode *)malloc(10 *
sizeof(struct TrafficNode));  // Initial allocation for 10 nodes
    int count = 0;  // To keep track of the number of traffic nodes
    int choice;
    char nodeID[10];

    do {
        printf("\n1. Add Traffic Node\n");
        printf("2. Display Traffic Nodes\n");
        printf("3. Remove Traffic Node\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTrafficNode(&nodes, &count);
                break;
            case 2:
                displayTrafficNodes(nodes, count);
                break;
            case 3:
                printf("Enter Node ID to remove: ");
                scanf("%s", nodeID);
                removeTrafficNode(&nodes, &count, nodeID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free all allocated memory
```

```
    for (int i = 0; i < count; i++) {
        free(nodes[i].nodeID);
    }
    free(nodes);

    return 0;
}
```

```
4. Exit
Enter your choice: 1
Enter node ID: 2
Enter location: B
Enter traffic condition (1 for clear, 2 for congested): 2

1. Add Traffic Node
2. Display Traffic Nodes
3. Remove Traffic Node
4. Exit
Enter your choice: 2
Current Traffic Nodes:
Node ID: 1 | Location: A | Condition: Clear
Node ID: 2 | Location: B | Condition: Congested

1. Add Traffic Node
2. Display Traffic Nodes
3. Remove Traffic Node
4. Exit
Enter your choice:
```

```
/*Develop a package delivery optimization tool using structures for
package details and unions for delivery methods.
 Use const pointers for package identifiers and double pointers to manage
dynamic delivery routes.
Specifications:
Structure: Package details (ID, weight, destination).
Union: Delivery methods (standard, express).
const Pointer: Package IDs.
Double Pointers: Dynamic route management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
union DeliveryMethod {
    char standard[20];
    char express[20];
};


struct Package {
    const char *packageID;
    float weight;
    char destination[50];
    union DeliveryMethod deliveryMethod;
};

void addPackage(struct Package **packages, int *count) {
    // Reallocate memory for the new package
    struct Package *temp = realloc(*packages, (*count + 1) * sizeof(struct
Package));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *packages = temp;

    struct Package *newPackage = &(*packages)[*count];


    newPackage->packageID = (char *)malloc(10 * sizeof(char));
    if (!newPackage->packageID) {
        printf("Memory allocation failed for package ID!\n");
        return;
    }


    printf("Enter package ID: ");
    scanf("%s", newPackage->packageID);
    printf("Enter weight (kg): ");
    scanf("%f", &newPackage->weight);
    printf("Enter destination: ");
    scanf("%s", newPackage->destination);
```

```c
    printf("Enter delivery method (1 for Standard, 2 for Express): ");
    int methodChoice;
    scanf("%d", &methodChoice);

    if (methodChoice == 1) {
        strcpy(newPackage->deliveryMethod.standard, "Standard");
    } else if (methodChoice == 2) {
        strcpy(newPackage->deliveryMethod.express, "Express");
    } else {
        printf("Invalid delivery method choice!\n");
    }

    (*count)++;
}

void removePackage(struct Package **packages, int *count, const char
*packageID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*packages)[i].packageID, packageID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Package ID not found!\n");
        return;
    }

    free((*packages)[index].packageID);


    for (int i = index; i < *count - 1; i++) {
        (*packages)[i] = (*packages)[i + 1];
    }
```

```
    struct Package *temp = realloc(*packages, (*count - 1) * sizeof(struct
Package));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *packages = temp;
    (*count)--;
}


void displayPackages(struct Package *packages, int count) {
    if (count == 0) {
        printf("No packages available.\n");
        return;
    }

    printf("Current Package Delivery Routes:\n");
    for (int i = 0; i < count; i++) {
        printf("Package ID: %s | Weight: %.2f kg | Destination: %s |
Delivery Method: ",
               packages[i].packageID, packages[i].weight,
packages[i].destination);

        if (strlen(packages[i].deliveryMethod.standard) > 0) {
            printf("%s\n", packages[i].deliveryMethod.standard);
        } else if (strlen(packages[i].deliveryMethod.express) > 0) {
            printf("%s\n", packages[i].deliveryMethod.express);
        }
    }
}

int main() {
    struct Package *packages = (struct Package *)malloc(10 * sizeof(struct
Package));
    int count = 0;
    int choice;
    char packageID[10];
```

```c
    do {
        printf("\n1. Add Package\n");
        printf("2. Display Packages\n");
        printf("3. Remove Package\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addPackage(&packages, &count);
                break;
            case 2:
                displayPackages(packages, count);
                break;
            case 3:
                printf("Enter Package ID to remove: ");
                scanf("%s", packageID);
                removePackage(&packages, &count, packageID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    for (int i = 0; i < count; i++) {
        free(packages[i].packageID);
    }
    free(packages);

    return 0;
}
```

```
Enter delivery method (1 for Standard, 2 for Express): 2

1. Add Package
2. Display Packages
3. Remove Package
4. Exit
Enter your choice: 2
Current Package Delivery Routes:
Package ID: 1 | Weight: 40.00 kg | Destination: A | Delivery Method: Standard
Package ID: 2 | Weight: 50.00 kg | Destination: B | Delivery Method: Express

1. Add Package
2. Display Packages
3. Remove Package
4. Exit
```

```c
/*Create a logistics data analytics system using structures for analytics
records and unions for different metrics.
 Use const pointers to ensure data integrity and double pointers for
managing dynamic analytics data.
Specifications:
Structure: Analytics records (timestamp, metric).
Union: Metrics (speed, efficiency).
const Pointer: Analytics data.
Double Pointers: Dynamic data storage.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define a union for different metrics (speed, efficiency)
union Metrics {
    float speed;       // Speed metric
    float efficiency; // Efficiency metric
};

// Define a structure for analytics records
struct AnalyticsRecord {
    const char *timestamp;     // Const pointer to timestamp to protect
data integrity
    union Metrics metric;      // Union to store either speed or efficiency
    int metricType;            // 0 for speed, 1 for efficiency
};

// Function to add a new analytics record
void addAnalyticsRecord(struct AnalyticsRecord **records, int *count) {
    // Reallocate memory for the new analytics record
```

```
    struct AnalyticsRecord *temp = realloc(*records, (*count + 1) *
sizeof(struct AnalyticsRecord));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *records = temp;

    struct AnalyticsRecord *newRecord = &(*records)[*count];

    // Allocate memory for timestamp (const pointer)
    newRecord->timestamp = (char *)malloc(20 * sizeof(char));
    if (!newRecord->timestamp) {
        printf("Memory allocation failed for timestamp!\n");
        return;
    }

    // Get analytics record details from the user
    printf("Enter timestamp (YYYY-MM-DD HH:MM:SS): ");
    scanf("%s", newRecord->timestamp);

    // Get metric type (0 for speed, 1 for efficiency)
    printf("Enter metric type (0 for speed, 1 for efficiency): ");
    scanf("%d", &newRecord->metricType);

    if (newRecord->metricType == 0) {
        // Speed metric
        printf("Enter speed (in km/h): ");
        scanf("%f", &newRecord->metric.speed);
    } else if (newRecord->metricType == 1) {
        // Efficiency metric
        printf("Enter efficiency (in percentage): ");
        scanf("%f", &newRecord->metric.efficiency);
    } else {
        printf("Invalid metric type!\n");
        return;
    }

    (*count)++;
}
```

```c
// Function to remove an analytics record by timestamp
void removeAnalyticsRecord(struct AnalyticsRecord **records, int *count,
const char *timestamp) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*records)[i].timestamp, timestamp) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Analytics record with the given timestamp not found!\n");
        return;
    }

    // Free the allocated memory for the timestamp
    free((*records)[index].timestamp);

    // Shift remaining records to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*records)[i] = (*records)[i + 1];
    }

    // Reallocate memory to shrink the records list
    struct AnalyticsRecord *temp = realloc(*records, (*count - 1) *
sizeof(struct AnalyticsRecord));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *records = temp;
    (*count)--;
}

// Function to display all analytics records
void displayAnalyticsRecords(struct AnalyticsRecord *records, int count) {
    if (count == 0) {
```

```
        printf("No analytics records available.\n");
        return;
    }

    printf("Analytics Records:\n");
    for (int i = 0; i < count; i++) {
        printf("Timestamp: %s | Metric: ", records[i].timestamp);

        if (records[i].metricType == 0) {
            // Speed metric
            printf("Speed = %.2f km/h\n", records[i].metric.speed);
        } else if (records[i].metricType == 1) {
            // Efficiency metric
            printf("Efficiency = %.2f%%\n", records[i].metric.efficiency);
        }
    }
}

int main() {
    struct AnalyticsRecord *records = (struct AnalyticsRecord *)malloc(10
* sizeof(struct AnalyticsRecord));  // Initial allocation for 10 records
    int count = 0;  // To keep track of the number of records
    int choice;
    char timestamp[20];

    do {
        printf("\n1. Add Analytics Record\n");
        printf("2. Display Analytics Records\n");
        printf("3. Remove Analytics Record\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addAnalyticsRecord(&records, &count);
                break;
            case 2:
                displayAnalyticsRecords(records, count);
                break;
```

```
            case 3:
                printf("Enter timestamp to remove: ");
                scanf("%s", timestamp);
                removeAnalyticsRecord(&records, &count, timestamp);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(records[i].timestamp);
    }
    free(records);

    return 0;
}
```

```
1. Add Analytics Record
2. Display Analytics Records
3. Remove Analytics Record
4. Exit
Enter your choice: 1
Enter timestamp (YYYY-MM-DD HH:MM:SS): 2025-01-20
Enter metric type (0 for speed, 1 for efficiency): 0
Enter speed (in km/h): 90

1. Add Analytics Record
2. Display Analytics Records
3. Remove Analytics Record
4. Exit
Enter your choice: 2
Analytics Records:
Timestamp: 2025-01-19 | Metric: Speed = 80.00 km/h
Timestamp: 2025-01-20 | Metric: Speed = 90.00 km/h

1. Add Analytics Record
2. Display Analytics Records
3. Remove Analytics Record
4. Exit
Enter your choice:
```

```c
/*Implement a transportation schedule management system using structures
for schedule details and unions for transport types.
Use const pointers for schedule IDs and double pointers for dynamic
schedule lists.
Specifications:
Structure: Schedule details (ID, start time, end time).
Union: Transport types (bus, truck).
const Pointer: Schedule IDs.
Double Pointers: Dynamic schedule handling.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for different transport types
union TransportType {
```

```c
    char bus[20];    // Bus type transport
    char truck[20]; // Truck type transport
};

// Structure for schedule details
struct ScheduleDetails {
    const char *ID;      // Schedule ID (protected by const pointer)
    char startTime[20]; // Start time
    char endTime[20];    // End time
    union TransportType transportType; // Transport type (bus or truck)
};

// Function to add a new schedule
void addSchedule(struct ScheduleDetails **schedules, int *count) {
    // Reallocate memory for the new schedule
    struct ScheduleDetails *temp = realloc(*schedules, (*count + 1) *
sizeof(struct ScheduleDetails));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *schedules = temp;

    struct ScheduleDetails *newSchedule = &(*schedules)[*count];

    // Allocate memory for schedule ID (const pointer)
    newSchedule->ID = (char *)malloc(10 * sizeof(char));
    if (!newSchedule->ID) {
        printf("Memory allocation failed for ID!\n");
        return;
    }

    // Get schedule details from the user
    printf("Enter schedule ID: ");
    scanf("%s", newSchedule->ID);

    printf("Enter start time (HH:MM): ");
    scanf("%s", newSchedule->startTime);

    printf("Enter end time (HH:MM): ");
```

```c
    scanf("%s", newSchedule->endTime);

    // Get transport type (0 for bus, 1 for truck)
    printf("Enter transport type (0 for bus, 1 for truck): ");
    int transportChoice;
    scanf("%d", &transportChoice);

    if (transportChoice == 0) {
        strcpy(newSchedule->transportType.bus, "Bus");
    } else if (transportChoice == 1) {
        strcpy(newSchedule->transportType.truck, "Truck");
    } else {
        printf("Invalid transport type!\n");
        return;
    }

    (*count)++;
}

// Function to remove a schedule by ID
void removeSchedule(struct ScheduleDetails **schedules, int *count, const
char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*schedules)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Schedule with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for the ID
    free((*schedules)[index].ID);

    // Shift remaining schedules to fill the gap
    for (int i = index; i < *count - 1; i++) {
```

```
        (*schedules)[i] = (*schedules)[i + 1];
    }

    // Reallocate memory to shrink the array
    struct ScheduleDetails *temp = realloc(*schedules, (*count - 1) *
sizeof(struct ScheduleDetails));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *schedules = temp;
    (*count)--;
}

// Function to display all schedules
void displaySchedules(struct ScheduleDetails *schedules, int count) {
    if (count == 0) {
        printf("No schedules available.\n");
        return;
    }

    printf("Transportation Schedules:\n");
    for (int i = 0; i < count; i++) {
        printf("Schedule ID: %s | Start Time: %s | End Time: %s |
Transport: ", schedules[i].ID, schedules[i].startTime,
schedules[i].endTime);

        if (strcmp(schedules[i].transportType.bus, "Bus") == 0) {
            printf("Bus\n");
        } else if (strcmp(schedules[i].transportType.truck, "Truck") == 0)
{
            printf("Truck\n");
        }
    }
}

int main() {
```

```c
    struct ScheduleDetails *schedules = (struct ScheduleDetails
*)malloc(10 * sizeof(struct ScheduleDetails)); // Initial allocation for
10 schedules
    int count = 0;  // To track the number of schedules
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Schedule\n");
        printf("2. Display Schedules\n");
        printf("3. Remove Schedule\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addSchedule(&schedules, &count);
                break;
            case 2:
                displaySchedules(schedules, count);
                break;
            case 3:
                printf("Enter Schedule ID to remove: ");
                scanf("%s", ID);
                removeSchedule(&schedules, &count, ID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(schedules[i].ID);
    }
    free(schedules);
```

```
    return 0;
}
```

```
4. Exit
Enter your choice: 1
Enter schedule ID: 2
Enter start time (HH:MM): 10:30
Enter end time (HH:MM): 12:40
Enter transport type (0 for bus, 1 for truck): 1

1. Add Schedule
2. Display Schedules
3. Remove Schedule
4. Exit
Enter your choice: 2
Transportation Schedules:
Schedule ID: 1 | Start Time: 12:45 | End Time: 16:50 | Transport: Bus
Schedule ID: 2 | Start Time: 10:30 | End Time: 12:40 | Transport: Truck

1. Add Schedule
2. Display Schedules
3. Remove Schedule
4. Exit
```

```
/*Develop a dynamic supply chain modeling tool using structures for
supplier and customer details, and unions for transaction types.
 Use const pointers for transaction IDs and double pointers for dynamic
relationship management.
Specifications:
Structure: Supplier/customer details (ID, name).
Union: Transaction types (purchase, return).
const Pointer: Transaction IDs.
Double Pointers: Dynamic supply chain modeling.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for different transaction types
union TransactionType {
    char purchase[20];  // Purchase transaction
    char return_[20];   // Return transaction (renamed return_ to avoid
conflict with keyword)
};

// Structure for supplier/customer details
struct SupplierCustomerDetails {
```

```
    const char *ID;       // Supplier/Customer ID (protected by const
pointer)
    char name[50];        // Name of the supplier or customer
    union TransactionType transactionType; // Type of transaction
(purchase or return)
};

// Function to add a new supplier/customer record
void addSupplierCustomer(struct SupplierCustomerDetails **records, int
*count) {
    // Reallocate memory for the new record
    struct SupplierCustomerDetails *temp = realloc(*records, (*count + 1)
* sizeof(struct SupplierCustomerDetails));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *records = temp;

    struct SupplierCustomerDetails *newRecord = &(*records)[*count];

    // Allocate memory for the ID (const pointer)
    newRecord->ID = (char *)malloc(10 * sizeof(char));
    if (!newRecord->ID) {
        printf("Memory allocation failed for ID!\n");
        return;
    }

    // Get supplier/customer details from the user
    printf("Enter ID: ");
    scanf("%s", newRecord->ID);

    printf("Enter name: ");
    scanf("%s", newRecord->name);

    // Get transaction type (0 for purchase, 1 for return)
    printf("Enter transaction type (0 for purchase, 1 for return): ");
    int transactionChoice;
    scanf("%d", &transactionChoice);
```

```c
    if (transactionChoice == 0) {
        strcpy(newRecord->transactionType.purchase, "Purchase");
    } else if (transactionChoice == 1) {
        strcpy(newRecord->transactionType.return_, "Return");
    } else {
        printf("Invalid transaction type!\n");
        return;
    }

    (*count)++;
}

// Function to remove a supplier/customer record by ID
void removeSupplierCustomer(struct SupplierCustomerDetails **records, int
*count, const char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*records)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Record with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for the ID
    free((*records)[index].ID);

    // Shift remaining records to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*records)[i] = (*records)[i + 1];
    }

    // Reallocate memory to shrink the array
    struct SupplierCustomerDetails *temp = realloc(*records, (*count - 1)
* sizeof(struct SupplierCustomerDetails));
    if (!temp && (*count - 1) > 0) {
```

```
        printf("Memory reallocation failed!\n");
        return;
    }

    *records = temp;
    (*count)--;
}

// Function to display all records
void displayRecords(struct SupplierCustomerDetails *records, int count) {
    if (count == 0) {
        printf("No records available.\n");
        return;
    }

    printf("Supplier/Customer Records:\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %s | Name: %s | Transaction Type: ", records[i].ID,
records[i].name);

        if (strcmp(records[i].transactionType.purchase, "Purchase") == 0)
{
            printf("Purchase\n");
        } else if (strcmp(records[i].transactionType.return_, "Return") ==
0) {
            printf("Return\n");
        }
    }
}

int main() {
    struct SupplierCustomerDetails *records = (struct
SupplierCustomerDetails *)malloc(10 * sizeof(struct
SupplierCustomerDetails)); // Initial allocation for 10 records
    int count = 0;  // To track the number of records
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Supplier/Customer Record\n");
```

```c
        printf("2. Display Records\n");
        printf("3. Remove Record\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addSupplierCustomer(&records, &count);
                break;
            case 2:
                displayRecords(records, count);
                break;
            case 3:
                printf("Enter ID to remove: ");
                scanf("%s", ID);
                removeSupplierCustomer(&records, &count, ID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(records[i].ID);
    }
    free(records);

    return 0;
}
```

```
1. Add Supplier/Customer Record
2. Display Records
3. Remove Record
4. Exit
Enter your choice: 1
Enter ID: 2
Enter name: blue
Enter transaction type (0 for purchase, 1 for return): 1

1. Add Supplier/Customer Record
2. Display Records
3. Remove Record
4. Exit
Enter your choice: 2
Supplier/Customer Records:
ID: 1 | Name: red | Transaction Type: Purchase
ID: 2 | Name: blue | Transaction Type: Return

1. Add Supplier/Customer Record
2. Display Records
3. Remove Record
4. Exit
Enter your choice:
```

```c
/*Create a freight cost calculation system using structures for cost
components and unions for different pricing models.
 Use const pointers for fixed cost parameters and double pointers for
dynamically allocated cost records.
Specifications:
Structure: Cost components (ID, base cost).
Union: Pricing models (fixed, variable).
const Pointer: Cost parameters.
Double Pointers: Dynamic cost management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for different pricing models (fixed, variable)
union PricingModels {
    float fixed;      // Fixed cost model
```

```
    float variable;   // Variable cost model (e.g., based on weight or
distance)
};

// Structure for cost components
struct CostComponents {
    const char *ID;      // ID of the cost component (const pointer)
    float baseCost;      // Base cost for the component
    union PricingModels pricing; // Pricing model (fixed or variable)
    int pricingModel;    // 0: fixed, 1: variable
};

// Function to add a new cost component
void addCostComponent(struct CostComponents **costs, int *count) {
    // Reallocate memory for the new cost component
    struct CostComponents *temp = realloc(*costs, (*count + 1) *
sizeof(struct CostComponents));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *costs = temp;

    struct CostComponents *newCost = &(*costs)[*count];

    // Allocate memory for the ID (const pointer)
    newCost->ID = (char *)malloc(10 * sizeof(char));
    if (!newCost->ID) {
        printf("Memory allocation failed for ID!\n");
        return;
    }

    // Get cost component details from the user
    printf("Enter ID: ");
    scanf("%s", newCost->ID);

    printf("Enter base cost: ");
    scanf("%f", &newCost->baseCost);

    printf("Enter pricing model (0 for fixed, 1 for variable): ");
```

```c
    scanf("%d", &newCost->pricingModel);

    if (newCost->pricingModel == 0) {
        printf("Enter fixed cost: ");
        scanf("%f", &newCost->pricing.fixed);
    } else if (newCost->pricingModel == 1) {
        printf("Enter variable cost: ");
        scanf("%f", &newCost->pricing.variable);
    } else {
        printf("Invalid pricing model!\n");
        return;
    }

    (*count)++;
}

// Function to remove a cost component by ID
void removeCostComponent(struct CostComponents **costs, int *count, const
char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*costs)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Cost component with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for the ID
    free((*costs)[index].ID);

    // Shift remaining records to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*costs)[i] = (*costs)[i + 1];
    }
```

```
    // Reallocate memory to shrink the array
    struct CostComponents *temp = realloc(*costs, (*count - 1) *
sizeof(struct CostComponents));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *costs = temp;
    (*count)--;
}

// Function to display all cost components
void displayCostComponents(struct CostComponents *costs, int count) {
    if (count == 0) {
        printf("No cost components available.\n");
        return;
    }

    printf("Cost Components:\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %s | Base Cost: %.2f | Pricing Model: ", costs[i].ID,
costs[i].baseCost);

        if (costs[i].pricingModel == 0) {
            printf("Fixed | Fixed Cost: %.2f\n", costs[i].pricing.fixed);
        } else if (costs[i].pricingModel == 1) {
            printf("Variable | Variable Cost: %.2f\n",
costs[i].pricing.variable);
        }
    }
}

// Function to calculate total freight cost
void calculateFreightCost(struct CostComponents *costs, int count) {
    if (count == 0) {
        printf("No cost components available.\n");
        return;
    }
```

```
    float totalCost = 0;

    for (int i = 0; i < count; i++) {
        totalCost += costs[i].baseCost;

        if (costs[i].pricingModel == 0) {
            totalCost += costs[i].pricing.fixed;  // Add fixed cost
        } else if (costs[i].pricingModel == 1) {
            totalCost += costs[i].pricing.variable;  // Add variable cost
        }
    }

    printf("Total Freight Cost: %.2f\n", totalCost);
}

int main() {
    struct CostComponents *costs = (struct CostComponents *)malloc(10 *
sizeof(struct CostComponents)); // Initial allocation for 10 cost
components
    int count = 0;  // To track the number of cost components
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Cost Component\n");
        printf("2. Display Cost Components\n");
        printf("3. Remove Cost Component\n");
        printf("4. Calculate Total Freight Cost\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCostComponent(&costs, &count);
                break;
            case 2:
                displayCostComponents(costs, count);
                break;
            case 3:
```

```
                printf("Enter ID to remove: ");
                scanf("%s", ID);
                removeCostComponent(&costs, &count, ID);
                break;
            case 4:
                calculateFreightCost(costs, count);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(costs[i].ID);
    }
    free(costs);

    return 0;
}
```

```
5. Exit
Enter your choice: 1
Enter ID: 2
Enter base cost: 300
Enter pricing model (0 for fixed, 1 for variable): 1
Enter variable cost: 300

1. Add Cost Component
2. Display Cost Components
3. Remove Cost Component
4. Calculate Total Freight Cost
5. Exit
Enter your choice: 2
Cost Components:
ID: 1 | Base Cost: 200.00 | Pricing Model: Fixed | Fixed Cost: 200.00
ID: 2 | Base Cost: 300.00 | Pricing Model: Variable | Variable Cost: 300.00

1. Add Cost Component
2. Display Cost Components
3. Remove Cost Component
4. Calculate Total Freight Cost
5. Exit
Enter your choice:
```

```c
/*Design a vehicle load balancing system using structures for load details
and unions for load types.
Use const pointers for load identifiers and double pointers for managing
dynamic load distribution.
Specifications:
Structure: Load details (ID, weight, destination).
Union: Load types (bulk, container).
const Pointer: Load IDs.
Double Pointers: Dynamic load handling.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union to represent different load types (bulk or container)
union LoadTypes {
    char bulk[20];         // Description of bulk load (e.g., "grains")
    char container[20];    // Description of container load (e.g.,
"electronics")
};

// Structure for load details
struct LoadDetails {
    const char *ID;        // Load ID (const pointer)
    float weight;          // Weight of the load (for balancing)
```

```c
    const char *destination; // Destination of the load (const pointer)
    union LoadTypes type; // Type of the load (bulk or container)
    int loadType;         // 0: Bulk, 1: Container
};

// Function to add a new load
void addLoad(struct LoadDetails **loads, int *count) {
    // Reallocate memory for the new load
    struct LoadDetails *temp = realloc(*loads, (*count + 1) *
sizeof(struct LoadDetails));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *loads = temp;

    struct LoadDetails *newLoad = &(*loads)[*count];

    // Allocate memory for Load ID and Destination (const pointers)
    newLoad->ID = (char *)malloc(10 * sizeof(char));
    newLoad->destination = (char *)malloc(20 * sizeof(char));

    if (!newLoad->ID || !newLoad->destination) {
        printf("Memory allocation failed for ID or destination!\n");
        return;
    }

    // Get load details from the user
    printf("Enter load ID: ");
    scanf("%s", newLoad->ID);

    printf("Enter weight: ");
    scanf("%f", &newLoad->weight);

    printf("Enter destination: ");
    scanf("%s", newLoad->destination);

    printf("Enter load type (0 for bulk, 1 for container): ");
    scanf("%d", &newLoad->loadType);
```

```c
    if (newLoad->loadType == 0) {
        printf("Enter bulk load description: ");
        scanf("%s", newLoad->type.bulk);
    } else if (newLoad->loadType == 1) {
        printf("Enter container load description: ");
        scanf("%s", newLoad->type.container);
    } else {
        printf("Invalid load type!\n");
        return;
    }

    (*count)++;
}

// Function to remove a load by ID
void removeLoad(struct LoadDetails **loads, int *count, const char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*loads)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Load with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for ID and destination
    free((*loads)[index].ID);
    free((*loads)[index].destination);

    // Shift the remaining loads to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*loads)[i] = (*loads)[i + 1];
    }

    // Reallocate memory to shrink the array
```

```
    struct LoadDetails *temp = realloc(*loads, (*count - 1) *
sizeof(struct LoadDetails));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *loads = temp;
    (*count)--;
}

// Function to display all loads
void displayLoads(struct LoadDetails *loads, int count) {
    if (count == 0) {
        printf("No loads available.\n");
        return;
    }

    printf("Loads Information:\n");
    for (int i = 0; i < count; i++) {
        printf("Load ID: %s | Weight: %.2f | Destination: %s | Load Type:
",
               loads[i].ID, loads[i].weight, loads[i].destination);
        if (loads[i].loadType == 0) {
            printf("Bulk | Description: %s\n", loads[i].type.bulk);
        } else if (loads[i].loadType == 1) {
            printf("Container | Description: %s\n",
loads[i].type.container);
        }
    }
}

// Function to balance load distribution (just for illustration, assume
simple weight-based distribution)
void balanceLoad(struct LoadDetails *loads, int count, float maxWeight) {
    float totalWeight = 0;
    int vehicleCount = 1;  // Assume we start with 1 vehicle

    // Calculate total weight
    for (int i = 0; i < count; i++) {
```

```
        totalWeight += loads[i].weight;
    }

    // Calculate number of vehicles needed
    int requiredVehicles = (totalWeight / maxWeight) + 1;

    printf("Total weight: %.2f\n", totalWeight);
    printf("Each vehicle can carry a maximum of %.2f. Total vehicles
required: %d\n", maxWeight, requiredVehicles);
}

int main() {
    struct LoadDetails *loads = (struct LoadDetails *)malloc(10 *
sizeof(struct LoadDetails)); // Initial allocation for 10 loads
    int count = 0;  // To track the number of loads
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Load\n");
        printf("2. Display Loads\n");
        printf("3. Remove Load\n");
        printf("4. Balance Load Distribution\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addLoad(&loads, &count);
                break;
            case 2:
                displayLoads(loads, count);
                break;
            case 3:
                printf("Enter Load ID to remove: ");
                scanf("%s", ID);
                removeLoad(&loads, &count, ID);
                break;
            case 4:
```

```c
                printf("Enter maximum vehicle weight: ");
                float maxWeight;
                scanf("%f", &maxWeight);
                balanceLoad(loads, count, maxWeight);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(loads[i].ID);
        free(loads[i].destination);
    }
    free(loads);

    return 0;
}
```

```
Enter your choice: 1
Enter load ID: 2
Enter weight: 30
Enter destination: B
Enter load type (0 for bulk, 1 for container): 1
Enter container load description: B

1. Add Load
2. Display Loads
3. Remove Load
4. Balance Load Distribution
5. Exit
Enter your choice: 2
Loads Information:
Load ID: 1 | Weight: 20.00 | Destination: A | Load Type: Bulk | Description: A
Load ID: 2 | Weight: 30.00 | Destination: B | Load Type: Container | Description: B

1. Add Load
2. Display Loads
3. Remove Load
4. Balance Load Distribution
5. Exit
Enter your choice:
```

```c
/*Implement an intermodal transport management system using structures for
transport details and unions for transport modes.
```

```c
Use const pointers for transport identifiers and double pointers for
dynamic transport route management.
Specifications:
Structure: Transport details (ID, origin, destination).
Union: Transport modes (rail, road).
const Pointer: Transport IDs.
Double Pointers: Dynamic transport management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union to represent different transport modes (rail or road)
union TransportModes {
    char rail[20];  // Description for rail transport (e.g., "Cargo
Train")
    char road[20];  // Description for road transport (e.g., "Truck")
};

// Structure for transport details
struct TransportDetails {
    const char *ID;      // Transport ID (const pointer)
    const char *origin;  // Origin of the transport route (const pointer)
    const char *destination; // Destination of the transport route (const
pointer)
    union TransportModes mode; // Transport mode (rail or road)
    int transportMode;   // 0: Rail, 1: Road
};

// Function to add a new transport route
void addTransport(struct TransportDetails **transports, int *count) {
    // Reallocate memory for the new transport route
    struct TransportDetails *temp = realloc(*transports, (*count + 1) *
sizeof(struct TransportDetails));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *transports = temp;

    struct TransportDetails *newTransport = &(*transports)[*count];
```

```
    // Allocate memory for Transport ID, origin, and destination (const
pointers)
    newTransport->ID = (char *)malloc(10 * sizeof(char));
    newTransport->origin = (char *)malloc(20 * sizeof(char));
    newTransport->destination = (char *)malloc(20 * sizeof(char));

    if (!newTransport->ID || !newTransport->origin ||
!newTransport->destination) {
        printf("Memory allocation failed for ID, origin or
destination!\n");
        return;
    }

    // Get transport details from the user
    printf("Enter transport ID: ");
    scanf("%s", newTransport->ID);

    printf("Enter origin: ");
    scanf("%s", newTransport->origin);

    printf("Enter destination: ");
    scanf("%s", newTransport->destination);

    printf("Enter transport mode (0 for rail, 1 for road): ");
    scanf("%d", &newTransport->transportMode);

    if (newTransport->transportMode == 0) {
        printf("Enter rail transport description: ");
        scanf("%s", newTransport->mode.rail);
    } else if (newTransport->transportMode == 1) {
        printf("Enter road transport description: ");
        scanf("%s", newTransport->mode.road);
    } else {
        printf("Invalid transport mode!\n");
        return;
    }

    (*count)++;
}
```

```c
// Function to remove a transport route by ID
void removeTransport(struct TransportDetails **transports, int *count,
const char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*transports)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Transport with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for ID, origin, and destination
    free((*transports)[index].ID);
    free((*transports)[index].origin);
    free((*transports)[index].destination);

    // Shift the remaining transports to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*transports)[i] = (*transports)[i + 1];
    }

    // Reallocate memory to shrink the array
    struct TransportDetails *temp = realloc(*transports, (*count - 1) *
sizeof(struct TransportDetails));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *transports = temp;
    (*count)--;
}

// Function to display all transport routes
```

```c
void displayTransports(struct TransportDetails *transports, int count) {
    if (count == 0) {
        printf("No transport routes available.\n");
        return;
    }

    printf("Transport Routes Information:\n");
    for (int i = 0; i < count; i++) {
        printf("Transport ID: %s | Origin: %s | Destination: %s |
Transport Mode: ",
               transports[i].ID, transports[i].origin,
transports[i].destination);
        if (transports[i].transportMode == 0) {
            printf("Rail | Description: %s\n", transports[i].mode.rail);
        } else if (transports[i].transportMode == 1) {
            printf("Road | Description: %s\n", transports[i].mode.road);
        }
    }
}

// Function to optimize transport routes (for simplicity, display total
routes and mode usage)
void optimizeRoutes(struct TransportDetails *transports, int count) {
    int railCount = 0, roadCount = 0;

    for (int i = 0; i < count; i++) {
        if (transports[i].transportMode == 0) {
            railCount++;
        } else if (transports[i].transportMode == 1) {
            roadCount++;
        }
    }

    printf("Total Transport Routes: %d\n", count);
    printf("Rail Transport Routes: %d\n", railCount);
    printf("Road Transport Routes: %d\n", roadCount);
}

int main() {
```

```c
    struct TransportDetails *transports = (struct TransportDetails
*)malloc(10 * sizeof(struct TransportDetails)); // Initial allocation for
10 transport routes
    int count = 0;  // To track the number of transport routes
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Transport Route\n");
        printf("2. Display Transport Routes\n");
        printf("3. Remove Transport Route\n");
        printf("4. Optimize Transport Routes\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTransport(&transports, &count);
                break;
            case 2:
                displayTransports(transports, count);
                break;
            case 3:
                printf("Enter Transport ID to remove: ");
                scanf("%s", ID);
                removeTransport(&transports, &count, ID);
                break;
            case 4:
                optimizeRoutes(transports, count);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free all allocated memory
```

```
    for (int i = 0; i < count; i++) {
        free(transports[i].ID);
        free(transports[i].origin);
        free(transports[i].destination);
    }
    free(transports);

    return 0;
}
```

```
Enter your choice: 1
Enter transport ID: 2
Enter origin: C
Enter destination: D
Enter transport mode (0 for rail, 1 for road): 1
Enter road transport description: truck

1. Add Transport Route
2. Display Transport Routes
3. Remove Transport Route
4. Optimize Transport Routes
5. Exit
Enter your choice: 2
Transport Routes Information:
Transport ID: 1 | Origin: A | Destination: B | Transport Mode: Rail | Description: rail
Transport ID: 2 | Origin: C | Destination: D | Transport Mode: Road | Description: truck

1. Add Transport Route
2. Display Transport Routes
3. Remove Transport Route
4. Optimize Transport Routes
5. Exit
Enter your choice:
```

```
/*Develop a logistics performance monitoring system using structures for
performance metrics and unions for different performance aspects.
 Use const pointers for metric identifiers and double pointers for
managing dynamic performance records.
Specifications:
Structure: Performance metrics (ID, value).
Union: Performance aspects (time, cost).
const Pointer: Metric IDs.
Double Pointers: Dynamic performance tracking.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
// Union to represent different performance aspects (time or cost)
union PerformanceAspects {
    float time;  // Time-related metric (e.g., delivery time in hours)
    float cost;  // Cost-related metric (e.g., transportation cost in
dollars)
};

// Structure for performance metrics
struct PerformanceMetrics {
    const char *ID;         // Metric ID (const pointer)
    float value;            // Value of the metric
    union PerformanceAspects aspect; // Performance aspect (time or cost)
    int aspectType;         // 0: Time, 1: Cost
};

// Function to add a new performance metric
void addPerformanceMetric(struct PerformanceMetrics **metrics, int *count)
{
    // Reallocate memory for a new performance metric
    struct PerformanceMetrics *temp = realloc(*metrics, (*count + 1) *
sizeof(struct PerformanceMetrics));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *metrics = temp;

    struct PerformanceMetrics *newMetric = &(*metrics)[*count];

    // Allocate memory for Metric ID (const pointer)
    newMetric->ID = (char *)malloc(10 * sizeof(char));

    if (!newMetric->ID) {
        printf("Memory allocation failed for Metric ID!\n");
        return;
    }

    // Get metric details from the user
    printf("Enter performance metric ID: ");
    scanf("%s", newMetric->ID);
```

```c
    printf("Enter value of the metric: ");
    scanf("%f", &newMetric->value);

    printf("Enter performance aspect (0 for time, 1 for cost): ");
    scanf("%d", &newMetric->aspectType);

    if (newMetric->aspectType == 0) {
        printf("Enter time-related metric value (in hours): ");
        scanf("%f", &newMetric->aspect.time);
    } else if (newMetric->aspectType == 1) {
        printf("Enter cost-related metric value (in dollars): ");
        scanf("%f", &newMetric->aspect.cost);
    } else {
        printf("Invalid performance aspect!\n");
        return;
    }

    (*count)++;
}

// Function to remove a performance metric by ID
void removePerformanceMetric(struct PerformanceMetrics **metrics, int
*count, const char *ID) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*metrics)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Performance metric with ID %s not found!\n", ID);
        return;
    }

    // Free the allocated memory for Metric ID
    free((*metrics)[index].ID);
```

```
    // Shift the remaining metrics to fill the gap
    for (int i = index; i < *count - 1; i++) {
        (*metrics)[i] = (*metrics)[i + 1];
    }

    // Reallocate memory to shrink the array
    struct PerformanceMetrics *temp = realloc(*metrics, (*count - 1) *
sizeof(struct PerformanceMetrics));
    if (!temp && (*count - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *metrics = temp;
    (*count)--;
}

// Function to display all performance metrics
void displayPerformanceMetrics(struct PerformanceMetrics *metrics, int
count) {
    if (count == 0) {
        printf("No performance metrics available.\n");
        return;
    }

    printf("Performance Metrics Information:\n");
    for (int i = 0; i < count; i++) {
        printf("Metric ID: %s | Value: %.2f | Aspect: ", metrics[i].ID,
metrics[i].value);
        if (metrics[i].aspectType == 0) {
            printf("Time | Time: %.2f hours\n", metrics[i].aspect.time);
        } else if (metrics[i].aspectType == 1) {
            printf("Cost | Cost: %.2f dollars\n", metrics[i].aspect.cost);
        }
    }
}

// Function to calculate average time or cost
void calculateAverage(struct PerformanceMetrics *metrics, int count, int
aspectType) {
```

```c
    if (count == 0) {
        printf("No metrics to calculate average.\n");
        return;
    }

    float total = 0;
    int validCount = 0;

    for (int i = 0; i < count; i++) {
        if (metrics[i].aspectType == aspectType) {
            if (aspectType == 0) {
                total += metrics[i].aspect.time;
            } else if (aspectType == 1) {
                total += metrics[i].aspect.cost;
            }
            validCount++;
        }
    }

    if (validCount == 0) {
        printf("No metrics of the selected aspect type.\n");
        return;
    }

    printf("Average value: %.2f\n", total / validCount);
}

int main() {
    struct PerformanceMetrics *metrics = (struct PerformanceMetrics
*)malloc(10 * sizeof(struct PerformanceMetrics)); // Initial allocation
for 10 metrics
    int count = 0;  // To track the number of metrics
    int choice;
    char ID[10];

    do {
        printf("\n1. Add Performance Metric\n");
        printf("2. Display Performance Metrics\n");
        printf("3. Remove Performance Metric\n");
        printf("4. Calculate Average Time/Cost\n");
```

```
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addPerformanceMetric(&metrics, &count);
                break;
            case 2:
                displayPerformanceMetrics(metrics, count);
                break;
            case 3:
                printf("Enter Metric ID to remove: ");
                scanf("%s", ID);
                removePerformanceMetric(&metrics, &count, ID);
                break;
            case 4:
                printf("Enter aspect type for average calculation (0 for
time, 1 for cost): ");
                int aspectType;
                scanf("%d", &aspectType);
                calculateAverage(metrics, count, aspectType);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free all allocated memory
    for (int i = 0; i < count; i++) {
        free(metrics[i].ID);
    }
    free(metrics);

    return 0;
}
```

```
Enter performance aspect (0 for time, 1 for cost): 1
Enter cost-related metric value (in dollars): 300

1. Add Performance Metric
2. Display Performance Metrics
3. Remove Performance Metric
4. Calculate Average Time/Cost
5. Exit
Enter your choice: 2
Performance Metrics Information:
Metric ID: 1 | Value: 20.00 | Aspect: Time | Time: 22.00 hours
Metric ID: 2 | Value: 30.00 | Aspect: Cost | Cost: 300.00 dollars

1. Add Performance Metric
2. Display Performance Metrics
3. Remove Performance Metric
4. Calculate Average Time/Cost
```

```c
/*Create a system to coordinate warehouse robotics using structures for
robot details and unions for task types.
Use const pointers for robot identifiers and double pointers for managing
dynamic task allocations.
Specifications:
Structure: Robot details (ID, type, status).
Union: Task types (picking, sorting).
const Pointer: Robot IDs.
Double Pointers: Dynamic task management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union to represent different task types (picking or sorting)
union TaskTypes {
    char picking[20];   // Picking task (e.g., "Pick Item")
    char sorting[20];   // Sorting task (e.g., "Sort Items")
};

// Structure for robot details
struct RobotDetails {
    const char *ID;       // Robot ID (const pointer)
    const char *type;     // Robot type (e.g., "Picker", "Sorter")
    const char *status;   // Robot status (e.g., "Idle", "Active",
"Charging")
    union TaskTypes task; // Task type (picking or sorting)
    int taskType;         // 0: Picking, 1: Sorting
```

```c
};

// Function to add a new robot
void addRobot(struct RobotDetails **robots, int *robotCount) {
    // Reallocate memory for a new robot
    struct RobotDetails *temp = realloc(*robots, (*robotCount + 1) *
sizeof(struct RobotDetails));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    *robots = temp;

    struct RobotDetails *newRobot = &(*robots)[*robotCount];

    // Allocate memory for Robot ID, type, and status (const pointers)
    newRobot->ID = (char *)malloc(10 * sizeof(char));
    newRobot->type = (char *)malloc(20 * sizeof(char));
    newRobot->status = (char *)malloc(20 * sizeof(char));

    if (!newRobot->ID || !newRobot->type || !newRobot->status) {
        printf("Memory allocation failed for Robot ID, type or
status!\n");
        return;
    }

    // Get robot details from the user
    printf("Enter robot ID: ");
    scanf("%s", newRobot->ID);

    printf("Enter robot type (e.g., Picker, Sorter): ");
    scanf("%s", newRobot->type);

    printf("Enter robot status (e.g., Idle, Active, Charging): ");
    scanf("%s", newRobot->status);

    (*robotCount)++;
}

// Function to assign a task to a robot
```

```c
void assignTask(struct RobotDetails *robots, int robotCount, const char
*robotID, int taskType) {
    int index = -1;
    for (int i = 0; i < robotCount; i++) {
        if (strcmp(robots[i].ID, robotID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Robot with ID %s not found!\n", robotID);
        return;
    }

    if (taskType == 0) {
        robots[index].taskType = 0;
        strcpy(robots[index].task.picking, "Pick Item");
        printf("Robot %s assigned to picking task.\n", robotID);
    } else if (taskType == 1) {
        robots[index].taskType = 1;
        strcpy(robots[index].task.sorting, "Sort Items");
        printf("Robot %s assigned to sorting task.\n", robotID);
    } else {
        printf("Invalid task type!\n");
    }
}

// Function to remove a robot by ID
void removeRobot(struct RobotDetails **robots, int *robotCount, const char
*robotID) {
    int index = -1;
    for (int i = 0; i < *robotCount; i++) {
        if (strcmp((*robots)[i].ID, robotID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
```

```
        printf("Robot with ID %s not found!\n", robotID);
        return;
    }

    // Free the allocated memory for Robot ID, type, and status
    free((*robots)[index].ID);
    free((*robots)[index].type);
    free((*robots)[index].status);

    // Shift the remaining robots to fill the gap
    for (int i = index; i < *robotCount - 1; i++) {
        (*robots)[i] = (*robots)[i + 1];
    }

    // Reallocate memory to shrink the array
    struct RobotDetails *temp = realloc(*robots, (*robotCount - 1) *
sizeof(struct RobotDetails));
    if (!temp && (*robotCount - 1) > 0) {
        printf("Memory reallocation failed!\n");
        return;
    }

    *robots = temp;
    (*robotCount)--;
}

// Function to display all robot details
void displayRobots(struct RobotDetails *robots, int robotCount) {
    if (robotCount == 0) {
        printf("No robots available.\n");
        return;
    }

    printf("Robot Information:\n");
    for (int i = 0; i < robotCount; i++) {
        printf("Robot ID: %s | Type: %s | Status: %s | Task: ",
robots[i].ID, robots[i].type, robots[i].status);
        if (robots[i].taskType == 0) {
            printf("Picking | Task: %s\n", robots[i].task.picking);
        } else if (robots[i].taskType == 1) {
```

```
            printf("Sorting | Task: %s\n", robots[i].task.sorting);
        }
    }
}

int main() {
    struct RobotDetails *robots = (struct RobotDetails *)malloc(10 *
sizeof(struct RobotDetails)); // Initial allocation for 10 robots
    int robotCount = 0;  // To track the number of robots
    int choice;
    char robotID[10];

    do {
        printf("\n1. Add Robot\n");
        printf("2. Assign Task to Robot\n");
        printf("3. Remove Robot\n");
        printf("4. Display Robots\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addRobot(&robots, &robotCount);
                break;
            case 2:
                printf("Enter Robot ID to assign task: ");
                scanf("%s", robotID);
                printf("Enter task type (0 for picking, 1 for sorting):
");
                int taskType;
                scanf("%d", &taskType);
                assignTask(robots, robotCount, robotID, taskType);
                break;
            case 3:
                printf("Enter Robot ID to remove: ");
                scanf("%s", robotID);
                removeRobot(&robots, &robotCount, robotID);
                break;
            case 4:
```

```c
                displayRobots(robots, robotCount);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free all allocated memory
    for (int i = 0; i < robotCount; i++) {
        free(robots[i].ID);
        free(robots[i].type);
        free(robots[i].status);
    }
    free(robots);

    return 0;
}
```

```
1. Add Robot
2. Assign Task to Robot
3. Remove Robot
4. Display Robots
5. Exit
Enter your choice: 4
Robot Information:
Robot ID: 1 | Type: Picker | Status: Idle | Task: Picking | Task: Pick Item
Robot ID: 2 | Type: Sorter | Status: Active | Task:
1. Add Robot
2. Assign Task to Robot
3. Remove Robot
4. Display Robots
5. Exit
```

```c
/*Design a system to analyze customer feedback using structures for
feedback details and unions for feedback types.
Use const pointers for feedback IDs and double pointers for dynamically
managing feedback data.
Specifications:
Structure: Feedback details (ID, content).
Union: Feedback types (positive, negative).
const Pointer: Feedback IDs.
Double Pointers: Dynamic feedback management.*/
```

```c
/*Design a system to analyze customer feedback using structures for
feedback details and unions for feedback types.
Use const pointers for feedback IDs and double pointers for dynamically
managing feedback data.
Specifications:
Structure: Feedback details (ID, content).
Union: Feedback types (positive, negative).
const Pointer: Feedback IDs.
Double Pointers: Dynamic feedback management.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Type
{
    int positive;
    int negative;
};
struct Feedback
{
    const char *id;
    char content[40];
    union Type type;

};
void add(struct Feedback **f,int *count)
{
    char *good[2]={"good","satisfied"};
    char *bad[2]={"bad","worst"};
    printf("Enter feddback id : ");
    (*f)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*f)[*count].id);
    getchar();
    printf("Enter content : ");
    fgets((*f)[*count].content,40,stdin);
    (*f)[*count].content[strcspn((*f)[*count].content, "\n")] = '\0';

    for(int i=0;i<2;i++)
    {
        if(strstr((*f)[*count].content,bad[i])!=NULL)
    {
```

```
        (*f)[*count].type.negative=1;
        break;
    }
     else if(strstr((*f)[*count].content,good[i])!=NULL)
    {
        (*f)[*count].type.negative=0;
        break;
    }
    }
    (*count)++;

}
void delete(struct Feedback **f,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*f)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count;i++)
    (*f)[i]=(*f)[i+1];
    (*count)--;
}
void display(struct Feedback *f,int count)
{
    printf("\nFeedback list\n");

printf("..............................................................\n
");
    for(int i=0;i<count;i++)
    {
        printf("Feedback Id : %s | Feedback type : ",f[i].id);
        if(f[i].type.negative==0)
        printf("Positive\n");
        else if(f[i].type.negative==1)
        printf("Negative\n");
```

```
        printf("Feedback : %s\n",f[i].content);

printf("..............................................................\n
");
    }
}
void main()
{
    struct Feedback *f=(struct Feedback *)malloc(10*sizeof(struct
Feedback));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Order\n");
        printf("2. Display Order\n");
        printf("3. Remove order\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&f, &count);
                break;
            case 2:
                display(f,count);
                break;
            case 3:
                printf("Enter Item ID to remove: ");
                scanf("%s", id);
                delete(&f,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
```

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


}
```

```
Enter feddback id : 2
Enter content : good work

1. Add Order
2. Display Order
3. Remove order
4. Exit
Enter your choice: 2

Feedback list
.............................................................
Feedback Id : 1 | Feedback type : Negative
Feedback : bad work
.............................................................
Feedback Id : 2 | Feedback type : Positive
Feedback : good work
.............................................................

1. Add Order
2. Display Order
3. Remove order
4. Exit
Enter your choice:
```

```
/*Implement a real-time fleet coordination system using structures for
fleet details and unions for coordination types.
Use const pointers for fleet IDs and double pointers for managing dynamic
coordination data.
Specifications:
Structure: Fleet details (ID, location, status).
Union: Coordination types (dispatch, reroute).
const Pointer: Fleet IDs.
Double Pointers: Dynamic coordination.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Cord
{
    char dispatch[10];
```

```
    char reroute[10];
};
struct Fleet
{
    const char *id;
    char loc[20];
    int status; //0 dispatch,1 reroute
    union Cord c;
};
void add(struct Fleet **f,int *count)
{
    printf("Enter fleet id :");
    (*f)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*f)[*count].id);
    printf("Enter location\n");
    getchar();
    scanf("%s",(*f)[*count].loc);
    printf("Enter status : ");
    scanf("%d",&(*f)[*count].status);
    if((*f)[*count].status==0)
    {
        printf("Dispatch from : ");
        scanf("%s",(*f)[*count].c.dispatch);
    }
    else
    {
        printf("Reroute to : ");
        scanf("%s",(*f)[*count].c.reroute);
    }

(*count)++;
}
void delete(struct Fleet **f,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*f)[i].id,id)==0)
        {
            index=i;
```

```
            break;
        }
    }
    for(int i=index;i<*count;i++)
    {
        (*f)[i]=(*f)[i+1];
    }
    (*count)--;
}
void display(struct Fleet *f,int count )
{
    printf("Current Fleets :\n ");
    for(int i=0;i<count;i++)
    {
        printf("Fleet id : %s | Location : %s |",f[i].id,f[i].loc);
        if(f[i].status==0)
        printf("Status : Dispatch | From : %s\n",f[i].c.dispatch);
        else
        printf("Status : Reroute | To : %s\n",f[i].c.reroute);
    }
}
void main()
{
    struct Fleet *f=(struct Fleet *)malloc(10*sizeof(struct Fleet));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Order\n");
        printf("2. Display Order\n");
        printf("3. Remove order\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&f, &count);
                break;
```

```c
            case 2:
                display(f,count);
                break;
            case 3:
                printf("Enter fleet ID to remove: ");
                scanf("%s", id);
                delete(&f,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
Reroute to : C

1. Add Order
2. Display Order
3. Remove order
4. Exit
Enter your choice: 2
Current Fleets :
 Fleet id : 1 | Location : A |Status : Dispatch | From : A
Fleet id : 2 | Location : B |Status : Reroute | To : C

1. Add Order
2. Display Order
3. Remove order
4. Exit
Enter your choice:
```

```c
/*Develop a security management system for logistics using structures for
security events and unions for event types.
Use const pointers for event identifiers and double pointers for managing
dynamic security data.
Specifications:
Structure: Security events (ID, description).
Union: Event types (breach, resolved).
const Pointer: Event IDs.
Double Pointers: Dynamic security event handling.*/
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Event
{
    char breach[20];
    char resolve[20];
};
struct Security
{
    const char *id;
    char description[20];
    int type;//0 breach,1 resolve;
    union Event e;

};
void add(struct Security **s,int *count)
{
    printf("Enter securit event id : ");
    (*s)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*s)[*count].id);
    getchar();
    printf("Enter description : ");
    fgets((*s)[*count].description,20,stdin);
    printf("Enter event type : ");
    scanf("%d",&(*s)[*count].type);
    if((*s)[*count].type==0)
    {
        printf("Enter breached sector name : ");
        scanf("%s",(*s)[*count].e.breach);
    }
    else
    {
        printf("Enter resolved issue : ");
        scanf("%s",(*s)[*count].e.resolve);
    }
(*count)++;

}
void delete(struct Security **s,int *count,const char *id)
```

```
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count;i++)
    (*s)[i]=(*s)[i+1];

    (*count)--;
}
void display(struct Security *s,int count)
{
    printf("Current events :\n");
    printf("..........................................................\n");
    for(int i=0;i<count;i++)
    {
        printf("Event id :%s \n",s[i].id);
        printf("Description : %s\n",s[i].description);
        if(s[i].type==0)
        {
            printf("Breach | Sector : %s\n",s[i].e.breach);
        }
        else
        {
            printf("Resolved | Issue : %s\n",s[i].e.resolve);
        }

printf(".......................................................\n");
    }

}
void main()
{
    struct Security *e=(struct Security *)malloc(10*sizeof(struct
Security));
```

```c
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Event\n");
        printf("2. Display Event\n");
        printf("3. Remove Event\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&e, &count);
                break;
            case 2:
                display(e,count);
                break;
            case 3:
                printf("Enter Security ID to remove: ");
                scanf("%s", id);
                delete(&e,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
1. Add Event
2. Display Event
3. Remove Event
4. Exit
Enter your choice: 2
Current events :
.....................................................
Event id :1
Description : a

Breach | Sector : A
.....................................................
Event id :2
Description : b

Resolved | Issue : bug
.....................................................

1. Add Event
2. Display Event
3. Remove Event
4. Exit
Enter your choice:
```

```c
/*Create an automated billing system using structures for billing details
and unions for payment methods.
Use const pointers for bill IDs and double pointers for dynamically
managing billing records.
Specifications:
Structure: Billing details (ID, amount, date).
Union: Payment methods (bank transfer, cash).
const Pointer: Bill IDs.
Double Pointers: Dynamic billing management.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Method
{
    char bid[10];
    int  notes;
};
struct Bill
{
    const char *id;
    float amount;
    char date[11];
    int type;//0 bank transfer ,1 cash
```

```
    union Method m;

};
void add(struct Bill **b,int *count)
{
    printf("Enter bill id :");
    (*b)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*b)[*count].id);
    printf("Enter amount : ");
    scanf("%f",&(*b)[*count].amount);
    printf("Enter date : ");
    getchar();
    scanf("%s",(*b)[*count].date);
    printf("Enter type : ");
    scanf("%d",&(*b)[*count].type);
    if((*b)[*count].type==0)
    {
        printf("Enter account number : ");
        scanf("%s",(*b)[*count].m.bid);
    }
    else
    {
        printf("Enter number of notes : ");
        scanf("%d",&(*b)[*count].m.notes);
    }
    (*count)++;
}
void delete(struct Bill **b,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*b)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    for(int i=index;i<*count;i++)
    (*b)[i]=(*b)[i+1];
```

```
    (*count)--;
}
void display(struct Bill *b,int count)
{
    printf("Current bills\n");

printf("..............................................................\n");
    for(int i=0;i<count;i++)
    {
        printf("Bill id  : %s | Amount : %.2f | Date : %s
|",b[i].id,b[i].amount,b[i].date);
        if(b[i].type==0)
        printf("Type : Bank transfer | Bank no : %s\n",b[i].m.bid);
        else
        printf("Type : Cash | Notes : %d\n",b[i].m.notes);

printf("..............................................................\n");
    }
}
void main()
{
    struct Bill *b=(struct Bill *)malloc(10*sizeof(struct Bill));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Event\n");
        printf("2. Display Event\n");
        printf("3. Remove Event\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&b, &count);
                break;
            case 2:
                display(b,count);
```

```
                break;
            case 3:
                printf("Enter Bill ID to remove: ");
                scanf("%s", id);
                delete(&b,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
Enter bill id :2
Enter amount : 30000
Enter date : 24-12-2024
Enter type : 1
Enter number of notes : 300

1. Add Event
2. Display Event
3. Remove Event
4. Exit
Enter your choice: 2
Current bills
.............................................................
Bill id  : 1 | Amount : 2000.00 | Date : 22-12-2024 |Type : Bank transfer | Bank no : 10001
.............................................................
Bill id  : 2 | Amount : 30000.00 | Date : 24-12-2024 |Type : Cash | Notes : 300
.............................................................

1. Add Event
2. Display Event
3. Remove Event
4. Exit
Enter your choice:
```

```
/*Design a navigation system that tracks a vessel's current position and
routes using structures and arrays.
Use const pointers for immutable route coordinates and strings for
location names. Double pointers handle dynamic route allocation.
Specifications:
Structure: Route details (start, end, waypoints).
Array: Stores multiple waypoints.
Strings: Names of locations.
const Pointers: Route coordinates.
```

```
Double Pointers: Dynamic allocation of route*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Waypoint
{
    char waypoint[20];
};
struct Route
{
    const char *cordinate;
    char start[20];
    char end[20];
    struct Waypoint w[5];
};
void add(struct Route **r,int *count)
{
    printf("Enter route cordinate : ");
    (*r)[*count].cordinate=(char *)malloc(10*sizeof(char));
    scanf("%s",(*r)[*count].cordinate);
    printf("Enter start location :");
    scanf("%s",(*r)[*count].start);
    printf("Enter end location : ");
    scanf("%s",(*r)[*count].end);
    printf("Enter waypoints : \n");
    for(int i=0;i<5;i++)
    {
        printf("waipoint %d : ",i+1);
        scanf("%s",(*r)[*count].w[i].waypoint);
    }
(*count)++;

}
void delete(struct Route **r,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*r)[i].cordinate,id)==0)
        {
```

```
            index=i;
            break;
        }
    }
    for(int i=index;i<*count;i++)
    (*r)[i]=(*r)[i+1];
    (*count)--;
}
void display(struct Route *r,int count)
{
    printf("Current Routes : \n");

printf("..................................................................
.\n");
    for(int i=0;i<count;i++)
    {
        printf("Coordinate : %s | Start : %s | End : %s
|\n",r[i].cordinate,r[i].start,r[i].end);
        for(int j=0;j<5;j++)
        {
            printf(" Waypoint %d : %s |",j+1,r[i].w[j].waypoint);
        }
        printf("\n");

printf("..................................................................
.\n");
    }
}
void main()
{
    struct Route *b=(struct Route *)malloc(10*sizeof(struct Route));
    int count=0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Route\n");
        printf("2. Display Route\n");
        printf("3. Remove Route\n");
        printf("4. Exit\n");
```

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

        switch (choice) {
            case 1:
                add(&b, &count);
                break;
            case 2:
                display(b,count);
                break;
            case 3:
                printf("Enter Route to remove: ");
                scanf("%s", id);
                delete(&b,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);


}
```

```
waipoint 4 : 35.10
waipoint 5 : 35.56

1. Add Route
2. Display Route
3. Remove Route
4. Exit
Enter your choice: 2
Current Routes :
.................................................................
Coordinate : 22.55 | Start : A | End : B |
 Waypoint 1 : 22.56 | Waypoint 2 : 22.57 | Waypoint 3 : 22.58 | Waypoint 4 : 22.59 | Waypoint 5 : 22.60 |
.................................................................
Coordinate : 33.66 | Start : D | End : J |
 Waypoint 1 : 33.77 | Waypoint 2 : 34.80 | Waypoint 3 : 34.90 | Waypoint 4 : 35.10 | Waypoint 5 : 35.56 |
.................................................................

1. Add Route
2. Display Route
3. Remove Route
4. Exit
```

```c
/*Develop a system to manage multiple vessels in a fleet, using arrays for
storing fleet data and structures for vessel details.
Unions represent variable attributes like cargo type or passenger count.
Specifications:
```

```c
Structure: Vessel details (name, ID, type).
Union: Cargo type or passenger count.
Array: Fleet data.
const Pointers: Immutable vessel IDs.
Double Pointers: Manage dynamic fleet records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Payload
{
    int passenger;
    char cargo[10];
};
struct Vessel
{
    char name[20];
    const char *id;
    int type;//0cargo 1 passenger;
   union Payload p;
};
void add(struct Vessel **v,int *count)
{
    printf("Enter vessel id : ");
    (*v)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*v)[*count].id);
    printf("Enter vessel name : ");
    scanf("%s",(*v)[*count].name);
    printf("Enter cargo type :");
    scanf("%d",&(*v)[*count].type);
    if((*v)[*count].type==0)
    {
        printf("Enter cargo  :");
        scanf("%s",(*v)[*count].p.cargo);
    }
    else
    {
        printf("Enter number of passengers : ");
        scanf("%d",&(*v)[*count].p.passenger);
    }
    (*count)++;
```

```c
}
void delete(struct Vessel **v,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*v)[i].id,id)==0)
        {
            index=i;
            break;
        }

    }
    for(int i=index;i<*count;i++)
    {
        (*v)[i]=(*v)[i+1];
    }
    (*count)--;
}
void display(struct Vessel *v,int count)
{
    for(int i=0;i<count;i++)
    {

        printf("Vessel id : %s | Vessel name : %s | ",v[i].id,v[i].name);
        if(v[i].type==0)
        {
            printf("Cargo : %s",v[i].p.cargo);
        }
        else
        {
            printf("Number of passengers : %d",v[i].p.passenger);
        }
        printf("\n");
    }
}
void main()
{
    struct Vessel *v=(struct Vessel *)malloc(10*sizeof(struct Vessel));
    int count=0;
```

```c
    int choice;
    char id[10];

    do {
        printf("\n1. Add Vessel\n");
        printf("2. Display Vessel\n");
        printf("3. Remove Vessel\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add(&v, &count);
                break;
            case 2:
                display(v,count);
                break;
            case 3:
                printf("Enter Vessel to remove: ");
                scanf("%s", id);
                delete(&v,&count,id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

}
```

```
1. Add Vessel
2. Display Vessel
3. Remove Vessel
4. Exit
Enter your choice: 1
Enter vessel id : 2
Enter vessel name : b
Enter cargo type :1
Enter number of passengers : 134

1. Add Vessel
2. Display Vessel
3. Remove Vessel
4. Exit
Enter your choice: 2
Vessel id : 1 | Vessel name : a | Cargo : gem
Vessel id : 2 | Vessel name : b | Number of passengers : 134

1. Add Vessel
2. Display Vessel
3. Remove Vessel
4. Exit
Enter your choice:
```

```c
/*Create a scheduler for ship maintenance tasks. Use structures to define
tasks and arrays for schedules.
Utilize double pointers for managing dynamic task lists.
Specifications:
Structure: Maintenance task (ID, description, schedule).
Array: Maintenance schedules.
const Pointers: Read-only task IDs.
Double Pointers: Dynamic task lists.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for a Maintenance Task
struct MaintenanceTask {
    const char *id;    // Read-only Task ID
    char description[100];
    char schedule[20]; // Schedule (e.g., "Weekly", "Monthly")
};

// Function to add a task dynamically
void addTask(struct MaintenanceTask **tasks, int *count) {
```

```
    *tasks = (struct MaintenanceTask *)realloc(*tasks, (*count + 1) *
sizeof(struct MaintenanceTask));

    (*tasks)[*count].id = (char *)malloc(10 * sizeof(char));

    printf("Enter Task ID: ");
    scanf("%s", (char *)(*tasks)[*count].id);

    getchar(); // Consume newline from previous input
    printf("Enter Description: ");
    fgets((*tasks)[*count].description,
sizeof((*tasks)[*count].description), stdin);
    (*tasks)[*count].description[strcspn((*tasks)[*count].description,
"\n")] = 0; // Remove newline

    printf("Enter Schedule (e.g., Weekly, Monthly): ");
    scanf("%s", (*tasks)[*count].schedule);

    (*count)++;
}

// Function to display all tasks
void displayTasks(struct MaintenanceTask *tasks, int count) {
    printf("\n=== Ship Maintenance Tasks ===\n");
    for (int i = 0; i < count; i++) {
        printf("Task ID: %s\n", tasks[i].id);
        printf("Description: %s\n", tasks[i].description);
        printf("Schedule: %s\n", tasks[i].schedule);
        printf("-------------------------------\n");
    }
}

// Function to delete a task by ID
void deleteTask(struct MaintenanceTask **tasks, int *count, const char
*id) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*tasks)[i].id, id) == 0) {
            index = i;
            break;
```

```
        }
    }

    if (index == -1) {
        printf("Task ID not found!\n");
        return;
    }

    free((char *)(*tasks)[index].id); // Free allocated memory for task ID

    for (int i = index; i < *count - 1; i++) {
        (*tasks)[i] = (*tasks)[i + 1];
    }

    *tasks = (struct MaintenanceTask *)realloc(*tasks, (*count - 1) *
sizeof(struct MaintenanceTask));
    (*count)--;
    printf("Task deleted successfully!\n");
}

// Main function
int main() {
    struct MaintenanceTask *tasks = NULL;
    int count = 0;
    int choice;
    char id[10];

    do {
        printf("\n1. Add Task\n");
        printf("2. Display Tasks\n");
        printf("3. Remove Task\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTask(&tasks, &count);
                break;
            case 2:
```

```
                displayTasks(tasks, count);
                break;
            case 3:
                printf("Enter Task ID to remove: ");
                scanf("%s", id);
                deleteTask(&tasks, &count, id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free allocated memory before exiting
    for (int i = 0; i < count; i++) {
        free((char *)tasks[i].id);
    }
    free(tasks);

    return 0;
}
```

```
Enter Description: B
Enter Schedule (e.g., Weekly, Monthly): Monthly

1. Add Task
2. Display Tasks
3. Remove Task
4. Exit
Enter your choice: 2

=== Ship Maintenance Tasks ===
Task ID: 1
Description: A
Schedule: Weekily
------------------------------
Task ID: 2
Description: B
Schedule: Monthly
------------------------------

1. Add Task
2. Display Tasks
3. Remove Task
4. Exit
Enter your choice:
```

```c
/*Design a system to optimize cargo loading using arrays for storing cargo
weights and structures for vessel specifications.
Unions represent variable cargo properties like dimensions or temperature
requirements.
Specifications:
Structure: Vessel specifications (capacity, dimensions).
Union: Cargo properties (weight, dimensions).
Array: Cargo data.
const Pointers: Protect cargo data.
Double Pointers: Dynamic cargo list allocation.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for cargo properties (weight, dimensions, temperature)
union CargoProperties {
    float weight;           // Weight in tons
    struct {                // Dimensions (length, width, height)
        float length;
```

```
        float width;
        float height;
    } dimensions;
    float temperature;       // Temperature requirement (for perishable
goods)
};

// Structure for cargo data
struct Cargo {
    const char *id;           // Read-only Cargo ID
    char type[20];            // Cargo Type (e.g., "Container", "Bulk",
"Refrigerated")
    union CargoProperties prop; // Variable properties
};

// Structure for vessel specifications
struct Vessel {
    float capacity;           // Max weight capacity (tons)
    struct {                  // Vessel dimensions
        float length;
        float width;
        float height;
    } dimensions;
};

// Function to add cargo dynamically
void addCargo(struct Cargo **cargoList, int *count) {
    *cargoList = (struct Cargo *)realloc(*cargoList, (*count + 1) *
sizeof(struct Cargo));

    (*cargoList)[*count].id = (char *)malloc(10 * sizeof(char));

    printf("Enter Cargo ID: ");
    scanf("%s", (char *)(*cargoList)[*count].id);

    printf("Enter Cargo Type (Container/Bulk/Refrigerated): ");
    scanf("%s", (*cargoList)[*count].type);

    // Assign cargo properties based on type
    if (strcmp((*cargoList)[*count].type, "Container") == 0) {
```

```c
        printf("Enter Cargo Dimensions (L W H in meters): ");
        scanf("%f %f %f", &(*cargoList)[*count].prop.dimensions.length,
                          &(*cargoList)[*count].prop.dimensions.width,
                          &(*cargoList)[*count].prop.dimensions.height);
    }
    else if (strcmp((*cargoList)[*count].type, "Bulk") == 0) {
        printf("Enter Cargo Weight (tons): ");
        scanf("%f", &(*cargoList)[*count].prop.weight);
    }
    else if (strcmp((*cargoList)[*count].type, "Refrigerated") == 0) {
        printf("Enter Required Temperature (°C): ");
        scanf("%f", &(*cargoList)[*count].prop.temperature);
    }

    (*count)++;
}

// Function to display cargo list
void displayCargo(const struct Cargo *cargoList, int count) {
    printf("\n=== Cargo List ===\n");
    for (int i = 0; i < count; i++) {
        printf("Cargo ID: %s\n", cargoList[i].id);
        printf("Type: %s\n", cargoList[i].type);

        if (strcmp(cargoList[i].type, "Container") == 0) {
            printf("Dimensions: %.2f x %.2f x %.2f meters\n",
cargoList[i].prop.dimensions.length,

cargoList[i].prop.dimensions.width,

cargoList[i].prop.dimensions.height);
        }
        else if (strcmp(cargoList[i].type, "Bulk") == 0) {
            printf("Weight: %.2f tons\n", cargoList[i].prop.weight);
        }
        else if (strcmp(cargoList[i].type, "Refrigerated") == 0) {
            printf("Temperature: %.2f°C\n",
cargoList[i].prop.temperature);
        }
        printf("----------------------------\n");
```

```
    }
}

// Function to delete cargo by ID
void deleteCargo(struct Cargo **cargoList, int *count, const char *id) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*cargoList)[i].id, id) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Cargo ID not found!\n");
        return;
    }

    free((char *)(*cargoList)[index].id); // Free allocated memory for
cargo ID

    for (int i = index; i < *count - 1; i++) {
        (*cargoList)[i] = (*cargoList)[i + 1];
    }

    *cargoList = (struct Cargo *)realloc(*cargoList, (*count - 1) *
sizeof(struct Cargo));
    (*count)--;
    printf("Cargo deleted successfully!\n");
}

// Main function
int main() {
    struct Cargo *cargoList = NULL;
    struct Vessel vessel = {1000, {50, 20, 15}}; // Vessel with 1000-ton
capacity, 50x20x15m dimensions
    int count = 0;
    int choice;
    char id[10];
```

```c
    do {
        printf("\n1. Add Cargo\n");
        printf("2. Display Cargo List\n");
        printf("3. Remove Cargo\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCargo(&cargoList, &count);
                break;
            case 2:
                displayCargo(cargoList, count);
                break;
            case 3:
                printf("Enter Cargo ID to remove: ");
                scanf("%s", id);
                deleteCargo(&cargoList, &count, id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free allocated memory before exiting
    for (int i = 0; i < count; i++) {
        free((char *)cargoList[i].id);
    }
    free(cargoList);

    return 0;
}
```

```
1. Add Cargo
2. Display Cargo List
3. Remove Cargo
4. Exit
Enter your choice: 2

=== Cargo List ===
Cargo ID: 1
Type: Container
Dimensions: 20.00 x 20.00 x 20.00 meters
----------------------------
Cargo ID: 2
Type: Bulk
Weight: 300.00 tons
----------------------------

1. Add Cargo
2. Display Cargo List
3. Remove Cargo
4. Exit
Enter your choice:
```

```c
/*Develop a weather alert system for ships using strings for alert
messages, structures for weather data, and arrays for historical records.
Specifications:
Structure: Weather data (temperature, wind speed).
Array: Historical records.
Strings: Alert messages.
const Pointers: Protect alert details.
Double Pointers: Dynamic weather record management.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure to store weather data
struct WeatherData {
    float temperature;  // Temperature in °C
    float windSpeed;    // Wind speed in knots
    char alert[50];     // Weather alert message
};

// Function to determine weather alert
void setWeatherAlert(struct WeatherData *weather) {
    if (weather->windSpeed > 50) {
```

```
        strcpy(weather->alert, "⚠️ Storm Alert! High wind speed
detected.");
    } else if (weather->temperature < 0) {
        strcpy(weather->alert, "⚠️ Ice Alert! Freezing temperatures.");
    } else {
        strcpy(weather->alert, "✅ Normal Weather Conditions.");
    }
}

// Function to add weather data dynamically
void addWeatherData(struct WeatherData **records, int *count) {
    *records = (struct WeatherData *)realloc(*records, (*count + 1) *
sizeof(struct WeatherData));

    printf("Enter Temperature (°C): ");
    scanf("%f", &((*records)[*count].temperature));

    printf("Enter Wind Speed (knots): ");
    scanf("%f", &((*records)[*count].windSpeed));

    setWeatherAlert(&(*records)[*count]);

    (*count)++;
}

// Function to display weather records
void displayWeatherRecords(const struct WeatherData *records, int count) {
    printf("\n=== Weather Records ===\n");
    for (int i = 0; i < count; i++) {
        printf("Record %d:\n", i + 1);
        printf("Temperature: %.2f°C\n", records[i].temperature);
        printf("Wind Speed: %.2f knots\n", records[i].windSpeed);
        printf("Alert: %s\n", records[i].alert);
        printf("----------------------------\n");
    }
}

// Function to delete a weather record by index
void deleteWeatherData(struct WeatherData **records, int *count, int
index) {
```

```
    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        (*records)[i] = (*records)[i + 1];
    }

    *records = (struct WeatherData *)realloc(*records, (*count - 1) *
sizeof(struct WeatherData));
    (*count)--;
    printf("Weather record deleted successfully!\n");
}

// Main function
int main() {
    struct WeatherData *records = NULL; // Dynamic weather records array
    int count = 0;
    int choice, index;

    do {
        printf("\n1. Add Weather Record\n");
        printf("2. Display Weather Records\n");
        printf("3. Remove Weather Record\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addWeatherData(&records, &count);
                break;
            case 2:
                displayWeatherRecords(records, count);
                break;
            case 3:
                printf("Enter record index to remove (starting from 0):
");
                scanf("%d", &index);
```

```
                deleteWeatherData(&records, &count, index);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    free(records); // Free dynamically allocated memory before exiting

    return 0;
}
```

```
3. Remove Weather Record
4. Exit
Enter your choice: 2

=== Weather Records ===
Record 1:
Temperature: 55.00┬▒C
Wind Speed: 300.00 knots
Alert: ΓÜá╗Å Storm Alert! High wind speed detected.
----------------------------
Record 2:
Temperature: 30.00┬▒C
Wind Speed: 200.00 knots
Alert: ΓÜá╗Å Storm Alert! High wind speed detected.
----------------------------

1. Add Weather Record
```

```
/*Implement a nautical chart management system using arrays for
coordinates and structures for chart metadata.
Use unions for depth or hazard data.
Specifications:
Structure: Chart metadata (ID, scale, region).
Union: Depth or hazard data.
Array: Coordinate points.
const Pointers: Immutable chart IDs.
Double Pointers: Manage dynamic charts.*/
```

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

// Union for depth or hazard data
union DepthOrHazard {
    float depth;          // Depth in meters
    char hazard[50];      // Hazard description
};

// Structure for storing chart metadata
struct NauticalChart {
    const char *id;       // Chart ID (immutable)
    char scale[20];       // Scale of the chart
    char region[50];      // Region of the chart
    float coordinates[10][2]; // Array for coordinate points (latitude,
longitude)
    union DepthOrHazard data; // Depth or hazard information
    int isDepth;          // Flag: 1 = Depth data, 0 = Hazard data
};

// Function to add a new chart dynamically
void addChart(struct NauticalChart **charts, int *count) {
    *charts = (struct NauticalChart *)realloc(*charts, (*count + 1) *
sizeof(struct NauticalChart));

    // Allocate memory for ID (Immutable)
    (*charts)[*count].id = (char *)malloc(10 * sizeof(char));

    printf("Enter Chart ID: ");
    scanf("%s", (char *)(*charts)[*count].id);

    printf("Enter Scale: ");
    scanf("%s", (*charts)[*count].scale);

    printf("Enter Region: ");
    getchar(); // Clear input buffer
    fgets((*charts)[*count].region, 50, stdin);
    (*charts)[*count].region[strcspn((*charts)[*count].region, "\n")] =
'\0'; // Remove newline
```

```c
    // Enter coordinates
    printf("Enter 3 coordinate points (latitude longitude):\n");
    for (int i = 0; i < 3; i++) {
        printf("Point %d: ", i + 1);
        scanf("%f %f", &(*charts)[*count].coordinates[i][0],
&(*charts)[*count].coordinates[i][1]);
    }

    // Choose depth or hazard
    printf("Enter 1 for Depth data, 0 for Hazard: ");
    scanf("%d", &(*charts)[*count].isDepth);

    if ((*charts)[*count].isDepth) {
        printf("Enter Depth (meters): ");
        scanf("%f", &(*charts)[*count].data.depth);
    } else {
        printf("Enter Hazard Description: ");
        getchar(); // Clear input buffer
        fgets((*charts)[*count].data.hazard, 50, stdin);

(*charts)[*count].data.hazard[strcspn((*charts)[*count].data.hazard,
"\n")] = '\0'; // Remove newline
    }

    (*count)++;
}

// Function to display all charts
void displayCharts(const struct NauticalChart *charts, int count) {
    printf("\n=== Nautical Charts ===\n");
    for (int i = 0; i < count; i++) {
        printf("Chart ID: %s\n", charts[i].id);
        printf("Scale: %s\n", charts[i].scale);
        printf("Region: %s\n", charts[i].region);
        printf("Coordinates:\n");
        for (int j = 0; j < 3; j++) {
            printf("  (%.2f, %.2f)\n", charts[i].coordinates[j][0],
charts[i].coordinates[j][1]);
        }
```

```c
        if (charts[i].isDepth) {
            printf("Depth: %.2f meters\n", charts[i].data.depth);
        } else {
            printf("Hazard: %s\n", charts[i].data.hazard);
        }
        printf("----------------------------\n");
    }
}

// Function to delete a chart by ID
void deleteChart(struct NauticalChart **charts, int *count, const char
*id) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*charts)[i].id, id) == 0) {
            index = i;
            free((char *)(*charts)[i].id); // Free allocated ID memory
            break;
        }
    }
    if (index == -1) {
        printf("Chart not found!\n");
        return;
    }
    for (int i = index; i < *count - 1; i++) {
        (*charts)[i] = (*charts)[i + 1];
    }
    *charts = (struct NauticalChart *)realloc(*charts, (*count - 1) *
sizeof(struct NauticalChart));
    (*count)--;
    printf("Chart deleted successfully!\n");
}

// Main function
int main() {
    struct NauticalChart *charts = NULL; // Dynamic array of charts
    int count = 0;
    int choice;
    char id[10];
```

```c
    do {
        printf("\n1. Add Nautical Chart\n");
        printf("2. Display Charts\n");
        printf("3. Remove Chart\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addChart(&charts, &count);
                break;
            case 2:
                displayCharts(charts, count);
                break;
            case 3:
                printf("Enter Chart ID to remove: ");
                scanf("%s", id);
                deleteChart(&charts, &count, id);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free dynamically allocated memory
    for (int i = 0; i < count; i++) {
        free((char *)charts[i].id);
    }
    free(charts);

    return 0;
}
```

```
=== Nautical Charts ===
Chart ID: 1
Scale: 20
Region: A
Coordinates:
  (12.00, 13.00)
  (15.00, 10.00)
  (22.00, 24.00)
Depth: 500.00 meters
---------------------------
Chart ID: 2
Scale: 30
Region: B
Coordinates:
  (11.00, 12.00)
  (23.00, 24.00)
  (55.00, 66.00)
Hazard: d
---------------------------

1. Add Nautical Chart
2. Display Charts
3. Remove Chart
4. Exit
Enter your choice:
```

```c
/*Develop a system to manage ship crew rosters using strings for names,
arrays for schedules, and structures for roles.
Specifications:
Structure: Crew details (name, role, schedule).
Array: Roster.
Strings: Crew names.
const Pointers: Protect role definitions.
Double Pointers: Dynamic roster allocation.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for Crew Member details
struct Crew {
    char name[50];        // Crew member name
    const char *role;     // Role (Immutable)
    char schedule[7][20]; // Weekly schedule
};
```

```c
// Function to add a crew member dynamically
void addCrew(struct Crew **roster, int *count) {
    *roster = (struct Crew *)realloc(*roster, (*count + 1) * sizeof(struct
Crew));

    printf("Enter Crew Member Name: ");
    getchar(); // Clear input buffer
    fgets((*roster)[*count].name, 50, stdin);
    (*roster)[*count].name[strcspn((*roster)[*count].name, "\n")] = '\0';
// Remove newline

    // Role Selection
    printf("Select Role:\n");
    printf("1. Captain\n2. First Mate\n3. Engineer\n4. Deckhand\nEnter
choice: ");
    int choice;
    scanf("%d", &choice);

    static const char *roles[] = {"Captain", "First Mate", "Engineer",
"Deckhand"};
    if (choice >= 1 && choice <= 4) {
        (*roster)[*count].role = roles[choice - 1]; // Assign role (const
pointer)
    } else {
        printf("Invalid role! Defaulting to 'Deckhand'\n");
        (*roster)[*count].role = roles[3];
    }

    // Enter Weekly Schedule
    printf("Enter Weekly Schedule (e.g., '8AM-4PM'):\n");
    for (int i = 0; i < 7; i++) {
        printf("Day %d: ", i + 1);
        scanf("%s", (*roster)[*count].schedule[i]);
    }

    (*count)++;
}

// Function to display the crew roster
```

```c
void displayRoster(const struct Crew *roster, int count) {
    printf("\n=== Ship Crew Roster ===\n");
    for (int i = 0; i < count; i++) {
        printf("Crew Member: %s\n", roster[i].name);
        printf("Role: %s\n", roster[i].role);
        printf("Schedule:\n");
        for (int j = 0; j < 7; j++) {
            printf("  Day %d: %s\n", j + 1, roster[i].schedule[j]);
        }
        printf("----------------------------\n");
    }
}

// Function to remove a crew member by name
void removeCrew(struct Crew **roster, int *count, const char *name) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*roster)[i].name, name) == 0) {
            index = i;
            break;
        }
    }
    if (index == -1) {
        printf("Crew member not found!\n");
        return;
    }
    for (int i = index; i < *count - 1; i++) {
        (*roster)[i] = (*roster)[i + 1];
    }
    *roster = (struct Crew *)realloc(*roster, (*count - 1) * sizeof(struct
Crew));
    (*count)--;
    printf("Crew member removed successfully!\n");
}

// Main function
int main() {
    struct Crew *roster = NULL; // Dynamic array for crew roster
    int count = 0;
    int choice;
```

```c
    char name[50];

    do {
        printf("\n1. Add Crew Member\n");
        printf("2. Display Crew Roster\n");
        printf("3. Remove Crew Member\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCrew(&roster, &count);
                break;
            case 2:
                displayRoster(roster, count);
                break;
            case 3:
                printf("Enter Crew Member Name to remove: ");
                getchar(); // Clear input buffer
                fgets(name, 50, stdin);
                name[strcspn(name, "\n")] = '\0'; // Remove newline
                removeCrew(&roster, &count, name);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free dynamically allocated memory
    free(roster);

    return 0;
}
```

```
1. Add Crew Member
2. Display Crew Roster
3. Remove Crew Member
4. Exit
Enter your choice: 2

=== Ship Crew Roster ===
Crew Member: red
Role: Captain
Schedule:
  Day 1: 8AM-4Pm
  Day 2: 10AM-5PM
  Day 3: 11AM-5PM
  Day 4: 8AM-4PM
  Day 5: 8AM-4PM
  Day 6: 8AM-4PM
  Day 7: 9AM-5PM
----------------------------

1. Add Crew Member
2. Display Crew Roster
3. Remove Crew Member
4. Exit
Enter your choice:
```

```c
/*Create a system for underwater sensor monitoring using arrays for
readings, structures for sensor details, and unions for variable sensor
types.
Specifications:
Structure: Sensor details (ID, location).
Union: Sensor types (temperature, pressure).
Array: Sensor readings.
const Pointers: Protect sensor IDs.
Double Pointers: Dynamic sensor lists.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Union for different sensor types
```

```c
union SensorType {
    float temperature; // Temperature in °C
    float pressure;    // Pressure in Pascals
};

// Structure for sensor details
struct Sensor {
    const char *id;   // Sensor ID (protected)
    char location[50]; // Sensor location
    union SensorType type;
    float readings[10]; // Array of last 10 sensor readings
};

// Function to add a sensor dynamically
void addSensor(struct Sensor **sensors, int *count) {
    *sensors = (struct Sensor *)realloc(*sensors, (*count + 1) *
sizeof(struct Sensor));

    // Allocate memory for ID
    char tempID[20];
    printf("Enter Sensor ID: ");
    scanf("%s", tempID);
    (*sensors)[*count].id = strdup(tempID); // Allocates memory for ID

    printf("Enter Sensor Location: ");
    getchar(); // Clear input buffer
    fgets((*sensors)[*count].location, 50, stdin);
    (*sensors)[*count].location[strcspn((*sensors)[*count].location,
"\n")] = '\0'; // Remove newline

    // Sensor Type Selection
    printf("Select Sensor Type:\n");
    printf("1. Temperature Sensor\n2. Pressure Sensor\nEnter choice: ");
    int choice;
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter Initial Temperature (°C): ");
        scanf("%f", &(*sensors)[*count].type.temperature);
    } else {
```

```
        printf("Enter Initial Pressure (Pa): ");
        scanf("%f", &(*sensors)[*count].type.pressure);
    }

    // Initialize sensor readings with the first value
    for (int i = 0; i < 10; i++) {
        (*sensors)[*count].readings[i] = (choice == 1) ?
(*sensors)[*count].type.temperature : (*sensors)[*count].type.pressure;
    }

    (*count)++;
}

// Function to update sensor readings
void updateSensor(struct Sensor *sensors, int count, const char *id) {
    for (int i = 0; i < count; i++) {
        if (strcmp(sensors[i].id, id) == 0) {
            printf("Updating readings for Sensor ID: %s\n",
sensors[i].id);

            // Shift old readings
            for (int j = 9; j > 0; j--) {
                sensors[i].readings[j] = sensors[i].readings[j - 1];
            }

            // Get new reading
            if (sensors[i].type.temperature) {
                printf("Enter New Temperature (°C): ");
                scanf("%f", &sensors[i].type.temperature);
                sensors[i].readings[0] = sensors[i].type.temperature;
            } else {
                printf("Enter New Pressure (Pa): ");
                scanf("%f", &sensors[i].type.pressure);
                sensors[i].readings[0] = sensors[i].type.pressure;
            }

            printf("Sensor readings updated successfully!\n");
            return;
        }
    }
```

```c
    printf("Sensor ID not found!\n");
}

// Function to display sensor details
void displaySensors(const struct Sensor *sensors, int count) {
    printf("\n=== Underwater Sensor Data ===\n");
    for (int i = 0; i < count; i++) {
        printf("Sensor ID: %s\n", sensors[i].id);
        printf("Location: %s\n", sensors[i].location);

        if (sensors[i].type.temperature) {
            printf("Type: Temperature Sensor\n");
            printf("Latest Reading: %.2f °C\n",
sensors[i].type.temperature);
        } else {
            printf("Type: Pressure Sensor\n");
            printf("Latest Reading: %.2f Pa\n", sensors[i].type.pressure);
        }

        printf("Last 10 Readings: ");
        for (int j = 0; j < 10; j++) {
            printf("%.2f ", sensors[i].readings[j]);
        }
        printf("\n----------------------------------\n");
    }
}

// Function to remove a sensor
void removeSensor(struct Sensor **sensors, int *count, const char *id) {
    int index = -1;
    for (int i = 0; i < *count; i++) {
        if (strcmp((*sensors)[i].id, id) == 0) {
            index = i;
            break;
        }
    }
    if (index == -1) {
        printf("Sensor not found!\n");
        return;
    }
```

```
    free((void *)(*sensors)[index].id); // Free allocated ID memory

    for (int i = index; i < *count - 1; i++) {
        (*sensors)[i] = (*sensors)[i + 1];
    }

    *sensors = (struct Sensor *)realloc(*sensors, (*count - 1) *
sizeof(struct Sensor));
    (*count)--;
    printf("Sensor removed successfully!\n");
}

// Main function
int main() {
    struct Sensor *sensors = NULL; // Dynamic sensor list
    int count = 0;
    int choice;
    char id[20];

    do {
        printf("\n1. Add Sensor\n");
        printf("2. Update Sensor Readings\n");
        printf("3. Display Sensors\n");
        printf("4. Remove Sensor\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addSensor(&sensors, &count);
                break;
            case 2:
                printf("Enter Sensor ID to update: ");
                scanf("%s", id);
                updateSensor(sensors, count, id);
                break;
            case 3:
                displaySensors(sensors, count);
```

```
                break;
            case 4:
                printf("Enter Sensor ID to remove: ");
                scanf("%s", id);
                removeSensor(&sensors, &count, id);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free dynamically allocated memory
    for (int i = 0; i < count; i++) {
        free((void *)sensors[i].id);
    }
    free(sensors);

    return 0;
}
```

```
3. Display Sensors
4. Remove Sensor
5. Exit
Enter your choice: 3

=== Underwater Sensor Data ===
Sensor ID: 1
Location: A
Type: Temperature Sensor
Latest Reading: 200.00 °C
Last 10 Readings: 200.00 200.00 200.00 200.00 200.
----------------------------------
Sensor ID: 2
Location: B
Type: Temperature Sensor
Latest Reading: 400.00 °C
Last 10 Readings: 400.00 400.00 400.00 400.00 400.
----------------------------------

1. Add Sensor
2. Update Sensor Readings
3. Display Sensors
4. Remove Sensor
5. Exit
Enter your choice:
```

```c
/*Design a ship log system using strings for log entries, arrays for daily
records, and structures for log metadata.
Specifications:
Structure: Log metadata (date, author).
Array: Daily log records.
Strings: Log entries.
const Pointers: Immutable metadata.
Double Pointers: Manage dynamic log entries.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for log metadata
```

```c
struct LogMetadata {
    const char *date;  // Log entry date (protected with const pointer)
    const char *author; // Log entry author (protected with const pointer)
};

// Structure for daily log entry
struct LogEntry {
    struct LogMetadata metadata;  // Log metadata (date, author)
    char entry[200];              // Log entry content (string)
};

// Function to add a new log entry
void addLog(struct LogEntry **logs, int *count) {
    *logs = (struct LogEntry *)realloc(*logs, (*count + 1) * sizeof(struct
LogEntry));

    // Assign metadata (date, author)
    char tempDate[20], tempAuthor[50];
    printf("Enter the date of the log (YYYY-MM-DD): ");
    scanf("%s", tempDate);
    (*logs)[*count].metadata.date = strdup(tempDate);  // Protect the date
using const pointer

    printf("Enter the author of the log: ");
    getchar(); // Clear input buffer
    fgets(tempAuthor, sizeof(tempAuthor), stdin);
    tempAuthor[strcspn(tempAuthor, "\n")] = 0; // Remove trailing newline
    (*logs)[*count].metadata.author = strdup(tempAuthor);  // Protect the
author using const pointer

    // Log entry content
    printf("Enter the log entry content: ");
    fgets((*logs)[*count].entry, sizeof((*logs)[*count].entry), stdin);
    (*count)++;
}

// Function to display the logs
void displayLogs(struct LogEntry *logs, int count) {
    printf("\n=== Ship Log Records ===\n");
    for (int i = 0; i < count; i++) {
```

```
        printf("\nLog #%d\n", i + 1);
        printf("Date: %s\n", logs[i].metadata.date);
        printf("Author: %s\n", logs[i].metadata.author);
        printf("Log Entry: %s\n", logs[i].entry);
        printf("-------------------------\n");
    }
}

// Function to delete a log entry by index
void deleteLog(struct LogEntry **logs, int *count, int index) {
    if (index < 0 || index >= *count) {
        printf("Invalid log entry index!\n");
        return;
    }

    // Free allocated memory for metadata
    free((void *)(*logs)[index].metadata.date);
    free((void *)(*logs)[index].metadata.author);

    // Shift the entries
    for (int i = index; i < *count - 1; i++) {
        (*logs)[i] = (*logs)[i + 1];
    }

    *logs = (struct LogEntry *)realloc(*logs, (*count - 1) * sizeof(struct
LogEntry));
    (*count)--;
    printf("Log entry deleted successfully!\n");
}

// Main function
int main() {
    struct LogEntry *logs = NULL; // Dynamic array of logs
    int count = 0;  // Counter for the number of logs
    int choice;
    int logIndex;

    do {
        printf("\n1. Add Log Entry\n");
        printf("2. Display Log Entries\n");
```

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

        switch (choice) {
            case 1:
                addLog(&logs, &count);
                break;
            case 2:
                displayLogs(logs, count);
                break;
            case 3:
                printf("Enter the log entry number to delete: ");
                scanf("%d", &logIndex);
                deleteLog(&logs, &count, logIndex - 1);  // Log entries
are 1-indexed for user convenience
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free allocated memory for all logs
    for (int i = 0; i < count; i++) {
        free((void *)logs[i].metadata.date);
        free((void *)logs[i].metadata.author);
    }
    free(logs);

    return 0;
}
```

```
Enter your choice: 2

=== Ship Log Records ===

Log #1
Date: 2024-12-29
Author: red
Log Entry: A

-------------------------

Log #2
Date: 2025
Author: 2
Log Entry: B

-------------------------

1. Add Log Entry
2. Display Log Entries
3. Delete Log Entry
4. Exit
Enter your choice:
```

```
/*Develop a waypoint management tool using arrays for storing waypoints,
strings for waypoint names, and structures for navigation details.
Specifications:
Structure: Navigation details (ID, waypoints).
Array: Waypoint data.
Strings: Names of waypoints.
const Pointers: Protect waypoint IDs.
Double Pointers: Dynamic waypoint storage.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for Navigation details: stores ID and corresponding waypoints
struct NavigationDetails {
```

```c
    const char *ID;       // Waypoint ID (protected by const pointer)
    char name[50];        // Waypoint name
};

// Function to add a new waypoint
void addWaypoint(struct NavigationDetails **waypoints, int *count) {
    *waypoints = (struct NavigationDetails *)realloc(*waypoints, (*count +
1) * sizeof(struct NavigationDetails));

    // Assign waypoint ID (protected by const pointer)
    char tempID[10], tempName[50];
    printf("Enter Waypoint ID: ");
    scanf("%s", tempID);
    (*waypoints)[*count].ID = strdup(tempID);  // Using strdup to protect
the ID

    // Waypoint Name
    printf("Enter Waypoint Name: ");
    getchar(); // Clear input buffer
    fgets(tempName, sizeof(tempName), stdin);
    tempName[strcspn(tempName, "\n")] = 0; // Remove trailing newline
    strcpy((*waypoints)[*count].name, tempName);

    (*count)++;
}

// Function to display all waypoints
void displayWaypoints(struct NavigationDetails *waypoints, int count) {
    printf("\n=== Waypoint List ===\n");
    for (int i = 0; i < count; i++) {
        printf("Waypoint ID: %s\n", waypoints[i].ID);
        printf("Waypoint Name: %s\n", waypoints[i].name);
        printf("-------------------------\n");
    }
}

// Function to delete a waypoint by ID
void deleteWaypoint(struct NavigationDetails **waypoints, int *count,
const char *ID) {
    int index = -1;
```

```
    // Find the index of the waypoint with the given ID
    for (int i = 0; i < *count; i++) {
        if (strcmp((*waypoints)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Waypoint with ID %s not found!\n", ID);
        return;
    }

    // Free memory for the waypoint ID
    free((void *)(*waypoints)[index].ID);

    // Shift the waypoints to delete the selected one
    for (int i = index; i < *count - 1; i++) {
        (*waypoints)[i] = (*waypoints)[i + 1];
    }

    // Resize the array
    *waypoints = (struct NavigationDetails *)realloc(*waypoints, (*count -
1) * sizeof(struct NavigationDetails));
    (*count)--;

    printf("Waypoint with ID %s has been deleted.\n", ID);
}

// Main function
int main() {
    struct NavigationDetails *waypoints = NULL;  // Dynamic array for
waypoints
    int count = 0;  // Counter for the number of waypoints
    int choice;
    char waypointID[10];

    do {
        printf("\n1. Add Waypoint\n");
```

```c
        printf("2. Display Waypoints\n");
        printf("3. Delete Waypoint\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addWaypoint(&waypoints, &count);
                break;
            case 2:
                displayWaypoints(waypoints, count);
                break;
            case 3:
                printf("Enter Waypoint ID to delete: ");
                scanf("%s", waypointID);
                deleteWaypoint(&waypoints, &count, waypointID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free allocated memory for waypoints
    for (int i = 0; i < count; i++) {
        free((void *)waypoints[i].ID);
    }
    free(waypoints);

    return 0;
}
```

```
4. Exit
Enter your choice: 1
Enter Waypoint ID: 2
Enter Waypoint Name: B

1. Add Waypoint
2. Display Waypoints
3. Delete Waypoint
4. Exit
Enter your choice: 2

=== Waypoint List ===
Waypoint ID: 1
Waypoint Name: A
-------------------------
Waypoint ID: 2
Waypoint Name: B
-------------------------

1. Add Waypoint
2. Display Waypoints
3. Delete Waypoint
4. Exit
Enter your choice:
```

```
/*Create a system for tracking marine wildlife using structures for animal
data and arrays for observation records.
Specifications:
Structure: Animal data (species, ID, location).
Array: Observation records.
Strings: Species names.
const Pointers: Protect species IDs.
Double Pointers: Manage dynamic tracking data.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
// Structure to store Animal data (species, ID, location)
struct AnimalData {
    const char *species;     // Species (protected by const pointer)
    char ID[20];             // Animal ID
    char location[50];       // Location where observed
};

// Function to add a new animal observation
void addObservation(struct AnimalData **animals, int *count) {
    *animals = (struct AnimalData *)realloc(*animals, (*count + 1) *
sizeof(struct AnimalData));

    // Species name (protected by const pointer)
    char tempSpecies[50], tempLocation[50];
    printf("Enter Animal Species: ");
    getchar();  // Clear input buffer before reading the species
    fgets(tempSpecies, sizeof(tempSpecies), stdin);
    tempSpecies[strcspn(tempSpecies, "\n")] = 0;  // Remove trailing
newline
    (*animals)[*count].species = strdup(tempSpecies);  // Protect species
name by using strdup

    // Animal ID
    printf("Enter Animal ID: ");
    scanf("%s", (*animals)[*count].ID);

    // Animal location
    printf("Enter Observation Location: ");
    getchar();  // Clear input buffer before reading the location
    fgets(tempLocation, sizeof(tempLocation), stdin);
    tempLocation[strcspn(tempLocation, "\n")] = 0;  // Remove trailing
newline
    strcpy((*animals)[*count].location, tempLocation);

    (*count)++;
}

// Function to display all animal observations
void displayObservations(struct AnimalData *animals, int count) {
```

```c
    printf("\n=== Animal Observation Records ===\n");
    for (int i = 0; i < count; i++) {
        printf("Species: %s\n", animals[i].species);
        printf("Animal ID: %s\n", animals[i].ID);
        printf("Location: %s\n", animals[i].location);
        printf("-------------------------\n");
    }
}

// Function to delete an animal observation by ID
void deleteObservation(struct AnimalData **animals, int *count, const char
*ID) {
    int index = -1;

    // Find the index of the observation with the given ID
    for (int i = 0; i < *count; i++) {
        if (strcmp((*animals)[i].ID, ID) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Observation with ID %s not found!\n", ID);
        return;
    }

    // Free memory for the species name (protected by const pointer)
    free((void *)(*animals)[index].species);

    // Shift the observations to delete the selected one
    for (int i = index; i < *count - 1; i++) {
        (*animals)[i] = (*animals)[i + 1];
    }

    // Resize the array
    *animals = (struct AnimalData *)realloc(*animals, (*count - 1) *
sizeof(struct AnimalData));
    (*count)--;
```

```
    printf("Observation with ID %s has been deleted.\n", ID);
}

// Main function
int main() {
    struct AnimalData *animals = NULL;  // Dynamic array for storing
animal data
    int count = 0;  // Counter for the number of observations
    int choice;
    char animalID[20];

    do {
        printf("\n1. Add Animal Observation\n");
        printf("2. Display Observations\n");
        printf("3. Delete Observation\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addObservation(&animals, &count);
                break;
            case 2:
                displayObservations(animals, count);
                break;
            case 3:
                printf("Enter Animal ID to delete: ");
                scanf("%s", animalID);
                deleteObservation(&animals, &count, animalID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    // Free allocated memory for species names and animal data
```

```
    for (int i = 0; i < count; i++) {
        free((void *)animals[i].species);
    }
    free(animals);

    return 0;
}
```

```
Enter Animal Species: B
Enter Animal ID: 2
Enter Observation Location: B1

1. Add Animal Observation
2. Display Observations
3. Delete Observation
4. Exit
Enter your choice: 2

=== Animal Observation Records ===
Species: A
Animal ID: 1
Location: A1
-------------------------
Species: B
Animal ID: 2
Location: B1
-------------------------

1. Add Animal Observation
2. Display Observations
3. Delete Observation
4. Exit
Enter your choice:
```

```
/*Design a system to manage coastal navigation beacons using structures
for beacon metadata, arrays for signals, and unions for variable beacon
types.
Specifications:
Structure: Beacon metadata (ID, type, location).
```

```c
Union: Variable beacon types.
Array: Signal data.
const Pointers: Immutable beacon IDs.
Double Pointers: Dynamic beacon data management.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
union BeaconType {
    char lightColor[20];
    char soundFrequency[20];
};

struct Beacon {
    const char *id;
    char type[20];                // Type of beacon (e.g., "light",
"sound")
    char location[50];
    union BeaconType beaconType;
};


void addBeacon(struct Beacon **beacons, int *count) {
    *beacons = (struct Beacon *)realloc(*beacons, (*count + 1) *
sizeof(struct Beacon));


    char tempID[20], tempType[20], tempLocation[50], tempBeaconType[20];
    printf("Enter Beacon ID: ");
    scanf("%s", tempID);
    (*beacons)[*count].id = strdup(tempID);


    printf("Enter Beacon Type (light/sound): ");
    scanf("%s", tempType);
    strcpy((*beacons)[*count].type, tempType);


    printf("Enter Beacon Location: ");
    getchar();
```

```c
    fgets(tempLocation, sizeof(tempLocation), stdin);
    tempLocation[strcspn(tempLocation, "\n")] = 0;
    strcpy((*beacons)[*count].location, tempLocation);

    if (strcmp((*beacons)[*count].type, "light") == 0) {
        printf("Enter Light Color: ");
        scanf("%s", tempBeaconType);
        strcpy((*beacons)[*count].beaconType.lightColor, tempBeaconType);
    } else if (strcmp((*beacons)[*count].type, "sound") == 0) {
        printf("Enter Sound Frequency: ");
        scanf("%s", tempBeaconType);
        strcpy((*beacons)[*count].beaconType.soundFrequency,
tempBeaconType);
    } else {
        printf("Invalid Beacon Type!\n");
    }

    (*count)++;
}

void displayBeacons(struct Beacon *beacons, int count) {
    printf("\n=== Beacon Information ===\n");
    for (int i = 0; i < count; i++) {
        printf("Beacon ID: %s\n", beacons[i].id);
        printf("Type: %s\n", beacons[i].type);
        printf("Location: %s\n", beacons[i].location);
        if (strcmp(beacons[i].type, "light") == 0) {
            printf("Light Color: %s\n", beacons[i].beaconType.lightColor);
        } else if (strcmp(beacons[i].type, "sound") == 0) {
            printf("Sound Frequency: %s\n",
beacons[i].beaconType.soundFrequency);
        }
        printf("--------------------------\n");
    }
}

void deleteBeacon(struct Beacon **beacons, int *count, const char *id) {
    int index = -1;
```

```
    for (int i = 0; i < *count; i++) {
        if (strcmp((*beacons)[i].id, id) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Beacon with ID %s not found!\n", id);
        return;
    }


    free((void *)(*beacons)[index].id);


    for (int i = index; i < *count - 1; i++) {
        (*beacons)[i] = (*beacons)[i + 1];
    }


    *beacons = (struct Beacon *)realloc(*beacons, (*count - 1) *
sizeof(struct Beacon));
    (*count)--;

    printf("Beacon with ID %s has been deleted.\n", id);
}


int main() {
    struct Beacon *beacons = NULL;
    int count = 0;
    int choice;
    char beaconID[20];

    do {
        printf("\n1. Add Beacon\n");
        printf("2. Display Beacons\n");
        printf("3. Delete Beacon\n");
        printf("4. Exit\n");
```

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

        switch (choice) {
            case 1:
                addBeacon(&beacons, &count);
                break;
            case 2:
                displayBeacons(beacons, count);
                break;
            case 3:
                printf("Enter Beacon ID to delete: ");
                scanf("%s", beaconID);
                deleteBeacon(&beacons, &count, beaconID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    for (int i = 0; i < count; i++) {
        free((void *)beacons[i].id);
    }
    free(beacons);

    return 0;
}
```

```
4. Exit
Enter your choice: 2

=== Beacon Information ===
Beacon ID: 1
Type: light
Location: A
Light Color: red
--------------------------
Beacon ID: 2
Type: sound
Location: B
Sound Frequency: 22
--------------------------
```

```c
/*Develop a fuel usage tracking system for ships using structures for fuel
data and arrays for consumption logs.
Specifications:
Structure: Fuel data (type, quantity).
Array: Consumption logs.
Strings: Fuel types.
const Pointers: Protect fuel data.
Double Pointers: Dynamic fuel log allocation.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct FuelData {
    char type[20];  // Fuel type (e.g., "diesel", "gas")
    float quantity; // Fuel quantity used in liters
};

void addFuelLog(struct FuelData **logs, int *count) {
    *logs = (struct FuelData *)realloc(*logs, (*count + 1) * sizeof(struct
FuelData));

    char tempType[20];
    float tempQuantity;

    printf("Enter fuel type (e.g., diesel, gas): ");
```

```c
    scanf("%s", tempType);
    strcpy((*logs)[*count].type, tempType);

    printf("Enter fuel quantity used (in liters): ");
    scanf("%f", &tempQuantity);
    (*logs)[*count].quantity = tempQuantity;

    (*count)++;
}


void displayFuelLogs(struct FuelData *logs, int count) {
    printf("\n=== Fuel Usage Logs ===\n");
    for (int i = 0; i < count; i++) {
        printf("Fuel Type: %s | Quantity Used: %.2f liters\n",
logs[i].type, logs[i].quantity);
        printf("------------------------\n");
    }
}


void deleteFuelLog(struct FuelData **logs, int *count, const char *type) {
    int index = -1;


    for (int i = 0; i < *count; i++) {
        if (strcmp((*logs)[i].type, type) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Fuel log with type %s not found!\n", type);
        return;
    }


    for (int i = index; i < *count - 1; i++) {
        (*logs)[i] = (*logs)[i + 1];
```

```
    }


    *logs = (struct FuelData *)realloc(*logs, (*count - 1) * sizeof(struct
FuelData));
    (*count)--;

    printf("Fuel log with type %s has been deleted.\n", type);
}


int main() {
    struct FuelData *logs = NULL;
    int count = 0;
    int choice;
    char fuelType[20];

    do {
        printf("\n1. Add Fuel Log\n");
        printf("2. Display Fuel Logs\n");
        printf("3. Delete Fuel Log\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addFuelLog(&logs, &count);
                break;
            case 2:
                displayFuelLogs(logs, count);
                break;
            case 3:
                printf("Enter fuel type to delete (e.g., diesel): ");
                scanf("%s", fuelType);
                deleteFuelLog(&logs, &count, fuelType);
                break;
            case 4:
                printf("Exiting...\n");
                break;
```

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

    free(logs);

    return 0;
}
```

```
1. Add Fuel Log
2. Display Fuel Logs
3. Delete Fuel Log
4. Exit
Enter your choice: 1
Enter fuel type (e.g., diesel, gas): gas
Enter fuel quantity used (in liters): 40

1. Add Fuel Log
2. Display Fuel Logs
3. Delete Fuel Log
4. Exit
Enter your choice: 2

=== Fuel Usage Logs ===
Fuel Type: diesel | Quantity Used: 100.00 liters
------------------------
Fuel Type: gas | Quantity Used: 40.00 liters
------------------------

1. Add Fuel Log
2. Display Fuel Logs
3. Delete Fuel Log
4. Exit
Enter your choice:
```

```
/*Create an emergency response system using strings for messages,
structures for response details, and arrays for alert history.
Specifications:
Structure: Response details (ID, location, type).
Array: Alert history.
Strings: Alert messages.
const Pointers: Protect emergency IDs.
```

```c
Double Pointers: Dynamic alert allocation.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct ResponseDetails {
    char *id;
    char location[50];
    char type[30];     // Type of the emergency (e.g., fire, medical)
};

void addResponse(struct ResponseDetails **alerts, int *alertCount) {
    *alerts = (struct ResponseDetails *)realloc(*alerts, (*alertCount + 1)
* sizeof(struct ResponseDetails));

    char tempID[20];
    char tempLocation[50];
    char tempType[30];


    printf("Enter Emergency ID: ");
    scanf("%s", tempID);
    (*alerts)[*alertCount].id = (char *)malloc((strlen(tempID) + 1) *
sizeof(char));
    strcpy((*alerts)[*alertCount].id, tempID);

    printf("Enter Emergency Location: ");
    scanf(" %[^\n]", tempLocation);
    strcpy((*alerts)[*alertCount].location, tempLocation);

    printf("Enter Emergency Type (e.g., fire, medical): ");
    scanf("%s", tempType);
    strcpy((*alerts)[*alertCount].type, tempType);

    (*alertCount)++;
}


void displayAlerts(struct ResponseDetails *alerts, int alertCount) {
    printf("\n=== Emergency Response Alerts ===\n");
```

```
    for (int i = 0; i < alertCount; i++) {
        printf("Emergency ID: %s\n", alerts[i].id);
        printf("Location: %s\n", alerts[i].location);
        printf("Emergency Type: %s\n", alerts[i].type);
        printf("----------------------------\n");
    }
}


void deleteAlert(struct ResponseDetails **alerts, int *alertCount, const
char *id) {
    int index = -1;

    for (int i = 0; i < *alertCount; i++) {
        if (strcmp((*alerts)[i].id, id) == 0) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        printf("Emergency alert with ID %s not found!\n", id);
        return;
    }

    free((*alerts)[index].id);

    for (int i = index; i < *alertCount - 1; i++) {
        (*alerts)[i] = (*alerts)[i + 1];
    }


    *alerts = (struct ResponseDetails *)realloc(*alerts, (*alertCount - 1)
* sizeof(struct ResponseDetails));
    (*alertCount)--;

    printf("Emergency alert with ID %s has been deleted.\n", id);
}
```

```c
int main() {
    struct ResponseDetails *alerts = NULL;
    int alertCount = 0;
    int choice;
    char alertID[20];

    do {
        printf("\n1. Add Emergency Response\n");
        printf("2. Display Emergency Alerts\n");
        printf("3. Delete Emergency Response\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addResponse(&alerts, &alertCount);
                break;
            case 2:
                displayAlerts(alerts, alertCount);
                break;
            case 3:
                printf("Enter Emergency ID to delete: ");
                scanf("%s", alertID);
                deleteAlert(&alerts, &alertCount, alertID);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);


    for (int i = 0; i < alertCount; i++) {
        free(alerts[i].id);
    }
    free(alerts);
```

```
    return 0;
}
```

```
Enter Emergency ID: 2
Enter Emergency Location: B
Enter Emergency Type (e.g., fire, medical): medical

1. Add Emergency Response
2. Display Emergency Alerts
3. Delete Emergency Response
4. Exit
Enter your choice: 2

=== Emergency Response Alerts ===
Emergency ID: 1
Location: A
Emergency Type: fire
----------------------------
Emergency ID: 2
Location: B
Emergency Type: medical
----------------------------

1. Add Emergency Response
2. Display Emergency Alerts
3. Delete Emergency Response
4. Exit
Enter your choice:
```

```
/*Design a system for ship performance analysis using arrays for
performance metrics, structures for ship specifications,
and unions for variable factors like weather impact.
Specifications:
Structure: Ship specifications (speed, capacity).
Union: Variable factors.
Array: Performance metrics.
const Pointers: Protect metric definitions.
Double Pointers: Dynamic performance records.*/
#include <stdio.h>
```

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

union VariableFactors {
    float weatherImpact;
    float seaCondition;
};


struct ShipSpecifications {
    char id[20];
    float speed;
    int capacity;
    union VariableFactors factors; // Variable factors (weather or sea
conditions)
};


struct PerformanceMetrics {
    float fuelConsumed;
    float actualSpeed;
    float distanceTraveled;
};


void inputShipData(struct ShipSpecifications **ships, int *shipCount) {

    *ships = (struct ShipSpecifications *)realloc(*ships, (*shipCount + 1)
* sizeof(struct ShipSpecifications));


    printf("Enter Ship ID: ");
    scanf("%s", (*ships)[*shipCount].id);

    printf("Enter Ship Speed (knots): ");
    scanf("%f", &(*ships)[*shipCount].speed);

    printf("Enter Ship Capacity (tons): ");
    scanf("%d", &(*ships)[*shipCount].capacity);
```

```
    char factorType;
    printf("Enter Variable Factor Type (w: weather impact, s: sea
condition): ");
    scanf(" %c", &factorType);

    if (factorType == 'w') {
        printf("Enter Weather Impact (as a percentage): ");
        scanf("%f", &(*ships)[*shipCount].factors.weatherImpact);
    } else if (factorType == 's') {
        printf("Enter Sea Condition Impact (as a percentage): ");
        scanf("%f", &(*ships)[*shipCount].factors.seaCondition);
    }

    (*shipCount)++;
}


void displayShipPerformance(struct ShipSpecifications *ships, int
shipCount) {
    printf("\n=== Ship Performance Records ===\n");
    for (int i = 0; i < shipCount; i++) {
        printf("Ship ID: %s\n", ships[i].id);
        printf("Speed: %.2f knots\n", ships[i].speed);
        printf("Capacity: %d tons\n", ships[i].capacity);


        if (ships[i].factors.weatherImpact != 0) {
            printf("Weather Impact: %.2f%% reduction in speed\n",
ships[i].factors.weatherImpact);
        } else if (ships[i].factors.seaCondition != 0) {
            printf("Sea Condition Impact: %.2f%% reduction in fuel
efficiency\n", ships[i].factors.seaCondition);
        }
        printf("----------------------------------\n");
    }
}
```

```c
void inputPerformanceMetrics(struct PerformanceMetrics **metrics, int
*metricCount) {
    *metrics = (struct PerformanceMetrics *)realloc(*metrics,
(*metricCount + 1) * sizeof(struct PerformanceMetrics));

    printf("Enter Fuel Consumed (in gallons): ");
    scanf("%f", &(*metrics)[*metricCount].fuelConsumed);

    printf("Enter Actual Speed (in knots): ");
    scanf("%f", &(*metrics)[*metricCount].actualSpeed);

    printf("Enter Distance Traveled (in nautical miles): ");
    scanf("%f", &(*metrics)[*metricCount].distanceTraveled);

    (*metricCount)++;
}


void displayPerformanceMetrics(struct PerformanceMetrics *metrics, int
metricCount) {
    printf("\n=== Ship Performance Metrics ===\n");
    for (int i = 0; i < metricCount; i++) {
        printf("Fuel Consumed: %.2f gallons\n", metrics[i].fuelConsumed);
        printf("Actual Speed: %.2f knots\n", metrics[i].actualSpeed);
        printf("Distance Traveled: %.2f nautical miles\n",
metrics[i].distanceTraveled);
        printf("----------------------------------\n");
    }
}


int main() {
    struct ShipSpecifications *ships = NULL;
    struct PerformanceMetrics *metrics = NULL;
    int shipCount = 0, metricCount = 0;
    int choice;

    do {
        printf("\n1. Add Ship Data\n");
        printf("2. Display Ship Specifications\n");
```

```c
        printf("3. Add Performance Metrics\n");
        printf("4. Display Performance Metrics\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputShipData(&ships, &shipCount);
                break;
            case 2:
                displayShipPerformance(ships, shipCount);
                break;
            case 3:
                inputPerformanceMetrics(&metrics, &metricCount);
                break;
            case 4:
                displayPerformanceMetrics(metrics, metricCount);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);
    for (int i = 0; i < shipCount; i++) {
        free(ships[i].id);
    }
    free(ships);
    free(metrics);

    return 0;
}
```

```
1. Add Ship Data
2. Display Ship Specifications
3. Add Performance Metrics
4. Display Performance Metrics
5. Exit
Enter your choice: 2

=== Ship Performance Records ===
Ship ID: 1
Speed: 100.00 knots
Capacity: 200 tons
Weather Impact: 54.00% reduction in speed
----------------------------------
Ship ID: 2
Speed: 300.00 knots
Capacity: 100 tons
Weather Impact: 20.00% reduction in speed
----------------------------------

1. Add Ship Data
2. Display Ship Specifications
3. Add Performance Metrics
4. Display Performance Metrics
5. Exit
Enter your choice:
```

```c
/*Develop a scheduler for port docking using arrays for schedules,
structures for port details, and strings for vessel names.
Specifications:
Structure: Port details (ID, capacity, location).
Array: Docking schedules.
Strings: Vessel names.
const Pointers: Protect schedule IDs.
Double Pointers: Manage dynamic schedules.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
struct PortDetails {
    char id[10];
    int capacity;
    char location[50];
};


struct DockingSchedule {
    char vesselName[50];
    char dockingTime[20];
    char portID[10];
};


void inputPortDetails(struct PortDetails **ports, int *portCount) {

    *ports = (struct PortDetails *)realloc(*ports, (*portCount + 1) *
sizeof(struct PortDetails));

    printf("Enter Port ID: ");
    scanf("%s", (*ports)[*portCount].id);

    printf("Enter Port Capacity: ");
    scanf("%d", &(*ports)[*portCount].capacity);

    printf("Enter Port Location: ");
    scanf(" %[^\n]", (*ports)[*portCount].location);

    (*portCount)++;
}


void inputDockingSchedule(struct DockingSchedule **schedules, int
*scheduleCount) {

    *schedules = (struct DockingSchedule *)realloc(*schedules,
(*scheduleCount + 1) * sizeof(struct DockingSchedule));

    printf("Enter Vessel Name: ");
```

```c
    scanf(" %[^\n]", (*schedules)[*scheduleCount].vesselName);

    printf("Enter Docking Time (e.g., 14:30): ");
    scanf("%s", (*schedules)[*scheduleCount].dockingTime);

    printf("Enter Port ID where the vessel docks: ");
    scanf("%s", (*schedules)[*scheduleCount].portID);

    (*scheduleCount)++;
}

void displayPortDetails(struct PortDetails *ports, int portCount) {
    printf("\n=== Port Details ===\n");
    for (int i = 0; i < portCount; i++) {
        printf("Port ID: %s\n", ports[i].id);
        printf("Capacity: %d\n", ports[i].capacity);
        printf("Location: %s\n", ports[i].location);
        printf("-----------------------------------\n");
    }
}


void displayDockingSchedules(struct DockingSchedule *schedules, int
scheduleCount) {
    printf("\n=== Docking Schedules ===\n");
    for (int i = 0; i < scheduleCount; i++) {
        printf("Vessel Name: %s\n", schedules[i].vesselName);
        printf("Docking Time: %s\n", schedules[i].dockingTime);
        printf("Port ID: %s\n", schedules[i].portID);
        printf("-----------------------------------\n");
    }
}

int main() {
    struct PortDetails *ports = NULL;
    struct DockingSchedule *schedules = NULL;
    int portCount = 0, scheduleCount = 0;
    int choice;

    do {
```

```c
        printf("\n1. Add Port Details\n");
        printf("2. Add Docking Schedule\n");
        printf("3. Display Port Details\n");
        printf("4. Display Docking Schedules\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputPortDetails(&ports, &portCount);
                break;
            case 2:
                inputDockingSchedule(&schedules, &scheduleCount);
                break;
            case 3:
                displayPortDetails(ports, portCount);
                break;
            case 4:
                displayDockingSchedules(schedules, scheduleCount);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    free(ports);
    free(schedules);

    return 0;
}
```

```
Enter Port ID where the vessel docks: 1

1. Add Port Details
2. Add Docking Schedule
3. Display Port Details
4. Display Docking Schedules
5. Exit
Enter your choice: 3

=== Port Details ===
Port ID: 1
Capacity: 20
Location: a
----------------------------------
Port ID: 2
Capacity: 30
Location: B
----------------------------------

1. Add Port Details
2. Add Docking Schedule
3. Display Port Details
4. Display Docking Schedules
5. Exit
Enter your choice:
```

```
/*Create a data logger for deep-sea exploration using structures for
exploration data and arrays for logs.
Specifications:
Structure: Exploration data (depth, location, timestamp).
Array: Logs.
const Pointers: Protect data entries.
Double Pointers: Dynamic log storage.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

struct ExplorationData {
```

```c
    float depth;
    char location[100];
    char timestamp[20];
};


void generateTimestamp(char *timestamp) {
    time_t t = time(NULL);
    struct tm *tm_info = localtime(&t);
    strftime(timestamp, 20, "%Y-%m-%d %H:%M:%S", tm_info);
}


void inputExplorationData(struct ExplorationData **logs, int *logCount) {

    *logs = (struct ExplorationData *)realloc(*logs, (*logCount + 1) *
sizeof(struct ExplorationData));


    printf("Enter exploration depth (in meters): ");
    scanf("%f", &(*logs)[*logCount].depth);


    printf("Enter exploration location: ");
    scanf(" %[^\n]", (*logs)[*logCount].location);


    generateTimestamp((*logs)[*logCount].timestamp);


    (*logCount)++;
}


void displayExplorationLogs(struct ExplorationData *logs, int logCount) {
    printf("\n=== Exploration Logs ===\n");
    for (int i = 0; i < logCount; i++) {
        printf("Log %d\n", i + 1);
        printf("Depth: %.2f meters\n", logs[i].depth);
        printf("Location: %s\n", logs[i].location);
```

```c
        printf("Timestamp: %s\n", logs[i].timestamp);
        printf("------------------------\n");
    }
}


int main() {
    struct ExplorationData *logs = NULL;
    int logCount = 0;
    int choice;

    do {
        printf("\n1. Add Exploration Data\n");
        printf("2. Display Exploration Logs\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputExplorationData(&logs, &logCount);
                break;
            case 2:
                displayExplorationLogs(logs, logCount);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 3);


    free(logs);

    return 0;
}
```

```
2. Display Exploration Logs
3. Exit
Enter your choice: 2

=== Exploration Logs ===
Log 1
Depth: 100.00 meters
Location: A
Timestamp: 2025-01-22 20:03:21
------------------------
Log 2
Depth: 200.00 meters
Location: B
Timestamp: 2025-01-22 20:03:28
------------------------

1. Add Exploration Data
2. Display Exploration Logs
3. Exit
Enter your choice:
```

```c
/*Develop a ship communication system using strings for messages,
structures for communication metadata, and arrays for message logs.
Specifications:
Structure: Communication metadata (ID, timestamp).
Array: Message logs.
Strings: Communication messages.
const Pointers: Protect communication IDs.
Double Pointers: Dynamic message storage.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

struct CommunicationMetadata {
    const char *id;
    char timestamp[20];
};
```

```c
struct Communication {
    struct CommunicationMetadata metadata;
    char message[256];
};


void generateTimestamp(char *timestamp) {
    time_t t = time(NULL);
    struct tm *tm_info = localtime(&t);
    strftime(timestamp, 20, "%Y-%m-%d %H:%M:%S", tm_info);
}

void inputCommunicationData(struct Communication **logs, int *logCount) {

    *logs = (struct Communication *)realloc(*logs, (*logCount + 1) *
sizeof(struct Communication));


    (*logs)[*logCount].metadata.id = (char *)malloc(10 * sizeof(char));
    printf("Enter communication ID (max 9 characters): ");
    scanf("%s", (*logs)[*logCount].metadata.id);


    generateTimestamp((*logs)[*logCount].metadata.timestamp);


    printf("Enter communication message: ");
    getchar();
    fgets((*logs)[*logCount].message, 256, stdin);


    (*logCount)++;
}


void displayCommunicationLogs(struct Communication *logs, int logCount) {
    printf("\n=== Communication Logs ===\n");
    for (int i = 0; i < logCount; i++) {
```

```c
        printf("Log %d\n", i + 1);
        printf("ID: %s\n", logs[i].metadata.id);
        printf("Timestamp: %s\n", logs[i].metadata.timestamp);
        printf("Message: %s\n", logs[i].message);
        printf("------------------------\n");
    }
}


int main() {
    struct Communication *logs = NULL;
    int logCount = 0;
    int choice;

    do {
        printf("\n1. Add Communication\n");
        printf("2. Display Communication Logs\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputCommunicationData(&logs, &logCount);
                break;
            case 2:
                displayCommunicationLogs(logs, logCount);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 3);


    for (int i = 0; i < logCount; i++) {
        free((char *)logs[i].metadata.id);
    }
```

```
    free(logs);

    return 0;
}
```

```
1. Add Communication
2. Display Communication Logs
3. Exit
Enter your choice: 2

=== Communication Logs ===
Log 1
ID: 1
Timestamp: 2025-01-22 20:04:31
Message: hello

------------------------
Log 2
ID: 2
Timestamp: 2025-01-22 20:04:37
Message: world

------------------------

1. Add Communication
2. Display Communication Logs
3. Exit
```