22nd – Jan

Problem Statements

1. Real-Time Inventory Tracking System Description:

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 Attributes { float
    weight; float
    volume;
};

typedef struct { int
    ID;
    char name[100]; char
    category[50];
    const char *item_code;
} Item;

typedef struct { Item
    details;
    union Attributes attributes; char
    attribute_type[50];
} Inventory;

void addItem(Inventory **inventory, int *count, int id, const char *item_code, const char
*name, const char *category, float attr_value, const char *attr_type); void
displayInventory(Inventory *inventory, int count);

int main() {
    Inventory *inventory = NULL; int count
    = 0;

    addItem(&inventory, &count, 1, "ITEM001", "Laptop", "Electronics", 2.5, "weight");
    addItem(&inventory, &count, 2, "ITEM002", "Water Bottle", "Kitchen", 1.5,
```

```
"volume");
    addItem(&inventory, &count, 3, "ITEM003", "Table", "Furniture", 15.0, "weight");

    displayInventory(inventory, count);

    free(inventory);
    return 0;
}

void addItem(Inventory **inventory, int *count, int id, const char *item_code, const char
*name, const char *category, float attr_value, const char *attr_type) {
    *inventory = realloc(*inventory, (*count + 1) * sizeof(Inventory));
    if (*inventory == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    (*inventory)[*count].details.ID = id;
    (*inventory)[*count].details.item_code = item_code;
    strncpy((*inventory)[*count].details.name, name,
sizeof((*inventory)[*count].details.name) - 1);
    (*inventory)[*count].details.name[sizeof((*inventory)[*count].details.name) - 1] = '\0';
    strncpy((*inventory)[*count].details.category, category,
sizeof((*inventory)[*count].details.category) - 1);
    (*inventory)[*count].details.category[sizeof((*inventory)[*count].details.category) - 1]
= '\0';

    if (strcmp(attr_type, "weight") == 0) {
        (*inventory)[*count].attributes.weight = attr_value;
        strncpy((*inventory)[*count].attribute_type, "weight",
sizeof((*inventory)[*count].attribute_type) - 1);
        (*inventory)[*count].attribute_type[sizeof((*inventory)[*count].attribute_type) - 1]
= '\0';
    } else if (strcmp(attr_type, "volume") == 0) {
        (*inventory)[*count].attributes.volume = attr_value;
        strncpy((*inventory)[*count].attribute_type, "volume",
sizeof((*inventory)[*count].attribute_type) - 1);
        (*inventory)[*count].attribute_type[sizeof((*inventory)[*count].attribute_type) - 1]
= '\0';
    } else {
        printf("Invalid attribute type! Use 'weight' or 'volume'.\n");
```

```
        return;
    }

    (*count)++;
}

void displayInventory(Inventory *inventory, int count) {
    printf("\nReal-Time Inventory:\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %d\n", inventory[i].details.ID);
        printf("Item Code: %s\n", inventory[i].details.item_code);
        printf("Name: %s\n", inventory[i].details.name);
        printf("Category: %s\n", inventory[i].details.category);
        if (strcmp(inventory[i].attribute_type, "weight") == 0) {
            printf("Weight: %.2f kg\n", inventory[i].attributes.weight);
        } else if (strcmp(inventory[i].attribute_type, "volume") == 0) {
            printf("Volume: %.2f liters\n", inventory[i].attributes.volume);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

1. Dynamic Route Management for Logistics
Description:
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 Modes {
    char air[50];
    char sea[50];
    char land[50];
};

typedef struct {
    int id;
    const char *route_id;
    char start[100];
    char end[100];
    char mode_type[10];
    union Modes mode;
} Route;

void add(Route **routes, int *count, int id, const char *route_id, const char *start, const
char *end, const char *mode, const char *mode_type);
void display(Route *routes, int count);

int main() {
    Route *routes = NULL;
    int count = 0;

    add(&routes, &count, 1, "RR01", "Bengaluru", "Chennai", "Air India", "air");
    add(&routes, &count, 2, "RR02", "Mumbai", "Dubai", "Sea Cargo", "sea");
    add(&routes, &count, 3, "RR03", "Delhi", "Banaras", "Bus", "land");

    display(routes, count);
    free(routes);

    return 0;
}

void add(Route **routes, int *count, int id, const char *route_id, const char *start, const
char *end, const char *mode, const char *mode_type) {
    *routes = realloc(*routes, (*count + 1) * sizeof(Route));
    if (*routes == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
```

```
    Route *new_route = &(*routes)[*count];
    new_route->id = id;
    new_route->route_id = route_id;
    strncpy(new_route->start, start, sizeof(new_route->start) - 1);
    new_route->start[sizeof(new_route->start) - 1] = '\0';
    strncpy(new_route->end, end, sizeof(new_route->end) - 1);
    new_route->end[sizeof(new_route->end) - 1] = '\0';

    if (strcmp(mode_type, "air") == 0) {
        strncpy(new_route->mode.air, mode, sizeof(new_route->mode.air) - 1);
        new_route->mode.air[sizeof(new_route->mode.air) - 1] = '\0';
        strncpy(new_route->mode_type, "air", sizeof(new_route->mode_type) - 1);
    } else if (strcmp(mode_type, "sea") == 0) {
        strncpy(new_route->mode.sea, mode, sizeof(new_route->mode.sea) - 1);
        new_route->mode.sea[sizeof(new_route->mode.sea) - 1] = '\0';
        strncpy(new_route->mode_type, "sea", sizeof(new_route->mode_type) - 1);
    } else if (strcmp(mode_type, "land") == 0) {
        strncpy(new_route->mode.land, mode, sizeof(new_route->mode.land) - 1);
        new_route->mode.land[sizeof(new_route->mode.land) - 1] = '\0';
        strncpy(new_route->mode_type, "land", sizeof(new_route->mode_type) - 1);
    } else {
        printf("Invalid mode type! Use 'air', 'sea', or 'land'.\n");
        return;
    }
    new_route->mode_type[sizeof(new_route->mode_type) - 1] = '\0';

    (*count)++;
}

void display(Route *routes, int count) {
    printf("\nRoutes:\n");
    printf("-----------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Route ID: %s\n", routes[i].route_id);
        printf("From: %s\n", routes[i].start);
        printf("To: %s\n", routes[i].end);

        if (strcmp(routes[i].mode_type, "air") == 0) {
            printf("Mode: Air (%s)\n", routes[i].mode.air);
        } else if (strcmp(routes[i].mode_type, "sea") == 0) {
            printf("Mode: Sea (%s)\n", routes[i].mode.sea);
```

```
        } else if (strcmp(routes[i].mode_type, "land") == 0) {
            printf("Mode: Land (%s)\n", routes[i].mode.land);
        }

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

2. Fleet Maintenance and Monitoring
Description:
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  <stdlib.h>
#include <string.h>

union Status {
    char active[50];
    char maintenance[50];
};

typedef struct {
    int id;
    const char *vehicle_id;
    char type[50];
    char status_type[50];
    union Status status;
} Vehicle;

void add(Vehicle **vehicle, int *count, int id, const char *vehicle_id, const char *type, const char *status, const char *status_type);
void display(Vehicle *vehicle, int count);
```

```
int main() {
    Vehicle *vehicle = NULL;
    int count = 0;

    add(&vehicle, &count, 1, "KA01", "Truck", "Running", "active");
    add(&vehicle, &count, 2, "AP05", "Car", "Not working", "maintenance");
    add(&vehicle, &count, 3, "KA56", "Bike", "Running", "active");
    add(&vehicle, &count, 4, "AP89", "Van", "Under Maintenance", "maintenance");

    display(vehicle, count);

    free(vehicle);

    return 0;
}

void add(Vehicle **vehicle, int *count, int id, const char *vehicle_id, const char *type,
const char *status, const char *status_type) {
    *vehicle = realloc(*vehicle, (*count + 1) * sizeof(Vehicle));
    if (*vehicle == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Vehicle *new_vehicle = &(*vehicle)[*count];
    new_vehicle->id = id;
    new_vehicle->vehicle_id = vehicle_id;
    strncpy(new_vehicle->type, type, sizeof(new_vehicle->type) - 1);
    new_vehicle->type[sizeof(new_vehicle->type) - 1] = '\0';

    if (strcmp(status_type, "active") == 0) {
        strncpy(new_vehicle->status.active, status, sizeof(new_vehicle->status.active) - 1);
        new_vehicle->status.active[sizeof(new_vehicle->status.active) - 1] = '\0';
        strncpy(new_vehicle->status_type, "active", sizeof(new_vehicle->status_type) - 1);
    } else if (strcmp(status_type, "maintenance") == 0) {
        strncpy(new_vehicle->status.maintenance, status,
sizeof(new_vehicle->status.maintenance) - 1);
        new_vehicle->status.maintenance[sizeof(new_vehicle->status.maintenance) - 1] =
'\0';
        strncpy(new_vehicle->status_type, "maintenance",
```

```
sizeof(new_vehicle->status_type) - 1);
    } else {
        printf("Invalid status type! Use 'active' or 'maintenance'.\n");
        return;
    }
    new_vehicle->status_type[sizeof(new_vehicle->status_type) - 1] = '\0';

    (*count)++;
}

void display(Vehicle *vehicle, int count) {
    printf("\nFleet Maintenance and Monitoring\n");
    printf("-----------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Vehicle ID : %s\n", vehicle[i].vehicle_id);
        printf("Type       : %s\n", vehicle[i].type);

        if (strcmp(vehicle[i].status_type, "active") == 0) {
            printf("Status     : Active (%s)\n", vehicle[i].status.active);
        } else if (strcmp(vehicle[i].status_type, "maintenance") == 0) {
            printf("Status     : Maintenance (%s)\n", vehicle[i].status.maintenance);
        }
        printf("-----------------------------------------------------------\n");
    }
}
```

3. Logistics Order Processing Queue
Description:
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>

union PaymentMethods {
    char credit_card[50];
    char cash[50];
};

typedef struct {
    int id;
    const char *order_id;
    char customer[100];
    char items[200];
    union PaymentMethods payment_method;
    char payment_type[20];
} Order;

void add(Order **orders, int *count, int id, const char *order_id, const char *customer,
const char *items, const char *payment, const char *payment_type);
void display(Order *orders, int count);

int main() {
    Order *orders = NULL;
    int count = 0;

    add(&orders, &count, 1, "ORD001", "John Doe", "Laptop, Mouse", "VISA",
"credit_card");
    add(&orders, &count, 2, "ORD002", "Jane Smith", "Shoes, T-shirt", "Cash", "cash");
    add(&orders, &count, 3, "ORD003", "Sam Wilson", "Phone, Headphones",
"MasterCard", "credit_card");

    display(orders, count);

    free(orders);

    return 0;
}

void add(Order **orders, int *count, int id, const char *order_id, const char *customer,
const char *items, const char *payment, const char *payment_type) {
    *orders = realloc(*orders, (*count + 1) * sizeof(Order));
```

```
    if (*orders == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Order *new_order = &(*orders)[*count];
    new_order->id = id;
    new_order->order_id = order_id;
    strncpy(new_order->customer, customer, sizeof(new_order->customer) - 1);
    new_order->customer[sizeof(new_order->customer) - 1] = '\0';
    strncpy(new_order->items, items, sizeof(new_order->items) - 1);
    new_order->items[sizeof(new_order->items) - 1] = '\0';

    if (strcmp(payment_type, "credit_card") == 0) {
        strncpy(new_order->payment_method.credit_card, payment,
sizeof(new_order->payment_method.credit_card) - 1);

new_order->payment_method.credit_card[sizeof(new_order->payment_method.credit_ca
rd) - 1] = '\0';
        strncpy(new_order->payment_type, "credit_card",
sizeof(new_order->payment_type) - 1);
    } else if (strcmp(payment_type, "cash") == 0) {
        strncpy(new_order->payment_method.cash, payment,
sizeof(new_order->payment_method.cash) - 1);
        new_order->payment_method.cash[sizeof(new_order->payment_method.cash) - 1]
= '\0';
        strncpy(new_order->payment_type, "cash", sizeof(new_order->payment_type) - 1);
    } else {
        printf("Invalid payment type! Use 'credit_card' or 'cash'.\n");
        return;
    }

    (*count)++;
}

void display(Order *orders, int count) {
    printf("\nLogistics Order Processing Queue\n");
    printf("-------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Order ID      : %s\n", orders[i].order_id);
        printf("Customer      : %s\n", orders[i].customer);
```

```
        printf("Items         : %s\n", orders[i].items);

        if (strcmp(orders[i].payment_type, "credit_card") == 0) {
            printf("Payment Method: Credit Card (%s)\n",
orders[i].payment_method.credit_card);
        } else if (strcmp(orders[i].payment_type, "cash") == 0) {
            printf("Payment Method: Cash (%s)\n", orders[i].payment_method.cash);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

4. Shipment Tracking System
Description:
Develop a shipment tracking system using structures for shipment details and unions for tracking events. Use const pointers to protect tracking numbers and double pointers to handle dynamic shipment lists.
Specifications:
Structure: Shipment details (tracking number, origin, destination).
Union: Tracking events (dispatched, delivered).
const Pointer: Tracking numbers.
Double Pointers: Dynamic shipment tracking.

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

union TrackingEvents {
    char dispatched[50];
    char delivered[50];
};

typedef struct {
    const char *tracking_number;
    char origin[100];
    char destination[100];
    union TrackingEvents tracking_event;
    char event_type[20];
} Shipment;
```

```
void add(Shipment **shipments, int *count, const char *tracking_number, const char *origin, const char *destination, const char *event, const char *event_type);
void display(Shipment *shipments, int count);

int main() {
    Shipment *shipments = NULL;
    int count = 0;

    add(&shipments, &count, "TN12345", "New York", "Los Angeles", "Dispatched", "dispatched");
    add(&shipments, &count, "TN12346", "Chicago", "Houston", "Delivered", "delivered");
    add(&shipments, &count, "TN12347", "San Francisco", "Seattle", "Dispatched", "dispatched");

    display(shipments, count);

    free(shipments);

    return 0;
}

void add(Shipment **shipments, int *count, const char *tracking_number, const char *origin, const char *destination, const char *event, const char *event_type) {
    *shipments = realloc(*shipments, (*count + 1) * sizeof(Shipment));
    if (*shipments == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Shipment *new_shipment = &(*shipments)[*count];
    new_shipment->tracking_number = tracking_number;
    strncpy(new_shipment->origin, origin, sizeof(new_shipment->origin) - 1);
    new_shipment->origin[sizeof(new_shipment->origin) - 1] = '\0';
    strncpy(new_shipment->destination, destination, sizeof(new_shipment->destination) - 1);
    new_shipment->destination[sizeof(new_shipment->destination) - 1] = '\0';

    if (strcmp(event_type, "dispatched") == 0) {
        strncpy(new_shipment->tracking_event.dispatched, event,
```

```
sizeof(new_shipment->tracking_event.dispatched)  -  1);

new_shipment->tracking_event.dispatched[sizeof(new_shipment->tracking_event.dispat
ched) - 1] = '\0';
        strncpy(new_shipment->event_type, "dispatched",
sizeof(new_shipment->event_type) - 1);
    } else if (strcmp(event_type, "delivered") == 0) {
        strncpy(new_shipment->tracking_event.delivered, event,
sizeof(new_shipment->tracking_event.delivered)  -  1);

new_shipment->tracking_event.delivered[sizeof(new_shipment->tracking_event.delivere
d) - 1] = '\0';
        strncpy(new_shipment->event_type, "delivered",
sizeof(new_shipment->event_type) - 1);
    } else {
        printf("Invalid event type! Use 'dispatched' or 'delivered'.\n");
        return;
    }

    (*count)++;
}

void display(Shipment *shipments, int count) {
    printf("\nShipment Tracking System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Tracking Number: %s\n", shipments[i].tracking_number);
        printf("Origin: %s\n", shipments[i].origin);
        printf("Destination: %s\n", shipments[i].destination);

        if (strcmp(shipments[i].event_type, "dispatched") == 0) {
            printf("Tracking Event: Dispatched (%s)\n",
shipments[i].tracking_event.dispatched);
        } else if (strcmp(shipments[i].event_type, "delivered") == 0) {
            printf("Tracking Event: Delivered (%s)\n",
shipments[i].tracking_event.delivered);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

5. Real-Time Traffic Management for Logistics
Description:
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>

union TrafficConditions {
    char clear[50];
    char congested[50];
};

typedef struct {
    const char *node_id;
    char location[100];
    union TrafficConditions condition;
    char condition_type[20];
} TrafficNode;

void add(TrafficNode **traffic_data, int *count, const char *node_id, const char
*location, const char *condition, const char *condition_type);
void display(TrafficNode *traffic_data, int count);

int main() {
    TrafficNode *traffic_data = NULL;
    int count = 0;

    add(&traffic_data, &count, "TN001", "Bengaluru - ORR", "Clear", "clear");
    add(&traffic_data, &count, "TN002", "Mumbai - Western Express Highway",
"Congested", "congested");
    add(&traffic_data, &count, "TN003", "Hyderabad - MG Road", "Clear", "clear");
```

```
    display(traffic_data, count);

    free(traffic_data);

    return 0;
}

void add(TrafficNode **traffic_data, int *count, const char *node_id, const char
*location, const char *condition, const char *condition_type) {
    *traffic_data = realloc(*traffic_data, (*count + 1) * sizeof(TrafficNode));
    if (*traffic_data == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    TrafficNode *new_node = &(*traffic_data)[*count];
    new_node->node_id = node_id;
    strncpy(new_node->location, location, sizeof(new_node->location) - 1);
    new_node->location[sizeof(new_node->location) - 1] = '\0';

    if (strcmp(condition_type, "clear") == 0) {
        strncpy(new_node->condition.clear, condition, sizeof(new_node->condition.clear) -
1);
        new_node->condition.clear[sizeof(new_node->condition.clear) - 1] = '\0';
        strncpy(new_node->condition_type, "clear", sizeof(new_node->condition_type) -
1);
    } else if (strcmp(condition_type, "congested") == 0) {
        strncpy(new_node->condition.congested, condition,
sizeof(new_node->condition.congested) - 1);
        new_node->condition.congested[sizeof(new_node->condition.congested) - 1] = '\0';
        strncpy(new_node->condition_type, "congested",
sizeof(new_node->condition_type) - 1);
    } else {
        printf("Invalid condition type! Use 'clear' or 'congested'.\n");
        return;
    }

    (*count)++;
}
```

```
void display(TrafficNode *traffic_data, int count) {
    printf("\nReal-Time Traffic Management for Logistics\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Node ID: %s\n", traffic_data[i].node_id);
        printf("Location: %s\n", traffic_data[i].location);

        if (strcmp(traffic_data[i].condition_type, "clear") == 0) {
            printf("Condition: Clear (%s)\n", traffic_data[i].condition.clear);
        } else if (strcmp(traffic_data[i].condition_type, "congested") == 0) {
            printf("Condition: Congested (%s)\n", traffic_data[i].condition.congested);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

6. Warehouse Slot Allocation System
Description:
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>

union ItemTypes {
    char perishable[50];
    char non_perishable[50];
};

typedef struct {
    const char *slot_id;
    char location[100];
```

```
    int size;
    union ItemTypes item_type;
    char type[20];  // 'perishable' or 'non-perishable'
} WarehouseSlot;

void add(WarehouseSlot **slots, int *count, const char *slot_id, const char *location, int
size, const char *item_type, const char *type);
void display(WarehouseSlot *slots, int count);

int main() {
    WarehouseSlot *slots = NULL;
    int count = 0;

    add(&slots, &count, "S001", "Aisle 1, Section 2", 100, "Fruits", "perishable");
    add(&slots, &count, "S002", "Aisle 2, Section 1", 200, "Canned Goods",
"non-perishable");
    add(&slots, &count, "S003", "Aisle 3, Section 3", 150, "Vegetables", "perishable");

    display(slots, count);

    free(slots);

    return 0;
}

void add(WarehouseSlot **slots, int *count, const char *slot_id, const char *location, int
size, const char *item_type, const char *type) {
    *slots = realloc(*slots, (*count + 1) * sizeof(WarehouseSlot));
    if (*slots == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    WarehouseSlot *new_slot = &(*slots)[*count];
    new_slot->slot_id = slot_id;
    strncpy(new_slot->location, location, sizeof(new_slot->location) - 1);
    new_slot->location[sizeof(new_slot->location) - 1] = '\0';
    new_slot->size = size;

    if (strcmp(type, "perishable") == 0) {
        strncpy(new_slot->item_type.perishable, item_type,
```

```
sizeof(new_slot->item_type.perishable) - 1);
        new_slot->item_type.perishable[sizeof(new_slot->item_type.perishable) - 1] = '\0';
        strncpy(new_slot->type, "perishable", sizeof(new_slot->type) - 1);
    } else if (strcmp(type, "non-perishable") == 0) {
        strncpy(new_slot->item_type.non_perishable, item_type,
sizeof(new_slot->item_type.non_perishable) - 1);
        new_slot->item_type.non_perishable[sizeof(new_slot->item_type.non_perishable) -
1] = '\0';
        strncpy(new_slot->type, "non-perishable", sizeof(new_slot->type) - 1);
    } else {
        printf("Invalid item type! Use 'perishable' or 'non-perishable'.\n");
        return;
    }

    (*count)++;
}

void display(WarehouseSlot *slots, int count) {
    printf("\nWarehouse Slot Allocation System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Slot ID: %s\n", slots[i].slot_id);
        printf("Location: %s\n", slots[i].location);
        printf("Size: %d\n", slots[i].size);
        printf("Item Type: %s (%s)\n", slots[i].type, (strcmp(slots[i].type, "perishable") ==
0) ? slots[i].item_type.perishable : slots[i].item_type.non_perishable);
        printf("------------------------------------------------------------\n");
    }
}
```

7. Package Delivery Optimization Tool

Description:

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>

union DeliveryMethods {
    char standard[50];
    char express[50];
};

typedef struct {
    const char *package_id;
    float weight;
    char destination[100];
    union DeliveryMethods delivery_method;
    char method_type[20];  // 'standard' or 'express'
} Package;

void add(Package **packages, int *count, const char *package_id, float weight, const
char *destination, const  char *delivery_method, const char *method_type);
void display(Package *packages, int count);

int main() {
    Package *packages = NULL;
    int count = 0;

    add(&packages, &count, "P001", 2.5, "New York", "Standard Delivery", "standard");
    add(&packages, &count, "P002", 1.2, "Los Angeles", "Express Delivery", "express");
    add(&packages, &count, "P003", 3.0, "Chicago", "Standard Delivery", "standard");

    display(packages, count);

    free(packages);

    return 0;
}

void add(Package **packages, int *count, const char *package_id, float weight, const
char *destination, const char *delivery_method, const char *method_type) {
```

```
    *packages = realloc(*packages, (*count + 1) * sizeof(Package));
    if (*packages == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Package *new_package = &(*packages)[*count];
    new_package->package_id = package_id;
    new_package->weight = weight;
    strncpy(new_package->destination, destination, sizeof(new_package->destination) -
1);
    new_package->destination[sizeof(new_package->destination) - 1] = '\0';

    if (strcmp(method_type, "standard") == 0) {
        strncpy(new_package->delivery_method.standard, delivery_method,
sizeof(new_package->delivery_method.standard) - 1);

new_package->delivery_method.standard[sizeof(new_package->delivery_method.standa
rd) - 1] = '\0';
        strncpy(new_package->method_type, "standard",
sizeof(new_package->method_type) - 1);
    } else if (strcmp(method_type, "express") == 0) {
        strncpy(new_package->delivery_method.express, delivery_method,
sizeof(new_package->delivery_method.express) - 1);

new_package->delivery_method.express[sizeof(new_package->delivery_method.express
) - 1] = '\0';
        strncpy(new_package->method_type, "express",
sizeof(new_package->method_type) - 1);
    } else {
        printf("Invalid delivery method! Use 'standard' or 'express'.\n");
        return;
    }

    (*count)++;
}

void display(Package *packages, int count) {
    printf("\nPackage Delivery Optimization Tool\n");
    printf("-------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
```

```
        printf("Package ID: %s\n", packages[i].package_id);
        printf("Weight: %.2f kg\n", packages[i].weight);
        printf("Destination: %s\n", packages[i].destination);
        printf("Delivery Method: %s (%s)\n", packages[i].method_type,
(strcmp(packages[i].method_type, "standard") == 0) ?
packages[i].delivery_method.standard : packages[i].delivery_method.express);
        printf("------------------------------------------------------------\n");
    }
}
```

8. Logistics Data Analytics System
Description:
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>

union Metrics {
    float speed;
    float efficiency;
};

typedef struct {
    const char *timestamp;
    union Metrics metric;
    char metric_type[20];  // 'speed' or 'efficiency'
} AnalyticsRecord;

void add(AnalyticsRecord **data, int *count, const char *timestamp, float value, const
char *metric_type);
void display(AnalyticsRecord *data, int count);
```

```
int main() {
    AnalyticsRecord *data = NULL;
    int count = 0;

    add(&data, &count, "2025-01-22 10:00", 45.5, "speed");
    add(&data, &count, "2025-01-22 11:00", 80.3, "efficiency");
    add(&data, &count, "2025-01-22 12:00", 50.1, "speed");

    display(data, count);

    free(data);

    return 0;
}

void add(AnalyticsRecord **data, int *count, const char *timestamp, float value, const
char *metric_type) {
    *data = realloc(*data, (*count + 1) * sizeof(AnalyticsRecord));
    if (*data == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    AnalyticsRecord *new_record = &(*data)[*count];
    new_record->timestamp = timestamp;

    if (strcmp(metric_type, "speed") == 0) {
        new_record->metric.speed = value;
        strncpy(new_record->metric_type, "speed", sizeof(new_record->metric_type) - 1);
    } else if (strcmp(metric_type, "efficiency") == 0) {
        new_record->metric.efficiency = value;
        strncpy(new_record->metric_type, "efficiency", sizeof(new_record->metric_type) -
1);
    } else {
        printf("Invalid metric type! Use 'speed' or 'efficiency'.\n");
        return;
    }

    (*count)++;
}
```

```
void display(AnalyticsRecord *data, int count) {
    printf("\nLogistics Data Analytics System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Timestamp: %s\n", data[i].timestamp);
        if (strcmp(data[i].metric_type, "speed") == 0) {
            printf("Metric: Speed (%.2f km/h)\n", data[i].metric.speed);
        } else if (strcmp(data[i].metric_type, "efficiency") == 0) {
            printf("Metric: Efficiency (%.2f%%)\n", data[i].metric.efficiency);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

9. Transportation Schedule Management

Description:

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 TransportType {
    char bus[50];
    char truck[50];
};

typedef struct {
    const char *schedule_id;
    const char *start_time;
    const char *end_time;
```

```
    union TransportType transport;
    char transport_type[10];  // 'bus' or 'truck'
} Schedule;

void add(Schedule **schedule, int *count, const char *schedule_id, const char
*start_time, const char *end_time, const char *transport, const char *transport_type);
void display(Schedule *schedule, int count);

int main() {
    Schedule *schedule = NULL;
    int count = 0;

    add(&schedule, &count, "S001", "10:00 AM", "12:00 PM", "City Bus", "bus");
    add(&schedule, &count, "S002", "01:00 PM", "03:00 PM", "Delivery Truck", "truck");
    add(&schedule, &count, "S003", "04:00 PM", "06:00 PM", "Airport Shuttle", "bus");

    display(schedule, count);

    free(schedule);

    return 0;
}

void add(Schedule **schedule, int *count, const char *schedule_id, const char
*start_time, const char *end_time, const char *transport, const char *transport_type) {
    *schedule = realloc(*schedule, (*count + 1) * sizeof(Schedule));
    if (*schedule == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Schedule *new_schedule = &(*schedule)[*count];
    new_schedule->schedule_id = schedule_id;
    new_schedule->start_time = start_time;
    new_schedule->end_time = end_time;

    if (strcmp(transport_type, "bus") == 0) {
        strncpy(new_schedule->transport.bus, transport,
sizeof(new_schedule->transport.bus) - 1);
        strncpy(new_schedule->transport_type, "bus",
sizeof(new_schedule->transport_type) - 1);
```

```
    } else if (strcmp(transport_type, "truck") == 0) {
        strncpy(new_schedule->transport.truck, transport,
sizeof(new_schedule->transport.truck) - 1);
        strncpy(new_schedule->transport_type, "truck",
sizeof(new_schedule->transport_type) - 1);
    } else {
        printf("Invalid transport type! Use 'bus' or 'truck'.\n");
        return;
    }

    (*count)++;
}

void display(Schedule *schedule, int count) {
    printf("\nTransportation Schedule Management\n");
    printf("----------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Schedule ID: %s\n", schedule[i].schedule_id);
        printf("Start Time: %s\n", schedule[i].start_time);
        printf("End Time: %s\n", schedule[i].end_time);

        if (strcmp(schedule[i].transport_type, "bus") == 0) {
            printf("Transport: Bus (%s)\n", schedule[i].transport.bus);
        } else if (strcmp(schedule[i].transport_type, "truck") == 0) {
            printf("Transport: Truck (%s)\n", schedule[i].transport.truck);
        }

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

10. Dynamic Supply Chain Modeling

Description:

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 TransactionType {
    char purchase[50];
    char return_transaction[50];
};

typedef struct {
    const   char   *transaction_id;
    const  char  *supplier_name;
    const char *customer_name;
    union TransactionType transaction;
    char transaction_type[10];  // 'purchase' or 'return'
} SupplyChain;

void add(SupplyChain **supply_chain, int *count, const char *transaction_id, const char
*supplier_name, const char *customer_name, const char *transaction, const char
*transaction_type);
void display(SupplyChain *supply_chain, int count);

int main() {
    SupplyChain *supply_chain = NULL;
    int count = 0;

    add(&supply_chain, &count, "T001", "Supplier A", "Customer X", "Order 1",
"purchase");
    add(&supply_chain, &count, "T002", "Supplier B", "Customer Y", "Order 2",
"purchase");
    add(&supply_chain, &count, "T003", "Supplier C", "Customer Z", "Return 1",
"return");

    display(supply_chain, count);

    free(supply_chain);
```

```
    return 0;
}

void add(SupplyChain **supply_chain, int *count, const char *transaction_id, const char
*supplier_name, const char *customer_name, const char *transaction, const char
*transaction_type) {
    *supply_chain = realloc(*supply_chain, (*count + 1) * sizeof(SupplyChain));
    if (*supply_chain == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    SupplyChain *new_supply_chain = &(*supply_chain)[*count];
    new_supply_chain->transaction_id = transaction_id;
    new_supply_chain->supplier_name = supplier_name;
    new_supply_chain->customer_name = customer_name;

    if (strcmp(transaction_type, "purchase") == 0) {
        strncpy(new_supply_chain->transaction.purchase, transaction,
sizeof(new_supply_chain->transaction.purchase) - 1);
        strncpy(new_supply_chain->transaction_type, "purchase",
sizeof(new_supply_chain->transaction_type) - 1);
    } else if (strcmp(transaction_type, "return") == 0) {
        strncpy(new_supply_chain->transaction.return_transaction, transaction,
sizeof(new_supply_chain->transaction.return_transaction) - 1);
        strncpy(new_supply_chain->transaction_type, "return",
sizeof(new_supply_chain->transaction_type) - 1);
    } else {
        printf("Invalid transaction type! Use 'purchase' or 'return'.\n");
        return;
    }

    (*count)++;
}

void display(SupplyChain *supply_chain, int count) {
    printf("\nDynamic Supply Chain Modeling\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Transaction ID: %s\n", supply_chain[i].transaction_id);
        printf("Supplier Name: %s\n", supply_chain[i].supplier_name);
```

```
        printf("Customer Name: %s\n", supply_chain[i].customer_name);

        if (strcmp(supply_chain[i].transaction_type, "purchase") == 0) {
            printf("Transaction: Purchase (%s)\n", supply_chain[i].transaction.purchase);
        } else if (strcmp(supply_chain[i].transaction_type, "return") == 0) {
            printf("Transaction: Return (%s)\n",
supply_chain[i].transaction.return_transaction);
        }

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

11. Freight Cost Calculation System
Description:
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 PricingModel {
    float fixed_cost;
    float variable_cost_per_km;
};

typedef struct {
    const char *cost_id;
    float base_cost;
    union PricingModel pricing_model;
    char pricing_type[10];  // 'fixed' or 'variable'
} FreightCost;
```

```
void add(FreightCost **costs, int *count, const char *cost_id, float base_cost, const char *pricing_type, float pricing_value);
void display(FreightCost *costs, int count);

int main() {
    FreightCost *costs = NULL;
    int count = 0;

    add(&costs, &count, "C001", 1000.0, "fixed", 200.0);
    add(&costs, &count, "C002", 500.0, "variable", 5.0);

    display(costs, count);

    free(costs);

    return 0;
}

void add(FreightCost **costs, int *count, const char *cost_id, float base_cost, const char *pricing_type, float pricing_value) {
    *costs = realloc(*costs, (*count + 1) * sizeof(FreightCost));
    if (*costs == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    FreightCost *new_cost = &(*costs)[*count];
    new_cost->cost_id = cost_id;
    new_cost->base_cost = base_cost;

    if (strcmp(pricing_type, "fixed") == 0) {
        new_cost->pricing_model.fixed_cost = pricing_value;
        strncpy(new_cost->pricing_type, "fixed", sizeof(new_cost->pricing_type) - 1);
    } else if (strcmp(pricing_type, "variable") == 0) {
        new_cost->pricing_model.variable_cost_per_km = pricing_value;
        strncpy(new_cost->pricing_type, "variable", sizeof(new_cost->pricing_type) - 1);
    } else {
        printf("Invalid pricing type! Use 'fixed' or 'variable'.\n");
        return;
    }
```

```
    (*count)++;
}

void display(FreightCost *costs, int count) {
    printf("\nFreight Cost Calculation System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Cost ID: %s\n", costs[i].cost_id);
        printf("Base Cost: %.2f\n", costs[i].base_cost);

        if (strcmp(costs[i].pricing_type, "fixed") == 0) {
            printf("Pricing Model: Fixed (%.2f)\n", costs[i].pricing_model.fixed_cost);
        } else if (strcmp(costs[i].pricing_type, "variable") == 0) {
            printf("Pricing Model: Variable (%.2f per km)\n",
costs[i].pricing_model.variable_cost_per_km);
        }

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

14. Intermodal Transport Management System
Description:
Implement an intermodal transport management system using structures for transport details and unions for transport modes. 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 TransportMode {
    char rail[50];
    char road[50];
};
```

```
typedef struct {
    const char *transport_id;
    char origin[50];
    char destination[50];
    union TransportMode mode;
    char type[10];
} Transport;

void addTransport(Transport **routes, int *count, const char *transport_id, const char
*origin, const char *destination, const char *type, const char *description);
void displayRoutes(Transport *routes, int count);

int main() {
    Transport *routes = NULL;
    int count = 0;

    addTransport(&routes, &count, "T001", "San Francisco", "Los Angeles", "road",
"Highway 101");
    addTransport(&routes, &count, "T002", "Chicago", "New York", "rail", "Freight Line
A");
    addTransport(&routes, &count, "T003", "Houston", "Dallas", "road", "Interstate 45");
    addTransport(&routes, &count, "T004", "Seattle", "Portland", "rail", "Pacific Rail");

    displayRoutes(routes, count);

    free(routes);
    return 0;
}

void addTransport(Transport **routes, int *count, const char *transport_id, const char
*origin, const char *destination, const char *type, const char *description) {
    *routes = realloc(*routes, (*count + 1) * sizeof(Transport));
    if (*routes == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Transport *new_transport = &(*routes)[*count];
    new_transport->transport_id = transport_id;
    strncpy(new_transport->origin, origin, sizeof(new_transport->origin) - 1);
```

```
    new_transport->origin[sizeof(new_transport->origin) - 1] = '\0';
    strncpy(new_transport->destination, destination, sizeof(new_transport->destination) -
1);
    new_transport->destination[sizeof(new_transport->destination) - 1] = '\0';

    if (strcmp(type, "road") == 0) {
        strncpy(new_transport->mode.road, description, sizeof(new_transport->mode.road)
- 1);
        new_transport->mode.road[sizeof(new_transport->mode.road) - 1] = '\0';
        strncpy(new_transport->type, "road", sizeof(new_transport->type) - 1);
    } else if (strcmp(type, "rail") == 0) {
        strncpy(new_transport->mode.rail, description, sizeof(new_transport->mode.rail) -
1);
        new_transport->mode.rail[sizeof(new_transport->mode.rail) - 1] = '\0';
        strncpy(new_transport->type, "rail", sizeof(new_transport->type) - 1);
    } else {
        printf("Invalid transport mode! Use 'rail' or 'road'.\n");
        return;
    }

    (*count)++;
}

void displayRoutes(Transport *routes, int count) {
    printf("\nIntermodal Transport Management System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Transport ID: %s\n", routes[i].transport_id);
        printf("Origin: %s\n", routes[i].origin);
        printf("Destination: %s\n", routes[i].destination);
        if (strcmp(routes[i].type, "road") == 0) {
            printf("Mode: Road (%s)\n", routes[i].mode.road);
        } else if (strcmp(routes[i].type, "rail") == 0) {
            printf("Mode: Rail (%s)\n", routes[i].mode.rail);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

15. Logistics Performance Monitoring
Description:
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>

union PerformanceAspect {
   double time;
   double cost;
};

typedef struct {
   const char *metric_id;
   double value;
   union PerformanceAspect aspect;
   char type[10];
} Performance;

void addPerformance(Performance **records, int *count, const char *metric_id, double value, const char *type, double aspect_value);
void displayPerformance(Performance *records, int count);

int main() {
   Performance *records = NULL;
   int count = 0;

   addPerformance(&records, &count, "M001", 98.5, "time", 5.2);
   addPerformance(&records, &count, "M002", 76.3, "cost", 1500.0);
   addPerformance(&records, &count, "M003", 89.7, "time", 4.8);
   addPerformance(&records, &count, "M004", 65.2, "cost", 1200.0);
```

```
    displayPerformance(records,  count);

    free(records);
    return 0;
}

void addPerformance(Performance **records, int *count, const char *metric_id, double
value, const char *type, double aspect_value) {
    *records = realloc(*records, (*count + 1) * sizeof(Performance));
    if (*records == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Performance *new_record = &(*records)[*count];
    new_record->metric_id = metric_id;
    new_record->value = value;

    if (strcmp(type, "time") == 0) {
        new_record->aspect.time = aspect_value;
        strncpy(new_record->type, "time", sizeof(new_record->type) - 1);
    } else if (strcmp(type, "cost") == 0) {
        new_record->aspect.cost = aspect_value;
        strncpy(new_record->type, "cost", sizeof(new_record->type) - 1);
    } else {
        printf("Invalid performance type! Use 'time' or 'cost'.\n");
        return;
    }

    (*count)++;
}

void displayPerformance(Performance *records, int count) {
    printf("\nLogistics Performance Monitoring System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Metric ID: %s\n", records[i].metric_id);
        printf("Value: %.2f\n", records[i].value);
        if (strcmp(records[i].type, "time") == 0) {
            printf("Aspect: Time (%.2f hours)\n", records[i].aspect.time);
        } else if (strcmp(records[i].type, "cost") == 0) {
```

```
            printf("Aspect: Cost (%.2f USD)\n", records[i].aspect.cost);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

16. Warehouse Robotics Coordination
Description:
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 TaskType {
    char picking[50];
    char sorting[50];
};

typedef struct {
    const char *robot_id;
    char type[50];
    char status[50];
    union TaskType task;
    char task_type[10];
} Robot;

void addRobot(Robot **robots, int *count, const char *robot_id, const char *type, const char *status, const char *task_type, const char *task);
void displayRobots(Robot *robots, int count);

int main() {
    Robot *robots = NULL;
```

```
    int count = 0;

    addRobot(&robots, &count, "R001", "Picker", "Active", "picking", "Picking items
from aisle A");
    addRobot(&robots, &count, "R002", "Sorter", "Idle", "sorting", "Sorting items in zone
B");
    addRobot(&robots, &count, "R003", "Picker", "Active", "picking", "Picking items
from aisle B");
    addRobot(&robots, &count, "R004", "Sorter", "Maintenance", "sorting", "Sorting
paused due to maintenance");

    displayRobots(robots,  count);

    free(robots);
    return 0;
}

void addRobot(Robot **robots, int *count, const char *robot_id, const char *type, const
char *status, const char *task_type, const char *task) {
    *robots = realloc(*robots, (*count + 1) * sizeof(Robot));
    if (*robots == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Robot *new_robot = &(*robots)[*count];
    new_robot->robot_id = robot_id;
    strncpy(new_robot->type, type, sizeof(new_robot->type) - 1);
    new_robot->type[sizeof(new_robot->type) - 1] = '\0';
    strncpy(new_robot->status, status, sizeof(new_robot->status) - 1);
    new_robot->status[sizeof(new_robot->status) - 1] = '\0';

    if (strcmp(task_type, "picking") == 0) {
        strncpy(new_robot->task.picking, task, sizeof(new_robot->task.picking) - 1);
        new_robot->task.picking[sizeof(new_robot->task.picking) - 1] = '\0';
        strncpy(new_robot->task_type,  "picking",  sizeof(new_robot->task_type)  -  1);
    } else if (strcmp(task_type, "sorting") == 0) {
        strncpy(new_robot->task.sorting, task, sizeof(new_robot->task.sorting) - 1);
        new_robot->task.sorting[sizeof(new_robot->task.sorting) - 1] = '\0';
        strncpy(new_robot->task_type,  "sorting",  sizeof(new_robot->task_type)  -  1);
    } else {
```

```
        printf("Invalid task type! Use 'picking' or 'sorting'.\n");
        return;
    }

    (*count)++;
}

void displayRobots(Robot *robots, int count) {
    printf("\nWarehouse Robotics Coordination System\n");
    printf("----------------------------------------------------
```

17. Customer Feedback Analysis System
Description:
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  <stdlib.h>
#include <string.h>

union FeedbackType {
    char positive[100];
    char negative[100];
};

typedef struct {
    const char *feedback_id;
    char content[200];
    union FeedbackType feedback;
    char feedback_type[10];
} Feedback;
```

```
void addFeedback(Feedback **feedbacks, int *count, const char *feedback_id, const
char *content, const char *feedback_type, const char *feedback_detail);
void displayFeedbacks(Feedback *feedbacks, int count);

int main() {
    Feedback *feedbacks = NULL;
    int count = 0;

    addFeedback(&feedbacks, &count, "F001", "Great service and fast delivery!",
"positive", "Highly satisfied with the experience.");
    addFeedback(&feedbacks, &count, "F002", "Delivery was delayed.", "negative",
"Unhappy with the delay in delivery.");
    addFeedback(&feedbacks, &count, "F003", "Packaging was excellent!", "positive",
"Impressed with the care in packaging.");
    addFeedback(&feedbacks, &count, "F004", "Received a damaged product.",
"negative", "Product arrived broken and unusable.");

    displayFeedbacks(feedbacks,  count);

    free(feedbacks);
    return 0;
}

void addFeedback(Feedback **feedbacks, int *count, const char *feedback_id, const
char *content, const char *feedback_type, const char *feedback_detail)  {
    *feedbacks = realloc(*feedbacks, (*count + 1) * sizeof(Feedback));
    if (*feedbacks == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Feedback *new_feedback = &(*feedbacks)[*count];
    new_feedback->feedback_id = feedback_id;
    strncpy(new_feedback->content, content, sizeof(new_feedback->content) - 1);
    new_feedback->content[sizeof(new_feedback->content) - 1] = '\0';

    if (strcmp(feedback_type, "positive") == 0) {
        strncpy(new_feedback->feedback.positive, feedback_detail,
sizeof(new_feedback->feedback.positive)  -  1);
        new_feedback->feedback.positive[sizeof(new_feedback->feedback.positive)  -  1]  =
'\0';
```

```
        strncpy(new_feedback->feedback_type, "positive",
sizeof(new_feedback->feedback_type) - 1);
    } else if (strcmp(feedback_type, "negative") == 0) {
        strncpy(new_feedback->feedback.negative, feedback_detail,
sizeof(new_feedback->feedback.negative) - 1);
        new_feedback->feedback.negative[sizeof(new_feedback->feedback.negative) - 1] =
'\0';
        strncpy(new_feedback->feedback_type, "negative",
sizeof(new_feedback->feedback_type) - 1);
    } else {
        printf("Invalid feedback type! Use 'positive' or 'negative'.\n");
        return;
    }

    (*count)++;
}

void displayFeedbacks(Feedback *feedbacks, int count) {
    printf("\nCustomer Feedback Analysis System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Feedback ID: %s\n", feedbacks[i].feedback_id);
        printf("Content: %s\n", feedbacks[i].content);
        if (strcmp(feedbacks[i].feedback_type, "positive") == 0) {
            printf("Type: Positive (%s)\n", feedbacks[i].feedback.positive);
        } else if (strcmp(feedbacks[i].feedback_type, "negative") == 0) {
            printf("Type: Negative (%s)\n", feedbacks[i].feedback.negative);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

18. Real-Time Fleet Coordination
Description:
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 CoordinationType {
    char dispatch[100];
    char reroute[100];
};

typedef struct {
    const char *fleet_id;
    char location[100];
    char status[50];
    union CoordinationType coordination;
    char coordination_type[10];
} Fleet;

void addFleet(Fleet **fleets, int *count, const char *fleet_id, const char *location, const
char *status, const char *coordination_type, const char *coordination_detail);
void displayFleets(Fleet *fleets, int count);

int main() {
    Fleet *fleets = NULL;
    int count = 0;

    addFleet(&fleets, &count, "FL001", "New York", "Active", "dispatch", "Dispatch to
warehouse A.");
    addFleet(&fleets, &count, "FL002", "San Francisco", "Idle", "reroute", "Reroute to
station B.");
    addFleet(&fleets, &count, "FL003", "Chicago", "In Transit", "dispatch", "Dispatch to
station C.");
    addFleet(&fleets, &count, "FL004", "Los Angeles", "Under Maintenance", "reroute",
"Reroute to service center.");

    displayFleets(fleets, count);

    free(fleets);
```

```
    return 0;
}

void addFleet(Fleet **fleets, int *count, const char *fleet_id, const char *location, const
char *status, const char *coordination_type, const char *coordination_detail) {
    *fleets = realloc(*fleets, (*count + 1) * sizeof(Fleet));
    if (*fleets == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Fleet *new_fleet = &(*fleets)[*count];
    new_fleet->fleet_id = fleet_id;
    strncpy(new_fleet->location, location, sizeof(new_fleet->location) - 1);
    new_fleet->location[sizeof(new_fleet->location) - 1] = '\0';
    strncpy(new_fleet->status, status, sizeof(new_fleet->status) - 1);
    new_fleet->status[sizeof(new_fleet->status) - 1] = '\0';

    if (strcmp(coordination_type, "dispatch") == 0) {
        strncpy(new_fleet->coordination.dispatch, coordination_detail,
sizeof(new_fleet->coordination.dispatch) - 1);
        new_fleet->coordination.dispatch[sizeof(new_fleet->coordination.dispatch) - 1] =
'\0';
        strncpy(new_fleet->coordination_type, "dispatch",
sizeof(new_fleet->coordination_type) - 1);
    } else if (strcmp(coordination_type, "reroute") == 0) {
        strncpy(new_fleet->coordination.reroute, coordination_detail,
sizeof(new_fleet->coordination.reroute) - 1);
        new_fleet->coordination.reroute[sizeof(new_fleet->coordination.reroute) - 1] = '\0';
        strncpy(new_fleet->coordination_type, "reroute",
sizeof(new_fleet->coordination_type) - 1);
    } else {
        printf("Invalid coordination type! Use 'dispatch' or 'reroute'.\n");
        return;
    }

    (*count)++;
}

void displayFleets(Fleet *fleets, int count) {
    printf("\nReal-Time Fleet Coordination System\n");
```

```
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Fleet ID: %s\n", fleets[i].fleet_id);
        printf("Location: %s\n", fleets[i].location);
        printf("Status: %s\n", fleets[i].status);
        if (strcmp(fleets[i].coordination_type, "dispatch") == 0) {
            printf("Coordination Type: Dispatch (%s)\n", fleets[i].coordination.dispatch);
        } else if (strcmp(fleets[i].coordination_type, "reroute") == 0) {
            printf("Coordination Type: Reroute (%s)\n", fleets[i].coordination.reroute);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

19. Logistics Security Management System
Description:
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.

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

union EventType {
    char breach[100];
    char resolved[100];
};

typedef struct {
    const char *event_id;
    char description[200];
    union EventType event_type;
    char event_category[10];
} SecurityEvent;
```

```
void addEvent(SecurityEvent **events, int *count, const char *event_id, const char
*description, const char *event_category, const char *event_detail);
void displayEvents(SecurityEvent *events, int count);

int main() {
    SecurityEvent *events = NULL;
    int count = 0;

    addEvent(&events, &count, "EV001", "Unauthorized access detected.", "breach",
"Warehouse door breach.");
    addEvent(&events, &count, "EV002", "Intrusion resolved by security personnel.",
"resolved", "Intrusion neutralized at main gate.");
    addEvent(&events, &count, "EV003", "Package tampering detected.", "breach",
"Container seal broken during transit.");
    addEvent(&events, &count, "EV004", "Tampering resolved by inspection team.",
"resolved", "Package resealed and secured.");

    displayEvents(events,  count);

    free(events);
    return 0;
}

void addEvent(SecurityEvent **events, int *count, const char *event_id, const char
*description, const char *event_category, const char *event_detail) {
    *events = realloc(*events, (*count + 1) * sizeof(SecurityEvent));
    if (*events == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    SecurityEvent *new_event = &(*events)[*count];
    new_event->event_id = event_id;
    strncpy(new_event->description, description, sizeof(new_event->description) - 1);
    new_event->description[sizeof(new_event->description) - 1] = '\0';

    if (strcmp(event_category, "breach") == 0) {
        strncpy(new_event->event_type.breach, event_detail,
sizeof(new_event->event_type.breach) - 1);
        new_event->event_type.breach[sizeof(new_event->event_type.breach) - 1] = '\0';
        strncpy(new_event->event_category,  "breach",  sizeof(new_event->event_category)
```

```
- 1);
    } else if (strcmp(event_category, "resolved") == 0) {
        strncpy(new_event->event_type.resolved, event_detail,
sizeof(new_event->event_type.resolved) - 1);
        new_event->event_type.resolved[sizeof(new_event->event_type.resolved) - 1] =
'\0';
        strncpy(new_event->event_category, "resolved",
sizeof(new_event->event_category) - 1);
    } else {
        printf("Invalid event category! Use 'breach' or 'resolved'.\n");
        return;
    }

    (*count)++;
}

void displayEvents(SecurityEvent *events, int count) {
    printf("\nLogistics Security Management System\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
        printf("Event ID: %s\n", events[i].event_id);
        printf("Description: %s\n", events[i].description);
        if (strcmp(events[i].event_category, "breach") == 0) {
            printf("Event Type: Breach (%s)\n", events[i].event_type.breach);
        } else if (strcmp(events[i].event_category, "resolved") == 0) {
            printf("Event Type: Resolved (%s)\n", events[i].event_type.resolved);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

20. Automated Billing System for Logistics
Description:
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  <stdlib.h>
#include <string.h>

union PaymentMethod {
    char bank_transfer[50];
    char cash[50];
};

typedef struct {
    const char *bill_id;
    float amount;
    char date[20];
    union PaymentMethod payment_method;
    char payment_type[15];
} BillingRecord;

void addBillingRecord(BillingRecord **records, int *count, const char *bill_id, float
amount, const char *date, const char *payment_type, const char *payment_detail);
void displayBillingRecords(BillingRecord *records, int count);

int main() {
    BillingRecord *records = NULL;
    int count = 0;

    addBillingRecord(&records, &count, "BILL001", 1500.50, "2025-01-20",
"bank_transfer", "Transaction ID: 12345ABC");
    addBillingRecord(&records, &count, "BILL002", 200.00, "2025-01-21", "cash", "Paid
in cash by customer.");
    addBillingRecord(&records, &count, "BILL003", 500.75, "2025-01-22",
"bank_transfer", "Transaction ID: 98765XYZ");

    displayBillingRecords(records,  count);

    free(records);
    return 0;
}
```

```
void addBillingRecord(BillingRecord **records, int *count, const char *bill_id, float
amount, const char *date, const char *payment_type, const char *payment_detail) {
    *records = realloc(*records, (*count + 1) * sizeof(BillingRecord));
    if (*records == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    BillingRecord *new_record = &(*records)[*count];
    new_record->bill_id = bill_id;
    new_record->amount = amount;
    strncpy(new_record->date, date, sizeof(new_record->date) - 1);
    new_record->date[sizeof(new_record->date) - 1] = '\0';

    if (strcmp(payment_type, "bank_transfer") == 0) {
        strncpy(new_record->payment_method.bank_transfer, payment_detail,
sizeof(new_record->payment_method.bank_transfer) - 1);

new_record->payment_method.bank_transfer[sizeof(new_record->payment_method.ban
k_transfer) - 1] = '\0';
        strncpy(new_record->payment_type, "bank_transfer",
sizeof(new_record->payment_type) - 1);
    } else if (strcmp(payment_type, "cash") == 0) {
        strncpy(new_record->payment_method.cash, payment_detail,
sizeof(new_record->payment_method.cash) - 1);
        new_record->payment_method.cash[sizeof(new_record->payment_method.cash) -
1] = '\0';
        strncpy(new_record->payment_type, "cash", sizeof(new_record->payment_type) -
1);
    } else {
        printf("Invalid payment type! Use 'bank_transfer' or 'cash'.\n");
        return;
    }

    (*count)++;
}

void displayBillingRecords(BillingRecord *records, int count) {
    printf("\nAutomated Billing System for Logistics\n");
    printf("-------------------------------------------------------------\n");
    for (int i = 0; i < count; i++) {
```

```
        printf("Bill ID: %s\n", records[i].bill_id);
        printf("Amount: %.2f\n", records[i].amount);
        printf("Date: %s\n", records[i].date);
        if (strcmp(records[i].payment_type, "bank_transfer") == 0) {
            printf("Payment Method: Bank Transfer (%s)\n",
records[i].payment_method.bank_transfer);
        } else if (strcmp(records[i].payment_type, "cash") == 0) {
            printf("Payment Method: Cash (%s)\n", records[i].payment_method.cash);
        }
        printf("------------------------------------------------------------\n");
    }
}
```

1. Vessel Navigation System

Description:

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 routes.

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

typedef struct {
    const char *start;
    const char *end;
    const char **waypoints;
    int waypoint_count;
} Route;

void addRoute(Route **route, int *count, const char *start, const char *end, const char
**waypoints, int waypoint_count);
```

```
void display(Route *route, int count);

int main() {
    Route *route = NULL;
    int count = 0;

    const char *waypoints1[] = {"Waypoint A", "Waypoint B", "Waypoint C"};
    addRoute(&route, &count, "Port 1", "Port 2", waypoints1, 3);

    const char *waypoints2[] = {"Waypoint X", "Waypoint Y"};
    addRoute(&route, &count, "Port 3", "Port 4", waypoints2, 2);

    display(route, count);
    free(route);
    return 0;
}

void addRoute(Route **route, int *count, const char *start, const char *end, const char
**waypoints, int waypoint_count) {
    *route = realloc(*route, (*count + 1) * sizeof(Route));
    if (*route == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Route *new = &(*route)[*count];

    new->start = start;
    new->end = end;
    new->waypoints = malloc(waypoint_count * sizeof(const char *));
    if (new->waypoints == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    for (int i = 0; i < waypoint_count; i++) {
        new->waypoints[i] = waypoints[i];
    }

    new->waypoint_count = waypoint_count;
    (*count)++;
```

```
}

void display(Route *route, int count) {
    printf("\nVessel Navigation System\n");
    printf("-----------------------------------------------------\n");

    for (int i = 0; i < count; i++) {
        printf("Route: %d\n", i + 1);
        printf("Start: %s\n", route[i].start);
        printf("End: %s\n", route[i].end);
        printf("Waypoints:\n");

        for (int j = 0; j < route[i].waypoint_count; j++) {
            printf("    - %s\n", route[i].waypoints[j]);
        }
        printf("-----------------------------------------------------\n");
    }
}
```

2. Fleet Management Software
Description:
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:
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>

typedef struct {
    const char *name;
    const char *id;
```

```
    const char *type;
    union {
        int passenger_count;  // Used for passenger vessels
        const char *cargo_type;  // Used for cargo vessels
    };
    int is_passenger_vessel;  // Flag to differentiate between cargo and passenger vessels
} Vessel;

void addVessel(Vessel **fleet, int *count, const char *name, const char *id, const char
*type, int is_passenger_vessel, int passenger_count, const char *cargo_type);
void displayFleet(Vessel *fleet, int count);

int main() {
    Vessel *fleet = NULL;
    int count = 0;

    addVessel(&fleet, &count, "Vessel 1", "V123", "Cargo", 0, 0, "Container");
    addVessel(&fleet, &count, "Vessel 2", "V124", "Passenger", 1, 100, NULL);

    displayFleet(fleet, count);

    free(fleet);

    return 0;
}

void addVessel(Vessel **fleet, int *count, const char *name, const char *id, const char
*type, int is_passenger_vessel, int passenger_count, const char *cargo_type) {
    *fleet = realloc(*fleet, (*count + 1) * sizeof(Vessel));
    if (*fleet == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    Vessel *new_vessel = &(*fleet)[*count];
    new_vessel->name = name;
    new_vessel->id = id;
    new_vessel->type = type;
    new_vessel->is_passenger_vessel = is_passenger_vessel;

    if (is_passenger_vessel) {
```

```
        new_vessel->passenger_count = passenger_count;
    } else {
        new_vessel->cargo_type = cargo_type;
    }

    (*count)++;
}

void displayFleet(Vessel *fleet, int count) {
    printf("\nFleet Management System\n");
    printf("-----------------------------------------------------\n");

    for (int i = 0; i < count; i++) {
        printf("Vessel %d\n", i + 1);
        printf("Name: %s\n", fleet[i].name);
        printf("ID: %s\n", fleet[i].id);
        printf("Type: %s\n", fleet[i].type);

        if (fleet[i].is_passenger_vessel) {
            printf("Passenger Count: %d\n", fleet[i].passenger_count);
        } else {
            printf("Cargo Type: %s\n", fleet[i].cargo_type);
        }
        printf("-----------------------------------------------------\n");
    }
}
```

3. Ship Maintenance Scheduler

Description:

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>

typedef struct {
    const char *task_id;
    const char *description;
    const char *schedule;
} MaintenanceTask;

void addTask(MaintenanceTask **taskList, int *taskCount, const char *task_id, const
char *description, const char *schedule);
void displayTasks(MaintenanceTask *taskList, int taskCount);

int main() {
    MaintenanceTask *taskList = NULL;
    int taskCount = 0;

    addTask(&taskList, &taskCount, "MT001", "Engine check-up", "2025-02-01");
    addTask(&taskList, &taskCount, "MT002", "Hull cleaning", "2025-02-15");
    addTask(&taskList, &taskCount, "MT003", "Radar system calibration",
"2025-03-01");

    displayTasks(taskList, taskCount);

    free(taskList);

    return 0;
}

void addTask(MaintenanceTask **taskList, int *taskCount, const char *task_id, const
char *description, const char *schedule) {
    *taskList = realloc(*taskList, (*taskCount + 1) * sizeof(MaintenanceTask));
    if (*taskList == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    MaintenanceTask *newTask = &(*taskList)[*taskCount];
    newTask->task_id = task_id;
    newTask->description = description;
    newTask->schedule = schedule;
```

```
    (*taskCount)++;
}

void displayTasks(MaintenanceTask *taskList, int taskCount) {
    printf("\nShip Maintenance Scheduler\n");
    printf("------------------------------------------------\n");

    for (int i = 0; i < taskCount; i++) {
        printf("Task %d\n", i + 1);
        printf("Task ID: %s\n", taskList[i].task_id);
        printf("Description: %s\n", taskList[i].description);
        printf("Schedule: %s\n", taskList[i].schedule);
        printf("------------------------------------------------\n");
    }
}
```

4. Cargo Loading Optimization
Description:
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>

typedef struct {
    const char *vessel_name;
    int capacity;
    const char *dimensions;
} Vessel;
```

```
typedef union {
    float weight;  // Weight of the cargo (in tons)
    const char *dimensions;  // Dimensions of the cargo (length x width x height)
} CargoProperties;

typedef struct {
    const char *cargo_id;
    CargoProperties properties;
    int is_weight_based;  // 1 for weight-based cargo, 0 for dimension-based cargo
} Cargo;

void addCargo(Cargo **cargoList, int *cargoCount, const char *cargo_id,
CargoProperties properties, int is_weight_based);
void displayCargo(Cargo *cargoList, int cargoCount);
void optimizeLoading(Vessel *vessel, Cargo *cargoList, int cargoCount);

int main() {
    Vessel vessel = {"Vessel A", 100, "50x30x15"};

    Cargo *cargoList = NULL;
    int cargoCount = 0;
    CargoProperties cargo1 = {.weight = 10.5};  // 10.5 tons of weight
    addCargo(&cargoList, &cargoCount, "Cargo1", cargo1, 1);  // Weight-based cargo

    CargoProperties cargo2 = {.dimensions = "10x5x3"};  // Dimensions of the cargo
    addCargo(&cargoList, &cargoCount, "Cargo2", cargo2, 0);  // Dimension-based cargo

    displayCargo(cargoList, cargoCount);

    optimizeLoading(&vessel, cargoList, cargoCount);

    free(cargoList);

    return 0;
}

void addCargo(Cargo **cargoList, int *cargoCount, const char *cargo_id,
CargoProperties properties, int is_weight_based) {
    *cargoList = realloc(*cargoList, (*cargoCount + 1) * sizeof(Cargo));
    if (*cargoList == NULL) {
        printf("Memory allocation failed\n");
```

```
        exit(1);
    }

    Cargo *newCargo = &(*cargoList)[*cargoCount];
    newCargo->cargo_id = cargo_id;
    newCargo->properties = properties;
    newCargo->is_weight_based = is_weight_based;

    (*cargoCount)++;
}

void displayCargo(Cargo *cargoList, int cargoCount) {
    printf("\nCargo List\n");
    printf("----------------------------------------------------\n");

    for (int i = 0; i < cargoCount; i++) {
        printf("Cargo ID: %s\n", cargoList[i].cargo_id);
        if (cargoList[i].is_weight_based) {
            printf("Cargo Type: Weight-based\n");
            printf("Weight: %.2f tons\n", cargoList[i].properties.weight);
        } else {
            printf("Cargo Type: Dimension-based\n");
            printf("Dimensions: %s\n", cargoList[i].properties.dimensions);
        }
        printf("----------------------------------------------------\n");
    }
}

void optimizeLoading(Vessel *vessel, Cargo *cargoList, int cargoCount) {
    float totalWeight = 0;

    printf("\nOptimizing Cargo Loading for %s\n", vessel->vessel_name);
    printf("Vessel Capacity: %d tons\n", vessel->capacity);

    for (int i = 0; i < cargoCount; i++) {
        if (cargoList[i].is_weight_based) {
            totalWeight += cargoList[i].properties.weight;
        }
    }

    if (totalWeight > vessel->capacity) {
```

```
        printf("Warning: Overloaded! Total weight: %.2f tons\n", totalWeight);
    } else {
        printf("Cargo loading successful! Total weight: %.2f tons\n", totalWeight);
    }
}
```

5. Real-Time Weather Alert System
Description:
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>

#define MAX_ALERT_MESSAGE_LENGTH  100

typedef struct {
    float temperature;  // Temperature in Celsius
    float wind_speed;    // Wind speed in km/h
} WeatherData;

void generateAlert(WeatherData *weather, char *alertMessage);
void displayHistoricalRecords(WeatherData **records, int recordCount);
void addWeatherRecord(WeatherData ***records, int *recordCount, WeatherData
newRecord);

int main() {
    WeatherData **historicalRecords = NULL;
    int recordCount = 0;

    WeatherData record1 = {30.5, 15.0};  // Temperature: 30.5°C, Wind Speed: 15 km/h
    WeatherData record2 = {25.0, 20.0};  // Temperature: 25.0°C, Wind Speed: 20 km/h
    WeatherData record3 = {35.0, 10.0};  // Temperature: 35.0°C, Wind Speed: 10 km/h
```

```
    // Add weather records to historical data
    addWeatherRecord(&historicalRecords, &recordCount, record1);
    addWeatherRecord(&historicalRecords, &recordCount, record2);
    addWeatherRecord(&historicalRecords, &recordCount, record3);

    char alertMessage[MAX_ALERT_MESSAGE_LENGTH];
    generateAlert(&record3, alertMessage);

    printf("Weather Alert: %s\n", alertMessage);

    displayHistoricalRecords(historicalRecords, recordCount);

    for (int i = 0; i < recordCount; i++) {
        free(historicalRecords[i]);
    }
    free(historicalRecords);

    return 0;
}

void generateAlert(WeatherData *weather, char *alertMessage) {
    if (weather->temperature > 30.0 && weather->wind_speed > 20.0) {
        snprintf(alertMessage, MAX_ALERT_MESSAGE_LENGTH, "Severe Weather
Alert: High Temperature (%.2f C) and Strong Winds (%.2f km/h)",
weather->temperature, weather->wind_speed);
    } else if (weather->temperature > 30.0) {
        snprintf(alertMessage, MAX_ALERT_MESSAGE_LENGTH, "Weather Alert:
High Temperature (%.2f C)", weather->temperature);
    } else if (weather->wind_speed > 20.0) {
        snprintf(alertMessage, MAX_ALERT_MESSAGE_LENGTH, "Weather Alert:
Strong Winds (%.2f km/h)", weather->wind_speed);
    } else {
        snprintf(alertMessage, MAX_ALERT_MESSAGE_LENGTH, "Weather is calm
(Temperature: %.2f C, Winds: %.2f km/h)", weather->temperature,
weather->wind_speed);
    }
}

void displayHistoricalRecords(WeatherData **records, int recordCount) {
    printf("\nHistorical Weather Records:\n");
```

```
    printf("-------------------------------------------------\n");

    for (int i = 0; i < recordCount; i++) {
        printf("Record %d:\n", i + 1);
        printf("Temperature: %.2f C\n", records[i]->temperature);
        printf("Wind Speed: %.2f km/h\n", records[i]->wind_speed);
        printf("-------------------------------------------------\n");
    }
}

void addWeatherRecord(WeatherData ***records, int *recordCount, WeatherData
newRecord) {
    *records = realloc(*records, (*recordCount + 1) * sizeof(WeatherData *));
    if (*records == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    (*records)[*recordCount] = malloc(sizeof(WeatherData));
    if ((*records)[*recordCount] == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    (*records)[*recordCount]->temperature  =  newRecord.temperature;
    (*records)[*recordCount]->wind_speed  =  newRecord.wind_speed;
    (*recordCount)++;
}
```

6. Nautical Chart Management

Description:

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.

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

typedef struct {
    double latitude;
    double longitude;
} Coordinate;

typedef struct {
} DepthOrHazard;

typedef struct {
    char chartID[10];
    char scale[20];
    char region[30];
    Coordinate *coordinates;
    DepthOrHazard *depthOrHazardData;
    int coordinateCount;
} NauticalChart;

void addChart(NauticalChart **charts, int *chartCount, const char *chartID, const char
*scale, const char *region, Coordinate *coordinates, DepthOrHazard
*depthOrHazardData, int coordinateCount);

int main() {
    int chartCount = 0;
    NauticalChart *charts = NULL;  // Pointer to pointer, to store dynamically allocated
charts

    Coordinate coordinates1[3] = { {1.0, 2.0}, {3.0, 4.0}, {5.0, 6.0} };
    DepthOrHazard depthData1[3];  // Assume populated with appropriate data

    Coordinate coordinates2[3] = { {7.0, 8.0}, {9.0, 10.0}, {11.0, 12.0} };
    DepthOrHazard hazardData2[3];  // Assume populated with appropriate data

    addChart(&charts, &chartCount, "CHART001", "1:100000", "North Sea",
coordinates1, depthData1, 3);
    addChart(&charts, &chartCount, "CHART002", "1:50000", "Bay of Bengal",
coordinates2, hazardData2, 3);

    printf("Chart ID: %s\n", charts[0].chartID);
```

```
    for (int i = 0; i < chartCount; i++) {
        free(charts[i].coordinates);
        free(charts[i].depthOrHazardData);
    }
    free(charts);

    return 0;
}

void addChart(NauticalChart **charts, int *chartCount, const char *chartID, const char
*scale, const char *region, Coordinate *coordinates, DepthOrHazard
*depthOrHazardData, int coordinateCount) {
    *charts = realloc(*charts, (*chartCount + 1) * sizeof(NauticalChart));

    NauticalChart *newChart = &(*charts)[*chartCount];
    snprintf(newChart->chartID, sizeof(newChart->chartID), "%s", chartID);
    snprintf(newChart->scale, sizeof(newChart->scale), "%s", scale);
    snprintf(newChart->region, sizeof(newChart->region), "%s", region);
    newChart->coordinates = malloc(coordinateCount * sizeof(Coordinate));
    newChart->depthOrHazardData = malloc(coordinateCount * sizeof(DepthOrHazard));
    for (int i = 0; i < coordinateCount; i++) {
        newChart->coordinates[i] = coordinates[i];
        newChart->depthOrHazardData[i] = depthOrHazardData[i];
    }
    newChart->coordinateCount = coordinateCount;

    (*chartCount)++;
}
```

7. Crew Roster Management
Description:
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>

#define MAX_NAME_LEN 100
#define MAX_SCHEDULE_LEN 50
#define MAX_ROLES 10

typedef struct {
    char name[MAX_NAME_LEN];
    const char *role;
    char schedule[MAX_SCHEDULE_LEN];
} Crew;

Crew **allocateRoster(int numCrew) {
    Crew **roster = (Crew **)malloc(numCrew * sizeof(Crew *));
    if (roster == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    return roster;
}

void freeRoster(Crew **roster, int numCrew) {
    for (int i = 0; i < numCrew; i++) {
        free(roster[i]);
    }
    free(roster);
}

void addCrewMember(Crew **roster, int index, const char *name, const char *role,
const char *schedule) {
    roster[index] = (Crew *)malloc(sizeof(Crew));
    if (roster[index] == NULL) {
        printf("Memory allocation failed for crew member %d!\n", index);
        exit(1);
    }
    strncpy(roster[index]->name, name, MAX_NAME_LEN);
    roster[index]->role = role;
```

```
    strncpy(roster[index]->schedule, schedule, MAX_SCHEDULE_LEN);
}

void displayRoster(Crew **roster, int numCrew) {
    for (int i = 0; i < numCrew; i++) {
        printf("Crew Member %d\n", i + 1);
        printf("Name: %s\n", roster[i]->name);
        printf("Role: %s\n", roster[i]->role);
        printf("Schedule: %s\n\n", roster[i]->schedule);
    }
}

int main() {
    int numCrew = 3;
    const char *roles[] = {
        "Captain",
        "First Mate",
        "Engineer"
    };

    Crew **roster = allocateRoster(numCrew);

    addCrewMember(roster, 0, "John Doe", roles[0], "Mon-Fri, 9am-5pm");
    addCrewMember(roster, 1, "Alice Smith", roles[1], "Mon-Fri, 10am-6pm");
    addCrewMember(roster, 2, "Bob Johnson", roles[2], "Mon-Sat, 8am-4pm");

    displayRoster(roster, numCrew);

    freeRoster(roster, numCrew);

    return 0;
}
```

8. Underwater Sensor Monitoring
Description:
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>

#define MAX_LOCATION_LEN 50
#define MAX_SENSOR_ID_LEN 20
#define MAX_SENSOR_READINGS 10

typedef union {
    float temperature;
    float pressure;
} SensorData;

typedef struct {
    char id[MAX_SENSOR_ID_LEN];
    char location[MAX_LOCATION_LEN];
    SensorData data;
} Sensor;

int main() {
    int numSensors = 3;
    Sensor **sensors = allocateSensors(numSensors);

    addSensor(sensors, 0, "SENSOR001", "North Pole", 25.5f, 1);
    addSensor(sensors, 1, "SENSOR002", "Mid-Atlantic", 1200.0f, 0);
    addSensor(sensors, 2, "SENSOR003", "Indian Ocean", 27.3f, 1);

    displaySensors(sensors, numSensors);

    freeSensors(sensors, numSensors);

    return 0;
}

Sensor **allocateSensors(int numSensors) {
```

```
    Sensor **sensors = (Sensor **)malloc(numSensors * sizeof(Sensor *));
    if (sensors == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    return sensors;
}

void freeSensors(Sensor **sensors, int numSensors) {
    free(sensors);
}

void addSensor(Sensor **sensors, int index, const char *id, const char *location, float
data, int isTemperature) {
    sensors[index] = (Sensor *)malloc(sizeof(Sensor));
    if (sensors[index] == NULL) {
        printf("Memory allocation failed for sensor %d!\n", index);
        exit(1);
    }
    strncpy(sensors[index]->id, id, MAX_SENSOR_ID_LEN);
    strncpy(sensors[index]->location, location, MAX_LOCATION_LEN);
    if (isTemperature) {
        sensors[index]->data.temperature = data;
    } else {
        sensors[index]->data.pressure = data;
    }
}

void displaySensors(Sensor **sensors, int numSensors) {
    for (int i = 0; i < numSensors; i++) {
        printf("Sensor %d\n", i + 1);
        printf("ID: %s\n", sensors[i]->id);
        printf("Location: %s\n", sensors[i]->location);
        if (sensors[i]->data.temperature != 0.0f) {
            printf("Temperature: %.2f\n", sensors[i]->data.temperature);
        } else {
            printf("Pressure: %.2f\n", sensors[i]->data.pressure);
        }
        printf("\n");
    }
}
```

9. Ship Log Management
Description:
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>

#define MAX_LOG_ENTRY_LEN 100
#define MAX_AUTHOR_LEN 50
#define MAX_DATE_LEN 20

typedef struct {
    char date[MAX_DATE_LEN];
    char author[MAX_AUTHOR_LEN];
} LogMetadata;

typedef struct {
    LogMetadata metadata;
    char *entry;
} LogRecord;

LogRecord **allocateLogs(int numLogs);
void freeLogs(LogRecord **logs, int numLogs);
void addLogEntry(LogRecord **logs, int index, const char *date, const char *author,
const char *entry);
void displayLogs(LogRecord **logs, int numLogs);

int main() {
    int numLogs = 3;
    LogRecord **logs = allocateLogs(numLogs);
```

```
    addLogEntry(logs, 0, "2025-01-22", "Captain", "Ship departed from port.");
    addLogEntry(logs, 1, "2025-01-23", "First Officer", "Sea conditions calm.");
    addLogEntry(logs, 2, "2025-01-24", "Captain", "Approaching destination.");

    displayLogs(logs, numLogs);

    freeLogs(logs, numLogs);

    return 0;
}

LogRecord **allocateLogs(int numLogs) {
    LogRecord **logs = (LogRecord **)malloc(numLogs * sizeof(LogRecord *));
    if (logs == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    return logs;
}

void freeLogs(LogRecord **logs, int numLogs) {
    for (int i = 0; i < numLogs; i++) {
        free(logs[i]->entry);
        free(logs[i]);
    }
    free(logs);
}

void addLogEntry(LogRecord **logs, int index, const char *date, const char *author,
const char *entry) {
    logs[index] = (LogRecord *)malloc(sizeof(LogRecord));
    if (logs[index] == NULL) {
        printf("Memory allocation failed for log entry %d!\n", index);
        exit(1);
    }
    strncpy(logs[index]->metadata.date, date, MAX_DATE_LEN);
    strncpy(logs[index]->metadata.author, author, MAX_AUTHOR_LEN);
    logs[index]->entry = (char *)malloc(strlen(entry) + 1);
    if (logs[index]->entry == NULL) {
        printf("Memory allocation failed for log entry text!\n");
        exit(1);
```

```
    }
    strcpy(logs[index]->entry, entry);
}

void displayLogs(LogRecord **logs, int numLogs) {
    for (int i = 0; i < numLogs; i++) {
        printf("Log %d\n", i + 1);
        printf("Date: %s\n", logs[i]->metadata.date);
        printf("Author: %s\n", logs[i]->metadata.author);
        printf("Entry: %s\n", logs[i]->entry);
        printf("\n");
    }
}
```

10. Navigation Waypoint Manager

Description:

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>

#define MAX_WAYPOINT_NAME_LEN 50

typedef struct {
    char id[MAX_WAYPOINT_NAME_LEN];
    double latitude;
    double longitude;
} Waypoint;

typedef struct {
    char id[MAX_WAYPOINT_NAME_LEN];
```

```
    Waypoint *waypoints;
    int waypointCount;
} NavigationDetails;

NavigationDetails **allocateWaypoints(int numWaypoints);
void freeWaypoints(NavigationDetails **waypoints, int numWaypoints);
void addWaypoint(NavigationDetails **waypoints, int index, const char *id, const char
*waypointID, double latitude, double longitude);
void displayWaypoints(NavigationDetails **waypoints, int numWaypoints);

int main() {
    int numWaypoints = 2;
    NavigationDetails **waypoints = allocateWaypoints(numWaypoints);

    addWaypoint(waypoints, 0, "NAV001", "WP001", 12.345, 98.765);
    addWaypoint(waypoints, 1, "NAV002", "WP002", 23.456, 87.654);

    displayWaypoints(waypoints, numWaypoints);

    freeWaypoints(waypoints, numWaypoints);

    return 0;
}

NavigationDetails **allocateWaypoints(int numWaypoints) {
    NavigationDetails **waypoints = (NavigationDetails **)malloc(numWaypoints *
sizeof(NavigationDetails *));
    if (waypoints == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    return waypoints;
}

void freeWaypoints(NavigationDetails **waypoints, int numWaypoints) {
    for (int i = 0; i < numWaypoints; i++) {
        free(waypoints[i]->waypoints);
        free(waypoints[i]);
    }
    free(waypoints);
}
```

```
void addWaypoint(NavigationDetails **waypoints, int index, const char *id, const char *waypointID, double latitude, double longitude) {
    waypoints[index] = (NavigationDetails *)malloc(sizeof(NavigationDetails));
    if (waypoints[index] == NULL) {
        printf("Memory allocation failed for navigation details %d!\n", index);
        exit(1);
    }
    strncpy(waypoints[index]->id, id, MAX_WAYPOINT_NAME_LEN);

    waypoints[index]->waypoints = (Waypoint *)malloc(sizeof(Waypoint));
    if (waypoints[index]->waypoints == NULL) {
        printf("Memory allocation failed for waypoints!\n");
        exit(1);
    }
    waypoints[index]->waypointCount = 1;

    strncpy(waypoints[index]->waypoints[0].id, waypointID, MAX_WAYPOINT_NAME_LEN);
    waypoints[index]->waypoints[0].latitude = latitude;
    waypoints[index]->waypoints[0].longitude = longitude;
}

void displayWaypoints(NavigationDetails **waypoints, int numWaypoints) {
    for (int i = 0; i < numWaypoints; i++) {
        printf("Navigation ID: %s\n", waypoints[i]->id);
        for (int j = 0; j < waypoints[i]->waypointCount; j++) {
            printf("Waypoint ID: %s\n", waypoints[i]->waypoints[j].id);
            printf("Latitude: %.6f\n", waypoints[i]->waypoints[j].latitude);
            printf("Longitude: %.6f\n", waypoints[i]->waypoints[j].longitude);
        }
        printf("\n");
    }
}
```

11. Marine Wildlife Tracking

Description:

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>

#define MAX_SPECIES_NAME_LEN 50

typedef struct {
    char species[MAX_SPECIES_NAME_LEN];
    char id[MAX_SPECIES_NAME_LEN];
    double latitude;
    double longitude;
} AnimalData;

typedef struct {
    AnimalData *animals;
    int animalCount;
} ObservationRecord;

ObservationRecord **allocateObservations(int numRecords);
void freeObservations(ObservationRecord **records, int numRecords);
void addAnimalObservation(ObservationRecord **records, int recordIndex, const char
*species, const char *id, double latitude, double longitude);
void displayAnimalObservations(ObservationRecord **records, int numRecords);

int main() {
    int numRecords = 2;
    ObservationRecord **records = allocateObservations(numRecords);

    addAnimalObservation(records, 0, "Dolphin", "ANIMAL001", 23.456, 76.543);
    addAnimalObservation(records, 1, "Whale", "ANIMAL002", 34.567, 65.432);

    displayAnimalObservations(records, numRecords);

    freeObservations(records,  numRecords);
```

```
    return 0;
}

ObservationRecord **allocateObservations(int numRecords) {
    ObservationRecord **records = (ObservationRecord **)malloc(numRecords *
sizeof(ObservationRecord *));
    if (records == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    return records;
}

void freeObservations(ObservationRecord **records, int numRecords) {
    for (int i = 0; i < numRecords; i++) {
        free(records[i]->animals);
        free(records[i]);
    }
    free(records);
}

void addAnimalObservation(ObservationRecord **records, int recordIndex, const char
*species, const char *id, double latitude, double longitude) {
    records[recordIndex] = (ObservationRecord *)malloc(sizeof(ObservationRecord));
    if (records[recordIndex] == NULL) {
        printf("Memory allocation failed for observation record %d!\n", recordIndex);
        exit(1);
    }

    records[recordIndex]->animals = (AnimalData *)malloc(sizeof(AnimalData));
    if (records[recordIndex]->animals == NULL) {
        printf("Memory allocation failed for animals!\n");
        exit(1);
    }
    records[recordIndex]->animalCount = 1;

    strncpy(records[recordIndex]->animals[0].species, species,
MAX_SPECIES_NAME_LEN);
    strncpy(records[recordIndex]->animals[0].id, id, MAX_SPECIES_NAME_LEN);
    records[recordIndex]->animals[0].latitude = latitude;
```

```
    records[recordIndex]->animals[0].longitude = longitude;
}

void displayAnimalObservations(ObservationRecord **records, int numRecords) {
    for (int i = 0; i < numRecords; i++) {
        printf("Observation Record %d:\n", i + 1);
        for (int j = 0; j < records[i]->animalCount; j++) {
            printf("Species: %s\n", records[i]->animals[j].species);
            printf("ID: %s\n", records[i]->animals[j].id);
            printf("Latitude: %.6f\n", records[i]->animals[j].latitude);
            printf("Longitude: %.6f\n", records[i]->animals[j].longitude);
        }
        printf("\n");
    }
}
```

12. Coastal Navigation Beacon Management

Description:

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).

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];
    int signalStrength;
};

typedef struct {
    char id[50];
    char type[50];
    double latitude;
```

```
    double longitude;
    union BeaconType beacon;
} BeaconMetadata;

void addBeacon(BeaconMetadata **beacons, int *count, const char *id, const char *type,
double latitude, double longitude, const char *lightColor, int signalStrength) {
    *beacons = realloc(*beacons, (*count + 1) * sizeof(BeaconMetadata));
    if (*beacons == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    BeaconMetadata *newBeacon = &(*beacons)[*count];
    strncpy(newBeacon->id, id, sizeof(newBeacon->id) - 1);
    strncpy(newBeacon->type, type, sizeof(newBeacon->type) - 1);
    newBeacon->latitude = latitude;
    newBeacon->longitude = longitude;

    if (lightColor != NULL) {
        strncpy(newBeacon->beacon.lightColor, lightColor,
sizeof(newBeacon->beacon.lightColor) - 1);
    } else {
        newBeacon->beacon.signalStrength = signalStrength;
    }

    (*count)++;
}

void displayBeacons(BeaconMetadata *beacons, int count) {
    for (int i = 0; i < count; i++) {
        printf("Beacon ID: %s\n", beacons[i].id);
        printf("Type: %s\n", beacons[i].type);
        printf("Location: %.6f, %.6f\n", beacons[i].latitude, beacons[i].longitude);

        if (strlen(beacons[i].beacon.lightColor) > 0) {
            printf("Light Color: %s\n", beacons[i].beacon.lightColor);
        } else {
            printf("Signal Strength: %d\n", beacons[i].beacon.signalStrength);
        }
        printf("\n");
    }
```

```
}

int main() {
    BeaconMetadata *beacons = NULL;
    int beaconCount = 0;

    addBeacon(&beacons, &beaconCount, "BEACON001", "Light", 25.1234, 78.4567,
"Red", 0);
    addBeacon(&beacons, &beaconCount, "BEACON002", "Radar", 35.6789, 89.1234,
NULL, 85);

    displayBeacons(beacons, beaconCount);

    free(beacons);

    return 0;
}
```

13. Fuel Usage Tracking

Description:

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>

typedef struct {
    char fuelType[50];
    double quantity;
} FuelData;
```

```
typedef struct {
    char date[20];
    FuelData fuelConsumed;
} FuelLog;

void addFuelLog(FuelLog **logs, int *count, const char *date, const char *fuelType,
double quantity) {
    *logs = realloc(*logs, (*count + 1) * sizeof(FuelLog));
    if (*logs == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    FuelLog *newLog = &(*logs)[*count];
    strncpy(newLog->date, date, sizeof(newLog->date) - 1);
    strncpy(newLog->fuelConsumed.fuelType, fuelType,
sizeof(newLog->fuelConsumed.fuelType) - 1);
    newLog->fuelConsumed.quantity = quantity;

    (*count)++;
}

void displayFuelLogs(FuelLog *logs, int count) {
    for (int i = 0; i < count; i++) {
        printf("Date: %s\n", logs[i].date);
        printf("Fuel Type: %s\n", logs[i].fuelConsumed.fuelType);
        printf("Fuel Consumed: %.2f liters\n", logs[i].fuelConsumed.quantity);
        printf("\n");
    }
}

int main() {
    FuelLog *logs = NULL;
    int logCount = 0;

    addFuelLog(&logs, &logCount, "2025-01-01", "Diesel", 500.0);
    addFuelLog(&logs, &logCount, "2025-01-02", "Petrol", 300.0);
    addFuelLog(&logs, &logCount, "2025-01-03", "Diesel", 450.0);

    displayFuelLogs(logs, logCount);
```

```
    free(logs);

    return 0;
}
```

14. Emergency Response System
Description:
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.
Double Pointers: Dynamic alert allocation.

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

typedef struct{
    char id[50];
    char location[100];
    char type[50];
}ResponseDetails;

typedef struct{
    char message[225];
    ResponseDetails response;
}Alert;

void add(Alert **alert,int *count,const char *message,const char *id,const char *location,const char *type);
void display(Alert *alert,int count);

int main(){
    Alert *alert = NULL;
```

```
    int count =0;
    add(&alert, &count, "Fire breakout in engine room.", "EMR001", "Engine Room",
"Fire");
    add(&alert, &count, "Medical emergency in crew quarters.", "EMR002", "Crew
Quarters", "Medical");
    add(&alert, &count, "Oil spill detected near starboard side.", "EMR003", "Starboard
Side", "Oil Spill");

    display(alert,count);

    free(alert);
    return 0;
}

void add(Alert **alert,int *count,const char *message,const char *id,const char
*location,const char *type){
    *alert = realloc(*alert,(*count+1)*sizeof(Alert));
    if(*alert ==NULL){
        printf("Memory allocation failed \n");
        exit(1);
    }
    Alert *new = &(*alert)[*count];

    strncpy(new->message,message,sizeof(new->message)-1);
    strncpy(new->response.id,id,sizeof(new->response.id)-1);
    strncpy(new->response.location,location,sizeof(new->response.location)-1);
    strncpy(new->response.type,type,sizeof(new->response.type)-1);

    (*count)++;
}

void display(Alert *alert,int count){
    printf("\nEmergency Response System\n");
     printf("------------------------------------------------------------\n");
    for(int i=0;i<count;i++){

        printf("Alert message : %s\n",alert[i].message);
        printf("Response Id: %s\n",alert[i].response.id);
        printf("Location: %s\n",alert[i].response.location);
        printf("Response Type: %s\n",alert[i].response.type);
        printf("------------------------------------------------------------\n");
```

```
    }
}
```

15. Ship Performance Analysis
Description:
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>
#include  <stdlib.h>
#include <string.h>

union Weather {
    float windSpeed;
    float waveHeight;
};

typedef struct {
    char shipID[50];
    float speed;
    float capacity;
    union Weather weatherImpact;
    float performanceMetric;
} ShipRecord;

void addPerformanceRecord(ShipRecord **records, int *count, const char *shipID, float
speed, float capacity, float weatherImpact, int isWind) {
    *records = realloc(*records, (*count + 1) * sizeof(ShipRecord));
    if (*records == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
```

```
    ShipRecord *newRecord = &(*records)[*count];
    strncpy(newRecord->shipID, shipID, sizeof(newRecord->shipID) - 1);
    newRecord->speed = speed;
    newRecord->capacity = capacity;

    if (isWind) {
        newRecord->weatherImpact.windSpeed = weatherImpact;
    } else {
        newRecord->weatherImpact.waveHeight = weatherImpact;
    }

    newRecord->performanceMetric = (speed / 20.0) * 100;  // Assuming maximum speed
is 20 knots for full performance

    (*count)++;
}

void displayPerformanceRecords(ShipRecord *records, int count) {
    for (int i = 0; i < count; i++) {
        printf("Ship ID: %s\n", records[i].shipID);
        printf("Speed: %.2f knots\n", records[i].speed);
        printf("Capacity: %.2f tons\n", records[i].capacity);

        printf("Weather Impact: ");
        if (records[i].weatherImpact.windSpeed > 0) {
            printf("Wind Speed: %.2f knots\n", records[i].weatherImpact.windSpeed);
        } else {
            printf("Wave Height: %.2f meters\n", records[i].weatherImpact.waveHeight);
        }

        printf("Performance Metric: %.2f%%\n\n", records[i].performanceMetric);
    }
}

int main() {
    ShipRecord *records = NULL;
    int recordCount = 0;

    addPerformanceRecord(&records, &recordCount, "SHIP001", 18.0, 5000.0, 15.0, 1);
// Wind impact
    addPerformanceRecord(&records, &recordCount, "SHIP002", 16.0, 4500.0, 2.5, 0);
```

```
// Wave impact
    addPerformanceRecord(&records, &recordCount, "SHIP003", 20.0, 6000.0, 12.0, 1);
// Wind impact

    displayPerformanceRecords(records, recordCount);

    free(records);

    return 0;
}
```

16. Port Docking Scheduler

Description:

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>

typedef struct {
    char portID[50];
    int capacity;
    char location[100];
} PortDetails;

typedef struct {
    char vesselName[50];
    char dockingTime[20];
    char departureTime[20];
} DockingSchedule;
```

```
void addDockingSchedule(DockingSchedule **schedule, int *count, const char
*vesselName, const char *dockingTime, const char *departureTime) {
    *schedule = realloc(*schedule, (*count + 1) * sizeof(DockingSchedule));
    if (*schedule == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    DockingSchedule *newSchedule = &(*schedule)[*count];
    strncpy(newSchedule->vesselName, vesselName, sizeof(newSchedule->vesselName) -
1);
    strncpy(newSchedule->dockingTime, dockingTime,
sizeof(newSchedule->dockingTime) - 1);
    strncpy(newSchedule->departureTime, departureTime,
sizeof(newSchedule->departureTime) - 1);

    (*count)++;
}

void displayDockingSchedules(DockingSchedule *schedule, int count) {
    for (int i = 0; i < count; i++) {
        printf("Vessel: %s\n", schedule[i].vesselName);
        printf("Docking Time: %s\n", schedule[i].dockingTime);
        printf("Departure Time: %s\n\n", schedule[i].departureTime);
    }
}

int main() {
    DockingSchedule *schedule = NULL;
    int scheduleCount = 0;

    addDockingSchedule(&schedule, &scheduleCount, "Vessel A", "2025-01-22 10:00",
"2025-01-22 18:00");
    addDockingSchedule(&schedule, &scheduleCount, "Vessel B", "2025-01-23 12:00",
"2025-01-23 20:00");

    displayDockingSchedules(schedule, scheduleCount);

    free(schedule);

    return 0;
```

}

17. Deep-Sea Exploration Data Logger
Description:
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>

typedef struct {
    double depth;
    char location[100];
    char timestamp[20];
} ExplorationData;

void addExplorationLog(ExplorationData **logs, int *count, double depth, const char *location, const char *timestamp) {
    *logs = realloc(*logs, (*count + 1) * sizeof(ExplorationData));
    if (*logs == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    ExplorationData *newLog = &(*logs)[*count];
    newLog->depth = depth;
    strncpy(newLog->location, location, sizeof(newLog->location) - 1);
    strncpy(newLog->timestamp, timestamp, sizeof(newLog->timestamp) - 1);

    (*count)++;
}

void displayExplorationLogs(ExplorationData *logs, int count) {
    for (int i = 0; i < count; i++) {
```

```
        printf("Depth: %.2f meters\n", logs[i].depth);
        printf("Location: %s\n", logs[i].location);
        printf("Timestamp: %s\n\n", logs[i].timestamp);
    }
}

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

    addExplorationLog(&logs, &logCount, 2000.5, "Oceanic Trench", "2025-01-22
10:00");
    addExplorationLog(&logs, &logCount, 1500.3, "Abyssal Plain", "2025-01-22 12:00");

    displayExplorationLogs(logs, logCount);

    free(logs);

    return 0;
}
```

18. Ship Communication System

Description:

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>

typedef struct {
    char id[50];
    char timestamp[20];
} CommunicationMetadata;
```

```
typedef struct {
    CommunicationMetadata metadata;
    char message[200];
} MessageLog;

void addMessageLog(MessageLog **logs, int *count, const char *id, const char
*timestamp, const char *message) {
    *logs = realloc(*logs, (*count + 1) * sizeof(MessageLog));
    if (*logs == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    MessageLog *newLog = &(*logs)[*count];
    strncpy(newLog->metadata.id, id, sizeof(newLog->metadata.id) - 1);
    strncpy(newLog->metadata.timestamp, timestamp,
sizeof(newLog->metadata.timestamp) - 1);
    strncpy(newLog->message, message, sizeof(newLog->message) - 1);

    (*count)++;
}

void displayMessageLogs(MessageLog *logs, int count) {
    for (int i = 0; i < count; i++) {
        printf("ID: %s\n", logs[i].metadata.id);
        printf("Timestamp: %s\n", logs[i].metadata.timestamp);
        printf("Message: %s\n\n", logs[i].message);
    }
}

int main() {
    MessageLog *logs = NULL;
    int logCount = 0;

    addMessageLog(&logs, &logCount, "COM001", "2025-01-22 10:00", "All systems
are operational.");
    addMessageLog(&logs, &logCount, "COM002", "2025-01-22 12:00", "Position report
sent.");

    displayMessageLogs(logs, logCount);
```

```
    free(logs);

    return 0;
}
```

19. Fishing Activity Tracker

Description:

Design a system to track fishing activities using arrays for catch records, structures for vessel details, and unions for variable catch data like species or weight.

Specifications:

Structure: Vessel details (ID, name).

Union: Catch data (species, weight).

Array: Catch records.

const Pointers: Protect vessel IDs.

Double Pointers: Dynamic catch management.

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

typedef struct {
    char vesselID[50];
    char vesselName[50];
} VesselDetails;

typedef union {
    char species[50];
    float weight;
} CatchData;

typedef struct {
    VesselDetails vessel;
    CatchData catchInfo;
    int isSpecies; // 0 for weight, 1 for species
} CatchRecord;

void addCatchRecord(CatchRecord **records, int *count, const char *vesselID, const
char *vesselName, const char *catchInfo, int isSpecies) {
    *records = realloc(*records, (*count + 1) * sizeof(CatchRecord));
```

```
    if (*records == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    CatchRecord *newRecord = &(*records)[*count];
    strncpy(newRecord->vessel.vesselID, vesselID, sizeof(newRecord->vessel.vesselID) -
1);
    strncpy(newRecord->vessel.vesselName, vesselName,
sizeof(newRecord->vessel.vesselName) - 1);
    newRecord->isSpecies = isSpecies;

    if (isSpecies) {
        strncpy(newRecord->catchInfo.species, catchInfo,
sizeof(newRecord->catchInfo.species) - 1);
    } else {
        newRecord->catchInfo.weight = atof(catchInfo);
    }

    (*count)++;
}

void displayCatchRecords(CatchRecord *records, int count) {
    for (int i = 0; i < count; i++) {
        printf("Vessel ID: %s\n", records[i].vessel.vesselID);
        printf("Vessel Name: %s\n", records[i].vessel.vesselName);
        if (records[i].isSpecies) {
            printf("Catch Species: %s\n", records[i].catchInfo.species);
        } else {
            printf("Catch Weight: %.2f kg\n", records[i].catchInfo.weight);
        }
        printf("\n");
    }
}

int main() {
    CatchRecord *records = NULL;
    int recordCount = 0;

    addCatchRecord(&records, &recordCount, "V001", "Fishing Vessel A", "Tuna", 1);
    addCatchRecord(&records, &recordCount, "V002", "Fishing Vessel B", "15.5", 0);
```

```
    displayCatchRecords(records, recordCount);

    free(records);

    return 0;
}
```

20. Submarine Navigation System
Description:
Create a submarine navigation system using structures for navigation data, unions for environmental conditions, and arrays for depth readings.
Specifications:
Structure: Navigation data (location, depth).
Union: Environmental conditions (temperature, pressure).
Array: Depth readings.
const Pointers: Immutable navigation data.
Double Pointers: Manage dynamic depth logs.

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

typedef struct {
    char location[100];
    double depth;
} NavigationData;

typedef union {
    float temperature;
    float pressure;
} EnvironmentalConditions;

typedef struct {
    NavigationData navData;
    EnvironmentalConditions  envConditions;
    int isTemperature; // 0 for pressure, 1 for temperature
} DepthLog;
```

```
void addDepthLog(DepthLog **logs, int *count, const char *location, double depth,
const char *envData, int isTemperature) {
    *logs = realloc(*logs, (*count + 1) * sizeof(DepthLog));
    if (*logs == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    DepthLog *newLog = &(*logs)[*count];
    strncpy(newLog->navData.location, location, sizeof(newLog->navData.location) - 1);
    newLog->navData.depth = depth;
    newLog->isTemperature = isTemperature;

    if (isTemperature) {
        newLog->envConditions.temperature = atof(envData);
    } else {
        newLog->envConditions.pressure = atof(envData);
    }

    (*count)++;
}

void displayDepthLogs(DepthLog *logs, int count) {
    for (int i = 0; i < count; i++) {
        printf("Location: %s\n", logs[i].navData.location);
        printf("Depth: %.2f meters\n", logs[i].navData.depth);
        if (logs[i].isTemperature) {
            printf("Temperature: %.2f °C\n", logs[i].envConditions.temperature);
        } else {
            printf("Pressure: %.2f bar\n", logs[i].envConditions.pressure);
        }
        printf("\n");
    }
}

int main() {
    DepthLog *logs = NULL;
    int logCount = 0;

    addDepthLog(&logs, &logCount, "Deep Ocean Trench", 2000.5, "5.5", 1); //
Temperature
```

```
    addDepthLog(&logs, &logCount, "Mid Ocean Ridge", 1500.3, "250.0", 0); // Pressure

    displayDepthLogs(logs, logCount);

    free(logs);

    return 0;
}
```