

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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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.

13. Vehicle Load Balancing System

Description:

Design a vehicle load balancing system using structures for load details and unions for load types. Use const pointers for load identifiers and double pointers for managing dynamic load distribution.

Specifications:

- **Structure:** Load details (ID, weight, destination).
- **Union:** Load types (bulk, container).
- **const Pointer:** Load IDs.
- **Double Pointers:** Dynamic load handling.

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.

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.

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.

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.

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.

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.

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.

1.

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

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

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

```
#define MAX_NAME_LEN 50
```

```
#define MAX_CATEGORY_LEN 30
```

```
union Attributes {
```

```
    double weight; // Weight of the item
```

```
    double volume; // Volume of the item
```

```
};
```

```
struct Item {
```

```
    const char *itemCode;
```

```
    char name[MAX_NAME_LEN];
```

```
    char category[MAX_CATEGORY_LEN];
```

```
    union Attributes attribute;
```

```
    int isWeight;
```

```
};
```

```
struct Item* createItem(const char *itemCode, const char *name, const char *category, double  
value, int isWeight) {
```

```
    struct Item *newItem = (struct Item *)malloc(sizeof(struct Item));
```

```
    newItem->itemCode = itemCode;
```

```
    strncpy(newItem->name, name, MAX_NAME_LEN);
```



```
strncpy(newItem->category, category, MAX_CATEGORY_LEN);

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

newItem->isWeight = isWeight;

return newItem;
}

void displayItem(struct Item *item) {
    printf("Item Code: %s\n", item->itemCode);
    printf("Name: %s\n", item->name);
    printf("Category: %s\n", item->category);

    if (item->isWeight) {
        printf("Weight: %.2f\n", item->attribute.weight);
    } else {
        printf("Volume: %.2f\n", item->attribute.volume);
    }
}
```

```

int main() {

    struct Item *inventory = (struct Item *)malloc(5 * sizeof(struct Item *));

    inventory[0] = createItem("A1001", "Item A", "Category 1", 10.5, 1); // Weight-based item
    inventory[1] = createItem("B2001", "Item B", "Category 2", 20.0, 0); // Volume-based item
    inventory[2] = createItem("C3001", "Item C", "Category 1", 15.3, 1); // Weight-based item
    inventory[3] = createItem("D4001", "Item D", "Category 3", 25.0, 0); // Volume-based item
    inventory[4] = createItem("E5001", "Item E", "Category 2", 30.7, 1); // Weight-based item


    for (int i = 0; i < 5; i++) {

        printf("\nItem %d:\n", i + 1);

        displayItem(inventory[i]);

    }

    for (int i = 0; i < 5; i++) {

        free(inventory[i]);

    }

    free(inventory);


    return 0;

}

```

2.

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

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

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

```
#define MAX_LOCATION_LEN 100
```

```
union TransportMode {
```

```
    double flightDuration;
```

```
    double shippingDistance;
```

```
    double roadDistance;
```

```
};
```

```
struct Route {
```

```
    const char *routeID;
```

```
    char start[MAX_LOCATION_LEN];
```

```
    char end[MAX_LOCATION_LEN];
```

```
    union TransportMode transport;
```

```
    int transportMode;
```

```
};
```

```
struct Route* createRoute(const char *routeID, const char *start, const char *end, double  
transportValue, int transportMode) {
```

```
    struct Route *newRoute = (struct Route *)malloc(sizeof(struct Route));
```

```
    newRoute->routeID = routeID;
```

```
    strncpy(newRoute->start, start, MAX_LOCATION_LEN);
```

```
    strncpy(newRoute->end, end, MAX_LOCATION_LEN);
```

```
newRoute->transportMode = transportMode;
```

```
if (transportMode == 0) {
```

```
    newRoute->transport.flightDuration = transportValue;
```

```
} else if (transportMode == 1) {
```

```
    newRoute->transport.shippingDistance = transportValue;
```

```
} else if (transportMode == 2) {
```

```
    newRoute->transport.roadDistance = transportValue;
```

```
}
```

```
return newRoute;
```

```
}
```

```
void displayRoute(struct Route *route) {
```

```
    printf("Route ID: %s\n", route->routeID);
```

```
    printf("Start Location: %s\n", route->start);
```

```
    printf("End Location: %s\n", route->end);
```

```
if (route->transportMode == 0) {
```

```
    printf("Transport Mode: Air\n");
```

```
    printf("Flight Duration: %.2f\n", route->transport.flightDuration);
```

```
} else if (route->transportMode == 1) {
```

```

        printf("Transport Mode: Sea\n");

        printf("Shipping Distance: %.2f\n", route->transport.shippingDistance);
    } else if (route->transportMode == 2) {

        printf("Transport Mode: Land\n");

        printf("Road Distance: %.2f\n", route->transport.roadDistance);

    }
}

int main() {

    struct Route *routes = (struct Route *)malloc(3 * sizeof(struct Route *));

    // Add routes to the dynamic array

    routes[0] = createRoute("R1001", "New York", "London", 7.5, 0); //Air Transport
    routes[1] = createRoute("R2001", "Asia", "Tokyo", 4000.0, 1); // Sea transport
    routes[2] = createRoute("R3001", "Chennai", "Banglore", 1200.0, 2); // Land transport

    for (int i = 0; i < 3; i++) {

        printf("\nRoute %d:\n", i + 1);

        displayRoute(routes[i]);

    }

    for (int i = 0; i < 3; i++) {

        free(routes[i]);

    }
}

```

```
    free(routes);

    return 0;
}
```

3.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
#define MAX_TYPE_LEN 50
```

```
union Status {

    int active;

    int maintenance;

};
```

```
struct Vehicle {

    const char *vehicleID;

    char type[MAX_TYPE_LEN];

    union Status status;

    int isInMaintenance;

};
```

```
struct Vehicle* createVehicle(const char *vehicleID, const char *type, int isInMaintenance) {
```

```
struct Vehicle *newVehicle = (struct Vehicle *)malloc(sizeof(struct Vehicle));

newVehicle->vehicleID = vehicleID;

strncpy(newVehicle->type, type, MAX_TYPE_LEN);


newVehicle->isInMaintenance = isInMaintenance;

if (isInMaintenance) {

    newVehicle->status.maintenance = 1;

} else {

    newVehicle->status.active = 1;

}


return newVehicle;

}

void displayVehicle(struct Vehicle *vehicle) {

    printf("Vehicle ID: %s\n", vehicle->vehicleID);

    printf("Vehicle Type: %s\n", vehicle->type);


    if (vehicle->isInMaintenance) {

        printf("Status: Under Maintenance\n");

    } else {

        printf("Status: Active\n");

    }

}
```

```

int main() {

    struct Vehicle *fleet = (struct Vehicle *)malloc(3 * sizeof(struct Vehicle *));

    fleet[0] = createVehicle("V1001", "Truck", 0); // Active

    fleet[1] = createVehicle("V2001", "Van", 1); // Under maintenance

    fleet[2] = createVehicle("V3001", "Car", 0); // Active


    for (int i = 0; i < 3; i++) {

        printf("\nVehicle %d:\n", i + 1);

        displayVehicle(fleet[i]);

    }

    for (int i = 0; i < 3; i++) {

        free(fleet[i]);

    }

    free(fleet);


    return 0;

}

```

4.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

```



```
#define MAX_NAME_LEN 100
```

```
#define MAX_ITEM_LEN 100
```

```
union PaymentMethod {
```

```
    char creditCard[16];
```

```
    double cashAmount;
```

```
};
```

```
struct Order {
```

```
    const char *orderID;
```

```
    char customer[MAX_NAME_LEN];
```

```
    char items[MAX_ITEM_LEN];
```

```
    union PaymentMethod payment;
```

```
    int isCreditCard;
```

```
};
```

```
struct Order* createOrder(const char *orderID, const char *customer, const char *items, const  
char *paymentDetail, int isCreditCard) {
```

```
    struct Order *newOrder = (struct Order *)malloc(sizeof(struct Order));
```

```
    newOrder->orderID = orderID;
```

```
    strncpy(newOrder->customer, customer, MAX_NAME_LEN);
```

```
    strncpy(newOrder->items, items, MAX_ITEM_LEN);
```

```
    newOrder->isCreditCard = isCreditCard;
```

```
    if (isCreditCard) {
```

```

        strncpy(newOrder->payment.creditCard, paymentDetail, 16);
    } else {
        newOrder->payment.cashAmount = atof(paymentDetail);
    }
    return newOrder;
}

```

```

void displayOrder(struct Order *order) {
    printf("Order ID: %s\n", order->orderID);
    printf("Customer: %s\n", order->customer);
    printf("Items: %s\n", order->items);
    if (order->isCreditCard) {
        printf("Payment Method: Credit Card\n");
        printf("Credit Card: %s\n", order->payment.creditCard);
    } else {
        printf("Payment Method: Cash\n");
        printf("Cash Amount: %.2f\n", order->payment.cashAmount);
    }
}

```

```

int main() {
    struct Order *orderQueue = (struct Order *)malloc(3 * sizeof(struct Order *));

```

```
orderQueue[0] = createOrder("O1001", "Sofi", "Laptop, Mouse", "1234567812345678", 1);
```

```
orderQueue[1] = createOrder("O1002", "Mickelen", "Phone", "100.50", 0);
```

```
orderQueue[2] = createOrder("O1003", "Christo", "Tablet", "9876543212345678", 1);
```

```
for (int i = 0; i < 3; i++) {  
    displayOrder(orderQueue[i]);  
}
```

```
for (int i = 0; i < 3; i++) {  
    free(orderQueue[i]);  
}
```

```
free(orderQueue);
```

```
return 0;
```

```
}
```

5.

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

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

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

```
#define MAX_LOCATION_LEN 100
```

```
union TrackingEvent {
```

```
    int dispatched;
```

```
    int delivered;
```

```

};

struct Shipment {

    const char *trackingNumber;

    char origin[MAX_LOCATION_LEN];

    char destination[MAX_LOCATION_LEN];

    union TrackingEvent event;

    int isDispatched;

};

struct Shipment* createShipment(const char *trackingNumber, const char *origin, const char
*destination, int isDispatched) {

    struct Shipment *newShipment = (struct Shipment *)malloc(sizeof(struct Shipment));

    newShipment->trackingNumber = trackingNumber;

    strncpy(newShipment->origin, origin, MAX_LOCATION_LEN);

    strncpy(newShipment->destination, destination, MAX_LOCATION_LEN);

    newShipment->isDispatched = isDispatched;

    if (isDispatched) {

        newShipment->event.dispatched = 1;

    } else {

        newShipment->event.delivered = 1;

    }

    return newShipment;

}

void displayShipment(struct Shipment *shipment) {

    printf("Tracking Number: %s\n", shipment->trackingNumber);

```

```

printf("Origin: %s\n", shipment->origin);

printf("Destination: %s\n", shipment->destination);

if (shipment->isDispatched) {

    printf("Status: Dispatched\n");

} else {

    printf("Status: Delivered\n");

}

}

int main() {

    struct Shipment *shipments = (struct Shipment *)malloc(3 * sizeof(struct Shipment *));

    shipments[0] = createShipment("T1001", "New York", "Los Angeles", 1);

    shipments[1] = createShipment("T1002", "Chicago", "Miami", 0);

    shipments[2] = createShipment("T1003", "San Francisco", "Houston", 1);

    for (int i = 0; i < 3; i++) {

        displayShipment(shipments[i]);

    }

    for (int i = 0; i < 3; i++) {

        free(shipments[i]);

    }

    free(shipments);

    return 0;

}

```

6.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_LOCATION_LEN 100

union TrafficCondition {
    int clear;
    int congested;
};

struct TrafficNode {
    const char *nodeID;
    char location[MAX_LOCATION_LEN];
    union TrafficCondition condition;
    int isCongested;
};

struct TrafficNode* createTrafficNode(const char *nodeID, const char *location, int
isCongested) {
    struct TrafficNode *newNode = (struct TrafficNode *)malloc(sizeof(struct TrafficNode));
    newNode->nodeID = nodeID;
    strncpy(newNode->location, location, MAX_LOCATION_LEN);
    newNode->isCongested = isCongested;
    if (isCongested) {
        newNode->condition.congested = 1;
    } else {
        newNode->condition.clear = 1;
    }
    return newNode;
}

```

```

void displayTrafficNode(struct TrafficNode *node) {
    printf("Node ID: %s\n", node->nodeID);
    printf("Location: %s\n", node->location);
    if (node->isCongested) {
        printf("Condition: Congested\n");
    } else {
        printf("Condition: Clear\n");
    }
}

int main() {
    struct TrafficNode *trafficData = (struct TrafficNode *)malloc(3 * sizeof(struct TrafficNode
*));

    trafficData[0] = createTrafficNode("N1001", "Main Street", 1);
    trafficData[1] = createTrafficNode("N1002", "Highway 50", 0);
    trafficData[2] = createTrafficNode("N1003", "Broadway", 1);

    for (int i = 0; i < 3; i++) {
        displayTrafficNode(trafficData[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(trafficData[i]);
    }

    free(trafficData);
    return 0;
}

```

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_LOCATION_LEN 100

union ItemType {
    int perishable;
    int nonPerishable;
};

struct Slot {
    const char *slotID;
    char location[MAX_LOCATION_LEN];
    double size;
    union ItemType itemType;
    int isPerishable;
};

struct Slot* createSlot(const char *slotID, const char *location, double size, int isPerishable) {
    struct Slot *newSlot = (struct Slot *)malloc(sizeof(struct Slot));
    newSlot->slotID = slotID;
    strncpy(newSlot->location, location, MAX_LOCATION_LEN);
    newSlot->size = size;
    newSlot->isPerishable = isPerishable;
    if (isPerishable) {
        newSlot->itemType.perishable = 1;
    } else {
        newSlot->itemType.nonPerishable = 1;
    }
    return newSlot;
}

```



```

}

void displaySlot(struct Slot *slot) {
    printf("Slot ID: %s\n", slot->slotID);
    printf("Location: %s\n", slot->location);
    printf("Size: %.2f cubic meters\n", slot->size);
    if (slot->isPerishable) {
        printf("Item Type: Perishable\n");
    } else {
        printf("Item Type: Non-Perishable\n");
    }
}

int main() {
    struct Slot *warehouseSlots = (struct Slot *)malloc(3 * sizeof(struct Slot *));
    warehouseSlots[0] = createSlot("S1001", "Aisle 1", 10.5, 1);
    warehouseSlots[1] = createSlot("S1002", "Aisle 2", 5.0, 0);
    warehouseSlots[2] = createSlot("S1003", "Aisle 3", 8.0, 1);

    for (int i = 0; i < 3; i++) {
        displaySlot(warehouseSlots[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(warehouseSlots[i]);
    }
    free(warehouseSlots);
    return 0;
}

```

8.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_DESTINATION_LEN 100
union DeliveryMethod {
    int standard;
    int express;
};
struct Package {
    const char *packageID;
    double weight;
    char destination[MAX_DESTINATION_LEN];
    union DeliveryMethod deliveryMethod;
    int isExpress;
};
struct Package* createPackage(const char *packageID, double weight, const char *destination,
int isExpress) {
    struct Package *newPackage = (struct Package *)malloc(sizeof(struct Package));
    newPackage->packageID = packageID;
    newPackage->weight = weight;
    strncpy(newPackage->destination, destination, MAX_DESTINATION_LEN);
    newPackage->isExpress = isExpress;
    if (isExpress) {
        newPackage->deliveryMethod.express = 1;
    } else {
        newPackage->deliveryMethod.standard = 1;
    }
    return newPackage;
}
```

```

}

void displayPackage(struct Package *package) {
    printf("Package ID: %s\n", package->packageID);
    printf("Weight: %.2f kg\n", package->weight);
    printf("Destination: %s\n", package->destination);
    if (package->isExpress) {
        printf("Delivery Method: Express\n");
    } else {
        printf("Delivery Method: Standard\n");
    }
}

int main() {
    struct Package *deliveryRoutes = (struct Package *)malloc(3 * sizeof(struct Package *));
    deliveryRoutes[0] = createPackage("P1001", 5.0, "New York", 1);
    deliveryRoutes[1] = createPackage("P1002", 2.5, "Los Angeles", 0);
    deliveryRoutes[2] = createPackage("P1003", 10.0, "Chicago", 1);
    for (int i = 0; i < 3; i++) {
        displayPackage(deliveryRoutes[i]);
    }
    for (int i = 0; i < 3; i++) {
        free(deliveryRoutes[i]);
    }
    free(deliveryRoutes);
    return 0;
}

```

9.

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

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

```

#include <string.h>

#define MAX_TIMESTAMP_LEN 100

union Metric {
    double speed;
    double efficiency;
};

struct AnalyticsRecord {
    const char *timestamp;
    union Metric metric;
    int isSpeed;
};

struct AnalyticsRecord* createAnalyticsRecord(const char *timestamp, double value, int
isSpeed) {
    struct AnalyticsRecord *newRecord = (struct AnalyticsRecord *)malloc(sizeof(struct
AnalyticsRecord));

    newRecord->timestamp = timestamp;
    newRecord->isSpeed = isSpeed;
    if (isSpeed) {
        newRecord->metric.speed = value;
    } else {
        newRecord->metric.efficiency = value;
    }
    return newRecord;
}

void displayAnalyticsRecord(struct AnalyticsRecord *record) {
    printf("Timestamp: %s\n", record->timestamp);
    if (record->isSpeed) {
        printf("Metric: Speed\n");
        printf("Speed: %.2f km/h\n", record->metric.speed);
    }
}

```

```

    } else {
        printf("Metric: Efficiency\n");
        printf("Efficiency: %.2f%%\n", record->metric.efficiency);
    }
}

int main() {
    struct AnalyticsRecord *analyticsData = (struct AnalyticsRecord *)malloc(3 * sizeof(struct
AnalyticsRecord *));

    analyticsData[0] = createAnalyticsRecord("2025-01-22 08:00", 60.5, 1);
    analyticsData[1] = createAnalyticsRecord("2025-01-22 09:00", 85.0, 0);
    analyticsData[2] = createAnalyticsRecord("2025-01-22 10:00", 58.3, 1);

    for (int i = 0; i < 3; i++) {
        displayAnalyticsRecord(analyticsData[i]);
    }
    for (int i = 0; i < 3; i++) {
        free(analyticsData[i]);
    }
    free(analyticsData);
    return 0;
}

```

10.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TIME_LEN 20
union TransportType {
    int bus;
    int truck;
}

```

```

};

struct Schedule {
    const char *scheduleID;
    char startTime[MAX_TIME_LEN];
    char endTime[MAX_TIME_LEN];
    union TransportType transportType;
    int isBus;
};

struct Schedule* createSchedule(const char *scheduleID, const char *startTime, const char
*endTime, int isBus) {
    struct Schedule *newSchedule = (struct Schedule *)malloc(sizeof(struct Schedule));
    newSchedule->scheduleID = scheduleID;
    strncpy(newSchedule->startTime, startTime, MAX_TIME_LEN);
    strncpy(newSchedule->endTime, endTime, MAX_TIME_LEN);
    newSchedule->isBus = isBus;
    if (isBus) {
        newSchedule->transportType.bus = 1;
    } else {
        newSchedule->transportType.truck = 1;
    }
    return newSchedule;
}

void displaySchedule(struct Schedule *schedule) {
    printf("Schedule ID: %s\n", schedule->scheduleID);
    printf("Start Time: %s\n", schedule->startTime);
    printf("End Time: %s\n", schedule->endTime);
    if (schedule->isBus) {
        printf("Transport Type: Bus\n");
    } else {

```

```

        printf("Transport Type: Truck\n");
    }
}

int main() {
    struct Schedule *scheduleList = (struct Schedule *)malloc(3 * sizeof(struct Schedule *));
    scheduleList[0] = createSchedule("S1001", "08:00", "12:00", 1);
    scheduleList[1] = createSchedule("S1002", "13:00", "17:00", 0);
    scheduleList[2] = createSchedule("S1003", "18:00", "22:00", 1);
    for (int i = 0; i < 3; i++) {
        displaySchedule(scheduleList[i]);
    }
    for (int i = 0; i < 3; i++) {
        free(scheduleList[i]);
    }
    free(scheduleList);
    return 0;
}

//11.

```

```

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

```

```

#define MAX_NAME_LEN 100

```

```

union TransactionType {
    int purchase;
    int returnTransaction;
}

```

```
};
```

```
struct Entity {  
    const char *entityID;  
    char name[MAX_NAME_LEN];  
    union TransactionType transactionType;  
    int isPurchase;  
};
```

```
struct Entity* createEntity(const char *entityID, const char *name, int isPurchase) {  
    struct Entity *newEntity = (struct Entity *)malloc(sizeof(struct Entity));  
    newEntity->entityID = entityID;  
    strncpy(newEntity->name, name, MAX_NAME_LEN);  
    newEntity->isPurchase = isPurchase;  
    if (isPurchase) {  
        newEntity->transactionType.purchase = 1;  
    } else {  
        newEntity->transactionType.returnTransaction = 1;  
    }  
    return newEntity;  
}
```

```
void displayEntity(struct Entity *entity) {  
    printf("Entity ID: %s\n", entity->entityID);  
    printf("Name: %s\n", entity->name);  
    if (entity->isPurchase) {  
        printf("Transaction Type: Purchase\n");  
    } else {
```



```

        printf("Transaction Type: Return\n");
    }
}

int main() {
    struct Entity *supplyChain = (struct Entity *)malloc(3 * sizeof(struct Entity *));

    supplyChain[0] = createEntity("T1001", "Supplier A", 1);
    supplyChain[1] = createEntity("T1002", "Customer B", 0);
    supplyChain[2] = createEntity("T1003", "Supplier C", 1);

    for (int i = 0; i < 3; i++) {
        displayEntity(supplyChain[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(supplyChain[i]);
    }
    free(supplyChain);

    return 0;
}

```

//12.

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

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

```
union PricingModel {
```

```

        double fixed;

        double variable;
};

struct Cost {
    const char *costID;
    double baseCost;
    union PricingModel pricingModel;
    int isFixed;
};

struct Cost* createCost(const char *costID, double baseCost, int isFixed) {
    struct Cost *newCost = (struct Cost *)malloc(sizeof(struct Cost));
    newCost->costID = costID;
    newCost->baseCost = baseCost;
    newCost->isFixed = isFixed;
    if (isFixed) {
        newCost->pricingModel.fixed = baseCost;
    } else {
        newCost->pricingModel.variable = baseCost;
    }
    return newCost;
}

void displayCost(struct Cost *cost) {
    printf("Cost ID: %s\n", cost->costID);
    printf("Base Cost: %.2f\n", cost->baseCost);
    if (cost->isFixed) {

```

```

        printf("Pricing Model: Fixed\n");
    } else {
        printf("Pricing Model: Variable\n");
    }
}

int main() {
    struct Cost *costList = (struct Cost *)malloc(3 * sizeof(struct Cost *));

    costList[0] = createCost("C1001", 100.0, 1);
    costList[1] = createCost("C1002", 0.5, 0);
    costList[2] = createCost("C1003", 150.0, 1);

    for (int i = 0; i < 3; i++) {
        displayCost(costList[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(costList[i]);
    }
    free(costList);

    return 0;
}

```

//13.

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

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

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

```
#define MAX_DESTINATION_LEN 100
```

```
union LoadType {  
    int bulk;  
    int container;  
};
```

```
struct Load {  
    const char *loadID;  
    double weight;  
    char destination[MAX_DESTINATION_LEN];  
    union LoadType loadType;  
    int isBulk;  
};
```

```
struct Load* createLoad(const char *loadID, double weight, const char *destination, int isBulk)  
{  
    struct Load *newLoad = (struct Load *)malloc(sizeof(struct Load));  
    newLoad->loadID = loadID;  
    newLoad->weight = weight;  
    strncpy(newLoad->destination, destination, MAX_DESTINATION_LEN);  
    newLoad->isBulk = isBulk;  
    if (isBulk) {  
        newLoad->loadType.bulk = 1;  
    } else {  
        newLoad->loadType.container = 1;  
    }  
}
```

```

        return newLoad;
    }

void displayLoad(struct Load *load) {
    printf("Load ID: %s\n", load->loadID);
    printf("Weight: %.2f kg\n", load->weight);
    printf("Destination: %s\n", load->destination);
    if (load->isBulk) {
        printf("Load Type: Bulk\n");
    } else {
        printf("Load Type: Container\n");
    }
}

int main() {
    struct Load *loads = (struct Load *)malloc(3 * sizeof(struct Load *));

    loads[0] = createLoad("L1001", 500.0, "Port A", 1);
    loads[1] = createLoad("L1002", 1500.0, "Port B", 0);
    loads[2] = createLoad("L1003", 800.0, "Port C", 1);

    for (int i = 0; i < 3; i++) {
        displayLoad(loads[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(loads[i]);
    }
}

```

```

        free(loads);

    return 0;
}

//14.

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

#define MAX_LOCATION_LEN 100

union TransportMode {
    int rail;
    int road;
};

struct Transport {
    const char *transportID;
    char origin[MAX_LOCATION_LEN];
    char destination[MAX_LOCATION_LEN];
    union TransportMode transportMode;
    int isRail;
};

struct Transport* createTransport(const char *transportID, const char *origin, const char
*destination, int isRail) {
    struct Transport *newTransport = (struct Transport *)malloc(sizeof(struct Transport));

```

```

newTransport->transportID = transportID;
strncpy(newTransport->origin, origin, MAX_LOCATION_LEN);
strncpy(newTransport->destination, destination, MAX_LOCATION_LEN);
newTransport->isRail = isRail;
if (isRail) {
    newTransport->transportMode.rail = 1;
} else {
    newTransport->transportMode.road = 1;
}
return newTransport;
}

void displayTransport(struct Transport *transport) {
    printf("Transport ID: %s\n", transport->transportID);
    printf("Origin: %s\n", transport->origin);
    printf("Destination: %s\n", transport->destination);
    if (transport->isRail) {
        printf("Transport Mode: Rail\n");
    } else {
        printf("Transport Mode: Road\n");
    }
}

int main() {
    struct Transport *routes = (struct Transport *)malloc(3 * sizeof(struct Transport *));

    routes[0] = createTransport("T1001", "City A", "City B", 1);
    routes[1] = createTransport("T1002", "City C", "City D", 0);

```

```

    routes[2] = createTransport("T1003", "City E", "City F", 1);

    for (int i = 0; i < 3; i++) {
        displayTransport(routes[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(routes[i]);
    }
    free(routes);

    return 0;
}

```

//15.

```

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

```

```

union PerformanceAspect {
    double time;
    double cost;
};

```

```

struct PerformanceMetric {
    const char *metricID;
    double value;
    union PerformanceAspect performanceAspect;
    int isTime;
}

```



```
};
```

```
struct PerformanceMetric* createPerformanceMetric(const char *metricID, double value, int isTime) {
```

```
    struct PerformanceMetric *newMetric = (struct PerformanceMetric *)malloc(sizeof(struct PerformanceMetric));
```

```
    newMetric->metricID = metricID;
```

```
    newMetric->value = value;
```

```
    newMetric->isTime = isTime;
```

```
    if (isTime) {
```

```
        newMetric->performanceAspect.time = value;
```

```
    } else {
```

```
        newMetric->performanceAspect.cost = value;
```

```
    }
```

```
    return newMetric;
```

```
}
```

```
void displayPerformanceMetric(struct PerformanceMetric *metric) {
```

```
    printf("Metric ID: %s\n", metric->metricID);
```

```
    printf("Value: %.2f\n", metric->value);
```

```
    if (metric->isTime) {
```

```
        printf("Aspect: Time\n");
```

```
    } else {
```

```
        printf("Aspect: Cost\n");
```

```
    }
```

```
}
```

```
int main() {
```

```
    struct PerformanceMetric *metrics = (struct PerformanceMetric *)malloc(3 * sizeof(struct PerformanceMetric *));
```

```
    metrics[0] = createPerformanceMetric("M1001", 120.5, 1);
```

```
    metrics[1] = createPerformanceMetric("M1002", 50.0, 0);
```

```
    metrics[2] = createPerformanceMetric("M1003", 110.0, 1);
```

```
    for (int i = 0; i < 3; i++) {
```

```
        displayPerformanceMetric(metrics[i]);
```

```
    }
```

```
    for (int i = 0; i < 3; i++) {
```

```
        free(metrics[i]);
```

```
    }
```

```
    free(metrics);
```

```
    return 0;
```

```
}
```

```
//16.
```

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

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

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

```
#define MAX_TYPE_LEN 50
```

```
union TaskType {
```

```
    int picking;
```

```
    int sorting;
};
```

```
struct Robot {
    const char *robotID;
    char type[MAX_TYPE_LEN];
    char status[MAX_TYPE_LEN];
    union TaskType taskType;
    int isPicking;
};
```

```
struct Robot* createRobot(const char *robotID, const char *type, const char *status, int
isPicking) {
    struct Robot *newRobot = (struct Robot *)malloc(sizeof(struct Robot));
    newRobot->robotID = robotID;
    strncpy(newRobot->type, type, MAX_TYPE_LEN);
    strncpy(newRobot->status, status, MAX_TYPE_LEN);
    newRobot->isPicking = isPicking;
    if (isPicking) {
        newRobot->taskType.picking = 1;
    } else {
        newRobot->taskType.sorting = 1;
    }
    return newRobot;
}
```

```
void displayRobot(struct Robot *robot) {
    printf("Robot ID: %s\n", robot->robotID);
    printf("Type: %s\n", robot->type);
}
```

```

printf("Status: %s\n", robot->status);
if (robot->isPicking) {
    printf("Task Type: Picking\n");
} else {
    printf("Task Type: Sorting\n");
}
}

int main() {
    struct Robot *robots = (struct Robot *)malloc(3 * sizeof(struct Robot *));

    robots[0] = createRobot("R1001", "Picker", "Active", 1);
    robots[1] = createRobot("R1002", "Sorter", "Inactive", 0);
    robots[2] = createRobot("R1003", "Picker", "Active", 1);

    for (int i = 0; i < 3; i++) {
        displayRobot(robots[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(robots[i]);
    }
    free(robots);

    return 0;
}

//17.

```

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

```
#define MAX_CONTENT_LEN 500
```

```
union FeedbackType {
    int positive;
    int negative;
};
```

```
struct Feedback {
    const char *feedbackID;
    char content[MAX_CONTENT_LEN];
    union FeedbackType feedbackType;
    int isPositive;
};
```

```
struct Feedback* createFeedback(const char *feedbackID, const char *content, int isPositive) {
    struct Feedback *newFeedback = (struct Feedback *)malloc(sizeof(struct Feedback));
    newFeedback->feedbackID = feedbackID;
    strncpy(newFeedback->content, content, MAX_CONTENT_LEN);
    newFeedback->isPositive = isPositive;
    if (isPositive) {
        newFeedback->feedbackType.positive = 1;
    } else {
        newFeedback->feedbackType.negative = 1;
    }
}
```

```

    }
    return newFeedback;
}

void displayFeedback(struct Feedback *feedback) {
    printf("Feedback ID: %s\n", feedback->feedbackID);
    printf("Content: %s\n", feedback->content);
    if (feedback->isPositive) {
        printf("Feedback Type: Positive\n");
    } else {
        printf("Feedback Type: Negative\n");
    }
}

int main() {
    struct Feedback *feedbacks = (struct Feedback *)malloc(3 * sizeof(struct Feedback *));

    feedbacks[0] = createFeedback("F1001", "Great service!", 1);
    feedbacks[1] = createFeedback("F1002", "Delivery was delayed.", 0);
    feedbacks[2] = createFeedback("F1003", "Excellent product quality.", 1);

    for (int i = 0; i < 3; i++) {
        displayFeedback(feedbacks[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(feedbacks[i]);
    }
}

```

```

    free(feedbacks);

    return 0;
}

//18.

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

#define MAX_LOCATION_LEN 100

union CoordinationType {
    int dispatch;
    int reroute;
};

struct Fleet {
    const char *fleetID;
    char location[MAX_LOCATION_LEN];
    char status[MAX_LOCATION_LEN];
    union CoordinationType coordinationType;
    int isDispatch;
};

struct Fleet* createFleet(const char *fleetID, const char *location, const char *status, int
isDispatch) {
    struct Fleet *newFleet = (struct Fleet *)malloc(sizeof(struct Fleet));

```

```

newFleet->fleetID = fleetID;
strncpy(newFleet->location, location, MAX_LOCATION_LEN);
strncpy(newFleet->status, status, MAX_LOCATION_LEN);
newFleet->isDispatch = isDispatch;
if (isDispatch) {
    newFleet->coordinationType.dispatch = 1;
} else {
    newFleet->coordinationType.reroute = 1;
}
return newFleet;
}

void displayFleet(struct Fleet *fleet) {
    printf("Fleet ID: %s\n", fleet->fleetID);
    printf("Location: %s\n", fleet->location);
    printf("Status: %s\n", fleet->status);
    if (fleet->isDispatch) {
        printf("Coordination Type: Dispatch\n");
    } else {
        printf("Coordination Type: Reroute\n");
    }
}

int main() {
    struct Fleet *fleets = (struct Fleet *)malloc(3 * sizeof(struct Fleet *));

    fleets[0] = createFleet("F1001", "Location A", "Active", 1);
    fleets[1] = createFleet("F1002", "Location B", "Inactive", 0);

```



```

    fleets[2] = createFleet("F1003", "Location C", "Active", 1);

    for (int i = 0; i < 3; i++) {
        displayFleet(fleets[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(fleets[i]);
    }
    free(fleets);

    return 0;
}

```

//19.

```

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

```

```

#define MAX_DESCRIPTION_LEN 300

```

```

union EventType {
    int breach;
    int resolved;
};

```

```

struct SecurityEvent {
    const char *eventID;

```

```

    char description[MAX_DESCRIPTION_LEN];

    union EventType eventType;

    int isBreach;

};

struct SecurityEvent* createSecurityEvent(const char *eventID, const char *description, int
isBreach) {

    struct SecurityEvent *newEvent = (struct SecurityEvent *)malloc(sizeof(struct
SecurityEvent));

    newEvent->eventID = eventID;

    strncpy(newEvent->description, description, MAX_DESCRIPTION_LEN);

    newEvent->isBreach = isBreach;

    if (isBreach) {
        newEvent->eventType.breach = 1;
    } else {
        newEvent->eventType.resolved = 1;
    }

    return newEvent;
}

void displaySecurityEvent(struct SecurityEvent *event) {

    printf("Event ID: %s\n", event->eventID);

    printf("Description: %s\n", event->description);

    if (event->isBreach) {
        printf("Event Type: Breach\n");
    } else {
        printf("Event Type: Resolved\n");
    }

}

```

```

int main() {

    struct SecurityEvent *events = (struct SecurityEvent *)malloc(3 * sizeof(struct SecurityEvent
    *));

    events[0] = createSecurityEvent("E1001", "Unauthorized access detected.", 1);
    events[1] = createSecurityEvent("E1002", "System access restored.", 0);
    events[2] = createSecurityEvent("E1003", "Suspicious activity detected.", 1);

    for (int i = 0; i < 3; i++) {
        displaySecurityEvent(events[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(events[i]);
    }
    free(events);

    return 0;
}

```

//20.

```

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

```

```

#define MAX_DATE_LEN 20

```

```
union PaymentMethod {  
    int bankTransfer;  
    int cash;  
};
```

```
struct Bill {  
    const char *billID;  
    double amount;  
    char date[MAX_DATE_LEN];  
    union PaymentMethod paymentMethod;  
    int isBankTransfer;  
};
```

```
struct Bill* createBill(const char *billID, double amount, const char *date, int isBankTransfer) {  
    struct Bill *newBill = (struct Bill *)malloc(sizeof(struct Bill));  
    newBill->billID = billID;  
    newBill->amount = amount;  
    strncpy(newBill->date, date, MAX_DATE_LEN);  
    newBill->isBankTransfer = isBankTransfer;  
    if (isBankTransfer) {  
        newBill->paymentMethod.bankTransfer = 1;  
    } else {  
        newBill->paymentMethod.cash = 1;  
    }  
    return newBill;  
}
```

```
void displayBill(struct Bill *bill) {
```

```

printf("Bill ID: %s\n", bill->billID);
printf("Amount: %.2f\n", bill->amount);
printf("Date: %s\n", bill->date);
if (bill->isBankTransfer) {
    printf("Payment Method: Bank Transfer\n");
} else {
    printf("Payment Method: Cash\n");
}
}

int main() {
    struct Bill *bills = (struct Bill *)malloc(3 * sizeof(struct Bill *));

    bills[0] = createBill("B1001", 150.0, "2025-01-22", 1);
    bills[1] = createBill("B1002", 200.0, "2025-01-22", 0);
    bills[2] = createBill("B1003", 300.0, "2025-01-23", 1);

    for (int i = 0; i < 3; i++) {
        displayBill(bills[i]);
    }

    for (int i = 0; i < 3; i++) {
        free(bills[i]);
    }
    free(bills);

    return 0;
}

```

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

1.

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

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

```

#include <string.h>

#define MAX_LOCATION_LEN 100

#define MAX_WAYPOINTS 10

struct Route {

    const char *start;

    const char *end;

    const char *waypoints[MAX_WAYPOINTS];

};

struct Route* createRoute(const char *start, const char *end, const char *waypoints[], int
numWaypoints) {

    struct Route *newRoute = (struct Route *)malloc(sizeof(struct Route));

    newRoute->start = start;

    newRoute->end = end;

    for (int i = 0; i < numWaypoints; i++) {

        newRoute->waypoints[i] = waypoints[i];

    }

    return newRoute;

}

void displayRoute(const struct Route *route) {

    printf("Start Location: %s\n", route->start);

    printf("End Location: %s\n", route->end);

    printf("Waypoints:\n");

    for (int i = 0; i < MAX_WAYPOINTS; i++) {

        if (route->waypoints[i] != NULL) {

```

```

        printf(" - %s\n", route->waypoints[i]);
    }
}

}

void allocateRoutes(struct Route ***routes, int numRoutes) {
    routes = (struct Route *)malloc(numRoutes * sizeof(struct Route *));
}

int main() {
    const char *waypoints1[] = {"Waypoint 1", "Waypoint 2", "Waypoint 3"};
    const char *waypoints2[] = {"Waypoint A", "Waypoint B"};
    struct Route **routes;
    allocateRoutes(&routes, 2);
    routes[0] = createRoute("Port A", "Port B", waypoints1, 3);
    routes[1] = createRoute("Port X", "Port Y", waypoints2, 2);
    for (int i = 0; i < 2; i++) {
        displayRoute(routes[i]);
    }
    for (int i = 0; i < 2; i++) {
        free(routes[i]);
    }
    free(routes);
    return 0;
}

```

2.

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

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

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

```
#define MAX_VESSELS 10
```

```
struct Vessel {
```

```
    const char *name;
```

```
    const char *ID;
```

```
    const char *type;
```

```
    union {
```

```
        int cargoWeight;
```

```
        int passengerCount;
```

```
    };
```

```
};
```

```
void displayVessel(struct Vessel *vessel) {
```

```
    printf("Vessel Name: %s\n", vessel->name);
```

```
    printf("Vessel ID: %s\n", vessel->ID);
```

```
    printf("Vessel Type: %s\n", vessel->type);
```

```
    if (strcmp(vessel->type, "Cargo") == 0) {
```

```
        printf("Cargo Weight: %d\n", vessel->cargoWeight);
```

```
    } else {
```

```
        printf("Passenger Count: %d\n", vessel->passengerCount);
```

```
    }
```

```
}
```

```
int main() {
```

```
    struct Vessel **fleet;
```

```
    int fleetSize = 2;
```

```
    fleet = (struct Vessel **)malloc(fleetSize * sizeof(struct Vessel *));
```

```
        fleet[0] = (struct Vessel *)malloc(sizeof(struct Vessel));
```

```
    fleet[0]->name = "Vessel A";
```

```
    fleet[0]->ID = "V123";
```

```
    fleet[0]->type = "Cargo";
```

```
    fleet[0]->cargoWeight = 5000;
```

```
    fleet[1] = (struct Vessel *)malloc(sizeof(struct Vessel));
```

```
    fleet[1]->name = "Vessel B";
```

```
    fleet[1]->ID = "V124";
```

```
    fleet[1]->type = "Passenger";
```

```
    fleet[1]->passengerCount = 200;
```

```
    for (int i = 0; i < fleetSize; i++) {
```

```
        displayVessel(fleet[i]);
```

```
    }
```

```
    for (int i = 0; i < fleetSize; i++) {
```

```
        free(fleet[i]);
```

```
    }
```

```
    free(fleet);
```



```
    return 0;

}

3.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_TASKS 5

struct MaintenanceTask {

    const char *ID;

    const char *description;

    const char *schedule;

};

void displayTask(struct MaintenanceTask *task) {

    printf("Task ID: %s\n", task->ID);

    printf("Description: %s\n", task->description);

    printf("Scheduled: %s\n", task->schedule);

}

int main() {

    struct MaintenanceTask **tasks;

    int taskCount = 2;

    tasks = (struct MaintenanceTask **)malloc(taskCount * sizeof(struct MaintenanceTask *));
```

```

tasks[0] = (struct MaintenanceTask *)malloc(sizeof(struct MaintenanceTask));

tasks[0]->ID = "T001";

tasks[0]->description = "Engine Check";

tasks[0]->schedule = "2025-02-01";

tasks[1] = (struct MaintenanceTask *)malloc(sizeof(struct MaintenanceTask));

tasks[1]->ID = "T002";

tasks[1]->description = "Hull Inspection";

tasks[1]->schedule = "2025-03-01";

for (int i = 0; i < taskCount; i++) {

    displayTask(tasks[i]);

}

for (int i = 0; i < taskCount; i++) {

    free(tasks[i]);

}

free(tasks);

return 0;

}

```

4.

```

#include <stdio.h>

#include <stdlib.h>

#define MAX_CARGO 5

struct VesselSpecifications {

    int capacity;

```

```

    int dimensions[3]; // length, width, height
};

union CargoProperties {

    int weight;

    int dimensions[3]; // length, width, height
};

void displayCargo(struct VesselSpecifications *vessel, union CargoProperties *cargo) {

    printf("Vessel Capacity: %d\n", vessel->capacity);

    printf("Cargo Weight: %d\n", cargo->weight);

}

int main() {

    struct VesselSpecifications **vessels;

    union CargoProperties **cargo;

    int cargoCount = 3;

    vessels = (struct VesselSpecifications **)malloc(cargoCount * sizeof(struct
VesselSpecifications *));

    cargo = (union CargoProperties **)malloc(cargoCount * sizeof(union CargoProperties *));

    vessels[0] = (struct VesselSpecifications *)malloc(sizeof(struct VesselSpecifications));

    vessels[0]->capacity = 10000;

    vessels[0]->dimensions[0] = 50;

    vessels[0]->dimensions[1] = 30;

    vessels[0]->dimensions[2] = 15;

    cargo[0] = (union CargoProperties *)malloc(sizeof(union CargoProperties));

    cargo[0]->weight = 3000;

```

```

vessels[1] = (struct VesselSpecifications *)malloc(sizeof(struct VesselSpecifications));

vessels[1]->capacity = 8000;

vessels[1]->dimensions[0] = 40;

vessels[1]->dimensions[1] = 25;

vessels[1]->dimensions[2] = 10;

cargo[1] = (union CargoProperties *)malloc(sizeof(union CargoProperties));

cargo[1]->weight = 5000;

for (int i = 0; i < cargoCount; i++) {
    displayCargo(vessels[i], cargo[i]);
}

for (int i = 0; i < cargoCount; i++) {
    free(vessels[i]);
    free(cargo[i]);
}

free(vessels);

free(cargo);

return 0;
}

```

5.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

```

```
#define MAX_RECORDS 5

struct WeatherData {

    float temperature;

    float windSpeed;

};

void displayWeatherAlert(struct WeatherData *weatherData, const char *alertMessage) {

    printf("Temperature: %.2f\n", weatherData->temperature);

    printf("Wind Speed: %.2f\n", weatherData->windSpeed);

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

}

int main() {

    struct WeatherData **weatherRecords;

    const char *alerts[] = {

        "Severe Storm Warning",

        "High Wind Advisory"

    };

    int recordCount = 2;

    weatherRecords = (struct WeatherData **)malloc(recordCount * sizeof(struct WeatherData
*));

    weatherRecords[0] = (struct WeatherData *)malloc(sizeof(struct WeatherData));

    weatherRecords[0]->temperature = 32.5;

    weatherRecords[0]->windSpeed = 60.0;

    weatherRecords[1] = (struct WeatherData *)malloc(sizeof(struct WeatherData));
```

```

weatherRecords[1]->temperature = 29.0;

weatherRecords[1]->windSpeed = 80.0;

for (int i = 0; i < recordCount; i++) {

    displayWeatherAlert(weatherRecords[i], alerts[i]);

}

for (int i = 0; i < recordCount; i++) {

    free(weatherRecords[i]);

}

free(weatherRecords);

return 0;

}

```

7.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_CREW 5

struct CrewMember {

    const char *name;

    const char *role;

    const char *schedule;

};

void displayCrewMember(struct CrewMember *crewMember) {

    printf("Name: %s\n", crewMember->name);

```

```

    printf("Role: %s\n", crewMember->role);

    printf("Schedule: %s\n", crewMember->schedule);
}

int main() {

    struct CrewMember **roster;

    int crewCount = 2;

    roster = (struct CrewMember **)malloc(crewCount * sizeof(struct CrewMember *));

    roster[0] = (struct CrewMember *)malloc(sizeof(struct CrewMember));

    roster[0]->name = "Sofia";

    roster[0]->role = "Captain";

    roster[0]->schedule = "2025-02-01 to 2025-03-01";

    roster[1] = (struct CrewMember *)malloc(sizeof(struct CrewMember));

    roster[1]->name = "Christo";

    roster[1]->role = "Engineer";

    roster[1]->schedule = "2025-02-15 to 2025-03-15";

    for (int i = 0; i < crewCount; i++) {

        displayCrewMember(roster[i]);

    }

    for (int i = 0; i < crewCount; i++) {

        free(roster[i]);

    }

    free(roster);
}

```

```
    return 0;

}

8.

#include <stdio.h>

#include <stdlib.h>

#define MAX_SENSORS 3

struct SensorDetails {

    const char *ID;

    const char *location;

    union {

        float temperature;

        float pressure;

    };

    float readings[5]; // Store sensor readings

};

void displaySensor(struct SensorDetails *sensor) {

    printf("Sensor ID: %s\n", sensor->ID);

    printf("Location: %s\n", sensor->location);

    if (sensor->temperature != 0) {

        printf("Temperature: %.2f°C\n", sensor->temperature);

    } else {

        printf("Pressure: %.2f Pa\n", sensor->pressure);

    }

}
```



```

    for (int i = 0; i < 5; i++) {

        printf("Reading %d: %.2f\n", i + 1, sensor->readings[i]);

    }

}

int main() {

    struct SensorDetails **sensors;

    int sensorCount = 2;

    sensors = (struct SensorDetails **)malloc(sensorCount * sizeof(struct SensorDetails *));

    sensors[0] = (struct SensorDetails *)malloc(sizeof(struct SensorDetails));

    sensors[0]->ID = "S001";

    sensors[0]->location = "A1-B2";

    sensors[0]->temperature = 15.6;

    sensors[0]->readings[0] = 15.6;

    sensors[0]->readings[1] = 15.7;

    sensors[1] = (struct SensorDetails *)malloc(sizeof(struct SensorDetails));

    sensors[1]->ID = "S002";

    sensors[1]->location = "C3-D4";

    sensors[1]->pressure = 101325;

    sensors[1]->readings[0] = 101325;

    sensors[1]->readings[1] = 101300;

    for (int i = 0; i < sensorCount; i++) {

        displaySensor(sensors[i]);

    }

```

```

    for (int i = 0; i < sensorCount; i++) {

        free(sensors[i]);

    }

    free(sensors);

    return 0;

}

```

9.

```

#include <stdio.h>

#include <stdlib.h>

#define MAX_LOG_ENTRIES 5

struct LogMetadata {

    const char *date;

    const char *author;

};

void displayLogEntry(struct LogMetadata *metadata, const char *logEntry) {

    printf("Date: %s\n", metadata->date);

    printf("Author: %s\n", metadata->author);

    printf("Log: %s\n", logEntry);

}

int main() {

    struct LogMetadata **logEntries;

    const char *logs[] = {

```

```

        "Ship departed from port.",
        "Ship arrived at destination."
    };

    int logCount = 2;

    logEntries = (struct LogMetadata **)malloc(logCount * sizeof(struct LogMetadata *));

    logEntries[0] = (struct LogMetadata *)malloc(sizeof(struct LogMetadata));

    logEntries[0]->date = "2025-01-22";

    logEntries[0]->author = "Captain";

    logEntries[1] = (struct LogMetadata *)malloc(sizeof(struct LogMetadata));

    logEntries[1]->date = "2025-01-23";

    logEntries[1]->author = "Engineer";

    for (int i = 0; i < logCount; i++) {

        displayLogEntry(logEntries[i], logs[i]);

    }

    for (int i = 0; i < logCount; i++) {

        free(logEntries[i]);

    }

    free(logEntries);

    return 0;

}

```

10.

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

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

```

#include <string.h>

#define MAX_WAYPOINTS 5

struct NavigationDetails {

    const char *ID;

    const char *waypoints[MAX_WAYPOINTS];

};

void displayWaypoint(struct NavigationDetails *navDetails) {

    printf("Navigation ID: %s\n", navDetails->ID);

    printf("Waypoints:\n");

    for (int i = 0; i < MAX_WAYPOINTS; i++) {

        if (navDetails->waypoints[i] != NULL) {

            printf("  - %s\n", navDetails->waypoints[i]);

        }

    }

}

int main() {

    struct NavigationDetails **navigation;

    int waypointCount = 2;

    navigation = (struct NavigationDetails **)malloc(waypointCount * sizeof(struct
NavigationDetails *));

    navigation[0] = (struct NavigationDetails *)malloc(sizeof(struct NavigationDetails));

    navigation[0]->ID = "N001";

    navigation[0]->waypoints[0] = "Waypoint 1";

    navigation[0]->waypoints[1] = "Waypoint 2";

```

```

navigation[1] = (struct NavigationDetails *)malloc(sizeof(struct NavigationDetails));

navigation[1]->ID = "N002";

navigation[1]->waypoints[0] = "Waypoint A";

navigation[1]->waypoints[1] = "Waypoint B";

for (int i = 0; i < waypointCount; i++) {

    displayWaypoint(navigation[i]);

}

for (int i = 0; i < waypointCount; i++) {

    free(navigation[i]);

}

free(navigation);

return 0;

}

```

11.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_OBSERVATIONS 5

struct AnimalData {

    const char *species;

    const char *ID;

    const char *location;

```

```

};

void displayAnimalData(struct AnimalData *animal) {

    printf("Species: %s\n", animal->species);

    printf("ID: %s\n", animal->ID);

    printf("Location: %s\n", animal->location);

}

int main() {

    struct AnimalData **animals;

    int animalCount = 2;

    animals = (struct AnimalData **)malloc(animalCount * sizeof(struct AnimalData *));

    animals[0] = (struct AnimalData *)malloc(sizeof(struct AnimalData));

    animals[0]->species = "Dolphin";

    animals[0]->ID = "A001";

    animals[0]->location = "Ocean A";

    animals[1] = (struct AnimalData *)malloc(sizeof(struct AnimalData));

    animals[1]->species = "Shark";

    animals[1]->ID = "A002";

    animals[1]->location = "Ocean B";

    for (int i = 0; i < animalCount; i++) {

        displayAnimalData(animals[i]);

    }

    for (int i = 0; i < animalCount; i++) {

        free(animals[i]);

```

```

    }

    free(animals);

return 0;

}

12.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_SIGNALS 3

// Structure to store beacon data

struct BeaconMetadata {

    const char *ID;

    const char *type;

    const char *location;

    union {

        float range; // for Radar type

        const char *signalType; // for Light type

    };

    char signals[MAX_SIGNALS][50]; // Array to store signal data

};

void displayBeacon(struct BeaconMetadata *beacon) {

    printf("Beacon ID: %s\n", beacon->ID);

    printf("Type: %s\n", beacon->type);

```

```

printf("Location: %s\n", beacon->location);

for (int i = 0; i < MAX_SIGNALS; i++) {

    if (beacon->signals[i][0] != '\0') { // Check if signal exists

        printf("Signal %d: %s\n", i + 1, beacon->signals[i]);

    }

}

if (beacon->type != NULL && strcmp(beacon->type, "Light") == 0) {

    printf("Signal Type: %s\n", beacon->signalType);

} else {

    printf("Range: %.2f km\n", beacon->range);

}

}

int main() {

    int beaconCount = 2;

    struct BeaconMetadata *beacons = (struct BeaconMetadata *)malloc(beaconCount *
sizeof(struct BeaconMetadata *));

    for (int i = 0; i < beaconCount; i++) {

        beacons[i] = (struct BeaconMetadata *)malloc(sizeof(struct BeaconMetadata));

    }

    beacons[0]->ID = "B001";

    beacons[0]->type = "Radar";

    beacons[0]->location = "Coastal Region A";

    beacons[0]->range = 50.0;

    strcpy(beacons[0]->signals[0], "Signal 1");

```



```

strcpy(beacons[0]->signals[1], "Signal 2");

beacons[1]->ID = "B002";

beacons[1]->type = "Light";

beacons[1]->location = "Coastal Region B";

beacons[1]->signalType = "Flashing";

strcpy(beacons[1]->signals[0], "Signal X");

strcpy(beacons[1]->signals[1], "Signal Y");

for (int i = 0; i < beaconCount; i++) {

    displayBeacon(beacons[i]);

}

for (int i = 0; i < beaconCount; i++) {

    free(beacons[i]);

}

free(beacons);

return 0;

}

```

13.

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_LOGS 5

struct FuelData {

    const char *type;

```

```
    float quantity;

};

void displayFuelData(struct FuelData *fuelData) {

    printf("Fuel Type: %s\n", fuelData->type);

    printf("Quantity: %.2f liters\n", fuelData->quantity);

}

int main() {

    struct FuelData **fuelLogs;

    int logCount = 2;

    fuelLogs = (struct FuelData **)malloc(logCount * sizeof(struct FuelData *));

    fuelLogs[0] = (struct FuelData *)malloc(sizeof(struct FuelData));

    fuelLogs[0]->type = "Diesel";

    fuelLogs[0]->quantity = 1500.0;

    fuelLogs[1] = (struct FuelData *)malloc(sizeof(struct FuelData));

    fuelLogs[1]->type = "Petrol";

    fuelLogs[1]->quantity = 800.0;

    for (int i = 0; i < logCount; i++) {

        displayFuelData(fuelLogs[i]);

    }

    for (int i = 0; i < logCount; i++) {

        free(fuelLogs[i]);

    }

    free(fuelLogs);

}
```

```

    return 0;

}

14.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_ALERTS 3

struct ResponseDetails {

    const char *ID;

    const char *location;

    const char *type;

};

void displayResponse(struct ResponseDetails *response) {

    printf("Response ID: %s\n", response->ID);

    printf("Location: %s\n", response->location);

    printf("Type: %s\n", response->type);

}

int main() {

    struct ResponseDetails **alerts;

    const char *alertMessages[MAX_ALERTS] = {

        "Fire on board",

        "Engine failure",

```

```

    "Medical emergency"
};

int alertCount = 3;

alerts = (struct ResponseDetails **)malloc(alertCount * sizeof(struct ResponseDetails *));

    alerts[0] = (struct ResponseDetails *)malloc(sizeof(struct ResponseDetails));

alerts[0]->ID = "R001";

alerts[0]->location = "Ship A";

alerts[0]->type = "Fire";

alerts[1] = (struct ResponseDetails *)malloc(sizeof(struct ResponseDetails));

    alerts[1]->ID = "R002";

alerts[1]->location = "Ship B";

    alerts[1]->type = "Engine failure";  alerts[2] = (struct ResponseDetails *)malloc(sizeof(struct
ResponseDetails));

    alerts[2]->ID = "R003";

alerts[2]->location = "Ship C";

alerts[2]->type = "Medical emergency";

for (int i = 0; i < alertCount; i++) {

    displayResponse(alerts[i]);

    printf("Alert Message: %s\n\n", alertMessages[i]);

}

for (int i = 0; i < alertCount; i++) {

    free(alerts[i]);

}

free(alerts);

```

```

    return 0;

}

15. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_METRICS 5

struct ShipSpecifications {

    float speed;

    int capacity;

};

union PerformanceFactors {

    float weatherImpact;

    float cargoWeight;

};

void displayPerformance(struct ShipSpecifications *shipSpec, union PerformanceFactors
*factor, float *metrics) {

    printf("Speed: %.2f knots\n", shipSpec->speed);

    printf("Capacity: %d tons\n", shipSpec->capacity);

    for (int i = 0; i < MAX_METRICS; i++) {

        printf("Metric %d: %.2f\n", i + 1, metrics[i]);

    }

    printf("Weather Impact: %.2f%%\n", factor->weatherImpact);

}

```

```

int main() {

    struct ShipSpecifications **ships;

    union PerformanceFactors *factors;

    float performanceMetrics[MAX_METRICS] = {90.5, 85.3, 88.7, 92.1, 87.2};

    int shipCount = 2;

    ships = (struct ShipSpecifications **)malloc(shipCount * sizeof(struct ShipSpecifications *));

    factors = (union PerformanceFactors *)malloc(shipCount * sizeof(union
PerformanceFactors));

    ships[0] = (struct ShipSpecifications *)malloc(sizeof(struct ShipSpecifications));

    ships[0]->speed = 25.5;

    ships[0]->capacity = 500;

    factors[0].weatherImpact = 10.5;

    ships[1] = (struct ShipSpecifications *)malloc(sizeof(struct ShipSpecifications));

    ships[1]->speed = 20.0;

    ships[1]->capacity = 700;

    factors[1].weatherImpact = 8.0;

    for (int i = 0; i < shipCount; i++) {

        displayPerformance(ships[i], &factors[i], performanceMetrics);

    }

    for (int i = 0; i < shipCount; i++) {

        free(ships[i]);

    }

    free(ships);

```

```

    free(factors);

    return 0;

}

16.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SCHEDULES 5

struct Port {
    const char *ID;
    int capacity;
    const char *location;
};

struct DockingSchedule {
    const char *scheduleID;
    const char *vesselName;
};

void displayPortSchedule(struct Port *port, struct DockingSchedule **schedule, int
scheduleCount) {
    printf("Port ID: %s\n", port->ID);
    printf("Capacity: %d\n", port->capacity);
    printf("Location: %s\n", port->location);
    for (int i = 0; i < scheduleCount; i++) {
        printf("Schedule %d: Vessel: %s, Schedule ID: %s\n", i + 1, schedule[i]->vesselName,
schedule[i]->scheduleID);
    }
}

int main() {

```

```

int scheduleCount = 3;

struct Port port = {"P001", 10, "New York Harbor"};

struct DockingSchedule *schedule = (struct DockingSchedule *)malloc(scheduleCount *
sizeof(struct DockingSchedule *));

    if (schedule == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

for (int i = 0; i < scheduleCount; i++) {
    schedule[i] = (struct DockingSchedule *)malloc(sizeof(struct DockingSchedule));

        if (schedule[i] == NULL) {
            printf("Memory allocation for schedule %d failed.\n", i);
            return 1;
        }
    }

schedule[0]->scheduleID = "S001";

schedule[0]->vesselName = "Vessel1";

schedule[1]->scheduleID = "S002";

schedule[1]->vesselName = "Vessel2";

schedule[2]->scheduleID = "S003";

schedule[2]->vesselName = "Vessel3";

displayPortSchedule(&port, schedule, scheduleCount);

for (int i = 0; i < scheduleCount; i++) {
    free(schedule[i]);
}

free(schedule);

return 0;
}

```


17.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_LOGS 5

struct ExplorationData {
    const char *location;

    float depth;

    const char *timestamp;
};

void displayLogs(struct ExplorationData **logs, int logCount) {
    for (int i = 0; i < logCount; i++) {
        printf("Log %d: Location: %s, Depth: %.2f meters, Timestamp: %s\n",
            i + 1, logs[i]->location, logs[i]->depth, logs[i]->timestamp);
    }
}

int main() {
    int logCount = 3;

    struct ExplorationData *logs = (struct ExplorationData *)malloc(logCount * sizeof(struct
ExplorationData *));

    if (logs == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (int i = 0; i < logCount; i++) {
        logs[i] = (struct ExplorationData *)malloc(sizeof(struct ExplorationData));

        if (logs[i] == NULL) {
            printf("Memory allocation for log %d failed.\n", i);
            return 1;
        }
    }
}
```

```

    }
}

logs[0]->location = "Mariana Trench";
logs[0]->depth = 10994.0;
logs[0]->timestamp = "2025-01-22 12:00:00";
logs[1]->location = "Great Barrier Reef";
logs[1]->depth = 100.0;
logs[1]->timestamp = "2025-01-22 12:30:00";
logs[2]->location = "Atlantic Ocean";
logs[2]->depth = 5000.0;
logs[2]->timestamp = "2025-01-22 13:00:00";
displayLogs(logs, logCount);
for (int i = 0; i < logCount; i++) {
    free(logs[i]);
}
free(logs);
return 0;
}

```

18.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_MESSAGES 5
struct CommunicationMetadata {
    const char *ID;
    const char *timestamp;
};
struct MessageLog {

```

```

    const char *message;
};

void displayMessages(struct CommunicationMetadata *metadata, struct MessageLog **logs, int
messageCount) {

    printf("Communication ID: %s\n", metadata->ID);
    printf("Timestamp: %s\n", metadata->timestamp);
    for (int i = 0; i < messageCount; i++) {
        printf("Message %d: %s\n", i + 1, logs[i]->message);
    }
}

int main() {
    int messageCount = 3;
    struct CommunicationMetadata metadata = {"C001", "2025-01-22 14:00:00"};
    struct MessageLog *logs = (struct MessageLog *)malloc(messageCount * sizeof(struct
MessageLog *));
    if (logs == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
    for (int i = 0; i < messageCount; i++) {
        logs[i] = (struct MessageLog *)malloc(sizeof(struct MessageLog));
        if (logs[i] == NULL) {
            printf("Memory allocation for message %d failed.\n", i);
            return 1;
        }
    }
    logs[0]->message = "Message 1";
    logs[1]->message = "Message 2";
    logs[2]->message = "Message 3";
}

```

```

    displayMessages(&metadata, logs, messageCount);
    for (int i = 0; i < messageCount; i++) {
        free(logs[i]);
    }
    free(logs);
    return 0;
}

```

19.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_CATCH_RECORDS 5

struct VesselDetails {
    const char *ID;
    const char *name;
};

union CatchData {
    const char *species;
    float weight;
};

struct CatchRecord {
    const char *vesselID;
    union CatchData catchData;
};

void displayCatchRecords(struct VesselDetails *vessel, struct CatchRecord **records, int
recordCount) {
    printf("Vessel ID: %s\n", vessel->ID);
    printf("Vessel Name: %s\n", vessel->name);
    for (int i = 0; i < recordCount; i++) {

```

```

        printf("Catch Record %d: Vessel ID: %s, ", i + 1, records[i]->vesselID);
        printf("Catch Species: %s\n", records[i]->catchData.species);
    }
}

int main() {
    int recordCount = 3;

    struct VesselDetails vessel = {"V001", "Fisher1"};

    struct CatchRecord *records = (struct CatchRecord *)malloc(recordCount * sizeof(struct
CatchRecord *));

    if (records == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (int i = 0; i < recordCount; i++) {
        records[i] = (struct CatchRecord *)malloc(sizeof(struct CatchRecord));

        if (records[i] == NULL) {
            printf("Memory allocation for record %d failed.\n", i);
            return 1;
        }
    }

    records[0]->vesselID = "V001";
    records[0]->catchData.species = "Tuna";
    records[1]->vesselID = "V001";
    records[1]->catchData.species = "Salmon";
    records[2]->vesselID = "V001";
    records[2]->catchData.species = "Cod";
    displayCatchRecords(&vessel, records, recordCount);

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

```

```

    }
    free(records);
    return 0;
}
20.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_DEPTH_LOGS 5
struct NavigationData {
    const char *location;
    float depth;
};
union EnvironmentalConditions {
    float temperature;
    float pressure;
};
struct DepthLog {
    float depth;
    union EnvironmentalConditions envConditions;
};
void displayDepthLogs(struct NavigationData *navData, struct DepthLog **logs, int logCount)
{
    printf("Navigation Location: %s\n", navData->location);
    printf("Navigation Depth: %.2f meters\n", navData->depth);
    for (int i = 0; i < logCount; i++) {
        printf("Depth Log %d: Depth: %.2f meters, Temperature: %.2f°C\n",
            i + 1, logs[i]->depth, logs[i]->envConditions.temperature);
    }
}

```

```

}

int main() {
    int logCount = 3;
    struct NavigationData navData = {"Ocean Floor", 10000.0};
    struct DepthLog *logs = (struct DepthLog *)malloc(logCount * sizeof(struct DepthLog *));
    if (logs == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
    for (int i = 0; i < logCount; i++) {
        logs[i] = (struct DepthLog *)malloc(sizeof(struct DepthLog));

        if (logs[i] == NULL) {
            printf("Memory allocation for log %d failed.\n", i);
            return 1;
        }
    }
    logs[0]->depth = 10000.0;
    logs[0]->envConditions.temperature = -2.5;
    logs[1]->depth = 10020.0;
    logs[1]->envConditions.temperature = -2.8;
    logs[2]->depth = 10050.0;
    logs[2]->envConditions.temperature = -3.0;
    displayDepthLogs(&navData, logs, logCount);
    for (int i = 0; i < logCount; i++) {
        free(logs[i]);
    }
    free(logs);
}

```

```
return 0;
```

```
}
```