**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;
* };
* struct Item {
* const char \*id;
* char name[30];
* char category[20];
* union Attributes attr;
* int isWeight; // 1 for weight, 0 for volume
* };
* void addItem(struct Item \*\*inventory, int \*size, const char \*id, const char \*name, const char \*category, float attrValue, int isWeight) {
* \*inventory = realloc(\*inventory, (\*size + 1) \* sizeof(struct Item));
* (\*inventory)[\*size].id = id;
* strcpy((\*inventory)[\*size].name, name);
* strcpy((\*inventory)[\*size].category, category);
* if (isWeight)
* (\*inventory)[\*size].attr.weight = attrValue;
* else
* (\*inventory)[\*size].attr.volume = attrValue;
* (\*inventory)[\*size].isWeight = isWeight;
* (\*size)++;
* }
* void displayInventory(struct Item \*inventory, int size) {
* for (int i = 0; i < size; i++) {
* printf("ID: %s, Name: %s, Category: %s, ", inventory[i].id, inventory[i].name, inventory[i].category);
* if (inventory[i].isWeight)
* printf("Weight: %.2f kg\n", inventory[i].attr.weight);
* else
* printf("Volume: %.2f liters\n", inventory[i].attr.volume);
* }
* }
* int main() {
* struct Item \*inventory = NULL;
* int size = 0;
* addItem(&inventory, &size, "101", "Sugar", "Grocery", 5.0, 1);
* addItem(&inventory, &size, "102", "Milk", "Dairy", 2.0, 0);
* printf("Inventory:\n");
* displayInventory(inventory, size);
* free(inventory);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* union TransportMode {
* const char \*air;
* const char \*sea;
* const char \*land;
* };
* struct Route {
* const char \*id;   // Immutable route ID
* const char \*start;
* const char \*end;
* union TransportMode mode;
* int modeType;     // 1 for air, 2 for sea, 3 for land
* };
* void addRoute(struct Route \*\*routes, int \*size, const char \*id, const char \*start, const char \*end, const char \*mode, int modeType) {
* \*routes = realloc(\*routes, (\*size + 1) \* sizeof(struct Route));
* (\*routes)[\*size].id = id;
* (\*routes)[\*size].start = start;
* (\*routes)[\*size].end = end;
* (\*routes)[\*size].modeType = modeType;
* if (modeType == 1)
* (\*routes)[\*size].mode.air = mode;
* else if (modeType == 2)
* (\*routes)[\*size].mode.sea = mode;
* else
* (\*routes)[\*size].mode.land = mode;
* (\*size)++;
* }
* void displayRoutes(struct Route \*routes, int size) {
* for (int i = 0; i < size; i++) {
* printf("Route ID: %s, Start: %s, End: %s, ", routes[i].id, routes[i].start, routes[i].end);
* if (routes[i].modeType == 1)
* printf("Mode: Air (%s)\n", routes[i].mode.air);
* else if (routes[i].modeType == 2)
* printf("Mode: Sea (%s)\n", routes[i].mode.sea);
* else
* printf("Mode: Land (%s)\n", routes[i].mode.land);
* }
* }
* int main() {
* struct Route \*routes = NULL;
* int size = 0;
* addRoute(&routes, &size, "R001", "New York", "London", "Boeing 747", 1);
* addRoute(&routes, &size, "R002", "Mumbai", "Singapore", "Cargo Ship", 2);
* addRoute(&routes, &size, "R003", "Berlin", "Paris", "Truck", 3);
* printf("Shipping Routes:\n");
* displayRoutes(routes, size);
* free(routes);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* union VehicleStatus {
* const char \*active;
* const char \*maintenance;
* };
* struct Vehicle {
* const char \*id;   // Immutable vehicle ID
* const char \*type;
* union VehicleStatus status;
* int statusType;   // 1 for active, 2 for maintenance
* };
* void addVehicle(struct Vehicle \*\*vehicles, int \*size, const char \*id, const char \*type, const char \*status, int statusType) {
* \*vehicles = realloc(\*vehicles, (\*size + 1) \* sizeof(struct Vehicle));
* (\*vehicles)[\*size].id = id;
* (\*vehicles)[\*size].type = type;
* (\*vehicles)[\*size].statusType = statusType;
* if (statusType == 1)
* (\*vehicles)[\*size].status.active = status;
* else
* (\*vehicles)[\*size].status.maintenance = status;
* (\*size)++;
* }
* void displayVehicles(struct Vehicle \*vehicles, int size) {
* for (int i = 0; i < size; i++) {
* printf("Vehicle ID: %s, Type: %s, ", vehicles[i].id, vehicles[i].type);
* if (vehicles[i].statusType == 1)
* printf("Status: Active (%s)\n", vehicles[i].status.active);
* else
* printf("Status: Maintenance (%s)\n", vehicles[i].status.maintenance);
* }
* }
* int main() {
* struct Vehicle \*vehicles = NULL;
* int size = 0;
* addVehicle(&vehicles, &size, "V001", "Truck", "Available for duty", 1);
* addVehicle(&vehicles, &size, "V002", "Bus", "Under repair", 2);
* addVehicle(&vehicles, &size, "V003", "Car", "Operational", 1);
* printf("Fleet Details:\n");
* displayVehicles(vehicles, size);
* free(vehicles);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* union PaymentMethod {
* const char \*creditCard;
* const char \*cash;
* };
* struct Order {
* const char \*id;   // Immutable order ID
* const char \*customer;
* const char \*items;
* union PaymentMethod payment;
* int paymentType;  // 1 for credit card, 2 for cash
* };
* void enqueue(struct Order \*\*queue, int \*size, const char \*id, const char \*customer, const char \*items, const char \*payment, int paymentType) {
* \*queue = realloc(\*queue, (\*size + 1) \* sizeof(struct Order));
* (\*queue)[\*size].id = id;
* (\*queue)[\*size].customer = customer;
* (\*queue)[\*size].items = items;
* (\*queue)[\*size].paymentType = paymentType;
* if (paymentType == 1)
* (\*queue)[\*size].payment.creditCard = payment;
* else
* (\*queue)[\*size].payment.cash = payment;
* (\*size)++;
* }
* void dequeue(struct Order \*\*queue, int \*size) {
* if (\*size == 0) {
* printf("Queue is empty!\n");
* return;
* }
* printf("Processing Order ID: %s\n", (\*queue)[0].id);
* for (int i = 1; i < \*size; i++) {
* (\*queue)[i - 1] = (\*queue)[i];
* }
* \*queue = realloc(\*queue, (\*size - 1) \* sizeof(struct Order));
* (\*size)--;
* }
* void displayOrders(struct Order \*queue, int size) {
* if (size == 0) {
* printf("No orders in the queue.\n");
* return;
* }
* for (int i = 0; i < size; i++) {
* printf("Order ID: %s, Customer: %s, Items: %s, Payment: ", queue[i].id, queue[i].customer, queue[i].items);
* if (queue[i].paymentType == 1)
* printf("Credit Card (%s)\n", queue[i].payment.creditCard);
* else
* printf("Cash (%s)\n", queue[i].payment.cash);
* }
* }
* int main() {
* struct Order \*orderQueue = NULL;
* int size = 0;
* enqueue(&orderQueue, &size, "O001", "Alice", "Laptop, Mouse", "Visa \*\*\*\*1234", 1);
* enqueue(&orderQueue, &size, "O002", "Bob", "Phone, Charger", "Paid in Cash", 2);
* enqueue(&orderQueue, &size, "O003", "Charlie", "Monitor", "MasterCard \*\*\*\*5678", 1);
* printf("Current Orders:\n");
* displayOrders(orderQueue, size);
* printf("\nDequeueing an Order...\n");
* dequeue(&orderQueue, &size);
* printf("\nRemaining Orders:\n");
* displayOrders(orderQueue, size);
* free(orderQueue);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* union TrackingEvent {
* const char \*dispatched;
* const char \*delivered;
* };
* struct Shipment {
* const char \*trackingNumber; // Immutable tracking number
* const char \*origin;
* const char \*destination;
* union TrackingEvent event;
* int eventType; // 1 for dispatched, 2 for delivered
* };
* void addShipment(struct Shipment \*\*list, int \*size, const char \*trackingNumber, const char \*origin, const char \*destination, const char \*event, int eventType) {
* \*list = realloc(\*list, (\*size + 1) \* sizeof(struct Shipment));
* (\*list)[\*size].trackingNumber = trackingNumber;
* (\*list)[\*size].origin = origin;
* (\*list)[\*size].destination = destination;
* (\*list)[\*size].eventType = eventType;
* if (eventType == 1)
* (\*list)[\*size].event.dispatched = event;
* else
* (\*list)[\*size].event.delivered = event;
* (\*size)++;
* }
* void removeShipment(struct Shipment \*\*list, int \*size, const char \*trackingNumber) {
* if (\*size == 0) {
* printf("No shipments in the list.\n");
* return;
* }
* int found = 0;
* for (int i = 0; i < \*size; i++) {
* if (strcmp((\*list)[i].trackingNumber, trackingNumber) == 0) {
* printf("Removing Shipment: %s\n", (\*list)[i].trackingNumber);
* for (int j = i; j < \*size - 1; j++) {
* (\*list)[j] = (\*list)[j + 1];
* }
* \*list = realloc(\*list, (\*size - 1) \* sizeof(struct Shipment));
* (\*size)--;
* found = 1;
* break;
* }
* }
* if (!found) {
* printf("Shipment with tracking number %s not found.\n", trackingNumber);
* }
* }
* void displayShipments(struct Shipment \*list, int size) {
* if (size == 0) {
* printf("No shipments available.\n");
* return;
* }
* for (int i = 0; i < size; i++) {
* printf("Tracking Number: %s, Origin: %s, Destination: %s, Event: ",
* list[i].trackingNumber, list[i].origin, list[i].destination);
* if (list[i].eventType == 1)
* printf("Dispatched (%s)\n", list[i].event.dispatched);
* else
* printf("Delivered (%s)\n", list[i].event.delivered);
* }
* }
* int main() {
* struct Shipment \*shipmentList = NULL;
* int size = 0;
* addShipment(&shipmentList, &size, "TN001", "New York", "Los Angeles", "On Truck", 1);
* addShipment(&shipmentList, &size, "TN002", "Chicago", "Houston", "Arrived at Hub", 2);
* addShipment(&shipmentList, &size, "TN003", "San Francisco", "Seattle", "Out for Delivery", 1);
* printf("Current Shipments:\n");
* displayShipments(shipmentList, size);
* printf("\nRemoving a Shipment...\n");
* removeShipment(&shipmentList, &size, "TN002");
* printf("\nRemaining Shipments:\n");
* displayShipments(shipmentList, size);
* free(shipmentList);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct TrafficNode {
* int id;
* char location[50];
* };
* union TrafficCondition {
* char clear[10];
* char congested[15];
* };
* void manageTrafficData(struct TrafficNode\*\* nodes, int\* count) {
* struct TrafficNode\* newNode = (struct TrafficNode\*)malloc(sizeof(struct TrafficNode));
* if (!newNode) {
* printf("Memory allocation failed.\n");
* return;
* }
* // Adding traffic node details
* newNode->id = \*count;
* snprintf(newNode->location, sizeof(newNode->location), "Location %d", \*count);
* // Assigning traffic condition using union
* union TrafficCondition condition;
* if (\*count % 2 == 0) {
* snprintf(condition.clear, sizeof(condition.clear), "Clear");
* } else {
* snprintf(condition.congested, sizeof(condition.congested), "Congested");
* }
* printf("Node ID: %d, Location: %s, Traffic Condition: %s\n", newNode->id, newNode->location, condition.clear);
* // Update dynamic list of traffic nodes
* (\*nodes) = realloc(\*nodes, (\*count + 1) \* sizeof(struct TrafficNode));
* (\*nodes)[\*count] = \*newNode;
* (\*count)++;
* }
* int main() {
* struct TrafficNode\* trafficNodes = NULL;
* int nodeCount = 0;
* // Adding traffic data
* for (int i = 0; i < 5; i++) {
* manageTrafficData(&trafficNodes, &nodeCount);
* }
* free(trafficNodes);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct Slot {
* int id;
* char location[50];
* float size;
* };
* union ItemType {
* char perishable[15];
* char nonPerishable[15];
* };
* void allocateSlot(struct Slot\*\* slots, int\* count) {
* struct Slot\* newSlot = (struct Slot\*)malloc(sizeof(struct Slot));
* if (!newSlot) {
* printf("Memory allocation failed.\n");
* return;
* }
* // Adding slot details
* newSlot->id = \*count;
* snprintf(newSlot->location, sizeof(newSlot->location), "Location %d", \*count);
* newSlot->size = 100.0 + \*count; // Example size
* // Assigning item type using union
* union ItemType itemType;
* if (\*count % 2 == 0) {
* snprintf(itemType.perishable, sizeof(itemType.perishable), "Perishable");
* } else {
* snprintf(itemType.nonPerishable, sizeof(itemType.nonPerishable), "Non-Perishable");
* }
* printf("Slot ID: %d, Location: %s, Size: %.2f, Item Type: %s\n", newSlot->id, newSlot->location, newSlot->size, itemType.perishable);
* // Update dynamic list of slots
* (\*slots) = realloc(\*slots, (\*count + 1) \* sizeof(struct Slot));
* (\*slots)[\*count] = \*newSlot;
* (\*count)++;
* }
* int main() {
* struct Slot\* warehouseSlots = NULL;
* int slotCount = 0;
* // Allocating slots
* for (int i = 0; i < 5; i++) {
* allocateSlot(&warehouseSlots, &slotCount);
* }
* // Free dynamically allocated memory
* free(warehouseSlots);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct Package {
* int id;
* float weight;
* char destination[50];
* };
* union DeliveryMethod {
* char standard[10];
* char express[10];
* };
* void allocatePackage(struct Package\*\* packages, int\* count) {
* struct Package\* newPackage = (struct Package\*)malloc(sizeof(struct Package));
* if (!newPackage) {
* printf("Memory allocation failed.\n");
* return;
* }
* // Adding package details
* newPackage->id = \*count;
* newPackage->weight = 5.0 + \*count; // Example weight
* snprintf(newPackage->destination, sizeof(newPackage->destination), "Destination %d", \*count);
* // Assigning delivery method using union
* union DeliveryMethod deliveryMethod;
* if (\*count % 2 == 0) {
* snprintf(deliveryMethod.standard, sizeof(deliveryMethod.standard), "Standard");
* } else {
* snprintf(deliveryMethod.express, sizeof(deliveryMethod.express), "Express");
* }
* printf("Package ID: %d, Weight: %.2f, Destination: %s, Delivery Method: %s\n",
* newPackage->id, newPackage->weight, newPackage->destination, deliveryMethod.standard);
* // Update dynamic list of packages
* (\*packages) = realloc(\*packages, (\*count + 1) \* sizeof(struct Package));
* (\*packages)[\*count] = \*newPackage;
* (\*count)++;
* }
* int main() {
* struct Package\* deliveryPackages = NULL;
* int packageCount = 0;
* // Allocating packages
* for (int i = 0; i < 5; i++) {
* allocatePackage(&deliveryPackages, &packageCount);
* }
* // Free dynamically allocated memory
* free(deliveryPackages);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct AnalyticsRecord {
* char timestamp[20];
* union Metric {
* float speed;
* float efficiency;
* } metric;
* };
* void addAnalyticsRecord(struct AnalyticsRecord\*\*\* records, int\* count) {
* struct AnalyticsRecord\* newRecord = (struct AnalyticsRecord\*)malloc(sizeof(struct AnalyticsRecord));
* if (!newRecord) {
* printf("Memory allocation failed.\n");
* return;
* }
* // Assign timestamp (for simplicity, using a dummy timestamp here)
* snprintf(newRecord->timestamp, sizeof(newRecord->timestamp), "2025-01-22 %d:00", \*count);
* // Choose metric (alternating between speed and efficiency)
* if (\*count % 2 == 0) {
* newRecord->metric.speed = 50.0 + \*count; // Example speed in km/h
* } else {
* newRecord->metric.efficiency = 85.0 + \*count; // Example efficiency
* }
* // Dynamically manage the array of records
* (\*records) = realloc(\*records, (\*count + 1) \* sizeof(struct AnalyticsRecord\*));
* (\*records)[\*count] = newRecord;
* (\*count)++;
* // Display the record
* printf("Timestamp: %s, ", newRecord->timestamp);
* if (\*count % 2 == 0) {
* printf("Speed: %.2f km/h\n", newRecord->metric.speed);
* } else {
* printf("Efficiency: %.2f%%\n", newRecord->metric.efficiency);
* }
* }
* int main() {
* struct AnalyticsRecord\*\* dataRecords = NULL;
* int recordCount = 0;
* // Adding data records
* for (int i = 0; i < 5; i++) {
* addAnalyticsRecord(&dataRecords, &recordCount);
* }
* // Free dynamically allocated memory
* for (int i = 0; i < recordCount; i++) {
* free(dataRecords[i]);
* }
* free(dataRecords);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct Schedule {
* const char\* scheduleID; // Immutable schedule ID
* int startTime;  // Start time (example: hour in 24-hour format)
* int endTime;    // End time
* };
* union TransportType {
* char bus[20];  // Transport type: Bus
* char truck[20]; // Transport type: Truck
* };
* struct TransportSchedule {
* struct Schedule scheduleDetails;  // Schedule details
* union TransportType transport;    // Transport type (Bus/Truck)
* };
* void addSchedule(struct TransportSchedule\*\* schedules, int\* count, const char\* scheduleID, int startTime, int endTime, const char\* transportType) {
* // Resize array for new schedule
* \*schedules = realloc(\*schedules, (\*count + 1) \* sizeof(struct TransportSchedule));
* struct TransportSchedule\* newSchedule = &(\*schedules)[\*count];
* // Assign schedule details
* newSchedule->scheduleDetails.scheduleID = scheduleID;
* newSchedule->scheduleDetails.startTime = startTime;
* newSchedule->scheduleDetails.endTime = endTime;
* // Assign transport type
* if (transportType == "bus") {
* for (int i = 0; i < 20 && transportType[i] != '\0'; i++) {
* newSchedule->transport.bus[i] = transportType[i];
* }
* } else if (transportType == "truck") {
* for (int i = 0; i < 20 && transportType[i] != '\0'; i++) {
* newSchedule->transport.truck[i] = transportType[i];
* }
* }
* // Increment schedule count
* (\*count)++;
* }
* void displaySchedule(struct TransportSchedule\* schedule) {
* printf("Schedule ID: %s\n", schedule->scheduleDetails.scheduleID);
* printf("Start Time: %d:00\n", schedule->scheduleDetails.startTime);
* printf("End Time: %d:00\n", schedule->scheduleDetails.endTime);
* // Display transport type
* if (schedule->transport.bus[0] != '\0') {
* printf("Transport Type: Bus\n");
* } else if (schedule->transport.truck[0] != '\0') {
* printf("Transport Type: Truck\n");
* }
* }
* int main() {
* struct TransportSchedule\* schedules = NULL;
* int scheduleCount = 0;
* // Adding schedules
* addSchedule(&schedules, &scheduleCount, "SCH001", 9, 11, "bus");
* addSchedule(&schedules, &scheduleCount, "SCH002", 12, 14, "truck");
* // Display all schedules
* for (int i = 0; i < scheduleCount; i++) {
* displaySchedule(&schedules[i]);
* }
* // Free allocated memory
* free(schedules);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct SupplierCustomer {
* int id; // Unique ID for supplier/customer
* char name[50]; // Name of supplier or customer
* };
* union TransactionType {
* char purchase[20];  // For purchase transactions
* char returnType[20]; // For return transactions
* };
* struct SupplyChain {
* const char\* transactionID;  // Immutable transaction ID
* struct SupplierCustomer entity; // Supplier or customer details
* union TransactionType transaction; // Transaction details
* };
* void addTransaction(struct SupplyChain\*\* transactions, int\* count, const char\* id, int entityId, const char\* entityName, const char\* transactionType) {
* \*transactions = realloc(\*transactions, (\*count + 1) \* sizeof(struct SupplyChain)); // Resize array for new transaction
* struct SupplyChain\* newTransaction = &(\*transactions)[\*count];
* newTransaction->transactionID = id;
* newTransaction->entity.id = entityId;
* for (int i = 0; i < 50 && entityName[i] != '\0'; i++) {
* newTransaction->entity.name[i] = entityName[i];
* }
* if (transactionType == "purchase") {
* for (int i = 0; i < 20 && transactionType[i] != '\0'; i++) {
* newTransaction->transaction.purchase[i] = transactionType[i];
* }
* } else if (transactionType == "return") {
* for (int i = 0; i < 20 && transactionType[i] != '\0'; i++) {
* newTransaction->transaction.returnType[i] = transactionType[i];
* }
* }
* (\*count)++; // Increment transaction count
* }
* void displayTransaction(struct SupplyChain\* transaction) {
* printf("Transaction ID: %s\n", transaction->transactionID);
* printf("Entity ID: %d\n", transaction->entity.id);
* printf("Entity Name: %s\n", transaction->entity.name);
* if (transaction->transaction.purchase[0] != '\0') {
* printf("Transaction Type: Purchase\n");
* } else if (transaction->transaction.returnType[0] != '\0') {
* printf("Transaction Type: Return\n");
* }
* }
* int main() {
* struct SupplyChain\* transactions = NULL;
* int transactionCount = 0;
* // Adding some transactions
* addTransaction(&transactions, &transactionCount, "TX001", 101, "Ravi", "purchase");
* addTransaction(&transactions, &transactionCount, "TX002", 102, "Priya", "return");
* // Display all transactions
* for (int i = 0; i < transactionCount; i++) {
* displayTransaction(&transactions[i]);
* }
* free(transactions); // Free allocated memory
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct CostComponent {
* const char\* costID;  // Immutable cost ID
* double baseCost;     // Base cost
* };
* union PricingModel {
* double fixedPrice;   // Fixed pricing model
* double variableRate; // Variable pricing model (rate per unit)
* };
* struct FreightCost {
* struct CostComponent costDetails;  // Cost component details
* union PricingModel pricing;        // Pricing model (fixed/variable)
* };
* void addCostRecord(struct FreightCost\*\* costs, int\* count, const char\* costID, double baseCost, const char\* pricingModel, double price) {
* // Resize array for new cost record
* \*costs = realloc(\*costs, (\*count + 1) \* sizeof(struct FreightCost));
* struct FreightCost\* newCost = &(\*costs)[\*count];
* // Assign cost details
* newCost->costDetails.costID = costID;
* newCost->costDetails.baseCost = baseCost;
* // Assign pricing model
* if (pricingModel == "fixed") {
* newCost->pricing.fixedPrice = price;
* } else if (pricingModel == "variable") {
* newCost->pricing.variableRate = price;
* }
* // Increment cost record count
* (\*count)++;
* }
* void displayCost(struct FreightCost\* cost) {
* printf("Cost ID: %s\n", cost->costDetails.costID);
* printf("Base Cost: %.2f\n", cost->costDetails.baseCost);
* // Display pricing model and corresponding price
* if (cost->pricing.fixedPrice != 0.0) {
* printf("Pricing Model: Fixed\n");
* printf("Fixed Price: %.2f\n", cost->pricing.fixedPrice);
* } else if (cost->pricing.variableRate != 0.0) {
* printf("Pricing Model: Variable\n");
* printf("Variable Rate: %.2f\n", cost->pricing.variableRate);
* }
* }
* int main() {
* struct FreightCost\* costs = NULL;
* int costCount = 0;
* // Adding cost records
* addCostRecord(&costs, &costCount, "COST001", 100.0, "fixed", 50.0);
* addCostRecord(&costs, &costCount, "COST002", 120.0, "variable", 30.0);
* // Display all cost records
* for (int i = 0; i < costCount; i++) {
* displayCost(&costs[i]);
* }
* // Free allocated memory
* free(costs);
* return 0;
* }

**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.
* #include <stdio.h>
* #include <stdlib.h>
* struct LoadDetails {
* const char\* loadID;  // Immutable load ID
* double weight;       // Weight of the load
* const char\* destination;  // Destination of the load
* };
* union LoadType {
* const char\* bulk;    // Bulk load type
* const char\* container; // Container load type
* };
* struct VehicleLoad {
* struct LoadDetails loadDetails;  // Load details
* union LoadType loadType;         // Load type (bulk/container)
* };
* void addLoadRecord(struct VehicleLoad\*\* loads, int\* count, const char\* loadID, double weight, const char\* destination, const char\* loadType, const char\* typeDetails) {
* // Resize array for new load record
* \*loads = realloc(\*loads, (\*count + 1) \* sizeof(struct VehicleLoad));
* struct VehicleLoad\* newLoad = &(\*loads)[\*count];
* // Assign load details
* newLoad->loadDetails.loadID = loadID;
* newLoad->loadDetails.weight = weight;
* newLoad->loadDetails.destination = destination;
* // Assign load type
* if (loadType == "bulk") {
* newLoad->loadType.bulk = typeDetails;
* } else if (loadType == "container") {
* newLoad->loadType.container = typeDetails;
* }
* // Increment load record count
* (\*count)++;
* }
* void displayLoad(struct VehicleLoad\* load) {
* printf("Load ID: %s\n", load->loadDetails.loadID);
* printf("Weight: %.2f\n", load->loadDetails.weight);
* printf("Destination: %s\n", load->loadDetails.destination);
* // Display load type
* if (load->loadType.bulk != NULL) {
* printf("Load Type: Bulk\n");
* printf("Bulk Load: %s\n", load->loadType.bulk);
* } else if (load->loadType.container != NULL) {
* printf("Load Type: Container\n");
* printf("Container Type: %s\n", load->loadType.container);
* }
* }
* int main() {
* struct VehicleLoad\* loads = NULL;
* int loadCount = 0;
* // Adding load records
* addLoadRecord(&loads, &loadCount, "LOAD001", 1000.0, "Mumbai", "bulk", "Grains");
* addLoadRecord(&loads, &loadCount, "LOAD002", 2000.0, "Delhi", "container", "Electronics");
* // Display all load records
* for (int i = 0; i < loadCount; i++) {
* displayLoad(&loads[i]);
* }
* // Free allocated memory
* free(loads);
* return 0;
* }

**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>
* struct TransportDetails {
* const char\* transportID;  // Immutable transport ID
* const char\* origin;       // Origin of transport
* const char\* destination;  // Destination of transport
* };
* union TransportMode {
* const char\* rail;    // Rail transport mode
* const char\* road;    // Road transport mode
* };
* struct Transport {
* struct TransportDetails transportDetails;  // Transport details
* union TransportMode transportMode;         // Transport mode (rail/road)
* };
* void addTransportRecord(struct Transport\*\* transports, int\* count, const char\* transportID, const char\* origin, const char\* destination, const char\* mode, const char\* modeDetails) {
* // Resize array for new transport record
* \*transports = realloc(\*transports, (\*count + 1) \* sizeof(struct Transport));
* struct Transport\* newTransport = &(\*transports)[\*count];
* // Assign transport details
* newTransport->transportDetails.transportID = transportID;
* newTransport->transportDetails.origin = origin;
* newTransport->transportDetails.destination = destination;
* // Assign transport mode
* if (mode == "rail") {
* newTransport->transportMode.rail = modeDetails;
* } else if (mode == "road") {
* newTransport->transportMode.road = modeDetails;
* }
* // Increment transport record count
* (\*count)++;
* }
* void displayTransport(struct Transport\* transport) {
* printf("Transport ID: %s\n", transport->transportDetails.transportID);
* printf("Origin: %s\n", transport->transportDetails.origin);
* printf("Destination: %s\n", transport->transportDetails.destination);
* // Display transport mode
* if (transport->transportMode.rail != NULL) {
* printf("Transport Mode: Rail\n");
* printf("Rail Type: %s\n", transport->transportMode.rail);
* } else if (transport->transportMode.road != NULL) {
* printf("Transport Mode: Road\n");
* printf("Road Type: %s\n", transport->transportMode.road);
* }
* }
* int main() {
* struct Transport\* transports = NULL;
* int transportCount = 0;
* // Adding transport records
* addTransportRecord(&transports, &transportCount, "T001", "Mumbai", "Delhi", "rail", "Freight");
* addTransportRecord(&transports, &transportCount, "T002", "Chennai", "Bangalore", "road", "Truck");
* // Display all transport records
* for (int i = 0; i < transportCount; i++) {
* displayTransport(&transports[i]);
* }
* // Free allocated memory
* free(transports);
* return 0;
* }

**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>
* struct PerformanceMetrics {
* const char\* metricID;
* double value;
* };
* union PerformanceAspect {
* double time;
* double cost;
* };
* struct PerformanceRecord {
* struct PerformanceMetrics metrics;
* union PerformanceAspect aspect;
* int isTime;
* };
* // Function to add a performance record
* void addPerformanceRecord(struct PerformanceRecord\*\* records, int\* count, const char\* metricID, double value, const char\* aspectType, double aspectValue) {
* // Resize array for new performance record
* \*records = realloc(\*records, (\*count + 1) \* sizeof(struct PerformanceRecord));
* struct PerformanceRecord\* newRecord = &(\*records)[\*count];
* // Assign metric details
* newRecord->metrics.metricID = metricID;
* newRecord->metrics.value = value;
* // Assign performance aspect (time or cost)
* if (aspectType == "time") {
* newRecord->aspect.time = aspectValue;
* newRecord->isTime = 1;
* } else if (aspectType == "cost") {
* newRecord->aspect.cost = aspectValue;
* newRecord->isTime = 0;
* }
* // Increment record count
* (\*count)++;
* }
* // Function to display performance records
* void displayPerformanceRecord(struct PerformanceRecord\* record) {
* printf("Metric ID: %s\n", record->metrics.metricID);
* printf("Metric Value: %.2f\n", record->metrics.value);
* if (record->isTime) {
* printf("Performance Aspect: Time\n");
* printf("Time Value: %.2f hours\n", record->aspect.time);
* } else {
* printf("Performance Aspect: Cost\n");
* printf("Cost Value: %.2f USD\n", record->aspect.cost);
* }
* }
* int main() {
* struct PerformanceRecord\* records = NULL;
* int recordCount = 0;
* // Adding performance records
* addPerformanceRecord(&records, &recordCount, "M001", 50.0, "time", 5.0);  // Time aspect
* addPerformanceRecord(&records, &recordCount, "M002", 30.0, "cost", 200.0); // Cost aspect
* // Display all performance records
* for (int i = 0; i < recordCount; i++) {
* displayPerformanceRecord(&records[i]);
* }
* // Free dynamically allocated memory
* free(records);
* return 0;
* }

**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>
* struct RobotDetails {
* const char\* robotID;
* const char\* type;
* const char\* status;
* };
* union TaskType {
* const char\* picking;
* const char\* sorting;
* };
* struct TaskAllocation {
* struct RobotDetails robot;
* union TaskType task;
* int isPicking;  // 1 for picking task, 0 for sorting task
* };
* void addTask(struct TaskAllocation\*\* tasks, int\* count, const char\* robotID, const char\* type, const char\* status, const char\* taskType) {
* \*tasks = realloc(\*tasks, (\*count + 1) \* sizeof(struct TaskAllocation));
* struct TaskAllocation\* newTask = &(\*tasks)[\*count];
* newTask->robot.robotID = robotID;
* newTask->robot.type = type;
* newTask->robot.status = status;
* if (taskType == "picking") {
* newTask->task.picking = "picking";
* newTask->isPicking = 1;
* } else {
* newTask->task.sorting = "sorting";
* newTask->isPicking = 0;
* }
* (\*count)++;
* }
* void displayTasks(struct TaskAllocation\* tasks, int count) {
* for (int i = 0; i < count; i++) {
* printf("Robot ID: %s, Type: %s, Status: %s, Task: %s\n",
* tasks[i].robot.robotID,
* tasks[i].robot.type,
* tasks[i].robot.status,
* tasks[i].isPicking ? tasks[i].task.picking : tasks[i].task.sorting);
* }
* }
* int main() {
* struct TaskAllocation\* tasks = NULL;
* int count = 0;
* addTask(&tasks, &count, "Robot001", "Picker", "Idle", "picking");
* addTask(&tasks, &count, "Robot002", "Sorter", "Idle", "sorting");
* addTask(&tasks, &count, "Robot003", "Picker", "Working", "picking");
* displayTasks(tasks, count);
* free(tasks);
* return 0;
* }

**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>
* struct FeedbackDetails {
* const char\* feedbackID;
* const char\* content;
* };
* union FeedbackType {
* const char\* positive;
* const char\* negative;
* };
* struct Feedback {
* struct FeedbackDetails details;
* union FeedbackType type;
* int isPositive; // 1 for positive, 0 for negative
* };
* void addFeedback(struct Feedback\*\* feedbacks, int\* count, const char\* feedbackID, const char\* content, const char\* type) {
* \*feedbacks = realloc(\*feedbacks, (\*count + 1) \* sizeof(struct Feedback));
* struct Feedback\* newFeedback = &(\*feedbacks)[\*count];
* newFeedback->details.feedbackID = feedbackID;
* newFeedback->details.content = content;
* if (type == "positive") {
* newFeedback->type.positive = "positive";
* newFeedback->isPositive = 1;
* } else {
* newFeedback->type.negative = "negative";
* newFeedback->isPositive = 0;
* }
* (\*count)++;
* }
* void displayFeedbacks(struct Feedback\* feedbacks, int count) {
* for (int i = 0; i < count; i++) {
* printf("Feedback ID: %s, Content: %s, Type: %s\n",
* feedbacks[i].details.feedbackID,
* feedbacks[i].details.content,
* feedbacks[i].isPositive ? feedbacks[i].type.positive : feedbacks[i].type.negative);
* }
* }
* int main() {
* struct Feedback\* feedbacks = NULL;
* int count = 0;
* addFeedback(&feedbacks, &count, "F001", "Great service!", "positive");
* addFeedback(&feedbacks, &count, "F002", "Very slow response.", "negative");
* addFeedback(&feedbacks, &count, "F003", "Excellent product quality.", "positive");
* displayFeedbacks(feedbacks, count);
* free(feedbacks);
* return 0;
* }

**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>
* struct FleetDetails {
* const char\* fleetID;
* const char\* location;
* const char\* status;
* };
* union CoordinationType {
* const char\* dispatch;
* const char\* reroute;
* };
* struct Fleet {
* struct FleetDetails details;
* union CoordinationType coordination;
* int isDispatch; // 1 for dispatch, 0 for reroute
* };
* void addFleet(struct Fleet\*\* fleets, int\* count, const char\* fleetID, const char\* location, const char\* status, const char\* coordination) {
* \*fleets = realloc(\*fleets, (\*count + 1) \* sizeof(struct Fleet));
* struct Fleet\* newFleet = &(\*fleets)[\*count];
* newFleet->details.fleetID = fleetID;
* newFleet->details.location = location;
* newFleet->details.status = status;
* if (coordination == "dispatch") {
* newFleet->coordination.dispatch = "dispatch";
* newFleet->isDispatch = 1;
* } else {
* newFleet->coordination.reroute = "reroute";
* newFleet->isDispatch = 0;
* }
* (\*count)++;
* }
* void displayFleetCoordination(struct Fleet\* fleets, int count) {
* for (int i = 0; i < count; i++) {
* printf("Fleet ID: %s, Location: %s, Status: %s, Coordination: %s\n",
* fleets[i].details.fleetID,
* fleets[i].details.location,
* fleets[i].details.status,
* fleets[i].isDispatch ? fleets[i].coordination.dispatch : fleets[i].coordination.reroute);
* }
* }
* int main() {
* struct Fleet\* fleets = NULL;
* int count = 0;
* addFleet(&fleets, &count, "F001", "New York", "Active", "dispatch");
* addFleet(&fleets, &count, "F002", "Los Angeles", "Inactive", "reroute");
* addFleet(&fleets, &count, "F003", "Chicago", "Active", "dispatch");
* displayFleetCoordination(fleets, count);
* free(fleets);
* return 0;
* }

**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>
* struct SecurityEvent {
* const char\* eventID;
* const char\* description;
* };
* union EventType {
* const char\* breach;
* const char\* resolved;
* };
* struct Security {
* struct SecurityEvent event;
* union EventType type;
* int isBreach; // 1 for breach, 0 for resolved
* };
* void addSecurityEvent(struct Security\*\* events, int\* count, const char\* eventID, const char\* description, const char\* type) {
* \*events = realloc(\*events, (\*count + 1) \* sizeof(struct Security));
* struct Security\* newEvent = &(\*events)[\*count];
* newEvent->event.eventID = eventID;
* newEvent->event.description = description;
* if (type == "breach") {
* newEvent->type.breach = "breach";
* newEvent->isBreach = 1;
* } else {
* newEvent->type.resolved = "resolved";
* newEvent->isBreach = 0;
* }
* (\*count)++;
* }
* void displaySecurityEvents(struct Security\* events, int count) {
* for (int i = 0; i < count; i++) {
* printf("Event ID: %s, Description: %s, Event Type: %s\n",
* events[i].event.eventID,
* events[i].event.description,
* events[i].isBreach ? events[i].type.breach : events[i].type.resolved);
* }
* }
* int main() {
* struct Security\* events = NULL;
* int count = 0;
* addSecurityEvent(&events, &count, "E001", "Unauthorized Access", "breach");
* addSecurityEvent(&events, &count, "E002", "Access Resolved", "resolved");
* addSecurityEvent(&events, &count, "E003", "Suspicious Activity", "breach");
* displaySecurityEvents(events, count);
* free(events);
* return 0;
* }

**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>
* struct BillingDetail {
* const char\* billID;
* double amount;
* const char\* date;
* };
* union PaymentMethod {
* const char\* bankTransfer;
* const char\* cash;
* };
* struct Billing {
* struct BillingDetail bill;
* union PaymentMethod method;
* int isBankTransfer; // 1 for bank transfer, 0 for cash
* };
* void addBillingRecord(struct Billing\*\* bills, int\* count, const char\* billID, double amount, const char\* date, const char\* paymentMethod) {
* \*bills = realloc(\*bills, (\*count + 1) \* sizeof(struct Billing));
* struct Billing\* newBill = &(\*bills)[\*count];
* newBill->bill.billID = billID;
* newBill->bill.amount = amount;
* newBill->bill.date = date;
* if (paymentMethod == "bank transfer") {
* newBill->method.bankTransfer = "bank transfer";
* newBill->isBankTransfer = 1;
* } else {
* newBill->method.cash = "cash";
* newBill->isBankTransfer = 0;
* }
* (\*count)++;
* }
* void displayBillingRecords(struct Billing\* bills, int count) {
* for (int i = 0; i < count; i++) {
* printf("Bill ID: %s, Amount: %.2f, Date: %s, Payment Method: %s\n",
* bills[i].bill.billID,
* bills[i].bill.amount,
* bills[i].bill.date,
* bills[i].isBankTransfer ? bills[i].method.bankTransfer : bills[i].method.cash);
* }
* }
* int main() {
* struct Billing\* bills = NULL;
* int count = 0;
* addBillingRecord(&bills, &count, "B001", 2500.75, "2025-01-25", "bank transfer");
* addBillingRecord(&bills, &count, "B002", 1500.50, "2025-01-26", "cash");
* addBillingRecord(&bills, &count, "B003", 3200.00, "2025-01-27", "bank transfer");
* displayBillingRecords(bills, count);
* 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.
* #include <stdio.h>
* #include <stdlib.h>
* struct Route {
* const char\* start;
* const char\* end;
* const char\*\* waypoints; // Array of waypoints
* int waypointCount;
* };
* void addRoute(struct Route\*\* routes, int\* count, const char\* start, const char\* end, const char\*\* waypoints, int waypointCount) {
* \*routes = realloc(\*routes, (\*count + 1) \* sizeof(struct Route));
* struct Route\* newRoute = &(\*routes)[\*count];
* newRoute->start = start;
* newRoute->end = end;
* newRoute->waypoints = malloc(waypointCount \* sizeof(const char\*));
* newRoute->waypointCount = waypointCount;
* for (int i = 0; i < waypointCount; i++) {
* newRoute->waypoints[i] = waypoints[i];
* }
* (\*count)++;
* }
* void displayRoutes(struct Route\* routes, int count) {
* for (int i = 0; i < count; i++) {
* printf("Route %d:\n", i + 1);
* printf("  Start: %s\n", routes[i].start);
* printf("  End: %s\n", routes[i].end);
* printf("  Waypoints:\n");
* for (int j = 0; j < routes[i].waypointCount; j++) {
* printf("    %s\n", routes[i].waypoints[j]);
* }
* }
* }
* void freeRoutes(struct Route\* routes, int count) {
* for (int i = 0; i < count; i++) {
* free(routes[i].waypoints);
* }
* free(routes);
* }
* int main() {
* struct Route\* routes = NULL;
* int routeCount = 0;
* const char\* waypoints1[] = {"Port A", "Island B", "Harbor C"};
* addRoute(&routes, &routeCount, "City X", "City Y", waypoints1, 3);
* const char\* waypoints2[] = {"Bay D", "Dock E"};
* addRoute(&routes, &routeCount, "City Z", "City W", waypoints2, 2);
* displayRoutes(routes, routeCount);
* freeRoutes(routes, routeCount);
* return 0;
* }

**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>
* // Union to represent variable attributes
* union Attributes {
* const char\* cargoType;
* int passengerCount;
* };
* // Structure for vessel details
* struct Vessel {
* const char\* name;
* const char\* id;
* const char\* type;
* union Attributes attr;
* };
* // Function to add a vessel to the fleet
* void addVessel(struct Vessel\*\* fleet, int\* count, const char\* name, const char\* id, const char\* type, union Attributes attr) {
* \*fleet = realloc(\*fleet, (\*count + 1) \* sizeof(struct Vessel));
* struct Vessel\* newVessel = &(\*fleet)[\*count];
* newVessel->name = name;
* newVessel->id = id;
* newVessel->type = type;
* newVessel->attr = attr;
* (\*count)++;
* }
* // Function to display fleet details
* void displayFleet(struct Vessel\* fleet, int count) {
* 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].type == "Cargo") {
* printf("  Cargo Type: %s\n", fleet[i].attr.cargoType);
* } else if (fleet[i].type == "Passenger") {
* printf("  Passenger Count: %d\n", fleet[i].attr.passengerCount);
* }
* }
* }
* // Free the dynamically allocated fleet memory
* void freeFleet(struct Vessel\* fleet) {
* free(fleet);
* }
* int main() {
* struct Vessel\* fleet = NULL;
* int fleetCount = 0;
* union Attributes attr1;
* attr1.cargoType = "Oil";
* union Attributes attr2;
* attr2.passengerCount = 150;
* addVessel(&fleet, &fleetCount, "Vessel A", "ID123", "Cargo", attr1);
* addVessel(&fleet, &fleetCount, "Vessel B", "ID456", "Passenger", attr2);
* displayFleet(fleet, fleetCount);
* freeFleet(fleet);
* return 0;
* }

**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>
* // Structure for maintenance tasks
* struct Task {
* const char\* id;          // Immutable task ID
* const char\* description; // Task description
* const char\* schedule;    // Scheduled date
* };
* // Function to add a task to the dynamic task list
* void addTask(struct Task\*\* tasks, int\* count, const char\* id, const char\* description, const char\* schedule) {
* \*tasks = realloc(\*tasks, (\*count + 1) \* sizeof(struct Task));
* (\*tasks)[\*count].id = id;
* (\*tasks)[\*count].description = description;
* (\*tasks)[\*count].schedule = schedule;
* (\*count)++;
* }
* // Function to display all tasks
* void displayTasks(struct Task\* tasks, int count) {
* for (int i = 0; i < count; i++) {
* printf("Task %d:\n", i + 1);
* printf("  ID: %s\n", tasks[i].id);
* printf("  Description: %s\n", tasks[i].description);
* printf("  Schedule: %s\n", tasks[i].schedule);
* }
* }
* // Free the dynamically allocated task list
* void freeTasks(struct Task\* tasks) {
* free(tasks);
* }
* int main() {
* struct Task\* taskList = NULL;
* int taskCount = 0;
* addTask(&taskList, &taskCount, "T001", "Inspect engine", "2025-02-10");
* addTask(&taskList, &taskCount, "T002", "Check navigation systems", "2025-02-15");
* addTask(&taskList, &taskCount, "T003", "Hull cleaning", "2025-02-20");
* displayTasks(taskList, taskCount);
* freeTasks(taskList);
* return 0;
* }

**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>
* struct Vessel {
* float capacity;
* float dimensions;
* };
* union CargoProperties {
* float weight;
* float dimensions;
* };
* struct Cargo {
* const char\* id;
* union CargoProperties property;
* const char\* propertyType;
* };
* /\*\*
* \* Adds cargo to the dynamic cargo list
* \*/
* void addCargo(struct Cargo\*\* cargos, int\* count, const char\* id, float value, const char\* propertyType) {
* \*cargos = realloc(\*cargos, (\*count + 1) \* sizeof(struct Cargo));
* (\*cargos)[\*count].id = id;
* if (propertyType == "Weight") {
* (\*cargos)[\*count].property.weight = value;
* } else {
* (\*cargos)[\*count].property.dimensions = value;
* }
* (\*cargos)[\*count].propertyType = propertyType;
* (\*count)++;
* }
* /\*\*
* \* Displays the cargo list
* \*/
* void displayCargo(struct Cargo\* cargos, int count) {
* for (int i = 0; i < count; i++) {
* printf("Cargo %d:\n", i + 1);
* printf("  ID: %s\n", cargos[i].id);
* if (cargos[i].propertyType == "Weight") {
* printf("  Property: Weight = %.2f tons\n", cargos[i].property.weight);
* } else {
* printf("  Property: Dimensions = %.2f cubic meters\n", cargos[i].property.dimensions);
* }
* }
* }
* /\*\*
* \* Frees the dynamically allocated cargo list
* \*/
* void freeCargo(struct Cargo\* cargos) {
* free(cargos);
* }
* int main() {
* struct Vessel vessel = {100.0, 500.0};
* struct Cargo\* cargoList = NULL;
* int cargoCount = 0;
* addCargo(&cargoList, &cargoCount, "C001", 20.0, "Weight");
* addCargo(&cargoList, &cargoCount, "C002", 50.0, "Weight");
* addCargo(&cargoList, &cargoCount, "C003", 300.0, "Dimensions");
* printf("Vessel Specifications:\n");
* printf("  Capacity: %.2f tons\n", vessel.capacity);
* printf("  Dimensions: %.2f cubic meters\n\n", vessel.dimensions);
* printf("Cargo List:\n");
* displayCargo(cargoList, cargoCount);
* freeCargo(cargoList);
* return 0;
* }

**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>
* struct WeatherData {
* float temperature;
* float windSpeed;
* };
* void addWeatherRecord(struct WeatherData\*\*\* records, int\* count, float temperature, float windSpeed) {
* \*records = realloc(\*records, (\*count + 1) \* sizeof(struct WeatherData\*));
* (\*records)[\*count] = malloc(sizeof(struct WeatherData));
* (\*records)[\*count]->temperature = temperature;
* (\*records)[\*count]->windSpeed = windSpeed;
* (\*count)++;
* }
* void displayWeatherRecords(struct WeatherData\*\* records, int count) {
* for (int i = 0; i < count; i++) {
* printf("Record %d:\n", i + 1);
* printf("  Temperature: %.2f°C\n", records[i]->temperature);
* printf("  Wind Speed: %.2f km/h\n", records[i]->windSpeed);
* }
* }
* void freeWeatherRecords(struct WeatherData\*\* records, int count) {
* for (int i = 0; i < count; i++) {
* free(records[i]);
* }
* free(records);
* }
* int main() {
* struct WeatherData\*\* weatherRecords = NULL;
* int recordCount = 0;
* // Adding weather records
* addWeatherRecord(&weatherRecords, &recordCount, 28.5, 15.2);
* addWeatherRecord(&weatherRecords, &recordCount, 32.0, 20.1);
* addWeatherRecord(&weatherRecords, &recordCount, 25.3, 10.4);
* // Displaying the weather records
* printf("Weather Records:\n");
* displayWeatherRecords(weatherRecords, recordCount);
* // Freeing the allocated memory
* freeWeatherRecords(weatherRecords, recordCount);
* return 0;
* }

**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>
* #include <string.h>
* struct WeatherData {
* float temperature;
* float windSpeed;
* };
* void addWeatherRecord(struct WeatherData\*\*\* records, int\* count, float temperature, float windSpeed) {
* \*records = realloc(\*records, (\*count + 1) \* sizeof(struct WeatherData\*));
* (\*records)[\*count] = malloc(sizeof(struct WeatherData));
* (\*records)[\*count]->temperature = temperature;
* (\*records)[\*count]->windSpeed = windSpeed;
* (\*count)++;
* }
* void displayWeatherRecords(struct WeatherData\*\* records, int count) {
* for (int i = 0; i < count; i++) {
* printf("Record %d:\n", i + 1);
* printf("  Temperature: %.2f°C\n", records[i]->temperature);
* printf("  Wind Speed: %.2f km/h\n", records[i]->windSpeed);
* }
* }
* void freeWeatherRecords(struct WeatherData\*\* records, int count) {
* for (int i = 0; i < count; i++) {
* free(records[i]);
* }
* free(records);
* }
* int main() {
* struct WeatherData\*\* weatherRecords = NULL;
* int recordCount = 0;
* // Adding weather records
* addWeatherRecord(&weatherRecords, &recordCount, 28.5, 15.2);
* addWeatherRecord(&weatherRecords, &recordCount, 32.0, 20.1);
* addWeatherRecord(&weatherRecords, &recordCount, 25.3, 10.4);
* // Displaying the weather records
* printf("Weather Records:\n");
* displayWeatherRecords(weatherRecords, recordCount);
* // Freeing the allocated memory
* freeWeatherRecords(weatherRecords, recordCount);
* return 0;
* }

**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>
* struct CrewMember {
* char name[50];
* const char\* role;
* char schedule[7][10];  // Schedule for 7 days
* };
* void addCrewMember(struct CrewMember\*\* roster, int\* count, char\* name, const char\* role, char schedule[7][10]) {
* \*roster = realloc(\*roster, (\*count + 1) \* sizeof(struct CrewMember));
* struct CrewMember\* newMember = &(\*roster)[\*count];
* // Copying name
* for (int i = 0; name[i] != '\0'; i++) {
* newMember->name[i] = name[i];
* }
* newMember->role = role;  // Assigning role (constant pointer)
* // Copying schedule
* for (int i = 0; i < 7; i++) {
* for (int j = 0; schedule[i][j] != '\0'; j++) {
* newMember->schedule[i][j] = schedule[i][j];
* }
* }
* (\*count)++;
* }
* void displayRoster(struct CrewMember\* roster, int count) {
* for (int i = 0; i < count; i++) {
* printf("Crew Member: %s\n", roster[i].name);
* printf("Role: %s\n", roster[i].role);
* printf("Schedule:\n");
* for (int j = 0; j < 7; j++) {
* printf("  Day %d: %s\n", j + 1, roster[i].schedule[j]);
* }
* printf("\n");
* }
* }
* void freeRoster(struct CrewMember\* roster) {
* free(roster);
* }
* int main() {
* struct CrewMember\* crewRoster = NULL;
* int crewCount = 0;
* char schedule1[7][10] = {"08:00-16:00", "08:00-16:00", "08:00-16:00", "OFF", "08:00-16:00", "08:00-16:00", "OFF"};
* addCrewMember(&crewRoster, &crewCount, "John Doe", "Captain", schedule1);
* char schedule2[7][10] = {"OFF", "12:00-20:00", "OFF", "12:00-20:00", "12:00-20:00", "OFF", "12:00-20:00"};
* addCrewMember(&crewRoster, &crewCount, "Jane Smith", "First Mate", schedule2);
* printf("\nCrew Roster:\n");
* displayRoster(crewRoster, crewCount);
* freeRoster(crewRoster);
* 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>
* struct Sensor {
* int id;
* char location[50];
* };
* union SensorData {
* float temperature;
* float pressure;
* };
* void allocateSensorReadings(float \*\*\*sensorReadings, int n) {
* \*sensorReadings = (float \*\*)malloc(n \* sizeof(float \*));
* for (int i = 0; i < n; i++) {
* (\*sensorReadings)[i] = (float \*)malloc(2 \* sizeof(float));
* }
* }
* void freeSensorReadings(float \*\*\*sensorReadings, int n) {
* for (int i = 0; i < n; i++) {
* free((\*sensorReadings)[i]);
* }
* free(\*sensorReadings);
* }
* int main() {
* int n;
* printf("Enter the number of sensors: ");
* scanf("%d", &n);
* struct Sensor sensors[n];
* union SensorData sensorData[n];
* float \*\*sensorReadings;
* const int \*sensorIds[n];
* allocateSensorReadings(&sensorReadings, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for sensor %d:\n", i + 1);
* printf("ID: ");
* scanf("%d", &sensors[i].id);
* sensorIds[i] = &sensors[i].id;
* printf("Location: ");
* scanf(" %[^\n]", sensors[i].location);
* printf("Enter temperature reading for sensor %d: ", i + 1);
* scanf("%f", &sensorReadings[i][0]);
* printf("Enter pressure reading for sensor %d: ", i + 1);
* scanf("%f", &sensorReadings[i][1]);
* }
* printf("\nSensor Details and Readings:\n");
* for (int i = 0; i < n; i++) {
* printf("\nSensor %d - Location: %s\n", \*sensorIds[i], sensors[i].location);
* printf("Temperature: %.2f°C\n", sensorReadings[i][0]);
* printf("Pressure: %.2f atm\n", sensorReadings[i][1]);
* }
* freeSensorReadings(&sensorReadings, n);
* return 0;
* }

**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>
* struct LogMetadata {
* char date[20];
* char author[50];
* };
* void allocateLogEntries(char \*\*\*\*logEntries, int n) {
* \*logEntries = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*logEntries)[i] = (char \*\*)malloc(sizeof(char \*));
* (\*logEntries)[i][0] = (char \*)malloc(200 \* sizeof(char));
* }
* }
* void freeLogEntries(char \*\*\*\*logEntries, int n) {
* for (int i = 0; i < n; i++) {
* free((\*logEntries)[i][0]);
* free((\*logEntries)[i]);
* }
* free(\*logEntries);
* }
* int main() {
* int n;
* printf("Enter the number of log records: ");
* scanf("%d", &n);
* struct LogMetadata logs[n];
* char \*\*\*logEntries;
* const char \*logDates[n];
* const char \*logAuthors[n];
* allocateLogEntries(&logEntries, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter log metadata for record %d:\n", i + 1);
* printf("Date: ");
* scanf(" %[^\n]", logs[i].date);
* logDates[i] = logs[i].date;
* printf("Author: ");
* scanf(" %[^\n]", logs[i].author);
* logAuthors[i] = logs[i].author;
* printf("Enter log entry: ");
* scanf(" %[^\n]", logEntries[i][0]);
* }
* printf("\nShip Log Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nLog %d - Date: %s, Author: %s\n", i + 1, logDates[i], logAuthors[i]);
* printf("Log Entry: %s\n", logEntries[i][0]);
* }
* freeLogEntries(&logEntries, n);
* return 0;
* }

**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>
* struct LogMetadata {
* char date[20];
* char author[50];
* };
* void allocateLogEntries(char \*\*\*\*logEntries, int n) {
* \*logEntries = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*logEntries)[i] = (char \*\*)malloc(2 \* sizeof(char \*));
* for (int j = 0; j < 2; j++) {
* (\*logEntries)[i][j] = (char \*)malloc(100 \* sizeof(char));
* }
* }
* }
* void freeLogEntries(char \*\*\*\*logEntries, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*logEntries)[i][j]);
* }
* free((\*logEntries)[i]);
* }
* free(\*logEntries);
* }
* int main() {
* int n;
* printf("Enter the number of log entries: ");
* scanf("%d", &n);
* struct LogMetadata logs[n];
* char \*\*\*logEntries;
* const char \*logDate[n];
* const char \*logAuthor[n];
* allocateLogEntries(&logEntries, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for log entry %d:\n", i + 1);
* printf("Date (DD/MM/YYYY): ");
* scanf(" %[^\n]", logs[i].date);
* logDate[i] = logs[i].date;
* printf("Author: ");
* scanf(" %[^\n]", logs[i].author);
* logAuthor[i] = logs[i].author;
* printf("Enter daily log record for day %d:\n", i + 1);
* printf("Entry: ");
* scanf(" %[^\n]", logEntries[i][0]);
* printf("Remarks: ");
* scanf(" %[^\n]", logEntries[i][1]);
* }
* printf("\nShip Log Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nLog Entry %d - Date: %s, Author: %s\n", i + 1, logDate[i], logAuthor[i]);
* printf("Log Entry: %s\n", logEntries[i][0]);
* printf("Remarks: %s\n", logEntries[i][1]);
* }
* freeLogEntries(&logEntries, n);
* return 0;
* }

**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>
* struct AnimalData {
* char species[50];
* int id;
* char location[50];
* };
* void allocateTrackingData(float \*\*\*\*trackingData, int n) {
* \*trackingData = (float \*\*\*)malloc(n \* sizeof(float \*\*));
* for (int i = 0; i < n; i++) {
* (\*trackingData)[i] = (float \*\*)malloc(2 \* sizeof(float \*));
* for (int j = 0; j < 2; j++) {
* (\*trackingData)[i][j] = (float \*)malloc(2 \* sizeof(float));  // Store latitude, longitude for each observation
* }
* }
* }
* void freeTrackingData(float \*\*\*\*trackingData, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*trackingData)[i][j]);
* }
* free((\*trackingData)[i]);
* }
* free(\*trackingData);
* }
* int main() {
* int n;
* printf("Enter the number of animals to track: ");
* scanf("%d", &n);
* struct AnimalData animals[n];
* float \*\*\*trackingData;
* const int \*speciesIds[n];
* allocateTrackingData(&trackingData, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for animal %d:\n", i + 1);
* printf("Species: ");
* scanf(" %[^\n]", animals[i].species);
* printf("ID: ");
* scanf("%d", &animals[i].id);
* speciesIds[i] = &animals[i].id;
* printf("Location: ");
* scanf(" %[^\n]", animals[i].location);
* printf("Enter observation data for animal %d:\n", i + 1);
* printf("Enter latitude: ");
* scanf("%f", &trackingData[i][0][0]);
* printf("Enter longitude: ");
* scanf("%f", &trackingData[i][0][1]);
* printf("Enter second observation data for animal %d:\n", i + 1);
* printf("Enter latitude: ");
* scanf("%f", &trackingData[i][1][0]);
* printf("Enter longitude: ");
* scanf("%f", &trackingData[i][1][1]);
* }
* printf("\nMarine Wildlife Tracking Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nAnimal %d - Species: %s, Location: %s\n", \*speciesIds[i], animals[i].species, animals[i].location);
* printf("Observation 1 - Latitude: %.2f, Longitude: %.2f\n", trackingData[i][0][0], trackingData[i][0][1]);
* printf("Observation 2 - Latitude: %.2f, Longitude: %.2f\n", trackingData[i][1][0], trackingData[i][1][1]);
* }
* freeTrackingData(&trackingData, n);
* return 0;
* }

.

**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>
* struct BeaconMetadata {
* int id;
* char type[50];
* char location[50];
* };
* union BeaconTypes {
* char lightSignal[50];
* char soundSignal[50];
* char radarSignal[50];
* };
* void allocateBeaconData(char \*\*\*\*beaconData, int n) {
* \*beaconData = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*beaconData)[i] = (char \*\*)malloc(2 \* sizeof(char \*));
* for (int j = 0; j < 2; j++) {
* (\*beaconData)[i][j] = (char \*)malloc(100 \* sizeof(char));
* }
* }
* }
* void freeBeaconData(char \*\*\*\*beaconData, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*beaconData)[i][j]);
* }
* free((\*beaconData)[i]);
* }
* free(\*beaconData);
* }
* int main() {
* int n;
* printf("Enter the number of beacons: ");
* scanf("%d", &n);
* struct BeaconMetadata beacons[n];
* union BeaconTypes beaconTypes[n];
* char \*\*\*beaconData;
* const int \*beaconIds[n];
* allocateBeaconData(&beaconData, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for beacon %d:\n", i + 1);
* printf("ID: ");
* scanf("%d", &beacons[i].id);
* beaconIds[i] = &beacons[i].id;
* printf("Type (Light/Sound/Radar): ");
* scanf(" %[^\n]", beacons[i].type);
* printf("Location: ");
* scanf(" %[^\n]", beacons[i].location);
* if (beacons[i].type[0] == 'L' || beacons[i].type[0] == 'l') {
* printf("Enter light signal description: ");
* scanf(" %[^\n]", beaconTypes[i].lightSignal);
* sprintf(beaconData[i][0], "%s", beaconTypes[i].lightSignal);
* } else if (beacons[i].type[0] == 'S' || beacons[i].type[0] == 's') {
* printf("Enter sound signal description: ");
* scanf(" %[^\n]", beaconTypes[i].soundSignal);
* sprintf(beaconData[i][0], "%s", beaconTypes[i].soundSignal);
* } else if (beacons[i].type[0] == 'R' || beacons[i].type[0] == 'r') {
* printf("Enter radar signal description: ");
* scanf(" %[^\n]", beaconTypes[i].radarSignal);
* sprintf(beaconData[i][0], "%s", beaconTypes[i].radarSignal);
* }
* printf("Enter additional signal description: ");
* scanf(" %[^\n]", beaconData[i][1]);
* }
* printf("\nCoastal Navigation Beacon Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nBeacon %d - Type: %s, Location: %s\n", \*beaconIds[i], beacons[i].type, beacons[i].location);
* printf("Signal: %s\n", beaconData[i][0]);
* printf("Additional Signal: %s\n", beaconData[i][1]);
* }
* freeBeaconData(&beaconData, n);
* 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>
* struct FuelData {
* char type[50];
* float quantity;
* };
* void allocateFuelLogs(float \*\*\*\*fuelLogs, int n) {
* \*fuelLogs = (float \*\*\*)malloc(n \* sizeof(float \*\*));
* for (int i = 0; i < n; i++) {
* (\*fuelLogs)[i] = (float \*\*)malloc(2 \* sizeof(float \*));  // Two entries: fuel usage and remaining fuel
* for (int j = 0; j < 2; j++) {
* (\*fuelLogs)[i][j] = (float \*)malloc(sizeof(float));
* }
* }
* }
* void freeFuelLogs(float \*\*\*\*fuelLogs, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*fuelLogs)[i][j]);
* }
* free((\*fuelLogs)[i]);
* }
* free(\*fuelLogs);
* }
* int main() {
* int n;
* printf("Enter the number of fuel logs to track: ");
* scanf("%d", &n);
* struct FuelData fuel[n];
* float \*\*\*fuelLogs;
* const float \*fuelQuantities[n];
* allocateFuelLogs(&fuelLogs, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for fuel log %d:\n", i + 1);
* printf("Fuel Type: ");
* scanf(" %[^\n]", fuel[i].type);
* printf("Fuel Quantity (in liters): ");
* scanf("%f", &fuel[i].quantity);
* fuelQuantities[i] = &fuel[i].quantity;
* // Log fuel consumption
* printf("Enter fuel consumed (in liters): ");
* scanf("%f", fuelLogs[i][0]);
* // Calculate remaining fuel
* \*fuelLogs[i][1] = fuel[i].quantity - \*fuelLogs[i][0];
* }
* printf("\nFuel Usage Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nFuel Log %d - Type: %s\n", i + 1, fuel[i].type);
* printf("Initial Quantity: %.2f liters\n", \*fuelQuantities[i]);
* printf("Fuel Consumed: %.2f liters\n", \*fuelLogs[i][0]);
* printf("Remaining Fuel: %.2f liters\n", \*fuelLogs[i][1]);
* }
* freeFuelLogs(&fuelLogs, n);
* 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>
* struct ResponseDetails {
* int id;
* char location[50];
* char type[50];
* };
* void allocateAlertHistory(char \*\*\*\*alertHistory, int n) {
* \*alertHistory = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*alertHistory)[i] = (char \*\*)malloc(2 \* sizeof(char \*));  // Two entries: alert message and timestamp
* for (int j = 0; j < 2; j++) {
* (\*alertHistory)[i][j] = (char \*)malloc(100 \* sizeof(char));
* }
* }
* }
* void freeAlertHistory(char \*\*\*\*alertHistory, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*alertHistory)[i][j]);
* }
* free((\*alertHistory)[i]);
* }
* free(\*alertHistory);
* }
* int main() {
* int n;
* printf("Enter the number of emergency alerts: ");
* scanf("%d", &n);
* struct ResponseDetails responses[n];
* char \*\*\*alertHistory;
* const int \*emergencyIDs[n];
* allocateAlertHistory(&alertHistory, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for emergency %d:\n", i + 1);
* printf("Emergency ID: ");
* scanf("%d", &responses[i].id);
* emergencyIDs[i] = &responses[i].id;
* printf("Location: ");
* scanf(" %[^\n]", responses[i].location);
* printf("Type (Fire/Medical/Other): ");
* scanf(" %[^\n]", responses[i].type);
* printf("Enter alert message: ");
* scanf(" %[^\n]", alertHistory[i][0]);
* printf("Enter alert timestamp: ");
* scanf(" %[^\n]", alertHistory[i][1]);
* }
* printf("\nEmergency Response System Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nEmergency %d - Location: %s, Type: %s\n", \*emergencyIDs[i], responses[i].location, responses[i].type);
* printf("Alert Message: %s\n", alertHistory[i][0]);
* printf("Alert Timestamp: %s\n", alertHistory[i][1]);
* }
* freeAlertHistory(&alertHistory, n);
* return 0;
* }

**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>
* struct ShipSpecifications {
* float speed;
* int capacity;
* };
* union VariableFactors {
* float weatherImpact;  // e.g., the impact of weather on speed
* int seaCondition;     // e.g., sea conditions that affect performance
* };
* void allocatePerformanceMetrics(float \*\*\*\*performanceRecords, int n) {
* \*performanceRecords = (float \*\*\*)malloc(n \* sizeof(float \*\*));
* for (int i = 0; i < n; i++) {
* (\*performanceRecords)[i] = (float \*\*)malloc(2 \* sizeof(float \*));  // Two entries: performance value and adjusted value
* for (int j = 0; j < 2; j++) {
* (\*performanceRecords)[i][j] = (float \*)malloc(sizeof(float));
* }
* }
* }
* void freePerformanceMetrics(float \*\*\*\*performanceRecords, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*performanceRecords)[i][j]);
* }
* free((\*performanceRecords)[i]);
* }
* free(\*performanceRecords);
* }
* int main() {
* int n;
* printf("Enter the number of ships to analyze: ");
* scanf("%d", &n);
* struct ShipSpecifications ships[n];
* union VariableFactors factors[n];
* float \*\*\*performanceRecords;
* const float \*performanceMetrics[n];
* allocatePerformanceMetrics(&performanceRecords, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter specifications for ship %d:\n", i + 1);
* printf("Speed (in knots): ");
* scanf("%f", &ships[i].speed);
* printf("Capacity (in tons): ");
* scanf("%d", &ships[i].capacity);
* // Log performance metric
* printf("Enter performance metric value: ");
* scanf("%f", performanceRecords[i][0]);
* // Log adjusted performance based on weather or sea condition
* printf("Enter weather impact (0 for no impact): ");
* scanf("%f", &factors[i].weatherImpact);
* \*performanceRecords[i][1] = \*performanceRecords[i][0] - factors[i].weatherImpact;
* }
* printf("\nShip Performance Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nShip %d - Speed: %.2f knots, Capacity: %d tons\n", i + 1, ships[i].speed, ships[i].capacity);
* printf("Initial Performance Metric: %.2f\n", \*performanceRecords[i][0]);
* printf("Adjusted Performance Metric (due to weather): %.2f\n", \*performanceRecords[i][1]);
* }
* freePerformanceMetrics(&performanceRecords, n);
* 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>
* struct PortDetails {
* int id;
* int capacity;
* char location[50];
* };
* void allocateDockingSchedules(char \*\*\*\*schedules, int n) {
* \*schedules = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*schedules)[i] = (char \*\*)malloc(2 \* sizeof(char \*));  // Two entries: vessel name and schedule time
* for (int j = 0; j < 2; j++) {
* (\*schedules)[i][j] = (char \*)malloc(100 \* sizeof(char));
* }
* }
* }
* void freeDockingSchedules(char \*\*\*\*schedules, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*schedules)[i][j]);
* }
* free((\*schedules)[i]);
* }
* free(\*schedules);
* }
* int main() {
* int n;
* printf("Enter the number of docking schedules: ");
* scanf("%d", &n);
* struct PortDetails ports[n];
* char \*\*\*schedules;
* const int \*scheduleIDs[n];
* allocateDockingSchedules(&schedules, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for port %d:\n", i + 1);
* printf("Port ID: ");
* scanf("%d", &ports[i].id);
* scheduleIDs[i] = &ports[i].id;
* printf("Port Capacity: ");
* scanf("%d", &ports[i].capacity);
* printf("Port Location: ");
* scanf(" %[^\n]", ports[i].location);
* // Schedule docking
* printf("Enter vessel name for docking schedule: ");
* scanf(" %[^\n]", schedules[i][0]);
* printf("Enter schedule time for the vessel: ");
* scanf(" %[^\n]", schedules[i][1]);
* }
* printf("\nPort Docking Schedules:\n");
* for (int i = 0; i < n; i++) {
* printf("\nPort %d - Location: %s, Capacity: %d\n", \*scheduleIDs[i], ports[i].location, ports[i].capacity);
* printf("Vessel: %s\n", schedules[i][0]);
* printf("Scheduled Time: %s\n", schedules[i][1]);
* }
* freeDockingSchedules(&schedules, n);
* 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>
* struct ExplorationData {
* float depth;
* char location[50];
* char timestamp[20];
* };
* void allocateLogs(struct ExplorationData \*\*\*\*logs, int n) {
* \*logs = (struct ExplorationData \*\*\*)malloc(n \* sizeof(struct ExplorationData \*\*));
* for (int i = 0; i < n; i++) {
* (\*logs)[i] = (struct ExplorationData \*\*)malloc(sizeof(struct ExplorationData \*));
* (\*logs)[i][0] = (struct ExplorationData \*)malloc(sizeof(struct ExplorationData));
* }
* }
* void freeLogs(struct ExplorationData \*\*\*\*logs, int n) {
* for (int i = 0; i < n; i++) {
* free((\*logs)[i][0]);
* free((\*logs)[i]);
* }
* free(\*logs);
* }
* int main() {
* int n;
* printf("Enter the number of deep-sea exploration logs: ");
* scanf("%d", &n);
* struct ExplorationData logs[n];
* struct ExplorationData \*\*\*logEntries;
* const struct ExplorationData \*dataEntries[n];
* allocateLogs(&logEntries, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for exploration log %d:\n", i + 1);
* printf("Depth (in meters): ");
* scanf("%f", &logs[i].depth);
* printf("Location: ");
* scanf(" %[^\n]", logs[i].location);  // Takes input until newline for location
* printf("Timestamp: ");
* scanf(" %[^\n]", logs[i].timestamp);  // Takes input until newline for timestamp
* dataEntries[i] = &logs[i];
* // Store exploration data in logs array
* (\*logEntries)[i][0]->depth = logs[i].depth;
* (\*logEntries)[i][0]->location[0] = '\0';  // Directly assigned by scanf
* (\*logEntries)[i][0]->timestamp[0] = '\0';  // Directly assigned by scanf
* }
* printf("\nDeep-Sea Exploration Logs:\n");
* for (int i = 0; i < n; i++) {
* printf("\nLog %d - Depth: %.2f meters, Location: %s, Timestamp: %s\n", i + 1, (\*logEntries)[i][0]->depth,
* (\*logEntries)[i][0]->location, (\*logEntries)[i][0]->timestamp);
* }
* freeLogs(&logEntries, n);
* 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>
* struct CommunicationMetadata {
* int id;
* char timestamp[20];
* };
* void allocateMessageLogs(char \*\*\*\*messageLogs, int n) {
* \*messageLogs = (char \*\*\*)malloc(n \* sizeof(char \*\*));
* for (int i = 0; i < n; i++) {
* (\*messageLogs)[i] = (char \*\*)malloc(2 \* sizeof(char \*));  // Two entries: message and timestamp
* for (int j = 0; j < 2; j++) {
* (\*messageLogs)[i][j] = (char \*)malloc(100 \* sizeof(char));  // Allocate memory for message and timestamp
* }
* }
* }
* void freeMessageLogs(char \*\*\*\*messageLogs, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*messageLogs)[i][j]);
* }
* free((\*messageLogs)[i]);
* }
* free(\*messageLogs);
* }
* int main() {
* int n;
* printf("Enter the number of communication logs: ");
* scanf("%d", &n);
* struct CommunicationMetadata communications[n];
* char \*\*\*messageLogs;
* const int \*communicationIDs[n];
* allocateMessageLogs(&messageLogs, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for communication log %d:\n", i + 1);
* printf("Communication ID: ");
* scanf("%d", &communications[i].id);
* communicationIDs[i] = &communications[i].id;
* printf("Timestamp: ");
* scanf(" %[^\n]", communications[i].timestamp);
* // Log the communication message
* printf("Enter communication message: ");
* scanf(" %[^\n]", messageLogs[i][0]);
* // Store timestamp for the message
* snprintf(messageLogs[i][1], 100, "%s", communications[i].timestamp);
* }
* printf("\nShip Communication Logs:\n");
* for (int i = 0; i < n; i++) {
* printf("\nLog %d - Communication ID: %d\n", i + 1, \*communicationIDs[i]);
* printf("Message: %s\n", messageLogs[i][0]);
* printf("Timestamp: %s\n", messageLogs[i][1]);
* }
* freeMessageLogs(&messageLogs, n);
* 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>
* struct VesselDetails {
* int id;
* char name[50];
* };
* union CatchData {
* char species[50];
* float weight;
* };
* void allocateCatchRecords(union CatchData \*\*\*\*catchRecords, int n) {
* \*catchRecords = (union CatchData \*\*\*)malloc(n \* sizeof(union CatchData \*\*));
* for (int i = 0; i < n; i++) {
* (\*catchRecords)[i] = (union CatchData \*\*)malloc(2 \* sizeof(union CatchData \*));  // Two entries: species or weight
* for (int j = 0; j < 2; j++) {
* (\*catchRecords)[i][j] = (union CatchData \*)malloc(sizeof(union CatchData));
* }
* }
* }
* void freeCatchRecords(union CatchData \*\*\*\*catchRecords, int n) {
* for (int i = 0; i < n; i++) {
* for (int j = 0; j < 2; j++) {
* free((\*catchRecords)[i][j]);
* }
* free((\*catchRecords)[i]);
* }
* free(\*catchRecords);
* }
* int main() {
* int n;
* printf("Enter the number of fishing activities to track: ");
* scanf("%d", &n);
* struct VesselDetails vessels[n];
* union CatchData \*\*\*catchRecords;
* const int \*vesselIDs[n];
* allocateCatchRecords(&catchRecords, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for vessel %d:\n", i + 1);
* printf("Vessel ID: ");
* scanf("%d", &vessels[i].id);
* vesselIDs[i] = &vessels[i].id;
* printf("Vessel Name: ");
* scanf(" %[^\n]", vessels[i].name);
* // Log the catch details (species or weight)
* printf("Enter catch data (1 for species, 2 for weight): ");
* int choice;
* scanf("%d", &choice);
* if (choice == 1) {
* printf("Enter species name: ");
* scanf(" %[^\n]", catchRecords[i][0]->species);
* } else if (choice == 2) {
* printf("Enter weight (in kg): ");
* scanf("%f", &catchRecords[i][1]->weight);
* }
* }
* printf("\nFishing Activity Records:\n");
* for (int i = 0; i < n; i++) {
* printf("\nVessel %d - Name: %s\n", \*vesselIDs[i], vessels[i].name);
* printf("Catch: ");
* if (catchRecords[i][0]->species[0] != '\0') {
* printf("Species: %s\n", catchRecords[i][0]->species);
* } else {
* printf("Weight: %.2f kg\n", catchRecords[i][1]->weight);
* }
* }
* freeCatchRecords(&catchRecords, n);
* 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>
* struct NavigationData {
* char location[50];
* float depth;
* };
* union EnvironmentalConditions {
* float temperature;
* float pressure;
* };
* void allocateDepthLogs(float \*\*\*\*depthLogs, int n) {
* \*depthLogs = (float \*\*\*)malloc(n \* sizeof(float \*\*));
* for (int i = 0; i < n; i++) {
* (\*depthLogs)[i] = (float \*\*)malloc(sizeof(float \*));
* (\*depthLogs)[i][0] = (float \*)malloc(sizeof(float));
* }
* }
* void freeDepthLogs(float \*\*\*\*depthLogs, int n) {
* for (int i = 0; i < n; i++) {
* free((\*depthLogs)[i][0]);
* free((\*depthLogs)[i]);
* }
* free(\*depthLogs);
* }
* int main() {
* int n;
* printf("Enter the number of depth readings to log: ");
* scanf("%d", &n);
* struct NavigationData navigation[n];
* union EnvironmentalConditions conditions[n];
* float \*\*\*depthLogs;
* const struct NavigationData \*navigationData[n];
* allocateDepthLogs(&depthLogs, n);
* for (int i = 0; i < n; i++) {
* printf("\nEnter details for navigation log %d:\n", i + 1);
* // Input navigation data (location and depth)
* printf("Location: ");
* scanf(" %[^\n]", navigation[i].location);
* printf("Depth (in meters): ");
* scanf("%f", &navigation[i].depth);
* navigationData[i] = &navigation[i];
* // Log environmental condition (either temperature or pressure)
* printf("Enter environmental condition (1 for temperature, 2 for pressure): ");
* int choice;
* scanf("%d", &choice);
* if (choice == 1) {
* printf("Enter temperature (in Celsius): ");
* scanf("%f", &conditions[i].temperature);
* } else if (choice == 2) {
* printf("Enter pressure (in bar): ");
* scanf("%f", &conditions[i].pressure);
* }
* // Log the depth reading
* \*depthLogs[i][0] = navigation[i].depth;
* }
* printf("\nSubmarine Navigation Logs:\n");
* for (int i = 0; i < n; i++) {
* printf("\nNavigation Log %d - Location: %s, Depth: %.2f meters\n", i + 1, navigationData[i]->location, \*depthLogs[i][0]);
* printf("Environmental Condition: ");
* if (conditions[i].temperature) {
* printf("Temperature: %.2f°C\n", conditions[i].temperature);
* } else {
* printf("Pressure: %.2f bar\n", conditions[i].pressure);
* }
* }
* freeDepthLogs(&depthLogs, n);
* return 0;
* }