Day 21 programs

----

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>


typedef struct {

    int id;

    char name[50];

    char category[30];

} Item;
```

```
typedef union {
    float weight;
    float volume;
} Attributes;

void addItem(Item **inventory, int *size, int id, const char *name, const
char *category) {
    *inventory = realloc(*inventory, (*size + 1) * sizeof(Item));
    (*inventory)[*size].id = id;
    snprintf((*inventory)[*size].name, 50, "%s", name);
    snprintf((*inventory)[*size].category, 30, "%s", category);
    (*size)++;
}

int main() {
    const char *immutableCode = "INV001";
    Item *inventory = NULL;
    int size = 0;

    addItem(&inventory, &size, 1, "Widget", "Tools");
    printf("Inventory: %s - %s (%s)\n", immutableCode,
inventory[0].name, inventory[0].category);
```

```
    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>

typedef struct {
    int id;
    char start[50];
    char end[50];
} Route;

typedef union {
    char air[20];
    char sea[20];
    char land[20];
} Transport;

void addRoute(Route **routes, int *size, int id, const char *start, const
char *end) {
    *routes = realloc(*routes, (*size + 1) * sizeof(Route));
    (*routes)[*size].id = id;
    snprintf((*routes)[*size].start, 50, "%s", start);
    snprintf((*routes)[*size].end, 50, "%s", end);
    (*size)++;
}

int main() {
    const char *routeID = "RT001";
    Route *routes = NULL;
    int size = 0;
    addRoute(&routes, &size, 1, "CityA", "CityB");
    printf("Route %s: %s to %s\n", routeID, routes[0].start, routes[0].end);


    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>
```

```
typedef struct {
    int id;
    char type[20];
    char status[20];
} Vehicle;

typedef union {
    char active[10];
    char maintenance[10];
} Status;

void addVehicle(Vehicle **fleet, int *size, int id, const char *type, const
char *status) {
    *fleet = realloc(*fleet, (*size + 1) * sizeof(Vehicle));
    (*fleet)[*size].id = id;
    snprintf((*fleet)[*size].type, 20, "%s", type);
    snprintf((*fleet)[*size].status, 20, "%s", status);
    (*size)++;
}

int main() {
    const char *vehicleID = "VH001";
    Vehicle *fleet = NULL;
    int size = 0;


    addVehicle(&fleet, &size, 1, "Truck", "Active");
    printf("Vehicle %s: %s (%s)\n", vehicleID, fleet[0].type, fleet[0].status);


    free(fleet);
    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>
```

```c
#include <stdlib.h>


typedef struct {
    int id;
    char customer[50];
    char items[100];
} Order;


typedef union {
    char creditCard[20];
    char cash[10];
} Payment;


void addOrder(Order **orders, int *size, int id, const char *customer,
const char *items) {
    *orders = realloc(*orders, (*size + 1) * sizeof(Order));
    (*orders)[*size].id = id;
    snprintf((*orders)[*size].customer, 50, "%s", customer);
    snprintf((*orders)[*size].items, 100, "%s", items);
    (*size)++;
}
```

```
int main() {
    const char *orderID = "ORD001";
    Order *orders = NULL;
    int size = 0;

    addOrder(&orders, &size, 1, "John Doe", "Laptop, Mouse");
    printf("Order %s: %s - %s\n", orderID, orders[0].customer,
orders[0].items);

    free(orders);
    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>


typedef struct {

    int trackingNumber;

    char origin[50];

    char destination[50];

} Shipment;


typedef union {

    char dispatched[15];

    char delivered[15];

} TrackingEvent;


void addShipment(Shipment **shipments, int *size, int trackingNumber,
const char *origin, const char *destination) {

    *shipments = realloc(*shipments, (*size + 1) * sizeof(Shipment));

    (*shipments)[*size].trackingNumber = trackingNumber;

    int i;

    for (i = 0; origin[i] != '\0'; i++) {
        (*shipments)[*size].origin[i] = origin[i];
    }
    (*shipments)[*size].origin[i] = '\0';

    for (i = 0; destination[i] != '\0'; i++) {
        (*shipments)[*size].destination[i] = destination[i];
    }
    (*shipments)[*size].destination[i] = '\0';
    (*size)++;
}

int main() {
    const int trackingID = 1001;
    Shipment *shipments = NULL;
    int size = 0;

    addShipment(&shipments, &size, trackingID, "WarehouseA",
"CustomerB");
    printf("Shipment %d: %s to %s\n", shipments[0].trackingNumber,
shipments[0].origin, shipments[0].destination);

    free(shipments);
    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>

typedef struct {
    int id;
    char location[50];
} TrafficNode;
```

```
typedef union {
    char clear[10];
    char congested[10];
} TrafficCondition;

void addTrafficNode(TrafficNode **nodes, int *size, int id, const char
*location) {
    *nodes = realloc(*nodes, (*size + 1) * sizeof(TrafficNode));
    (*nodes)[*size].id = id;
    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*nodes)[*size].location[i] = location[i];
    }
    (*nodes)[*size].location[i] = '\0';
    (*size)++;
}

int main() {
    const int nodeID = 2001;
    TrafficNode *nodes = NULL;
    int size = 0;
    addTrafficNode(&nodes, &size, nodeID, "CityCenter");
    printf("Node %d: Location - %s\n", nodes[0].id, nodes[0].location);


    free(nodes);
    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>
```

```
typedef struct {
    int id;
    char location[50];
    int size;
} Slot;

typedef union {
    char perishable[15];
    char nonPerishable[20];
} ItemType;

void addSlot(Slot **slots, int *size, int id, const char *location, int
slotSize) {
    *slots = realloc(*slots, (*size + 1) * sizeof(Slot));
    (*slots)[*size].id = id;
    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*slots)[*size].location[i] = location[i];
    }
    (*slots)[*size].location[i] = '\0';
    (*slots)[*size].size = slotSize;
    (*size)++;
}


int main() {
    const int slotID = 3001;
    Slot *slots = NULL;
    int size = 0;

    addSlot(&slots, &size, slotID, "SectionA", 50);
    printf("Slot %d: Location - %s, Size - %d\n", slots[0].id,
slots[0].location, slots[0].size);

    free(slots);
    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>

typedef struct {
    int id;
    float weight;
    char destination[50];
} Package;

typedef union {
    char standard[10];
    char express[10];
} DeliveryMethod;

void addPackage(Package **packages, int *size, int id, float weight, const
char *destination) {
    *packages = realloc(*packages, (*size + 1) * sizeof(Package));
    (*packages)[*size].id = id;
    (*packages)[*size].weight = weight;
    int i;
    for (i = 0; destination[i] != '\0'; i++) {
        (*packages)[*size].destination[i] = destination[i];
    }
    (*packages)[*size].destination[i] = '\0';
    (*size)++;
}

int main() {
    const int packageID = 4001;
    Package *packages = NULL;
    int size = 0;

    addPackage(&packages, &size, packageID, 2.5, "CityB");
    printf("Package %d: Weight - %.1f, Destination - %s\n",
packages[0].id, packages[0].weight, packages[0].destination);

    free(packages);
    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>


typedef struct {
    int timestamp;
    char metric[20];
} AnalyticsRecord;


typedef union {
    float speed;
    float efficiency;
```

```
} Metric;


void addRecord(AnalyticsRecord **records, int *size, int timestamp,
const char *metric) {
    *records = realloc(*records, (*size + 1) * sizeof(AnalyticsRecord));
    (*records)[*size].timestamp = timestamp;
    int i;
    for (i = 0; metric[i] != '\0'; i++) {
        (*records)[*size].metric[i] = metric[i];
    }
    (*records)[*size].metric[i] = '\0';
    (*size)++;
}


int main() {
    const int timestamp = 162378;
    AnalyticsRecord *records = NULL;
    int size = 0;


    addRecord(&records, &size, timestamp, "Speed");
    printf("Record %d: Metric - %s\n", records[0].timestamp,
records[0].metric);
```

```
    free(records);
    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>

typedef struct {
    int id;
    char startTime[10];
    char endTime[10];
} Schedule;


typedef union {
    char bus[10];
    char truck[10];
} TransportType;


void addSchedule(Schedule **schedules, int *size, int id, const char *start,
const char *end) {
    *schedules = realloc(*schedules, (*size + 1) * sizeof(Schedule));
    (*schedules)[*size].id = id;
    int i;
    for (i = 0; start[i] != '\0'; i++) {
        (*schedules)[*size].startTime[i] = start[i];
    }
    (*schedules)[*size].startTime[i] = '\0';

    for (i = 0; end[i] != '\0'; i++) {
        (*schedules)[*size].endTime[i] = end[i];
    }
    (*schedules)[*size].endTime[i] = '\0';
    (*size)++;
}

int main() {
    const int scheduleID = 5001;
    Schedule *schedules = NULL;
    int size = 0;

    addSchedule(&schedules, &size, scheduleID, "08:00", "12:00");
    printf("Schedule %d: %s to %s\n", schedules[0].id,
schedules[0].startTime, schedules[0].endTime);

    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>


typedef struct {

    int id;

    char name[50];

} Entity;


typedef union {

    char purchase[10];

    char returnType[10];

} TransactionType;


void addEntity(Entity **entities, int *size, int id, const char *name) {

    *entities = realloc(*entities, (*size + 1) * sizeof(Entity));
    (*entities)[*size].id = id;
    int i;
    for (i = 0; name[i] != '\0'; i++) {
        (*entities)[*size].name[i] = name[i];
    }
    (*entities)[*size].name[i] = '\0';
    (*size)++;
}

int main() {
    Entity *entities = NULL;
    int size = 0;

    addEntity(&entities, &size, 101, "SupplierA");
    printf("Entity %d: %s\n", entities[0].id, entities[0].name);

    free(entities);
    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>


typedef struct {

    int id;

    float baseCost;

} CostComponent;


typedef union {

    float fixed;

    float variable;
} PricingModel;


void addCostComponent(CostComponent **components, int *size, int id,
float baseCost) {
    *components = realloc(*components, (*size + 1) *
sizeof(CostComponent));
    (*components)[*size].id = id;
    (*components)[*size].baseCost = baseCost;
    (*size)++;
}


int main() {
    CostComponent *components = NULL;
    int size = 0;


    addCostComponent(&components, &size, 201, 100.5);
    printf("Cost Component %d: %.2f\n", components[0].id,
components[0].baseCost);


    free(components);
    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.

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


typedef struct {
    int id;
    float weight;
    char destination[50];
} Load;


typedef union {
    char bulk[10];
    char container[10];
} LoadType;

void addLoad(Load **loads, int *size, int id, float weight, const char
*destination) {
    *loads = realloc(*loads, (*size + 1) * sizeof(Load));
    (*loads)[*size].id = id;
    (*loads)[*size].weight = weight;
    int i;
    for (i = 0; destination[i] != '\0'; i++) {
        (*loads)[*size].destination[i] = destination[i];
    }
    (*loads)[*size].destination[i] = '\0';
    (*size)++;
}

int main() {
    Load *loads = NULL;
    int size = 0;

    addLoad(&loads, &size, 301, 500.0, "WarehouseB");
    printf("Load %d: %.2f to %s\n", loads[0].id, loads[0].weight,
loads[0].destination);

    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>
```

```
typedef struct {
    int id;
    char origin[50];
    char destination[50];
} Transport;

typedef union {
    char rail[10];
    char road[10];
} TransportMode;

void addTransport(Transport **transports, int *size, int id, const char
*origin, const char *destination) {
    *transports = realloc(*transports, (*size + 1) * sizeof(Transport));
    (*transports)[*size].id = id;
    int i;
    for (i = 0; origin[i] != '\0'; i++) {
        (*transports)[*size].origin[i] = origin[i];
    }
    (*transports)[*size].origin[i] = '\0';

    for (i = 0; destination[i] != '\0'; i++) {
        (*transports)[*size].destination[i] = destination[i];
    }
    (*transports)[*size].destination[i] = '\0';
    (*size)++;
}

int main() {
    Transport *transports = NULL;
    int size = 0;

    addTransport(&transports, &size, 401, "PortA", "DepotB");
    printf("Transport %d: %s to %s\n", transports[0].id,
transports[0].origin, transports[0].destination);

    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>


typedef struct {

    int id;

    float value;

} PerformanceMetric;


typedef union {

    float time;

    float cost;

} PerformanceAspect;
```

```
void addMetric(PerformanceMetric **metrics, int *size, int id, float
value) {
    *metrics = realloc(*metrics, (*size + 1) * sizeof(PerformanceMetric));
    (*metrics)[*size].id = id;
    (*metrics)[*size].value = value;
    (*size)++;
}

int main() {
    PerformanceMetric *metrics = NULL;
    int size = 0;

    addMetric(&metrics, &size, 501, 95.0);
    printf("Metric %d: %.2f\n", metrics[0].id, metrics[0].value);

    free(metrics);
    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>


typedef struct {

    int id;

    char type[20];

    char status[20];

} Robot;


typedef union {

    char picking[10];

    char sorting[10];

} TaskType;
```

```
void addRobot(Robot **robots, int *size, int id, const char *type, const
char *status) {
    *robots = realloc(*robots, (*size + 1) * sizeof(Robot));
    (*robots)[*size].id = id;

    int i;
    for (i = 0; type[i] != '\0'; i++) {
        (*robots)[*size].type[i] = type[i];
    }
    (*robots)[*size].type[i] = '\0';

    for (i = 0; status[i] != '\0'; i++) {
        (*robots)[*size].status[i] = status[i];
    }
    (*robots)[*size].status[i] = '\0';

    (*size)++;
}

int main() {
    Robot *robots = NULL;
    int size = 0;
    addRobot(&robots, &size, 601, "Loader", "Active");
    printf("Robot %d: %s - %s\n", robots[0].id, robots[0].type,
robots[0].status);

    free(robots);
    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>

typedef struct {
    int id;
    char content[100];
} Feedback;

typedef union {
    char positive[10];
    char negative[10];
} FeedbackType;

void addFeedback(Feedback **feedbacks, int *size, int id, const char
*content) {
    *feedbacks = realloc(*feedbacks, (*size + 1) * sizeof(Feedback));
    (*feedbacks)[*size].id = id;

    int i;
    for (i = 0; content[i] != '\0'; i++) {
        (*feedbacks)[*size].content[i] = content[i];
    }
    (*feedbacks)[*size].content[i] = '\0';
    (*size)++;
}

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

    addFeedback(&feedbacks, &size, 701, "Excellent service!");
    printf("Feedback %d: %s\n", feedbacks[0].id, feedbacks[0].content);

    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>


typedef struct {

    int id;

    char location[50];

    char status[20];

} Fleet;


typedef union {

    char dispatch[10];

    char reroute[10];

} CoordinationType;


void addFleet(Fleet **fleets, int *size, int id, const char *location, const
char *status) {

    *fleets = realloc(*fleets, (*size + 1) * sizeof(Fleet));

    (*fleets)[*size].id = id;
    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*fleets)[*size].location[i] = location[i];
    }
    (*fleets)[*size].location[i] = '\0';

    for (i = 0; status[i] != '\0'; i++) {
        (*fleets)[*size].status[i] = status[i];
    }
    (*fleets)[*size].status[i] = '\0';

    (*size)++;
}

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

    addFleet(&fleets, &size, 801, "CityA", "Available");
    printf("Fleet %d: %s - %s\n", fleets[0].id, fleets[0].location,
fleets[0].status);
```

```
    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>

typedef struct {
    int id;
    char description[100];
```

```
} SecurityEvent;

typedef union {
    char breach[10];
    char resolved[10];
} EventType;

void addSecurityEvent(SecurityEvent **events, int *size, int id, const
char *description) {
    *events = realloc(*events, (*size + 1) * sizeof(SecurityEvent));
    (*events)[*size].id = id;

    int i;
    for (i = 0; description[i] != '\0'; i++) {
        (*events)[*size].description[i] = description[i];
    }
    (*events)[*size].description[i] = '\0';

    (*size)++;
}

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


    addSecurityEvent(&events, &size, 901, "Unauthorized access
detected");
    printf("Event %d: %s\n", events[0].id, events[0].description);


    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>


typedef struct {

    int id;

    float amount;

    char date[20];

} BillingDetail;


typedef union {

    char bankTransfer[15];

    char cash[10];

} PaymentMethod;


void addBillingDetail(BillingDetail **bills, int *size, int id, float amount,
const char *date) {

    *bills = realloc(*bills, (*size + 1) * sizeof(BillingDetail));

    (*bills)[*size].id = id;

    (*bills)[*size].amount = amount;


    int i;

    for (i = 0; date[i] != '\0'; i++) {
        (*bills)[*size].date[i] = date[i];
    }
    (*bills)[*size].date[i] = '\0';

    (*size)++;
}

int main() {
    BillingDetail *bills = NULL;
    int size = 0;

    addBillingDetail(&bills, &size, 1001, 2500.75, "2025-01-22");
    printf("Bill %d: %.2f on %s\n", bills[0].id, bills[0].amount, bills[0].date);

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

Set 2 programs

1.Vessel Navigation System

Description:

Design a navigation system that tracks a vessel's current position and routes using structures and arrays. Use const pointers for immutable route coordinates and strings for location names. Double pointers handle dynamic route allocation.

Specifications:

Structure: Route details (start, end, waypoints).

Array: Stores multiple waypoints.

Strings: Names of locations.

const Pointers: Route coordinates.

Double Pointers: Dynamic allocation of routes.

```c
#include <stdio.h>

#include <stdlib.h>


typedef struct {

    char start[20];

    char end[20];

    char **waypoints;

    int waypoint_count;

} Route;
```

```
void addRoute(Route **routes, int *size, const char *start, const char
*end, char **waypoints, int waypoint_count) {
    *routes = realloc(*routes, (*size + 1) * sizeof(Route));
    Route *newRoute = &(*routes)[*size];

    int i;
    for (i = 0; start[i] != '\0'; i++) {
        newRoute->start[i] = start[i];
    }
    newRoute->start[i] = '\0';

    for (i = 0; end[i] != '\0'; i++) {
        newRoute->end[i] = end[i];
    }
    newRoute->end[i] = '\0';

    newRoute->waypoint_count = waypoint_count;
    newRoute->waypoints = malloc(waypoint_count * sizeof(char *));
    for (i = 0; i < waypoint_count; i++) {
        newRoute->waypoints[i] = waypoints[i];
    }
    (*size)++;
}


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


    char *waypoints[] = {"Waypoint1", "Waypoint2", "Waypoint3"};
    addRoute(&routes, &size, "PortA", "PortB", waypoints, 3);


    printf("Route: %s to %s\n", routes[0].start, routes[0].end);
    printf("Waypoints: ");
    for (int i = 0; i < routes[0].waypoint_count; i++) {
        printf("%s ", routes[0].waypoints[i]);
    }
    printf("\n");


    free(routes[0].waypoints);
    free(routes);
    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>


typedef struct {

    int id;

    char name[20];

    char type[20];

} Vessel;
```

```
typedef union {
    char cargoType[20];
    int passengerCount;
} VesselAttributes;


void addVessel(Vessel **fleet, int *size, int id, const char *name, const
char *type) {
    *fleet = realloc(*fleet, (*size + 1) * sizeof(Vessel));
    (*fleet)[*size].id = id;


    int i;
    for (i = 0; name[i] != '\0'; i++) {
        (*fleet)[*size].name[i] = name[i];
    }
    (*fleet)[*size].name[i] = '\0';


    for (i = 0; type[i] != '\0'; i++) {
        (*fleet)[*size].type[i] = type[i];
    }
    (*fleet)[*size].type[i] = '\0';


    (*size)++;
}


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


    addVessel(&fleet, &size, 101, "Vessel1", "Cargo");
    printf("Vessel %d: %s - %s\n", fleet[0].id, fleet[0].name, fleet[0].type);


    free(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>


typedef struct {

    int id;

    char description[50];

    char schedule[20];

} MaintenanceTask;


void addTask(MaintenanceTask **tasks, int *size, int id, const char
*description, const char *schedule) {

    *tasks = realloc(*tasks, (*size + 1) * sizeof(MaintenanceTask));

    (*tasks)[*size].id = id;


    int i;

    for (i = 0; description[i] != '\0'; i++) {

        (*tasks)[*size].description[i] = description[i];

    }

    (*tasks)[*size].description[i] = '\0';
    for (i = 0; schedule[i] != '\0'; i++) {
        (*tasks)[*size].schedule[i] = schedule[i];
    }
    (*tasks)[*size].schedule[i] = '\0';

    (*size)++;
}

int main() {
    MaintenanceTask *tasks = NULL;
    int size = 0;

    addTask(&tasks, &size, 201, "Engine Maintenance", "2025-01-30");
    printf("Task %d: %s - %s\n", tasks[0].id, tasks[0].description,
tasks[0].schedule);

    free(tasks);
    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>


typedef struct {

    int capacity;

    int dimensions[2];

} VesselSpec;


typedef union {

    float weight;

    int dimensions[3];

} Cargo;
```

```
typedef struct {
    int id;
    Cargo cargo;
} CargoRecord;

void addCargo(CargoRecord **cargos, int *size, int id, float weight) {
    *cargos = realloc(*cargos, (*size + 1) * sizeof(CargoRecord));
    (*cargos)[*size].id = id;
    (*cargos)[*size].cargo.weight = weight;

    (*size)++;
}

int main() {
    CargoRecord *cargos = NULL;
    int size = 0;

    addCargo(&cargos, &size, 301, 2500.5);
    printf("Cargo %d: %.2f tons\n", cargos[0].id, cargos[0].cargo.weight);
    free(cargos);
    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>

typedef struct {
    float temperature;
    float windSpeed;
} WeatherData;
```

```
void addWeatherRecord(WeatherData **records, int *size, float
temperature, float windSpeed) {
    *records = realloc(*records, (*size + 1) * sizeof(WeatherData));
    (*records)[*size].temperature = temperature;
    (*records)[*size].windSpeed = windSpeed;

    (*size)++;
}

int main() {
    WeatherData *records = NULL;
    int size = 0;

    addWeatherRecord(&records, &size, 28.5, 15.0);
    printf("Weather: %.1f°C, %.1f m/s\n", records[0].temperature,
records[0].windSpeed);

    free(records);
    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>

typedef struct {
    int id;
    float scale;
    char region[30];
} ChartMetadata;

typedef union {
    float depth;
    char hazard[30];
} ChartData;

void addChart(ChartMetadata **charts, int *size, int id, float scale, const
char *region) {
    *charts = realloc(*charts, (*size + 1) * sizeof(ChartMetadata));
    (*charts)[*size].id = id;
    (*charts)[*size].scale = scale;

    int i;
    for (i = 0; region[i] != '\0'; i++) {
        (*charts)[*size].region[i] = region[i];
    }
    (*charts)[*size].region[i] = '\0';

    (*size)++;
}

int main() {
    ChartMetadata *charts = NULL;
    int size = 0;
    addChart(&charts, &size, 401, 1.5, "North Atlantic");
    printf("Chart %d: Scale %.2f, Region: %s\n", charts[0].id,
charts[0].scale, charts[0].region);

    free(charts);
    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>
```

```
typedef struct {
    char name[30];
    char role[20];
    char schedule[20];
} CrewMember;

void addCrewMember(CrewMember **roster, int *size, const char *name,
const char *role, const char *schedule) {
    *roster = realloc(*roster, (*size + 1) * sizeof(CrewMember));

    int i;
    for (i = 0; name[i] != '\0'; i++) {
        (*roster)[*size].name[i] = name[i];
    }
    (*roster)[*size].name[i] = '\0';

    for (i = 0; role[i] != '\0'; i++) {
        (*roster)[*size].role[i] = role[i];
    }
    (*roster)[*size].role[i] = '\0';
    for (i = 0; schedule[i] != '\0'; i++) {
        (*roster)[*size].schedule[i] = schedule[i];
    }
    (*roster)[*size].schedule[i] = '\0';

    (*size)++;
}

int main() {
    CrewMember *roster = NULL;
    int size = 0;

    addCrewMember(&roster, &size, "John Doe", "Captain", "2025-02-
01");
    printf("Crew: %s, Role: %s, Schedule: %s\n", roster[0].name,
roster[0].role, roster[0].schedule);

    free(roster);
    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>


typedef struct {

    int id;

    char location[30];

} Sensor;


typedef union {

    float temperature;

    float pressure;
```

```
} SensorReading;


void addSensor(Sensor **sensors, int *size, int id, const char *location) {
    *sensors = realloc(*sensors, (*size + 1) * sizeof(Sensor));
    (*sensors)[*size].id = id;


    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*sensors)[*size].location[i] = location[i];
    }
    (*sensors)[*size].location[i] = '\0';


    (*size)++;
}


int main() {
    Sensor *sensors = NULL;
    int size = 0;


    addSensor(&sensors, &size, 501, "Pacific Ocean");
    printf("Sensor %d: Location: %s\n", sensors[0].id, sensors[0].location);
    free(sensors);
    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>

typedef struct {
    char date[20];
    char author[30];
```

```
} LogMetadata;


void addLogEntry(LogMetadata **logs, int *size, const char *date, const
char *author) {
    *logs = realloc(*logs, (*size + 1) * sizeof(LogMetadata));


    int i;
    for (i = 0; date[i] != '\0'; i++) {
        (*logs)[*size].date[i] = date[i];
    }
    (*logs)[*size].date[i] = '\0';


    for (i = 0; author[i] != '\0'; i++) {
        (*logs)[*size].author[i] = author[i];
    }
    (*logs)[*size].author[i] = '\0';


    (*size)++;
}


int main() {
    LogMetadata *logs = NULL;
    int size = 0;

    addLogEntry(&logs, &size, "2025-01-22", "Captain Smith");
    printf("Log Date: %s, Author: %s\n", logs[0].date, logs[0].author);

    free(logs);
    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>
```

```c
typedef struct {
    int id;
    char **waypoints;
    int waypoint_count;
} Navigation;

void addWaypoint(Navigation **routes, int *size, int id, char
**waypoints, int waypoint_count) {
    *routes = realloc(*routes, (*size + 1) * sizeof(Navigation));
    (*routes)[*size].id = id;

    (*routes)[*size].waypoint_count = waypoint_count;
    (*routes)[*size].waypoints = malloc(waypoint_count * sizeof(char *));
    for (int i = 0; i < waypoint_count; i++) {
        (*routes)[*size].waypoints[i] = waypoints[i];
    }

    (*size)++;
}

int main() {
    Navigation *routes = NULL;
    int size = 0;

    char *waypoints[] = {"WaypointA", "WaypointB", "WaypointC"};
    addWaypoint(&routes, &size, 601, waypoints, 3);

    printf("Route ID: %d\nWaypoints: ", routes[0].id);
    for (int i = 0; i < routes[0].waypoint_count; i++) {
        printf("%s ", routes[0].waypoints[i]);
    }
    printf("\n");

    free(routes[0].waypoints);
    free(routes);
    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.

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

typedef struct {
    int id;
    char species[30];
    char location[30];
} Animal;

void addAnimal(Animal **animals, int *size, int id, const char *species,
const char *location) {
    *animals = realloc(*animals, (*size + 1) * sizeof(Animal));
    (*animals)[*size].id = id;

    int i;
    for (i = 0; species[i] != '\0'; i++) {
        (*animals)[*size].species[i] = species[i];
    }
    (*animals)[*size].species[i] = '\0';

    for (i = 0; location[i] != '\0'; i++) {
        (*animals)[*size].location[i] = location[i];
    }
    (*animals)[*size].location[i] = '\0';

    (*size)++;
}

int main() {
    Animal *animals = NULL;
    int size = 0;

    addAnimal(&animals, &size, 1, "Dolphin", "Pacific Ocean");
    printf("ID: %d, Species: %s, Location: %s\n", animals[0].id,
animals[0].species, animals[0].location);

    free(animals);
    return 0;
```

```c
}
```

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.

```c
#include <stdio.h>

#include <stdlib.h>


typedef struct {

    int id;

    char type[20];

    char location[30];
```

```
} Beacon;

typedef union {
    int signalStrength;
    char signalType[20];
} Signal;

void addBeacon(Beacon **beacons, int *size, int id, const char *type,
const char *location) {
    *beacons = realloc(*beacons, (*size + 1) * sizeof(Beacon));
    (*beacons)[*size].id = id;

    int i;
    for (i = 0; type[i] != '\0'; i++) {
        (*beacons)[*size].type[i] = type[i];
    }
    (*beacons)[*size].type[i] = '\0';

    for (i = 0; location[i] != '\0'; i++) {
        (*beacons)[*size].location[i] = location[i];
    }
    (*beacons)[*size].location[i] = '\0';
    (*size)++;
}

int main() {
    Beacon *beacons = NULL;
    int size = 0;

    addBeacon(&beacons, &size, 101, "Light", "Harbor");
    printf("Beacon ID: %d, Type: %s, Location: %s\n", beacons[0].id,
beacons[0].type, beacons[0].location);

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

13. Fuel Usage Tracking

Description:

Develop a fuel usage tracking system for ships using structures for fuel data and arrays for consumption logs.

Specifications:

Structure: Fuel data (type, quantity).

Array: Consumption logs.

Strings: Fuel types.

const Pointers: Protect fuel data.

Double Pointers: Dynamic fuel log allocation.

```
#include <stdio.h>

#include <stdlib.h>


typedef struct {

    char type[20];

    float quantity;

} Fuel;


void addFuel(Fuel **fuelLogs, int *size, const char *type, float quantity) {

    *fuelLogs = realloc(*fuelLogs, (*size + 1) * sizeof(Fuel));


    int i;

    for (i = 0; type[i] != '\0'; i++) {

        (*fuelLogs)[*size].type[i] = type[i];

    }

    (*fuelLogs)[*size].type[i] = '\0';
    (*fuelLogs)[*size].quantity = quantity;

    (*size)++;
}

int main() {
    Fuel *fuelLogs = NULL;
    int size = 0;

    addFuel(&fuelLogs, &size, "Diesel", 500.0);
    printf("Fuel Type: %s, Quantity: %.2f\n", fuelLogs[0].type,
fuelLogs[0].quantity);

    free(fuelLogs);
    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>


typedef struct {

    int id;

    char location[30];

    char type[20];

} Emergency;


void addEmergency(Emergency **emergencies, int *size, int id, const
char *location, const char *type) {

    *emergencies = realloc(*emergencies, (*size + 1) * sizeof(Emergency));

    (*emergencies)[*size].id = id;
    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*emergencies)[*size].location[i] = location[i];
    }
    (*emergencies)[*size].location[i] = '\0';

    for (i = 0; type[i] != '\0'; i++) {
        (*emergencies)[*size].type[i] = type[i];
    }
    (*emergencies)[*size].type[i] = '\0';

    (*size)++;
}

int main() {
    Emergency *emergencies = NULL;
    int size = 0;

    addEmergency(&emergencies, &size, 201, "Atlantic Ocean",
"Collision");
    printf("ID: %d, Location: %s, Type: %s\n", emergencies[0].id,
emergencies[0].location, emergencies[0].type);
```

```
    free(emergencies);
    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>

typedef struct {
    float speed;
    float capacity;
} Ship;

typedef union {
    float weatherImpact;
    int engineEfficiency;
} PerformanceFactors;

void addShip(Ship **ships, int *size, float speed, float capacity) {
    *ships = realloc(*ships, (*size + 1) * sizeof(Ship));
    (*ships)[*size].speed = speed;
    (*ships)[*size].capacity = capacity;
    (*size)++;
}

int main() {
    Ship *ships = NULL;
    int size = 0;

    addShip(&ships, &size, 20.5, 300.0);
    printf("Ship Speed: %.2f, Capacity: %.2f\n", ships[0].speed,
ships[0].capacity);
```

```
    free(ships);
    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>

typedef struct {
    int id;
    int capacity;
    char location[30];
} Port;

void addDockingSchedule(Port **ports, int *size, int id, int capacity,
const char *location) {
    *ports = realloc(*ports, (*size + 1) * sizeof(Port));
    (*ports)[*size].id = id;
    (*ports)[*size].capacity = capacity;

    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*ports)[*size].location[i] = location[i];
    }
    (*ports)[*size].location[i] = '\0';

    (*size)++;
}

int main() {
    Port *ports = NULL;
    int size = 0;
    addDockingSchedule(&ports, &size, 101, 10, "Harbor City");
    printf("Port ID: %d, Capacity: %d, Location: %s\n", ports[0].id,
ports[0].capacity, ports[0].location);

    free(ports);
    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>

typedef struct {
    float depth;
    char location[30];
    char timestamp[20];
} ExplorationData;

void addLog(ExplorationData **logs, int *size, float depth, const char
*location, const char *timestamp) {
    *logs = realloc(*logs, (*size + 1) * sizeof(ExplorationData));
    (*logs)[*size].depth = depth;

    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*logs)[*size].location[i] = location[i];
    }
    (*logs)[*size].location[i] = '\0';

    for (i = 0; timestamp[i] != '\0'; i++) {
        (*logs)[*size].timestamp[i] = timestamp[i];
    }
    (*logs)[*size].timestamp[i] = '\0';

    (*size)++;
```

```
}


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


    addLog(&logs, &size, 1200.5, "Atlantic Ridge", "2025-01-22");
    printf("Depth: %.1f, Location: %s, Timestamp: %s\n", logs[0].depth,
logs[0].location, logs[0].timestamp);


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

18. Ship Communication System

Description:

Develop a ship communication system using strings for messages, structures for communication metadata, and arrays for message logs.

Specifications:

Structure: Communication metadata (ID, timestamp).

Array: Message logs.

Strings: Communication messages.

const Pointers: Protect communication IDs.

Double Pointers: Dynamic message storage.

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

typedef struct {
    int id;
    char timestamp[20];
} Communication;

void addMessage(Communication **messages, int *size, int id, const char
*timestamp) {
    *messages = realloc(*messages, (*size + 1) * sizeof(Communication));
    (*messages)[*size].id = id;

    int i;
    for (i = 0; timestamp[i] != '\0'; i++) {
        (*messages)[*size].timestamp[i] = timestamp[i];
    }
    (*messages)[*size].timestamp[i] = '\0';
    (*size)++;
}

int main() {
    Communication *messages = NULL;
    int size = 0;

    addMessage(&messages, &size, 301, "2025-01-22T15:00:00");
    printf("Message ID: %d, Timestamp: %s\n", messages[0].id,
messages[0].timestamp);

    free(messages);
    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>


typedef struct {

    int id;

    char name[30];

} Vessel;


typedef union {

    char species[20];

    float weight;

} CatchData;


void addCatchRecord(Vessel **vessels, int *size, int id, const char *name)
{

    *vessels = realloc(*vessels, (*size + 1) * sizeof(Vessel));

    (*vessels)[*size].id = id;
    int i;
    for (i = 0; name[i] != '\0'; i++) {
        (*vessels)[*size].name[i] = name[i];
    }
    (*vessels)[*size].name[i] = '\0';

    (*size)++;
}

int main() {
    Vessel *vessels = NULL;
    int size = 0;

    addCatchRecord(&vessels, &size, 101, "Fishing Boat A");
    printf("Vessel ID: %d, Name: %s\n", vessels[0].id, vessels[0].name);

    free(vessels);
    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>


typedef struct {

    char location[30];

    float depth;

} NavigationData;


typedef union {

    float temperature;
    float pressure;
} EnvironmentalConditions;

void addDepthLog(NavigationData **logs, int *size, const char *location,
float depth) {
    *logs = realloc(*logs, (*size + 1) * sizeof(NavigationData));

    int i;
    for (i = 0; location[i] != '\0'; i++) {
        (*logs)[*size].location[i] = location[i];
    }
    (*logs)[*size].location[i] = '\0';

    (*logs)[*size].depth = depth;
    (*size)++;
}

int main() {
    NavigationData *logs = NULL;
    int size = 0;

    addDepthLog(&logs, &size, "Pacific Trench", 8000.0);
    printf("Location: %s, Depth: %.1f\n", logs[0].location, logs[0].depth);


    free(logs);

    return 0;

}
```