```
#include <stdio.h>

struct Employee{
    int ID;
    int deptNO;
    float LoginTime;
    float LogoutTime;
};
int main(){
    struct Employee emp1;
    emp1.ID=1234;
    emp1.deptNO=4;
}
```

==========

typedef

==========

```
#include <stdio.h>

typedef struct Employee{
    int ID;
    int deptNO;
    float LoginTime;
    float LogoutTime;
} emp;
int main(){
    emp emp1;
    emp1.ID=1234;
    emp1.deptNO=4;
}
```

ALLIAS FOR POINTER

==========

```
#include <stdio.h>
typedef float* fp;

int main(){
    float pi=3.14;
    fp pFlt=&pi;
    printf("pi=%f \n",*pFlt);
}
```

==========

ALLIAS FOR AN ARRAY

==========

```
#include <stdio.h>
 typedef int arr[10];//allias for an array
int main(){

    arr a={1,2,3,4};
    for(int i=0;i<10;i++){
    printf("%d \n",a[i]);
}
return 0;
}
```

diff btw #define and typedef

==========

```
#include <stdio.h>
 typedef int arr[10];//allias for an array
 #define pi 3.14
int main(){

    arr a={1,2,3,4,pi};
    for(int i=0;i<10;i++){
    printf("%d \n",a[i]);
}
return 0;
}
```

SET OF PROBLEMS

==========

Problem 1: Inventory Management System
Description: Develop an inventory management system for an e-commerce platform.
Requirements:
Use a structure to define an item with fields: itemID, itemName, price, and quantity.
Use an array of structures to store the inventory.
Implement functions to add new items, update item details (call by reference), and display the entire inventory (call by value).
Use a loop to iterate through the inventory.
Use static to keep track of the total number of items added.
Output Expectations:
Display the updated inventory after each addition or update.
Show the total number of items.

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

// Use typedef to define Item structure
typedef struct {
    int itemID;
    char itemName[50];
    float price;
    int quantity;
} Item;

// Static variable to keep track of the total number of items added
static int totalItems = 0;

// Function to add a new item to the inventory
void addItem(Item inventory[], int *currentItemCount) {
    Item newItem;

    printf("\nEnter item ID: ");
    scanf("%d", &newItem.itemID);

    printf("Enter item name (max 49 characters): ");
    // Read string using scanf (takes care of avoiding buffer overflow)
    scanf(" %49[^\n]", newItem.itemName);
```

```
    printf("Enter item price: ");
    scanf("%f", &newItem.price);

    printf("Enter item quantity: ");
    scanf("%d", &newItem.quantity);

    // Add new item to the inventory
    inventory[*currentItemCount] = newItem;
    (*currentItemCount)++;
    totalItems++;  // Increase total items count

    printf("\nNew item added successfully.\n");
}

// Function to update an existing item in the inventory
void updateItem(Item inventory[], int currentItemCount) {
    int itemID, i;
    int itemFound = 0;

    printf("\nEnter item ID to update: ");
    scanf("%d", &itemID);

    for (i = 0; i < currentItemCount; i++) {
        if (inventory[i].itemID == itemID) {
            itemFound = 1;
            printf("Item found. Update the details:\n");

            printf("Enter new name: ");
            // Read string using scanf
            scanf(" %49[^\n]", inventory[i].itemName);

            printf("Enter new price: ");
            scanf("%f", &inventory[i].price);

            printf("Enter new quantity: ");
            scanf("%d", &inventory[i].quantity);

            printf("\nItem updated successfully.\n");
            break;
        }
    }

    if (!itemFound) {
        printf("\nItem with ID %d not found.\n", itemID);
    }
}

// Function to display the inventory
void displayInventory(Item inventory[], int currentItemCount) {
    printf("\nInventory List:\n");
    printf("-------------------------------------------------\n");
    printf("ID\tName\t\tPrice\tQuantity\n");
    printf("-------------------------------------------------\n");
```

```
    for (int i = 0; i < currentItemCount; i++) {
        printf("%d\t%-15s\t%.2f\t%d\n", inventory[i].itemID, inventory[i].itemName, inventory[i].price,
inventory[i].quantity);
    }

    printf("-----------------------------------------------------\n");
    printf("Total number of items in inventory: %d\n", totalItems);
}

int main() {
    Item inventory[100];  // Array of items to store the inventory
    int currentItemCount = 0;
    int choice;

    while (1) {
        printf("\nInventory Management System\n");
        printf("1. Add Item\n");
        printf("2. Update Item\n");
        printf("3. Display Inventory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addItem(inventory, &currentItemCount);
                displayInventory(inventory, currentItemCount);
                break;
            case 2:
                updateItem(inventory, currentItemCount);
                displayInventory(inventory, currentItemCount);
                break;
            case 3:
                displayInventory(inventory, currentItemCount);
                break;
            case 4:
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

2.Problem 2: Order Processing System
Description: Create an order processing system that calculates the total order cost and applies discounts.
Requirements:
Use a structure for Order containing fields for orderID, customerName, items (array), and totalCost.
Use const for the discount rate.
Implement functions for calculating the total cost (call by value) and applying the discount (call by reference).
Use a loop to process multiple orders.
Output Expectations:

Show the total cost before and after applying the discount for each order.

```
#include <stdio.h>

#define DISCOUNT_RATE 0.10  // Discount rate: 10%

// Define a structure for Order
typedef struct {
    int orderID;
    char customerName[100];
    float items[10];  // Array to hold prices of up to 10 items
    int itemCount;
    float totalCost;
} Order;

// Function to calculate total cost (call by value)
float calculateTotalCost(Order order) {
    float total = 0.0;
    for (int i = 0; i < order.itemCount; i++) {
        total += order.items[i];
    }
    return total;
}

// Function to apply discount (call by reference)
void applyDiscount(Order *order) {
    order->totalCost -= order->totalCost * DISCOUNT_RATE;
}

// Function to process multiple orders
void processOrders() {
    int numOrders;
    printf("Enter number of orders: ");
    scanf("%d", &numOrders);

    for (int i = 0; i < numOrders; i++) {
        Order order;
        printf("\nEnter order ID for order #%d: ", i + 1);
        scanf("%d", &order.orderID);

        printf("Enter customer name for order #%d: ", i + 1);
        scanf(" %[^\n]%*c", order.customerName);  // Reading string with spaces

        printf("Enter number of items in the order: ");
        scanf("%d", &order.itemCount);

        // Reading item prices
        for (int j = 0; j < order.itemCount; j++) {
            printf("Enter price for item %d: ", j + 1);
            scanf("%f", &order.items[j]);
        }

        // Calculate total cost before discount
        order.totalCost = calculateTotalCost(order);
```

```
        // Output total cost before discount
        printf("\nOrder ID: %d\n", order.orderID);
        printf("Customer Name: %s\n", order.customerName);
        printf("Total Cost before discount: $%.2f\n", order.totalCost);

        // Apply discount
        applyDiscount(&order);

        // Output total cost after discount
        printf("Total Cost after discount: $%.2f\n", order.totalCost);
    }
}

int main() {
    processOrders();  // Process the orders
    return 0;
}
```

Problem 3: Customer Feedback System
Description: Develop a feedback system that categorizes customer feedback based on ratings.
Requirements:
Use a structure to define Feedback with fields for customerID, feedbackText, and rating.
Use a switch case to categorize feedback (e.g., Excellent, Good, Average, Poor).
Store feedback in an array.
Implement functions to add feedback and display feedback summaries using loops.
Output Expectations:
Display categorized feedback summaries.

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

// Define the structure for feedback using typedef
typedef struct {
    int customerID;
    char feedbackText[255];
    int rating; // Rating can be an integer (1-5)
} Feedback;

// Function to add feedback to the array
void addFeedback(Feedback feedbacks[], int *count, int customerID, const char* feedbackText, int rating)
{
    Feedback newFeedback;
    newFeedback.customerID = customerID;
    strncpy(newFeedback.feedbackText, feedbackText, sizeof(newFeedback.feedbackText) - 1);
    newFeedback.rating = rating;

    feedbacks[*count] = newFeedback;
    (*count)++;
}

// Function to display categorized feedback summaries
void displayFeedbackSummary(Feedback feedbacks[], int count) {
    int excellentCount = 0, goodCount = 0, averageCount = 0, poorCount = 0;

    // Loop through the feedback array and categorize based on the rating
```

```
    for (int i = 0; i < count; i++) {
        switch (feedbacks[i].rating) {
            case 5:
                excellentCount++;
                break;
            case 4:
                goodCount++;
                break;
            case 3:
                averageCount++;
                break;
            case 2:
            case 1:
                poorCount++;
                break;
            default:
                printf("Invalid rating detected.\n");
                break;
        }
    }

    // Display the categorized feedback summaries
    printf("Feedback Summary:\n");
    printf("Excellent (5 stars): %d\n", excellentCount);
    printf("Good (4 stars): %d\n", goodCount);
    printf("Average (3 stars): %d\n", averageCount);
    printf("Poor (1-2 stars): %d\n", poorCount);
}

int main() {
    // Array to hold customer feedbacks
    Feedback feedbacks[100];
    int feedbackCount = 0;

    // Add some feedback data
    addFeedback(feedbacks, &feedbackCount, 1, "Great service, very satisfied!", 5);
    addFeedback(feedbacks, &feedbackCount, 2, "Good experience, could be better.", 4);
    addFeedback(feedbacks, &feedbackCount, 3, "Average service, not what I expected.", 3);
    addFeedback(feedbacks, &feedbackCount, 4, "Very poor service, will not come again.", 2);
    addFeedback(feedbacks, &feedbackCount, 5, "Excellent, highly recommend!", 5);

    // Display feedback summary
    displayFeedbackSummary(feedbacks, feedbackCount);

    return 0;
}
```

Problem 4: Payment Method Selection
Description: Write a program that handles multiple payment methods and calculates transaction charges.
Requirements:
Use a structure for Payment with fields for method, amount, and transactionCharge.
Use const for fixed transaction charges.
Use a switch case to determine the transaction charge based on the payment method.
Implement functions for processing payments and updating transaction details (call by reference).
Output Expectations:

Show the payment details including the method and transaction charge.

```
#include <stdio.h>

// Define constants for transaction charges
#define CREDIT_CARD_CHARGE 2.5
#define DEBIT_CARD_CHARGE 1.5
#define PAYPAL_CHARGE 3.0
#define BANK_TRANSFER_CHARGE 1.0

// Define the Payment structure
typedef struct {
    char method[20];      // Payment method
    double amount;        // Amount of payment
    double transactionCharge; // Transaction charge based on method
} Payment;

// Function to process the payment and update the transaction details
void processPayment(Payment payment) {
    // Use switch-case to determine the transaction charge based on payment method
    switch(payment.method[0]) {
        case 'C': // Credit Card
            payment.transactionCharge = CREDIT_CARD_CHARGE;
            break;
        case 'D': // Debit Card
            payment.transactionCharge = DEBIT_CARD_CHARGE;
            break;
        case 'P': // PayPal
            payment.transactionCharge = PAYPAL_CHARGE;
            break;
        case 'B': // Bank Transfer
            payment.transactionCharge = BANK_TRANSFER_CHARGE;
            break;
        default:
            payment.transactionCharge = 0.0;
            break;
    }
    // Print the payment details after processing
    printf("\nPayment Method: %s\n", payment.method);
    printf("Amount: $%.2f\n", payment.amount);
    printf("Transaction Charge: $%.2f\n", payment.transactionCharge);
    printf("Total Amount: $%.2f\n", payment.amount + payment.transactionCharge);
}

int main() {
    Payment payment;

    // Input payment details
    printf("Enter payment method (C for Credit Card, D for Debit Card, P for PayPal, B for Bank Transfer):
");
    scanf("%s", payment.method);
    printf("Enter payment amount: $");
    scanf("%lf", &payment.amount);

    // Process the payment
```

```
    processPayment(payment);

    return 0;
}
```

Problem 5: Shopping Cart System
Description: Implement a shopping cart system that allows adding, removing, and viewing items.
Requirements:
Use a structure for CartItem with fields for itemID, itemName, price, and quantity.
Use an array to store the cart items.
Implement functions to add, remove (call by reference), and display items (call by value).
Use loops for iterating through cart items.
Output Expectations:
Display the updated cart after each operation.

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

#define MAX_CART_SIZE 10  // Maximum number of items in the cart

// Define the CartItem structure
typedef struct {
    int itemID;
    char itemName[50];
    float price;
    int quantity;
} CartItem;

// Function prototypes
void addItem(CartItem cart[], int *numItems);
void removeItem(CartItem cart[], int *numItems);
void displayCart(CartItem cart[], int numItems);

int main() {
    CartItem cart[MAX_CART_SIZE];  // Array to hold the cart items
    int numItems = 0;  // Track number of items in the cart
    int choice;

    while (1) {
        printf("\nShopping Cart\n");
        printf("1. Add Item\n");
        printf("2. Remove Item\n");
        printf("3. Display Cart\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addItem(cart, &numItems);
                break;
            case 2:
                removeItem(cart, &numItems);
                break;
```

```
            case 3:
                displayCart(cart, numItems);
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}

// Function to add an item to the cart
void addItem(CartItem cart[], int *numItems) {
    if (*numItems < MAX_CART_SIZE) {
        CartItem newItem;
        printf("Enter item ID: ");
        scanf("%d", &newItem.itemID);

        printf("Enter item name: ");
        scanf(" %[^\n]", newItem.itemName);

        printf("Enter price: ");
        scanf("%f", &newItem.price);

        printf("Enter quantity: ");
        scanf("%d", &newItem.quantity);

        cart[*numItems] = newItem;
        (*numItems)++;
        printf("Item added!\n");
    } else {
        printf("Cart is full.\n");
    }
}

// Function to remove an item from the cart
void removeItem(CartItem cart[], int *numItems) {
    if (*numItems == 0) {
        printf("Cart is empty.\n");
        return;
    }

    int itemID;
    printf("Enter item ID to remove: ");
    scanf("%d", &itemID);

    for (int i = 0; i < *numItems; i++) {
        if (cart[i].itemID == itemID) {
            for (int j = i; j < *numItems - 1; j++) {
                cart[j] = cart[j + 1];
            }
            (*numItems)--;
            printf("Item removed.\n");
```

```
            return;
        }
    }

    printf("Item not found.\n");
}

// Function to display the cart items
void displayCart(CartItem cart[], int numItems) {
    if (numItems == 0) {
        printf("Your cart is empty.\n");
        return;
    }

    printf("\nItems in your cart:\n");
    for (int i = 0; i < numItems; i++) {
        printf("ID: %d, Name: %s, Price: %.2f, Quantity: %d\n",
               cart[i].itemID, cart[i].itemName, cart[i].price, cart[i].quantity);
    }
}
```

Problem 6: Product Search System
Description: Create a system that allows searching for products by name or ID.
Requirements:
Use a structure for Product with fields for productID, productName, category, and price.
Store products in an array.
Use a loop to search for a product.
Implement functions for searching by name (call by value) and updating details (call by reference).
Output Expectations:
Display product details if found or a message indicating the product is not found.

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

// Define the Product structure using typedef
typedef struct {
    int productID;
    char productName[50];
    char category[50];
    double price;
} Product;

// Function to search for a product by name (pass-by-value)
int searchByName(Product products[], int size, const char *name, Product *result) {
    for (int i = 0; i < size; i++) {
        if (strcmp(products[i].productName, name) == 0) {
            *result = products[i];
            return 1;  // Product found
        }
    }
    return 0;  // Product not found
}

// Function to update product details by reference
```

```
void updateProductDetails(Product *product, const char *newName, const char *newCategory, double
newPrice) {
    strcpy(product->productName, newName);
    strcpy(product->category, newCategory);
    product->price = newPrice;
}

// Function to search for a product by ID (pass-by-value)
int searchByID(Product products[], int size, int id, Product *result) {
    for (int i = 0; i < size; i++) {
        if (products[i].productID == id) {
            *result = products[i];
            return 1;  // Product found
        }
    }
    return 0;  // Product not found
}

// Function to display the product details
void displayProduct(Product product) {
    printf("Product ID: %d\n", product.productID);
    printf("Product Name: %s\n", product.productName);
    printf("Category: %s\n", product.category);
    printf("Price: $%.2f\n", product.price);
}

int main() {
    // Array of products
    Product products[] = {
        {101, "Laptop", "Electronics", 999.99},
        {102, "Headphones", "Electronics", 199.99},
        {103, "Coffee Maker", "Appliances", 49.99},
        {104, "Smartphone", "Electronics", 799.99}
    };

    int size = sizeof(products) / sizeof(products[0]);  // Size of products array
    char searchName[50];
    int searchID;
    Product foundProduct;

    // Search by product name
    printf("Enter product name to search: ");
    scanf("%s", searchName);  // Read product name

    if (searchByName(products, size, searchName, &foundProduct)) {
        printf("Product found!\n");
        displayProduct(foundProduct);
    } else {
        printf("Product not found!\n");
    }

    // Search by product ID
    printf("Enter product ID to search: ");
    scanf("%d", &searchID);  // Read product ID
```

```
    if (searchByID(products, size, searchID, &foundProduct)) {
        printf("Product found!\n");
        displayProduct(foundProduct);
    } else {
        printf("Product not found!\n");
    }

    // Update product details
    printf("Enter product ID to update: ");
    scanf("%d", &searchID);  // Read product ID for update

    if (searchByID(products, size, searchID, &foundProduct)) {
        char newName[50], newCategory[50];
        double newPrice;

        printf("Enter new product name: ");
        scanf("%s", newName);  // Read new product name
        printf("Enter new product category: ");
        scanf("%s", newCategory);  // Read new product category
        printf("Enter new product price: ");
        scanf("%lf", &newPrice);  // Read new product price

        updateProductDetails(&foundProduct, newName, newCategory, newPrice);
        printf("Product updated!\n");
        displayProduct(foundProduct);
    } else {
        printf("Product not found to update!\n");
    }

    return 0;
}
```

Problem 7: Sales Report Generator
Description: Develop a system that generates a sales report for different categories.
Requirements:
Use a structure for Sale with fields for saleID, productCategory, amount, and date.
Store sales in an array.
Use a loop and switch case to categorize and summarize sales.
Implement functions to add sales data and generate reports.
Output Expectations:
Display summarized sales data by category.

```
#include <stdio.h>
#include <string.h>
// Define Sale structure using typedef
typedef struct {
    int saleID;
    char productCategory[30];  // To store category names
    float amount;
    char date[20];  // To store the date of sale
} Sale;

// Function to add sales data
```

```
void addSale(Sale *sales, int *count) {
    printf("Enter Sale ID: ");
    scanf("%d", &sales[*count].saleID);

    printf("Enter Product Category: ");
    scanf("%s", sales[*count].productCategory);

    printf("Enter Sale Amount: ");
    scanf("%f", &sales[*count].amount);

    printf("Enter Sale Date (YYYY-MM-DD): ");
    scanf("%s", sales[*count].date);

    (*count)++;
}

// Function to generate and display sales report
void generateReport(Sale *sales, int count) {
    float categoryTotal;
    int i;

    printf("\nSales Report:\n");
    for (i = 0; i < count; i++) {
        categoryTotal = 0.0;

        // Loop through sales and categorize them
        for (int j = 0; j < count; j++) {
            if (strcmp(sales[i].productCategory, sales[j].productCategory) == 0) {
                categoryTotal += sales[j].amount;
            }
        }

        // Display report by category
        printf("Category: %s\n", sales[i].productCategory);
        printf("Total Sales Amount: $%.2f\n\n", categoryTotal);
    }
}

int main() {
    Sale sales[100];  // Array to store up to 100 sales data
    int count = 0;    // Number of sales entered

    // Loop to add sales data
    int choice;
    do {
        printf("1. Add Sale\n");
        printf("2. Generate Sales Report\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addSale(sales, &count);  // Add sale data
                break;
```

```
            case 2:
                generateReport(sales, count);  // Generate report
                break;
            case 0:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 0);

    return 0;
}
```

Problem 8: Customer Loyalty Program
Description: Implement a loyalty program that rewards customers based on their total purchase amount.
Requirements:
Use a structure for Customer with fields for customerID, name, totalPurchases, and rewardPoints.
Use const for the reward rate.
Implement functions to calculate and update reward points (call by reference).
Use a loop to process multiple customers.
Output Expectations:
Display customer details including reward points after updating.

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

#define REWARD_RATE 0.05  // Reward rate is 5% of total purchases

// Define Customer structure using typedef
typedef struct {
    int customerID;
    char name[50];
    double totalPurchases;
    double rewardPoints;
} Customer;

// Function to calculate and update reward points
void calculateRewardPoints(Customer* customer) {
    customer->rewardPoints = customer->totalPurchases * REWARD_RATE;
}

int main() {
    int numCustomers;

    // Ask for the number of customers
    printf("Enter the number of customers: ");
    scanf("%d", &numCustomers);

    Customer customers[numCustomers]; // Array of customers

    // Process each customer
    for (int i = 0; i < numCustomers; i++) {
        // Input customer details
        printf("\nEnter details for Customer %d:\n", i + 1);
```

```
        printf("Customer ID: ");
        scanf("%d", &customers[i].customerID);
        printf("Customer Name: ");
        scanf("%s", customers[i].name);  // We use %s for reading a name (no spaces)
        printf("Total Purchases: ");
        scanf("%lf", &customers[i].totalPurchases);

        // Calculate and update reward points
        calculateRewardPoints(&customers[i]);

        // Output customer details after updating reward points
        printf("\nCustomer Details After Updating Reward Points:\n");
        printf("Customer ID: %d\n", customers[i].customerID);
        printf("Customer Name: %s\n", customers[i].name);
        printf("Total Purchases: %.2lf\n", customers[i].totalPurchases);
        printf("Reward Points: %.2lf\n", customers[i].rewardPoints);
    }

    return 0;
}
```

Problem 9: Warehouse Management System
Description: Create a warehouse management system to track stock levels of different products.
Requirements:
Use a structure for WarehouseItem with fields for itemID, itemName, currentStock, and reorderLevel.
Use an array to store warehouse items.
Implement functions to update stock levels (call by reference) and check reorder status (call by value).
Use a loop for updating stock.
Output Expectations:
Display the stock levels and reorder status for each item.

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

// Define the structure for WarehouseItem
typedef struct {
    int itemID;
    char itemName[50];
    int currentStock;
    int reorderLevel;
} WarehouseItem;

// Function to update stock levels (call by reference)
void updateStock(WarehouseItem* item, int stockChange) {
    item->currentStock += stockChange;
}

// Function to check reorder status (call by value)
int checkReorderStatus(WarehouseItem item) {
    if (item.currentStock <= item.reorderLevel) {
        return 1;  // Needs reorder
    }
    return 0;  // No need to reorder
}
```

```
int main() {
    int n, i, stockChange;

    // Ask for the number of items in the warehouse
    printf("Enter the number of items in the warehouse: ");
    scanf("%d", &n);

    // Declare an array of WarehouseItem
    WarehouseItem warehouse[n];

    // Input warehouse items' information
    for (i = 0; i < n; i++) {
        printf("\nEnter details for item %d:\n", i + 1);
        printf("Item ID: ");
        scanf("%d", &warehouse[i].itemID);
        printf("Item Name: ");
        scanf(" %49[^\n]", warehouse[i].itemName);  // Read the item name
        printf("Current Stock: ");
        scanf("%d", &warehouse[i].currentStock);
        printf("Reorder Level: ");
        scanf("%d", &warehouse[i].reorderLevel);
    }

    // Update stock levels using a loop
    for (i = 0; i < n; i++) {
        printf("\nEnter stock change for item %d (%s): ", warehouse[i].itemID, warehouse[i].itemName);
        scanf("%d", &stockChange);
        updateStock(&warehouse[i], stockChange);
    }

    // Display the stock levels and reorder status
    printf("\nWarehouse Stock Summary:\n");
    for (i = 0; i < n; i++) {
        printf("\nItem ID: %d\n", warehouse[i].itemID);
        printf("Item Name: %s\n", warehouse[i].itemName);
        printf("Current Stock: %d\n", warehouse[i].currentStock);
        if (checkReorderStatus(warehouse[i])) {
            printf("Reorder Status: Reorder Needed\n");
        } else {
            printf("Reorder Status: Sufficient Stock\n");
        }
    }

    return 0;
}
```

Problem 10: Discount Management System
Description: Design a system that manages discounts for different product categories.
Requirements:
Use a structure for Discount with fields for category, discountPercentage, and validTill.
Use const for predefined categories.
Use a switch case to apply discounts based on the category.
Implement functions to update and display discounts (call by reference).
Output Expectations:
Show the updated discount details for each category.

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

// Defining predefined categories as constants
#define ELECTRONICS 1
#define CLOTHING 2
#define GROCERY 3
#define FURNITURE 4

// Defining a structure for Discount
typedef struct {
    int category;         // Category of the product
    float discountPercentage; // Discount percentage
    char validTill[11];   // Valid till date (DD-MM-YYYY)
} Discount;

// Function to update discount for a category
void updateDiscount(Discount *d, int category, float discount, const char *validTill) {
    d->category = category;
    d->discountPercentage = discount;
    strncpy(d->validTill, validTill, 10);
    d->validTill[10] = '\0';  // Ensure the string is null-terminated
}

// Function to display the discount details
void displayDiscount(Discount d) {
    printf("Category: ");
    switch (d.category) {
        case ELECTRONICS:
            printf("Electronics\n");
            break;
        case CLOTHING:
            printf("Clothing\n");
            break;
        case GROCERY:
            printf("Grocery\n");
            break;
        case FURNITURE:
            printf("Furniture\n");
            break;
        default:
            printf("Unknown Category\n");
            return;
    }
    printf("Discount: %.2f%%\n", d.discountPercentage);
    printf("Valid Till: %s\n", d.validTill);
}

int main() {
    // Create discount instances for different categories
    Discount electronicsDiscount;
    Discount clothingDiscount;
    Discount groceryDiscount;
    Discount furnitureDiscount;
```

```
    // Update discount details for each category
    updateDiscount(&electronicsDiscount, ELECTRONICS, 10.0, "31-12-2025");
    updateDiscount(&clothingDiscount, CLOTHING, 20.0, "30-06-2025");
    updateDiscount(&groceryDiscount, GROCERY, 5.0, "15-03-2025");
    updateDiscount(&furnitureDiscount, FURNITURE, 15.0, "01-09-2025");

    // Display updated discount details
    printf("Updated Discount Details:\n\n");
    displayDiscount(electronicsDiscount);
    displayDiscount(clothingDiscount);
    displayDiscount(groceryDiscount);
    displayDiscount(furnitureDiscount);

    return 0;
}
```

UNION

==========

```
#include<stdio.h>

union num {
    int a;
    char b;
};

int main() {
    union num x1;
    union num x2;  // Declare another instance of the union

    // Declare a pointer to union num and point it to x2
    union num *ptr = &x2;

    // Printing addresses of members in x1
    printf("Address of x1.a = %p \n", &x1.a);
    printf("Address of x1.b = %p \n", &x1.b);

    // Initialize values for x2
    ptr->a = 10;  // Assigning an integer value to x2.a via pointer
    ptr->b = 'a'; // Assigning a character value to x2.b via pointer

    // Print values of x2 using pointer
    printf("Value of x2.a = %d \n", ptr->a); // Printing integer value using pointer
    printf("Value of x2.b = %c \n", ptr->b); // Printing character value using pointer

    return 0;
}
```

typedef is also used

==========

```
#include<stdio.h>

 typedef union num {
    int a;
```

```
    char b;
}n;

int main() {
     n x1;
     n x2;  // Declare another instance of the union

    // Declare a pointer to union num and point it to x2
    n *ptr = &x2;

    // Printing addresses of members in x1
    printf("Address of x1.a = %p \n", &x1.a);
    printf("Address of x1.b = %p \n", &x1.b);

    // Initialize values for x2
    ptr->a = 10;  // Assigning an integer value to x2.a via pointer
    ptr->b = 'a'; // Assigning a character value to x2.b via pointer

    // Print values of x2 using pointer
    printf("Value of x2.a = %d \n", ptr->a); // Printing integer value using pointer
    printf("Value of x2.b = %c \n", ptr->b); // Printing character value using pointer

    return 0;
}

==================================
//union
/*
union tag_name{

    datatype memeberElement1;

    datatype memeberElement2;

}var1, var2;

union tag_name var1,var;

Access memeber elements:-> using (.) operator

var1.memeberElement1 = X;

*/

#include<stdio.h>

typedef union num{

    int a;

    char b;

}n;

int main(){
```

```
    n x1;
    n *ptr = &x1;
    ptr->a = 10;
    printf("x1.a = %d\n",(*ptr).a);
    printf("x1.b = %d\n",(*ptr).b);
    ptr->b =’a’;
    printf("x1.a = %d\n",(*ptr).a);
    printf("x1.b = %c\n",(*ptr).b);
    return 0;

}
```

NESTED UNION INSIDE STRUCTURE AND INTILAIZATION

==========

```
#include <stdio.h>

struct Student{
    char section;
    int RollNo;
    union{
        float mathMarks;
        float chemMarks;
        float TotalMarks;
    }marks;
};


int main(){
    struct Student s1;
    s1.section = ’A’;
    s1.RollNo = 123;
    s1.marks.mathMarks = 45.5;
    s1.marks.chemMarks = 54.5;
    s1.marks.TotalMarks = s1.marks.mathMarks +  s1.marks.chemMarks;
    printf("Student 1: Section = %c     Roll No.: = %d   \n",s1.section,s1.RollNo);
    printf("Math Marks: = %f        Chem Marks: = %f   Total Marks: =
%f\n",s1.marks.mathMarks,s1.marks.chemMarks,s1.marks.TotalMarks);

    return 0;
}
```

SET OF PROBLEMS(UNION)

==========

==========

Problem 1: Union for Mixed Data

Description: Create a union that can store an integer, a float, or a character. Write a program that assigns values to each member and displays them.

```
#include <stdio.h>

union MixedData {
    int intVal;
    float floatVal;
    char charVal;
```

```
};

int main() {
    union MixedData data;

    data.intVal = 42;
    printf("Integer value: %d\n", data.intVal);
    data.floatVal = 3.14f;
    printf("Float value: %.2f\n", data.floatVal);
    data.charVal = 'A';
    printf("Character value: %c\n", data.charVal);

    return 0;
}
```

Problem 2: Student Data with Union
Description: Define a union to store either a student's roll number (integer) or name (string). Write a program to input and display student details using the union.

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

union Student {
    int rollNumber;
    char name[50];
};

struct StudentDetails {
    int choice;          // choice of input (1 for roll number, 2 for name)
    union Student student;
};

int main() {
    struct StudentDetails studentDetails;

    printf("Enter 1 to input roll number or 2 to input name: ");
    scanf("%d", &studentDetails.choice);

    if (studentDetails.choice == 1) {
        printf("Enter roll number: ");
        scanf("%d", &studentDetails.student.rollNumber);
        printf("Student roll number: %d\n", studentDetails.student.rollNumber);
    } else if (studentDetails.choice == 2) {
        printf("Enter student name: ");
        scanf(" %[^\n]s", studentDetails.student.name);  // To input string with spaces
        printf("Student name: %s\n", studentDetails.student.name);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }

    return 0;
}
```

Problem 3: Union for Measurement Units
Description: Create a union that can store a distance in either kilometers (float) or miles (float). Write a

program to convert and display the distance in both units.

```
#include <stdio.h>
union Distance {
    float kilometers;
    float miles;
};

#define KM_TO_MILES 0.621371
#define MILES_TO_KM 1.60934

int main() {
    int choice;
    union Distance dist;

    printf("Enter 1 to input distance in kilometers or 2 for miles: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter distance in kilometers: ");
        scanf("%f", &dist.kilometers);

        printf("%.2f kilometers = %.2f miles\n", dist.kilometers, dist.kilometers * KM_TO_MILES);
    } else if (choice == 2) {

        printf("Enter distance in miles: ");
        scanf("%f", &dist.miles);

        // Convert and display both miles and kilometers
        printf("%.2f miles = %.2f kilometers\n", dist.miles, dist.miles * MILES_TO_KM);
    } else {
        printf("Invalid choice!\n");
    }

    return 0;
}
```

4.Problem 4: Union for Shape Dimensions
Description: Define a union to store dimensions of different shapes: a radius (float) for a circle, length and width (float) for a rectangle. Write a program to calculate and display the area based on the selected shape

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

union ShapeDimensions {
    float radius;    // For circle
    struct {
        float length;
        float width;
    } rectangle;     // For rectangle
};

int main() {
    int shapeType;
```

```
    union ShapeDimensions shape;
    float area;
    printf("Select the shape (1 for Circle, 2 for Rectangle): ");
    scanf("%d", &shapeType);

    if (shapeType == 1) {
        // Input radius for circle
        printf("Enter the radius of the circle: ");
        scanf("%f", &shape.radius);

        // Calculate the area of the circle (π * radius^2)
        area = M_PI * shape.radius * shape.radius;
        printf("Area of the circle: %.2f\n", area);
    }
    else if (shapeType == 2) {
        // Input length and width for rectangle
        printf("Enter the length and width of the rectangle: ");
        scanf("%f %f", &shape.rectangle.length, &shape.rectangle.width);

        // Calculate the area of the rectangle (length * width)
        area = shape.rectangle.length * shape.rectangle.width;
        printf("Area of the rectangle: %.2f\n", area);
    }
    else {
        printf("Invalid shape type selected.\n");
    }

    return 0;
}
```

Problem 5: Union for Employee Data
Description: Create a union to store either an employee's ID (integer) or salary (float). Write a program to input and display either ID or salary based on user choice.

```
#include <stdio.h>

union EmployeeData {
    int id;
    float salary;
};

int main() {
    int choice;
    union EmployeeData empData;

    printf("Enter 1 to input Employee ID or 2 to input Employee Salary: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter Employee ID (integer): ");
        scanf("%d", &empData.id);  // Store ID
        printf("Employee ID is: %d\n", empData.id);
    } else if (choice == 2) {
        printf("Enter Employee Salary (float): ");
        scanf("%f", &empData.salary);  // Store Salary
```

```
        printf("Employee Salary is: %.2f\n", empData.salary);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }

    return 0;
}
```

Problem 6: Union for Sensor Data
Description: Define a union to store sensor data, either temperature (float) or pressure (float). Write a program to simulate sensor readings and display the data.

```
#include <stdio.h>

union SensorData {
    float temperature;
    float pressure;
};

int main() {
    union SensorData sensor;


    sensor.temperature = 25.5;  // Set temperature data (in Celsius)
    printf("Sensor Type: Temperature\n");
    printf("Temperature: %.2f °C\n\n", sensor.temperature);

    // pressure sensor reading
    sensor.pressure = 1013.25;  // Set pressure data (in hPa)
    printf("Sensor Type: Pressure\n");
    printf("Pressure: %.2f hPa\n", sensor.pressure);

    return 0;
}
```

Problem 7: Union for Bank Account Information
Description: Create a union to store either a bank account number (integer) or balance (float). Write a program to input and display either the account number or balance based on user input.

```
#include <stdio.h>

union BankInfo {
    int accountNumber;
    float balance;
};

int main() {
    int choice;
    union BankInfo bankInfo;

    printf("Enter 1 to input account number or 2 to input balance: ");
    scanf("%d", &choice);


    if (choice == 1) {
```

```
        printf("Enter the account number: ");
        scanf("%d", &bankInfo.accountNumber);


        printf("The account number is: %d\n", bankInfo.accountNumber);
    }
    else if (choice == 2) {

        printf("Enter the balance: ");
        scanf("%f", &bankInfo.balance);

        printf("The balance is: %.2f\n", bankInfo.balance);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }

    return 0;
}
```

Problem 8: Union for Vehicle Information
Description: Define a union to store either the vehicle's registration number (integer) or fuel capacity (float). Write a program to input and display either the registration number or fuel capacity.

```
#include <stdio.h>

union VehicleInfo {
    int regNumber;
    float fuelCapacity;
};

int main() {
    union VehicleInfo vehicle;

    int choice;

    printf("Enter 1 to input vehicle registration number.\n");
    printf("Enter 2 to input vehicle fuel capacity.\n");
    printf("Your choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter vehicle registration number (integer): ");
        scanf("%d", &vehicle.regNumber);
        printf("Vehicle registration number: %d\n", vehicle.regNumber);
    } else if (choice == 2) {
        printf("Enter vehicle fuel capacity (float): ");
        scanf("%f", &vehicle.fuelCapacity);
        printf("Vehicle fuel capacity: %.2f liters\n", vehicle.fuelCapacity);
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}
```

Problem 9: Union for Exam Results
Description: Create a union to store either a student's marks (integer) or grade (char). Write a program to input marks or grade and display the corresponding value.

```
#include <stdio.h>


union ExamResult {
    int marks;
    char grade;
};

int main() {
    union ExamResult result;
    int choice;

    printf("Enter 1 to input marks, 2 to input grade: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter the marks (integer): ");
        scanf("%d", &result.marks);  // Store marks in the union
        printf("The marks entered: %d\n", result.marks); // Display marks
    } else if (choice == 2) {
        printf("Enter the grade (character): ");
        scanf(" %c", &result.grade);  // Store grade in the union (note the space before %c)
        printf("The grade entered: %c\n", result.grade);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }

    return 0;
}
```

Problem 10: Union for Currency Conversion
Description: Define a union to store currency values in either USD (float) or EUR (float). Write a program to input a value in one currency and display the equivalent in the other

```
#include <stdio.h>

union Currency {
    float usd;
    float eur;
};

int main() {
    union Currency currency;
    char currencyType;
    float conversionRateToEUR = 0.85; // 1 USD = 0.85 EUR
    float conversionRateToUSD = 1.1765; // 1 EUR = 1.1765 USD

    printf("Enter the currency type (U for USD, E for EUR): ");
    scanf("%c", &currencyType);
```

```
    if (currencyType == 'U' || currencyType == 'u') {
        printf("Enter the amount in USD: ");
        scanf("%f", &currency.usd);
        // Convert USD to EUR
        printf("Equivalent amount in EUR: %.2f\n", currency.usd * conversionRateToEUR);
    }
    else if (currencyType == 'E' || currencyType == 'e') {
        printf("Enter the amount in EUR: ");
        scanf("%f", &currency.eur);
        // Convert EUR to USD
        printf("Equivalent amount in USD: %.2f\n", currency.eur * conversionRateToUSD);
    }
    else {
        printf("Invalid currency type entered!\n");
    }

    return 0;
}
```

THIRD SET OF PROBLEMS
==========

Problem 1: Aircraft Fleet Management
Description: Develop a system to manage a fleet of aircraft, tracking their specifications and operational status.
Requirements:
Define a struct for Aircraft with fields: aircraftID, model, capacity, and status.
Use an array of Aircraft structures.
Implement functions to add new aircraft (call by reference), update status, and display fleet details (call by value).
Use static to track the total number of aircraft.
Utilize a switch case to manage different operational statuses.
Employ loops to iterate through the fleet.
Output Expectations:
Display updated fleet information after each operation.

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

// Define the Aircraft struct
struct Aircraft {
    int aircraftID;
    char model[50];
    int capacity;
    char status[20];
};

// Define a static variable to track the total number of aircraft
static int totalAircraft = 0;

// Function to add a new aircraft to the fleet
void addAircraft(struct Aircraft *fleet, int aircraftID, const char *model, int capacity, const char *status) {
    fleet[totalAircraft].aircraftID = aircraftID;
```

```
    strcpy(fleet[totalAircraft].model, model);
    fleet[totalAircraft].capacity = capacity;
    strcpy(fleet[totalAircraft].status, status);
    totalAircraft++;
}

// Function to update the status of an aircraft
void updateStatus(struct Aircraft *fleet, int aircraftID, const char *newStatus) {
    for (int i = 0; i < totalAircraft; i++) {
        if (fleet[i].aircraftID == aircraftID) {
            strcpy(fleet[i].status, newStatus);
            printf("Status of aircraft ID %d updated to '%s'.\n", aircraftID, newStatus);
            return;
        }
    }
    printf("Aircraft ID %d not found!\n", aircraftID);
}

// Function to display fleet details
void displayFleetDetails(struct Aircraft fleet[]) {
    printf("\nAircraft Fleet Details:\n");
    printf("----------------------------\n");
    for (int i = 0; i < totalAircraft; i++) {
        printf("Aircraft ID: %d\n", fleet[i].aircraftID);
        printf("Model: %s\n", fleet[i].model);
        printf("Capacity: %d\n", fleet[i].capacity);
        printf("Status: %s\n", fleet[i].status);
        printf("----------------------------\n");
    }
}

// Function to manage operational status updates using switch case
void setAircraftStatus(struct Aircraft *fleet, int aircraftID, int statusCode) {
    for (int i = 0; i < totalAircraft; i++) {
        if (fleet[i].aircraftID == aircraftID) {
            switch (statusCode) {
                case 1:
                    strcpy(fleet[i].status, "Active");
                    break;
                case 2:
                    strcpy(fleet[i].status, "Maintenance");
                    break;
                case 3:
                    strcpy(fleet[i].status, "Retired");
                    break;
                default:
                    printf("Invalid status code!\n");
                    return;
            }
            printf("Aircraft ID %d status updated to '%s'.\n", aircraftID, fleet[i].status);
            return;
        }
    }
    printf("Aircraft ID %d not found!\n", aircraftID);
}
```

```
int main() {
    // Create an array to hold the fleet of aircraft
    struct Aircraft fleet[100]; // Assuming maximum of 100 aircraft
    int choice, id, capacity, statusCode;
    char model[50], status[20];

    while (1) {
        printf("\nAircraft Fleet Management System\n");
        printf("1. Add New Aircraft\n");
        printf("2. Update Aircraft Status\n");
        printf("3. Display Fleet Details\n");
        printf("4. Set Aircraft Status (Active, Maintenance, Retired)\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Aircraft ID: ");
                scanf("%d", &id);
                printf("Enter Aircraft Model: ");
                scanf("%s", model);
                printf("Enter Aircraft Capacity: ");
                scanf("%d", &capacity);
                printf("Enter Aircraft Status: ");
                scanf("%s", status);
                addAircraft(fleet, id, model, capacity, status);
                printf("Aircraft added successfully!\n");
                break;
            case 2:
                printf("Enter Aircraft ID to update status: ");
                scanf("%d", &id);
                printf("Enter new status: ");
                scanf("%s", status);
                updateStatus(fleet, id, status);
                break;
            case 3:
                displayFleetDetails(fleet);
                break;
            case 4:
                printf("Enter Aircraft ID to set status: ");
                scanf("%d", &id);
                printf("Enter status code (1 for Active, 2 for Maintenance, 3 for Retired): ");
                scanf("%d", &statusCode);
                setAircraftStatus(fleet, id, statusCode);
                break;
            case 5:
                printf("Exiting the system.\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
```

```
    return 0;
}
```

Problem 2: Satellite Data Processing
Description: Create a system to process and analyze satellite data.
Requirements:
Define a union for SatelliteData to store either image data (array) or telemetry data (nested structure).
Use struct to define Telemetry with fields: temperature, velocity, and altitude.
Implement functions to process image and telemetry data (call by reference).
Use const for fixed telemetry limits.
Employ loops to iterate through data points.
Output Expectations:
Display processed image or telemetry data based on user input.

```
#include <stdio.h>

// Define constants for telemetry limits
#define MAX_TEMP 100.0
#define MAX_VELOCITY 20000.0
#define MAX_ALTITUDE 500000.0

// Define the Telemetry struct
typedef struct {
    float temperature; // in Celsius
    float velocity;    // in km/h
    float altitude;    // in meters
} Telemetry;

// Define a union to store either image data or telemetry data
typedef union {
    unsigned char image[100][100]; // 100x100 image array
    Telemetry telemetry;
} SatelliteData;

// Function to process image data
void processImageData(unsigned char image[100][100]) {
    printf("Processing image data...\n");
    // Simple processing: print the first 5x5 pixels (just as an example)
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            printf("%d ", image[i][j]);
        }
        printf("\n");
    }
}

// Function to process telemetry data
void processTelemetryData(Telemetry *telemetry) {
    printf("Processing telemetry data...\n");

    // Check if telemetry values are within limits
    if (telemetry->temperature > MAX_TEMP) {
        printf("Warning: Temperature exceeds maximum limit!\n");
    }
    if (telemetry->velocity > MAX_VELOCITY) {
```

```
        printf("Warning: Velocity exceeds maximum limit!\n");
    }
    if (telemetry->altitude > MAX_ALTITUDE) {
        printf("Warning: Altitude exceeds maximum limit!\n");
    }

    // Output the telemetry data
    printf("Temperature: %.2f °C\n", telemetry->temperature);
    printf("Velocity: %.2f km/h\n", telemetry->velocity);
    printf("Altitude: %.2f meters\n", telemetry->altitude);
}

int main() {
    // User input for data type
    int choice;
    printf("Enter 1 for image data or 2 for telemetry data: ");
    scanf("%d", &choice);

    SatelliteData data;

    if (choice == 1) {
        // Example image data (100x100 image array)
        printf("Enter pixel data for a 5x5 part of the image (0 to 255 values):\n");
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                scanf("%hhu", &data.image[i][j]);
            }
        }
        // Process the image data
        processImageData(data.image);
    } else if (choice == 2) {
        // Example telemetry data
        printf("Enter temperature (°C), velocity (km/h), and altitude (m): ");
        scanf("%f %f %f", &data.telemetry.temperature, &data.telemetry.velocity, &data.telemetry.altitude);

        // Process the telemetry data
        processTelemetryData(&data.telemetry);
    } else {
        printf("Invalid choice!\n");
    }

    return 0;
}
```

Problem 3: Mission Control System
Description: Develop a mission control system to manage spacecraft missions.
Requirements:
Define a struct for Mission with fields: missionID, name, duration, and a nested union for payload (either crew details or cargo).
Implement functions to add missions (call by reference), update mission details, and display mission summaries (call by value).
Use static to count total missions.
Use loops and switch case for managing different mission types.
Output Expectations:
Provide detailed mission summaries including payload information.

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

// Maximum lengths for mission name and crew name
#define MAX_MISSION_NAME_LENGTH 50
#define MAX_CREW_NAME_LENGTH 30
#define MAX_CARGO_DESCRIPTION_LENGTH 100

// Payload Union
typedef union {
    struct {
        char crewName[MAX_CREW_NAME_LENGTH];
        int crewCount;
    } crew;

    struct {
        char cargoDescription[MAX_CARGO_DESCRIPTION_LENGTH];
        float cargoWeight; // in kilograms
    } cargo;
} Payload;

// Mission Struct
typedef struct {
    int missionID;
    char name[MAX_MISSION_NAME_LENGTH];
    int duration; // in days
    Payload payload; // Crew or Cargo
    int payloadType; // 0 for Crew, 1 for Cargo
} Mission;

// Static variable to keep track of the total number of missions
static int totalMissions = 0;

// Function to add a mission
void addMission(Mission *mission, int missionID, const char *name, int duration, int payloadType) {
    mission->missionID = missionID;
    strncpy(mission->name, name, MAX_MISSION_NAME_LENGTH);
    mission->duration = duration;
    mission->payloadType = payloadType;

    // Increment totalMissions statically
    totalMissions++;

    if (payloadType == 0) {
        // Crew mission
        printf("Enter crew member name: ");
        scanf("%s", mission->payload.crew.crewName);
        printf("Enter crew count: ");
        scanf("%d", &(mission->payload.crew.crewCount));
    } else if (payloadType == 1) {
        // Cargo mission
        printf("Enter cargo description: ");
        scanf("%s", mission->payload.cargo.cargoDescription);
```

```
        printf("Enter cargo weight (in kg): ");
        scanf("%f", &(mission->payload.cargo.cargoWeight));
    }
}

// Function to update mission details
void updateMission(Mission *mission) {
    printf("Updating mission ID: %d\n", mission->missionID);
    printf("Enter new mission name: ");
    scanf("%s", mission->name);

    printf("Enter new mission duration (in days): ");
    scanf("%d", &(mission->duration));

    if (mission->payloadType == 0) {
        // Update crew mission
        printf("Enter new crew member name: ");
        scanf("%s", mission->payload.crew.crewName);
        printf("Enter new crew count: ");
        scanf("%d", &(mission->payload.crew.crewCount));
    } else if (mission->payloadType == 1) {
        // Update cargo mission
        printf("Enter new cargo description: ");
        scanf("%s", mission->payload.cargo.cargoDescription);
        printf("Enter new cargo weight (in kg): ");
        scanf("%f", &(mission->payload.cargo.cargoWeight));
    }
}

// Function to display mission summary
void displayMissionSummary(Mission mission) {
    printf("\nMission ID: %d\n", mission.missionID);
    printf("Mission Name: %s\n", mission.name);
    printf("Duration: %d days\n", mission.duration);

    if (mission.payloadType == 0) {
        // Crew mission summary
        printf("Payload Type: Crew\n");
        printf("Crew Member: %s\n", mission.payload.crew.crewName);
        printf("Crew Count: %d\n", mission.payload.crew.crewCount);
    } else if (mission.payloadType == 1) {
        // Cargo mission summary
        printf("Payload Type: Cargo\n");
        printf("Cargo Description: %s\n", mission.payload.cargo.cargoDescription);
        printf("Cargo Weight: %.2f kg\n", mission.payload.cargo.cargoWeight);
    }
}

// Main function to drive the system
int main() {
    Mission missions[10]; // Array to hold missions
    int choice, missionID, duration, payloadType;
    char name[MAX_MISSION_NAME_LENGTH];

    while (1) {
```

```
    printf("\nMission Control System\n");
    printf("1. Add Mission\n");
    printf("2. Update Mission\n");
    printf("3. Display Mission Summary\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            // Add a mission
            printf("Enter Mission ID: ");
            scanf("%d", &missionID);
            printf("Enter Mission Name: ");
            scanf("%s", name);

            printf("Enter Mission Duration (in days): ");
            scanf("%d", &duration);

            printf("Enter Payload Type (0 for Crew, 1 for Cargo): ");
            scanf("%d", &payloadType);

            addMission(&missions[totalMissions], missionID, name, duration, payloadType);
            break;

        case 2:
            // Update a mission
            printf("Enter Mission ID to update: ");
            scanf("%d", &missionID);

            if (missionID >= 1 && missionID <= totalMissions) {
                updateMission(&missions[missionID - 1]);
            } else {
                printf("Invalid Mission ID\n");
            }
            break;

        case 3:
            // Display mission summaries
            for (int i = 0; i < totalMissions; i++) {
                displayMissionSummary(missions[i]);
            }
            break;

        case 4:
            // Exit the program
            printf("Exiting Mission Control System\n");
            return 0;

        default:
            printf("Invalid choice. Please try again.\n");
    }
}

return 0;
```

}

Problem 4: Aircraft Maintenance Tracker
Description: Create a tracker for aircraft maintenance schedules and logs.
Requirements:
Use a struct for MaintenanceLog with fields: logID, aircraftID, date, and a nested union for maintenance type (routine or emergency).
Implement functions to add maintenance logs (call by reference) and display logs (call by value).
Use const for maintenance frequency.
Employ loops to iterate through maintenance logs.
Output Expectations:
Display maintenance logs categorized by type.

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

#define MAX_LOGS 100  // Maximum number of maintenance logs
#define MAINTENANCE_FREQUENCY 30  // Frequency of routine maintenance in days

// Define a struct for MaintenanceLog
typedef struct {
    int logID;
    int aircraftID;
    char date[11];  // Date in format YYYY-MM-DD
    union {
        char maintenanceType[10];  // e.g., "Routine" or "Emergency"
    } type;
} MaintenanceLog;

// Function to add a maintenance log (call by reference)
void addMaintenanceLog(MaintenanceLog *log, int logID, int aircraftID, const char *date, const char *maintenanceType) {
    log->logID = logID;
    log->aircraftID = aircraftID;
    strcpy(log->date, date);
    strcpy(log->type.maintenanceType, maintenanceType);
}

// Function to display a maintenance log (call by value)
void displayMaintenanceLog(MaintenanceLog log) {
    printf("Log ID: %d\n", log.logID);
    printf("Aircraft ID: %d\n", log.aircraftID);
    printf("Date: %s\n", log.date);
    printf("Maintenance Type: %s\n\n", log.type.maintenanceType);
}

// Function to display all logs of a certain type
void displayMaintenanceLogsByType(MaintenanceLog logs[], int numLogs, const char *type) {
    for (int i = 0; i < numLogs; i++) {
        if (strcmp(logs[i].type.maintenanceType, type) == 0) {
            displayMaintenanceLog(logs[i]);
        }
    }
}
```

```
int main() {
    MaintenanceLog logs[MAX_LOGS];
    int numLogs = 0;

    // Adding sample maintenance logs
    addMaintenanceLog(&logs[numLogs++], 1, 101, "2025-01-01", "Routine");
    addMaintenanceLog(&logs[numLogs++], 2, 102, "2025-01-03", "Emergency");
    addMaintenanceLog(&logs[numLogs++], 3, 103, "2025-01-05", "Routine");
    addMaintenanceLog(&logs[numLogs++], 4, 104, "2025-01-07", "Routine");

    // Display all routine maintenance logs
    printf("Routine Maintenance Logs:\n");
    displayMaintenanceLogsByType(logs, numLogs, "Routine");

    // Display all emergency maintenance logs
    printf("Emergency Maintenance Logs:\n");
    displayMaintenanceLogsByType(logs, numLogs, "Emergency");

    return 0;
}
```

Problem 5: Spacecraft Navigation System
Description: Develop a navigation system for spacecraft to track their position and velocity.
Requirements:
Define a struct for NavigationData with fields: position, velocity, and a nested union for navigation mode (manual or automatic).
Implement functions to update navigation data (call by reference) and display the current status (call by value).
Use static to count navigation updates.
Use loops and switch case for managing navigation modes.
Output Expectations:
Show updated position and velocity with navigation mode details.

```
#include <stdio.h>

// Define a struct for navigation data
typedef struct {
    float position[3];  // 3D position (x, y, z)
    float velocity[3];  // 3D velocity (vx, vy, vz)
    int mode;           // 0 for Manual, 1 for Automatic
} NavigationData;

// Static variable to count navigation updates
static int update_count = 0;

// Function to update navigation data
void updateNavigationData(NavigationData *navData, float pos[3], float vel[3], int mode) {
    // Update position and velocity
    for (int i = 0; i < 3; i++) {
        navData->position[i] = pos[i];
        navData->velocity[i] = vel[i];
    }

    // Set the navigation mode
    navData->mode = mode;
```

```
    // Increment update count
    update_count++;
}

// Function to display navigation status
void displayNavigationStatus(NavigationData navData) {
    printf("\nNavigation Update #%d\n", update_count);
    printf("Position: (%.2f, %.2f, %.2f)\n", navData.position[0], navData.position[1], navData.position[2]);
    printf("Velocity: (%.2f, %.2f, %.2f)\n", navData.velocity[0], navData.velocity[1], navData.velocity[2]);

    // Display navigation mode
    if (navData.mode == 0)
        printf("Mode: Manual\n");
    else
        printf("Mode: Automatic\n");
}

int main() {
    // Create an instance of NavigationData
    NavigationData spacecraftNav;

    // Sample position and velocity
    float position[3] = {100.0, 150.0, 200.0};
    float velocity[3] = {1.5, -2.3, 0.7};

    // Update navigation data in manual mode (0)
    updateNavigationData(&spacecraftNav, position, velocity, 0);
    displayNavigationStatus(spacecraftNav);

    // New position and velocity for automatic mode (1)
    position[0] = 105.0;
    position[1] = 155.0;
    position[2] = 205.0;
    velocity[0] = 2.5;
    velocity[1] = -1.3;
    velocity[2] = 1.2;

    // Update navigation data in automatic mode (1)
    updateNavigationData(&spacecraftNav, position, velocity, 1);
    displayNavigationStatus(spacecraftNav);

    return 0;
}
```

Problem 6: Flight Simulation Control
Description: Create a control system for flight simulations with different aircraft models.
Requirements:
Define a struct for Simulation with fields: simulationID, aircraftModel, duration, and a nested union for control settings (manual or automated).
Implement functions to start simulations (call by reference), update settings, and display simulation results (call by value).
Use const for fixed simulation parameters.
Utilize loops to run multiple simulations and a switch case for selecting control settings.
Output Expectations:

Display simulation results with control settings.

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

#define MAX_SIMULATIONS 5  // Maximum number of simulations

// Constants for fixed simulation parameters
const int DEFAULT_DURATION = 60; // Default duration for the simulation in minutes

// Enum for control settings
typedef enum {
    MANUAL = 1,
    AUTOMATED = 2
} ControlType;

// Union for control settings
union ControlSettings {
    char manual[50];  // Manual control description
    char automated[50];  // Automated control description
};

// Struct for simulation
struct Simulation {
    int simulationID;
    char aircraftModel[50];
    int duration;
    union ControlSettings control;  // Nested union for control settings
    ControlType controlType;  // Type of control: MANUAL or AUTOMATED
};

// Function to start a simulation (call by reference)
void startSimulation(struct Simulation *sim, int simulationID, const char *aircraftModel) {
    sim->simulationID = simulationID;
    strcpy(sim->aircraftModel, aircraftModel);
    sim->duration = DEFAULT_DURATION;  // Set to default duration
    sim->controlType = MANUAL;  // Default control is manual
}

// Function to update control settings (call by reference)
void updateSettings(struct Simulation *sim, ControlType controlType, const char *settingDescription) {
    sim->controlType = controlType;

    if (controlType == MANUAL) {
        strcpy(sim->control.manual, settingDescription);  // Set manual control description
    } else if (controlType == AUTOMATED) {
        strcpy(sim->control.automated, settingDescription);  // Set automated control description
    }
}

// Function to display simulation results (call by value)
void displaySimulationResults(struct Simulation sim) {
    printf("\nSimulation ID: %d\n", sim.simulationID);
    printf("Aircraft Model: %s\n", sim.aircraftModel);
    printf("Duration: %d minutes\n", sim.duration);
```

```
    if (sim.controlType == MANUAL) {
        printf("Control Type: Manual\n");
        printf("Manual Control Settings: %s\n", sim.control.manual);
    } else if (sim.controlType == AUTOMATED) {
        printf("Control Type: Automated\n");
        printf("Automated Control Settings: %s\n", sim.control.automated);
    }
}

int main() {
    struct Simulation simulations[MAX_SIMULATIONS];

    // Loop to run multiple simulations
    for (int i = 0; i < MAX_SIMULATIONS; i++) {
        int simulationID = i + 1;
        const char *aircraftModel = (i % 2 == 0) ? "Model A" : "Model B";  // Alternate aircraft models

        // Start the simulation
        startSimulation(&simulations[i], simulationID, aircraftModel);

        // Prompt user to select control settings
        int controlChoice;
        printf("\nSimulation %d\n", simulationID);
        printf("Select Control Type (1 for Manual, 2 for Automated): ");
        scanf("%d", &controlChoice);

        if (controlChoice == MANUAL) {
            char manualDescription[50];
            printf("Enter Manual Control Description: ");
            scanf(" %[^\n]", manualDescription);  // Read input with spaces
            updateSettings(&simulations[i], MANUAL, manualDescription);
        } else if (controlChoice == AUTOMATED) {
            char automatedDescription[50];
            printf("Enter Automated Control Description: ");
            scanf(" %[^\n]", automatedDescription);  // Read input with spaces
            updateSettings(&simulations[i], AUTOMATED, automatedDescription);
        } else {
            printf("Invalid choice. Defaulting to Manual control.\n");
            updateSettings(&simulations[i], MANUAL, "Default manual control.");
        }

        // Display the results of the simulation
        displaySimulationResults(simulations[i]);
    }

    return 0;
}
```

Problem 7: Aerospace Component Testing
Description: Develop a system for testing different aerospace components.
Requirements:
Use a struct for ComponentTest with fields: testID, componentName, and a nested union for test data (physical or software).
Implement functions to record test results (call by reference) and display summaries (call by value).

Use static to count total tests conducted.
Employ loops and switch case for managing different test types.
Output Expectations:
Display test results categorized by component type.

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

#define MAX_COMPONENTS 5  // Limiting number of components for simplicity

// Define a union for test data
union TestData {
    struct {
        float temperature;  // Physical test data: Temperature
        float pressure;     // Physical test data: Pressure
    } physicalTestData;

    struct {
        int version;        // Software test data: Version number
        char result[50];    // Software test data: Test result description
    } softwareTestData;
};

// Define a struct for ComponentTest
struct ComponentTest {
    int testID;
    char componentName[50];
    union TestData testData;
    char testType; // 'P' for physical, 'S' for software
};

// Static variable to count total tests conducted
static int totalTests = 0;

// Function to record a test result
void recordTest(struct ComponentTest* test, int id, const char* name, char type) {
    test->testID = id;
    // Use scanf to copy the component name (it will be limited to one word, no spaces)
    strncpy(test->componentName, name, sizeof(test->componentName) - 1);
    test->componentName[sizeof(test->componentName) - 1] = '\0';  // Ensure null termination
    test->testType = type;

    // Assign the appropriate test data based on test type
    if (type == 'P') {  // Physical test
        printf("Enter temperature and pressure for component %s:\n", name);
        printf("Temperature: ");
        scanf("%f", &test->testData.physicalTestData.temperature);
        printf("Pressure: ");
        scanf("%f", &test->testData.physicalTestData.pressure);
    } else if (type == 'S') {  // Software test
        printf("Enter software version and result for component %s:\n", name);
        printf("Version: ");
        scanf("%d", &test->testData.softwareTestData.version);
        printf("Result: ");
        scanf(" %49[^\n]", test->testData.softwareTestData.result); // Reads a line with spaces until newline
```

```
    }

    totalTests++;  // Increment the total test counter
}

// Function to display the test summary
void displaySummary(struct ComponentTest test) {
    printf("\nTest ID: %d\n", test.testID);
    printf("Component: %s\n", test.componentName);

    if (test.testType == 'P') {  // Physical test
        printf("Test Type: Physical\n");
        printf("Temperature: %.2f\n", test.testData.physicalTestData.temperature);
        printf("Pressure: %.2f\n", test.testData.physicalTestData.pressure);
    } else if (test.testType == 'S') {  // Software test
        printf("Test Type: Software\n");
        printf("Version: %d\n", test.testData.softwareTestData.version);
        printf("Result: %s\n", test.testData.softwareTestData.result);
    }
}

// Function to display the total number of tests
void displayTotalTests() {
    printf("\nTotal tests conducted: %d\n", totalTests);
}

int main() {
    struct ComponentTest tests[MAX_COMPONENTS];
    int numTests = 0;

    while (numTests < MAX_COMPONENTS) {
        int id;
        char name[50];
        char type;

        printf("\nEnter test ID (0 to stop): ");
        scanf("%d", &id);
        if (id == 0) break;  // Exit the loop if test ID is 0

        printf("Enter component name (one word): ");
        scanf("%49s", name);  // Limiting input to 49 characters (no spaces)

        printf("Enter test type (P for physical, S for software): ");
        scanf(" %c", &type);  // Read a single character (leading space to consume previous newline)

        // Record the test result
        recordTest(&tests[numTests], id, name, type);
        numTests++;
    }

    // Display the summary for each test
    for (int i = 0; i < numTests; i++) {
        displaySummary(tests[i]);
    }
```

```
    // Display the total tests conducted
    displayTotalTests();

    return 0;
}
```

Problem 8: Space Station Crew Management
Description: Create a system to manage crew members aboard a space station.
Requirements:
Define a struct for CrewMember with fields: crewID, name, role, and a nested union for role-specific details (engineer or scientist).
Implement functions to add crew members (call by reference), update details, and display crew lists (call by value).
Use const for fixed role limits.
Use loops to iterate through the crew list and a switch case for role management.
Output Expectations:
Show updated crew information including role-specific details.

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

#define MAX_CREW 10
#define MAX_NAME_LEN 50

// Define role limits
const int MAX_ENGINEERS = 5;
const int MAX_SCIENTISTS = 5;

typedef struct {
    char engineerSkill[MAX_NAME_LEN];
} Engineer;

typedef struct {
    char scientistField[MAX_NAME_LEN];
} Scientist;

// Define a nested union for role-specific details
typedef union {
    Engineer engineer;
    Scientist scientist;
} RoleDetails;

// Define the struct for CrewMember
typedef struct {
    int crewID;
    char name[MAX_NAME_LEN];
    char role[MAX_NAME_LEN]; // either "Engineer" or "Scientist"
    RoleDetails roleDetails;
} CrewMember;

// Crew list
CrewMember crewList[MAX_CREW];
int currentCrewCount = 0;

// Function to add crew member
```

```
void addCrewMember(CrewMember* crew) {
    if (currentCrewCount < MAX_CREW) {
        printf("Enter Crew Member ID: ");
        scanf("%d", &crew->crewID);

        printf("Enter Crew Member Name: ");
        scanf("%s", crew->name);  // Using scanf for name

        printf("Enter Role (Engineer/Scientist): ");
        scanf("%s", crew->role);  // Using scanf for role

        if (strcmp(crew->role, "Engineer") == 0) {
            printf("Enter Engineer Skill: ");
            scanf("%s", crew->roleDetails.engineer.engineerSkill);  // Using scanf for engineer skill
        } else if (strcmp(crew->role, "Scientist") == 0) {
            printf("Enter Scientist Field: ");
            scanf("%s", crew->roleDetails.scientist.scientistField);  // Using scanf for scientist field
        } else {
            printf("Invalid role. Please enter either Engineer or Scientist.\n");
            return;
        }

        crewList[currentCrewCount] = *crew;
        currentCrewCount++;
        printf("Crew member added successfully!\n");
    } else {
        printf("Crew list is full. Cannot add more members.\n");
    }
}

// Function to update crew member details
void updateCrewMember(int crewID) {
    for (int i = 0; i < currentCrewCount; i++) {
        if (crewList[i].crewID == crewID) {
            printf("Updating Crew Member with ID: %d\n", crewID);

            printf("Enter new name: ");
            scanf("%s", crewList[i].name);  // Using scanf for name

            printf("Enter new role (Engineer/Scientist): ");
            scanf("%s", crewList[i].role);  // Using scanf for role

            if (strcmp(crewList[i].role, "Engineer") == 0) {
                printf("Enter new Engineer Skill: ");
                scanf("%s", crewList[i].roleDetails.engineer.engineerSkill);  // Using scanf for engineer skill
            } else if (strcmp(crewList[i].role, "Scientist") == 0) {
                printf("Enter new Scientist Field: ");
                scanf("%s", crewList[i].roleDetails.scientist.scientistField);  // Using scanf for scientist field
            } else {
                printf("Invalid role. Please enter either Engineer or Scientist.\n");
            }
            printf("Crew member updated successfully!\n");
            return;
        }
    }
```

```
    printf("Crew member with ID %d not found.\n", crewID);
}

// Function to display the crew list
void displayCrewList() {
    if (currentCrewCount == 0) {
        printf("No crew members to display.\n");
        return;
    }

    printf("\nCrew List:\n");
    for (int i = 0; i < currentCrewCount; i++) {
        printf("\nCrew ID: %d\n", crewList[i].crewID);
        printf("Name: %s\n", crewList[i].name);
        printf("Role: %s\n", crewList[i].role);

        // Role-specific details
        if (strcmp(crewList[i].role, "Engineer") == 0) {
            printf("Engineer Skill: %s\n", crewList[i].roleDetails.engineer.engineerSkill);
        } else if (strcmp(crewList[i].role, "Scientist") == 0) {
            printf("Scientist Field: %s\n", crewList[i].roleDetails.scientist.scientistField);
        }
    }
}

int main() {
    int choice, crewID;
    CrewMember crew;

    while (1) {
        printf("\nSpace Station Crew Management\n");
        printf("1. Add Crew Member\n");
        printf("2. Update Crew Member\n");
        printf("3. Display Crew List\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCrewMember(&crew);
                break;
            case 2:
                printf("Enter Crew ID to update: ");
                scanf("%d", &crewID);
                updateCrewMember(crewID);
                break;
            case 3:
                displayCrewList();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
```

```
    }

    return 0;
}
```

Problem 9: Aerospace Research Data Analysis
Description: Develop a system to analyze research data from aerospace experiments.
Requirements:
Use a struct for ResearchData with fields: experimentID, description, and a nested union for data type (numerical or qualitative).
Implement functions to analyze data (call by reference) and generate reports (call by value).
Use static to track the number of analyses conducted.
Employ loops and switch case for managing different data types.
Output Expectations:
Provide detailed reports of analyzed data.

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

#define MAX_DESC_LENGTH 100

// Define a union for data type
union Data {
    float numericalData;      // For numerical data (e.g., float or double)
    char qualitativeData[50]; // For qualitative data (e.g., string)
};

// Define a struct for ResearchData
struct ResearchData {
    int experimentID;
    char description[MAX_DESC_LENGTH];
    union Data data;
};

// Static variable to count the number of analyses
static int analysisCount = 0;

// Function to analyze the data (call by reference)
void analyzeData(struct ResearchData* data) {
    analysisCount++; // Increment the static count of analyses

    // Use switch-case or if-else to handle different data types
    printf("Analyzing data for Experiment ID: %d\n", data->experimentID);
    printf("Description: %s\n", data->description);

    if (data->experimentID % 2 == 0) {
        // Assume numerical data for even ID
        printf("Numerical Data: %.2f\n", data->data.numericalData);
    } else {
        // Assume qualitative data for odd ID
        printf("Qualitative Data: %s\n", data->data.qualitativeData);
    }
}

// Function to generate a report (call by value)
```

```
void generateReport(struct ResearchData data) {
    printf("\n--- Report ---\n");
    printf("Experiment ID: %d\n", data.experimentID);
    printf("Description: %s\n", data.description);

    if (data.experimentID % 2 == 0) {
        // Assume numerical data for even ID
        printf("Numerical Data: %.2f\n", data.data.numericalData);
    } else {
        // Assume qualitative data for odd ID
        printf("Qualitative Data: %s\n", data.data.qualitativeData);
    }

    printf("Total Analyses Conducted: %d\n", analysisCount);
}

int main() {
    // Creating instances of ResearchData
    struct ResearchData experiment1 = {1, "Test of flight velocity", .data.qualitativeData = "Successful"};
    struct ResearchData experiment2 = {2, "Pressure test at high altitude", .data.numericalData = 98.5};

    // Analyzing the data (call by reference)
    analyzeData(&experiment1);
    analyzeData(&experiment2);

    // Generating reports (call by value)
    generateReport(experiment1);
    generateReport(experiment2);

    return 0;
}
```

Problem 10: Rocket Launch Scheduler
Description: Create a scheduler for managing rocket launches.
Requirements:
Define a struct for Launch with fields: launchID, rocketName, date, and a nested union for launch status (scheduled or completed).
Implement functions to schedule launches (call by reference), update statuses, and display launch schedules (call by value).
Use const for fixed launch parameters.
Use loops to iterate through launch schedules and a switch case for managing status updates.
Output Expectations:
Display detailed launch schedules and statuses.

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

#define MAX_LAUNCHES 5

// Define a union for the launch status
union LaunchStatus {
    char scheduled[20];
    char completed[20];
};
```

```
// Define a struct for a launch
struct Launch {
    int launchID;
    char rocketName[50];
    char date[20];
    union LaunchStatus status; // Nested union for status
};

// Function to schedule a launch
void scheduleLaunch(struct Launch* launches, int* currentIndex) {
    if (*currentIndex >= MAX_LAUNCHES) {
        printf("Cannot schedule more launches.\n");
        return;
    }

    struct Launch newLaunch;
    newLaunch.launchID = *currentIndex + 1;  // Launch ID is auto-generated
    printf("Enter rocket name: ");
    scanf("%s", newLaunch.rocketName);
    printf("Enter launch date (yyyy-mm-dd): ");
    scanf("%s", newLaunch.date);

    // Set the status to "scheduled"
    strcpy(newLaunch.status.scheduled, "Scheduled");

    launches[*currentIndex] = newLaunch;
    (*currentIndex)++;
}

// Function to update the status of a launch
void updateStatus(struct Launch* launches, int totalLaunches) {
    int launchID;
    printf("Enter the launch ID to update status: ");
    scanf("%d", &launchID);

    if (launchID < 1 || launchID > totalLaunches) {
        printf("Invalid launch ID.\n");
        return;
    }

    int choice;
    printf("Select status update for launch %d:\n", launchID);
    printf("1. Scheduled\n2. Completed\n");
    scanf("%d", &choice);

    if (choice == 1) {
        strcpy(launches[launchID - 1].status.scheduled, "Scheduled");
    } else if (choice == 2) {
        strcpy(launches[launchID - 1].status.completed, "Completed");
    } else {
        printf("Invalid choice.\n");
    }
}

// Function to display all launches and their statuses
```

```
void displayLaunchSchedule(struct Launch* launches, int totalLaunches) {
    if (totalLaunches == 0) {
        printf("No launches scheduled.\n");
        return;
    }

    for (int i = 0; i < totalLaunches; i++) {
        printf("\nLaunch ID: %d\n", launches[i].launchID);
        printf("Rocket Name: %s\n", launches[i].rocketName);
        printf("Launch Date: %s\n", launches[i].date);

        // Display the status
        if (strlen(launches[i].status.scheduled) > 0) {
            printf("Status: %s\n", launches[i].status.scheduled);
        } else if (strlen(launches[i].status.completed) > 0) {
            printf("Status: %s\n", launches[i].status.completed);
        }
    }
}

int main() {
    struct Launch launches[MAX_LAUNCHES]; // Array to store launches
    int totalLaunches = 0;
    int choice;

    while (1) {
        printf("\nRocket Launch Scheduler Menu:\n");
        printf("1. Schedule a new launch\n");
        printf("2. Update launch status\n");
        printf("3. Display all launch schedules\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleLaunch(launches, &totalLaunches);
                break;
            case 2:
                updateStatus(launches, totalLaunches);
                break;
            case 3:
                displayLaunchSchedule(launches, totalLaunches);
                break;
            case 4:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```