Day 14 programs

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>

#define MAX_ITEMS 100

typedef struct {
    int itemID;
    char itemName[50];
    float price;
    int quantity;
} Item;

// Function prototypes
void addItem(Item inventory[], int *totalItems);
void updateItem(Item inventory[], int totalItems);
void displayInventory(const Item inventory[], int totalItems);
```

```
int main() {
    Item inventory[MAX_ITEMS];
    int totalItems = 0;
    int choice;

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

        switch (choice) {
            case 1:
                addItem(inventory, &totalItems);
                break;
            case 2:
                updateItem(inventory, totalItems);
                break;
            case 3:
                displayInventory(inventory, totalItems);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
```

```
            printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

// Function to add a new item to the inventory
void addItem(Item inventory[], int *totalItems) {
    static int itemCount = 0;
    if (*totalItems < MAX_ITEMS) {
        Item newItem;
        newItem.itemID = ++itemCount;
        printf("Enter Item Name: ");
        scanf(" %[^\n]", newItem.itemName); // Read string with spaces
        printf("Enter Price: ");
        scanf("%f", &newItem.price);
        printf("Enter Quantity: ");
        scanf("%d", &newItem.quantity);

        inventory[*totalItems] = newItem;
        (*totalItems)++;
        printf("Item added successfully. Total items: %d\n", *totalItems);
    } else {
        printf("Inventory is full. Cannot add more items.\n");
    }
}

// Function to update an existing item's details
```

```
void updateItem(Item inventory[], int totalItems) {
    if (totalItems > 0) {
        int itemID;
        printf("Enter Item ID to update: ");
        scanf("%d", &itemID);
        int found = 0;
        for (int i = 0; i < totalItems; i++) {
            if (inventory[i].itemID == itemID) {
                printf("Updating Item ID %d\n", inventory[i].itemID);
                printf("Enter New Item Name (current: %s): ", inventory[i].itemName);
                scanf(" %[^\n]", inventory[i].itemName); // Read string with spaces
                printf("Enter New Price (current: %.2f): ", inventory[i].price);
                scanf("%f", &inventory[i].price);
                printf("Enter New Quantity (current: %d): ", inventory[i].quantity);
                scanf("%d", &inventory[i].quantity);
                printf("Item updated successfully.\n");
                found = 1;
                break;
            }
        }
        if (!found) {
            printf("Item with ID %d not found.\n", itemID);
        }
    } else {
        printf("Inventory is empty. Add items first.\n");
    }
}

// Function to display the entire inventory
```

```
void displayInventory(const Item inventory[], int totalItems) {
    if (totalItems > 0) {
        printf("\nCurrent Inventory:\n");
        printf("ID\tName\t\tPrice\tQuantity\n");
        printf("-----------------------------------------\n");
        for (int i = 0; i < totalItems; i++) {
            printf("%d\t%s\t\t%.2f\t%d\n", inventory[i].itemID, inventory[i].itemName,
inventory[i].price, inventory[i].quantity);
        }
    } else {
        printf("Inventory is empty.\n");
    }
}
```

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;
}
```

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>

#define MAX_FEEDBACKS 100

// Define a structure for feedback
typedef struct {
    int customerID;
    char feedbackText[255];
    int rating;  // Rating from 1 (Poor) to 5 (Excellent)
} Feedback;
```

```
// Function prototypes
void addFeedback(Feedback feedbacks[], int *totalFeedbacks);
void displayFeedbackSummary(Feedback feedbacks[], int totalFeedbacks);

int main() {
    Feedback feedbacks[MAX_FEEDBACKS];
    int totalFeedbacks = 0;
    int choice;

    do {
        printf("\nCustomer Feedback System\n");
        printf("1. Add Feedback\n");
        printf("2. Display Feedback Summaries\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addFeedback(feedbacks, &totalFeedbacks);
                break;
            case 2:
                displayFeedbackSummary(feedbacks, totalFeedbacks);
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
```

```
            printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 3);

    return 0;
}

// Function to add feedback
void addFeedback(Feedback feedbacks[], int *totalFeedbacks) {
    if (*totalFeedbacks < MAX_FEEDBACKS) {
        Feedback newFeedback;
        printf("Enter Customer ID: ");
        scanf("%d", &newFeedback.customerID);
        getchar();  // To consume the newline character after entering the customer ID

        printf("Enter Feedback Text: ");
        fgets(newFeedback.feedbackText, sizeof(newFeedback.feedbackText), stdin);

        printf("Enter Rating (1 to 5): ");
        scanf("%d", &newFeedback.rating);

        if (newFeedback.rating < 1 || newFeedback.rating > 5) {
            printf("Invalid rating. Please enter a rating between 1 and 5.\n");
            return;
        }

        feedbacks[*totalFeedbacks] = newFeedback;
        (*totalFeedbacks)++;
        printf("Feedback added successfully!\n");
```

```
    } else {
        printf("Feedback storage is full.\n");
    }
}

// Function to display categorized feedback summary
void displayFeedbackSummary(Feedback feedbacks[], int totalFeedbacks) {
    int excellent = 0, good = 0, average = 0, poor = 0;

    for (int i = 0; i < totalFeedbacks; i++) {
        switch (feedbacks[i].rating) {
            case 5:
                excellent++;
                break;
            case 4:
                good++;
                break;
            case 3:
                average++;
                break;
            case 2:
                poor++;
                break;
            case 1:
                poor++;
                break;
        }
    }
```

```
    printf("\nFeedback Summary:\n");
    printf("Excellent (Rating 5): %d\n", excellent);
    printf("Good (Rating 4): %d\n", good);
    printf("Average (Rating 3): %d\n", average);
    printf("Poor (Rating 1-2): %d\n", poor);
}
```

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>
#include<string.h>

#define CREDIT_CARD_CHARGE 0.03
#define DEBIT_CARD_CHARGE 0.02
#define PAYPAL_CHARGE 0.05


typedef struct {
    char method[20];
    float amount;
```

```
    float transactionCharge;
} Payment;

// Function prototypes
void processPayment(Payment* payment);
void displayPaymentDetails(const Payment* payment);

int main() {
    Payment payment;

    printf("Payment Method Selection\n");
    printf("Enter payment method (Credit Card, Debit Card, PayPal): ");
    scanf("%s", payment.method);

    printf("Enter payment amount: ");
    scanf("%f", &payment.amount);

    // Process the payment and calculate the transaction charge
    processPayment(&payment);

    // Display the payment details
    displayPaymentDetails(&payment);

    return 0;
}

// Function to process the payment and calculate transaction charges based on
payment method
void processPayment(Payment* payment) {
```

```
    if (payment == NULL) return;

    // Apply the transaction charge based on the payment method
    switch (payment->method[0]) {
        case 'C': // Credit Card
            if (strcmp(payment->method, "Credit") == 0) {
                payment->transactionCharge = payment->amount *
CREDIT_CARD_CHARGE;
                break;
            }
            break;
        case 'D': // Debit Card
            if (strcmp(payment->method, "Debit") == 0) {
                payment->transactionCharge = payment->amount *
DEBIT_CARD_CHARGE;
                break;
            }
            break;
        case 'P': // PayPal
            if (strcmp(payment->method, "PayPal") == 0) {
                payment->transactionCharge = payment->amount * PAYPAL_CHARGE;
                break;
            }
            break;
        default:
            payment->transactionCharge = 0;
            break;
    }
}
```

```
// Function to display payment details including method and transaction charge
void displayPaymentDetails(const Payment* payment) {
    if (payment == NULL) return;

    printf("\nPayment Details:\n");
    printf("Payment Method: %s\n", payment->method);
    printf("Payment Amount: %.2f\n", payment->amount);
    printf("Transaction Charge: %.2f\n", payment->transactionCharge);
    printf("Total Amount After Charge: %.2f\n", payment->amount + payment->transactionCharge);
}
```

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_ITEMS 100

typedef struct {
    int itemID;
    char itemName[50];
```

```
    float price;
    int quantity;
} CartItem;

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

int main() {
    CartItem cart[MAX_ITEMS];
    int totalItems = 0;
    int choice;

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

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

```
            break;
        case 3:
            displayCart(cart, totalItems);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 4);

return 0;
}

// Function to add an item to the cart
void addItem(CartItem cart[], int* totalItems) {
    if (*totalItems < MAX_ITEMS) {
        CartItem newItem;
        printf("Enter Item ID: ");
        scanf("%d", &newItem.itemID);
        printf("Enter Item Name: ");
        scanf(" %[^\n]", newItem.itemName);  // Read string with spaces
        printf("Enter Item Price: ");
        scanf("%f", &newItem.price);
        printf("Enter Item Quantity: ");
        scanf("%d", &newItem.quantity);

        cart[*totalItems] = newItem;
```

```
        (*totalItems)++;
        printf("Item added to cart.\n");
    } else {
        printf("Cart is full, cannot add more items.\n");
    }
}

// Function to remove an item from the cart
void removeItem(CartItem cart[], int* totalItems) {
    if (*totalItems > 0) {
        int itemID, found = 0;
        printf("Enter Item ID to remove: ");
        scanf("%d", &itemID);

        for (int i = 0; i < *totalItems; i++) {
            if (cart[i].itemID == itemID) {
                // Remove item by shifting items
                for (int j = i; j < *totalItems - 1; j++) {
                    cart[j] = cart[j + 1];
                }
                (*totalItems)--;
                found = 1;
                printf("Item removed from cart.\n");
                break;
            }
        }

        if (!found) {
            printf("Item with ID %d not found in cart.\n", itemID);
```

```
        }
    } else {
        printf("Cart is empty. No items to remove.\n");
    }
}

// Function to display the current cart items
void displayCart(const CartItem cart[], int totalItems) {
    if (totalItems > 0) {
        printf("\nShopping Cart Contents:\n");
        printf("ID\tName\t\tPrice\tQuantity\tTotal\n");
        printf("-----------------------------------------------------\n");
        for (int i = 0; i < totalItems; i++) {
            printf("%d\t%s\t\t%.2f\t%d\t\t%.2f\n",
                   cart[i].itemID, cart[i].itemName,
                   cart[i].price, cart[i].quantity,
                   cart[i].price * cart[i].quantity);
        }
    } else {
        printf("Your cart is empty.\n");
    }
}
```

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;
}
```

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 MAX_SALES 10

// Define a structure for Sale
typedef struct {
    int saleID;
    char productCategory[30];
    float amount;
    char date[15];
} Sale;

// Function prototypes
void addSale(Sale sales[], int* totalSales);
void generateReport(Sale sales[], int totalSales);
void displaySummaryByCategory(Sale sales[], int totalSales, char category[]);

int main() {
    Sale sales[MAX_SALES];
```

```
    int totalSales = 0;
    int choice;

    do {
        printf("\nSales Report Generator\n");
        printf("1. Add Sale\n");
        printf("2. Generate Sales Report\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // Consume newline character left by scanf

        switch (choice) {
            case 1:
                addSale(sales, &totalSales);
                break;
            case 2:
                generateReport(sales, totalSales);
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

```
// Function to add sale data
void addSale(Sale sales[], int* totalSales) {
    if (*totalSales >= MAX_SALES) {
        printf("Sales record is full, cannot add more sales.\n");
        return;
    }

    printf("Enter sale ID: ");
    scanf("%d", &sales[*totalSales].saleID);
    getchar();  // Consume newline character left by scanf

    printf("Enter product category: ");
    fgets(sales[*totalSales].productCategory,
sizeof(sales[*totalSales].productCategory), stdin);
    sales[*totalSales].productCategory[strcspn(sales[*totalSales].productCategory,
"\n")] = '\0';  // Remove newline character

    printf("Enter sale amount: ");
    scanf("%f", &sales[*totalSales].amount);
    getchar();  // Consume newline character left by scanf

    printf("Enter sale date (DD/MM/YYYY): ");
    fgets(sales[*totalSales].date, sizeof(sales[*totalSales].date), stdin);
    sales[*totalSales].date[strcspn(sales[*totalSales].date, "\n")] = '\0';  // Remove
newline character

    (*totalSales)++;
    printf("Sale added successfully!\n");
}
```

```
// Function to generate sales report
void generateReport(Sale sales[], int totalSales) {
    if (totalSales == 0) {
        printf("No sales data available to generate a report.\n");
        return;
    }

    printf("\nSales Report by Category:\n");

    // Summarize sales by category
    char categories[4][20] = {"Electronics", "Apparel", "Appliances", "Footwear"};
    for (int i = 0; i < 4; i++) {
        displaySummaryByCategory(sales, totalSales, categories[i]);
    }
}

// Function to display summarized sales by category
void displaySummaryByCategory(Sale sales[], int totalSales, char category[]) {
    float totalAmount = 0;
    int count = 0;

    for (int i = 0; i < totalSales; i++) {
        if (strcmp(sales[i].productCategory, category) == 0) {
            totalAmount += sales[i].amount;
            count++;
        }
    }
```

```
    if (count > 0) {
        printf("\nCategory: %s\n", category);
        printf("Number of Sales: %d\n", count);
        printf("Total Sales Amount: %.2f\n", totalAmount);
    }
}
```

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>

#define MAX_CUSTOMERS 5
#define REWARD_RATE 0.05 // 5% reward for total purchases

// Define a structure for Customer
typedef struct {
    int customerID;
    char name[50];
    float totalPurchases;
    float rewardPoints;
} Customer;
```

```
// Function prototypes
void calculateRewardPoints(Customer* customer);
void updateCustomer(Customer* customer);
void displayCustomerDetails(Customer customer);

int main() {
    Customer customers[MAX_CUSTOMERS];
    int totalCustomers = 0;
    int choice;

    do {
        printf("\nCustomer Loyalty Program\n");
        printf("1. Add/Update Customer\n");
        printf("2. Display Customer Details\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (totalCustomers < MAX_CUSTOMERS) {
                    updateCustomer(&customers[totalCustomers]);
                    totalCustomers++;
                } else {
                    printf("Maximum number of customers reached.\n");
                }
                break;
            case 2:
```

```
            for (int i = 0; i < totalCustomers; i++) {
                displayCustomerDetails(customers[i]);
            }
            break;
        case 3:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 3);

return 0;
}

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

// Function to add or update customer details
void updateCustomer(Customer* customer) {
    printf("Enter customer ID: ");
    scanf("%d", &customer->customerID);

    printf("Enter customer name: ");
    // We use scanf with %49s to avoid overflow
    scanf("%49s", customer->name);
```

```
    printf("Enter total purchases: ");
    scanf("%f", &customer->totalPurchases);

    calculateRewardPoints(customer);  // Update reward points based on total purchases

    printf("Customer details updated successfully!\n");
}

// Function to display customer details, including reward points
void displayCustomerDetails(Customer customer) {
    printf("\nCustomer ID: %d\n", customer.customerID);
    printf("Name: %s\n", customer.name);
    printf("Total Purchases: %.2f\n", customer.totalPurchases);
    printf("Reward Points: %.2f\n", customer.rewardPoints);
}
```

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>
```

```
#define MAX_ITEMS 5

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

// Function prototypes
void updateStock(WarehouseItem* item, int quantity);
void checkReorderStatus(WarehouseItem item);
void displayStock(WarehouseItem item);

int main() {
    WarehouseItem items[MAX_ITEMS] = {
        {1, "Laptop", 50, 20},
        {2, "Phone", 30, 15},
        {3, "Shirt", 100, 50},
        {4, "Shoes", 20, 10},
        {5, "Coffee Maker", 10, 5}
    };

    int totalItems = MAX_ITEMS;
    int choice, itemID, quantity;

    do {
```

```
printf("\nWarehouse Management System\n");
printf("1. Update Stock\n");
printf("2. Check Reorder Status\n");
printf("3. Display Stock\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter item ID to update stock: ");
        scanf("%d", &itemID);
        if (itemID > 0 && itemID <= totalItems) {
            printf("Enter quantity to update: ");
            scanf("%d", &quantity);
            updateStock(&items[itemID - 1], quantity);
        } else {
            printf("Invalid item ID.\n");
        }
        break;
    case 2:
        printf("Enter item ID to check reorder status: ");
        scanf("%d", &itemID);
        if (itemID > 0 && itemID <= totalItems) {
            checkReorderStatus(items[itemID - 1]);
        } else {
            printf("Invalid item ID.\n");
        }
        break;
```

```
            case 3:
                for (int i = 0; i < totalItems; i++) {
                    displayStock(items[i]);
                }
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

// Function to update stock levels
void updateStock(WarehouseItem* item, int quantity) {
    if (item != NULL) {
        item->currentStock += quantity;
        printf("Stock updated successfully!\n");
    }
}

// Function to check reorder status
void checkReorderStatus(WarehouseItem item) {
    if (item.currentStock <= item.reorderLevel) {
        printf("Reorder required for %s. Current stock: %d\n", item.itemName,
item.currentStock);
```

```
    } else {
        printf("Stock level for %s is sufficient. Current stock: %d\n", item.itemName,
item.currentStock);
    }
}

// Function to display stock information for an item
void displayStock(WarehouseItem item) {
    printf("\nItem ID: %d\n", item.itemID);
    printf("Item Name: %s\n", item.itemName);
    printf("Current Stock: %d\n", item.currentStock);
    printf("Reorder Level: %d\n", item.reorderLevel);
}
```

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>

#define MAX_CATEGORIES 3
```

```
// Define a structure for Discount
typedef struct {
    char category[50];
    float discountPercentage;
    char validTill[20];
} Discount;

// Define constant categories
const char* categories[MAX_CATEGORIES] = {"Electronics", "Apparel",
"Groceries"};

// Function prototypes
void applyDiscount(Discount* discount, const char* category, float percentage, const
char* validTill);
void displayDiscount(Discount discount);

int main() {
    Discount discounts[MAX_CATEGORIES] = {
        {"Electronics", 10.0, "2025-12-31"},
        {"Apparel", 15.0, "2025-06-30"},
        {"Groceries", 5.0, "2025-03-31"}
    };

    int choice;
    char category[50];
    float percentage;
    char validTill[20];

    do {
        printf("\nDiscount Management System\n");
```

```
printf("1. Apply Discount\n");
printf("2. Display Discount Details\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter category to apply discount (Electronics/Apparel/Groceries): ");
        fgets(category, sizeof(category), stdin);
        category[strcspn(category, "\n")] = '\0';
        printf("Enter discount percentage: ");
        scanf("%f", &percentage);
        getchar(); // Consume newline
        printf("Enter validity date (YYYY-MM-DD): ");
        fgets(validTill, sizeof(validTill), stdin);
        validTill[strcspn(validTill, "\n")] = '\0';

        // Apply the discount to the corresponding category
        for (int i = 0; i < MAX_CATEGORIES; i++) {
            if (strcmp(discounts[i].category, category) == 0) {
                applyDiscount(&discounts[i], category, percentage, validTill);
                break;
            }
        }
        break;
    case 2:
        // Display discount details for each category
        for (int i = 0; i < MAX_CATEGORIES; i++) {
```

```
                displayDiscount(discounts[i]);
            }
            break;
        case 3:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 3);

return 0;
}

// Function to apply discount to a product category
void applyDiscount(Discount* discount, const char* category, float percentage, const char* validTill) {
    strcpy(discount->category, category);
    discount->discountPercentage = percentage;
    strcpy(discount->validTill, validTill);
    printf("Discount applied successfully!\n");
}

// Function to display discount details
void displayDiscount(Discount discount) {
    printf("\nCategory: %s\n", discount.category);
    printf("Discount Percentage: %.2f%%\n", discount.discountPercentage);
    printf("Valid Till: %s\n", discount.validTill);
}
```

Union programs

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>

typedef union {
    int a;
    float b;
    char c;
} MixedData;

int main() {
    MixedData data;



    printf("Enter the value for a (integer): ");
    scanf("%d", &data.a);
    printf("The value of a is: %d\n", data.a);



    printf("Enter the value for b (float): ");
    scanf("%f", &data.b);
    printf("The value of b is: %.2f\n", data.b);
```

```
    printf("Enter the value for c (character): ");
    scanf(" %c", &data.c);
    printf("The value of c is: %c\n", data.c);

    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>

// Define a union to store either roll number or name
typedef union {
    int rollNumber;
    char name[50];
} StudentData;

int main() {
    StudentData student;
    int choice;

    printf("1. Enter Roll Number\n");
    printf("2. Enter Name\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter Roll Number: ");
```

```
        scanf("%d", &student.rollNumber);
        printf("Student Roll Number: %d\n", student.rollNumber);
    } else if (choice == 2) {
        printf("Enter Name: ");
        scanf("%s", student.name);
        printf("Student Name: %s\n", student.name);
    } else {
        printf("Invalid choice!\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>



typedef union {
    float kilometers;
    float miles;
} Distance;

int main() {
    Distance distance;
    int choice;

    printf("1. Enter Distance in Kilometers\n");
```

```
    printf("2. Enter Distance in Miles\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter distance in kilometers: ");
        scanf("%f", &distance.kilometers);
        printf("Distance in kilometers: %.2f km\n", distance.kilometers);
        printf("Distance in miles: %.2f miles\n", distance.kilometers * 0.621371);
    } else if (choice == 2) {
        printf("Enter distance in miles: ");
        scanf("%f", &distance.miles);
        printf("Distance in miles: %.2f miles\n", distance.miles);
        printf("Distance in kilometers: %.2f km\n", distance.miles / 0.621371);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }

    return 0;
}
```

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>

typedef union {
    float radius;
    float length;
```

```
    float width;
} shapeArea;

int main() {
    shapeArea shape;
    int choice;
    float area;

    printf("1. Circle\n");
    printf("2. Rectangle\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter radius of circle: ");
        scanf("%f", &shape.radius);
        area=3.14*shape.radius*shape.radius;
        printf("Area of circle is: %lf\n",area);
    } else if (choice == 2) {
        printf("Enter the length: ");
        scanf("%f",&shape.length);
        printf("Enter the width: ");
        scanf("%f",&shape.width);
        area=shape.length*shape.width;
        printf("Area of rectangle=: %lf\n",area);
    } else {
        printf("Invalid choice!\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>

typedef union {
    int empid;
    float salary;
} Employee;

int main() {
    Employee employee;
    int choice;

    printf("1. Id\n");
    printf("2. Salary\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter the Employee Id: ");
        scanf("%d", &employee.empid);
        printf("The employee id is %d\n",employee.empid);
    } else if (choice == 2) {
        printf("Enter the salary: ");
        scanf("%f",&employee.salary);
```

```
        printf("The employee salary is %f\n",employee.salary);
    } else {
        printf("Invalid choice!\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>

typedef union {
    float temperature;  // To store temperature
    float pressure;     // To store pressure
} SensorData;

int main() {
    SensorData sensor;
    int choice;

    printf("1. Enter Temperature (°C)\n");
    printf("2. Enter Pressure (Pa)\n");
    printf("Enter choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter temperature (°C): ");
```

```
        scanf("%f", &sensor.temperature);
        printf("Temperature: %.2f °C\n", sensor.temperature);
    } else if (choice == 2) {
        printf("Enter pressure (Pa): ");
        scanf("%f", &sensor.pressure);
        printf("Pressure: %.2f Pa\n", sensor.pressure);
    } else {
        printf("Invalid choice!\n");
    }

    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>

typedef union {
    int accno;
    float balance;
} Bank;

int main() {
    Bank bank;
    int choice;

    printf("1. Account Number\n");
```

```
    printf("2. Balance\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter the Account Number: ");
        scanf("%d", &bank.accno);
        printf("The account number is %d\n",bank.accno);
    } else if (choice == 2) {
        printf("Enter the balance: ");
        scanf("%f",&bank.balance);
        printf("The balance is %f\n",bank.balance);
    } else {
        printf("Invalid choice!\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>

typedef union {
    int regno;
    float fuelcap;
} Vehicle;
```

```
int main() {
    Vehicle vehicle;
    int choice;

    printf("1. Regno\n");
    printf("2. Fuel Capacity\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter the Reg No: ");
        scanf("%d", &vehicle.regno);
        printf("The Reg no is %d\n",vehicle.regno);
    } else if (choice == 2) {
        printf("Enter the fuel capacity: ");
        scanf("%f",&vehicle.fuelcap);
        printf("The fuel capacity is %f\n",vehicle.fuelcap);
    } 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>
```

```
typedef union {
    int marks;
    char grade;
} Student;

int main() {
    Student student;
    int choice;

    printf("1. Marks\n");
    printf("2. Grade\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter the Marks: ");
        scanf("%d", &student.marks);
        printf("The marks is: %d\n",student.marks);
    } else if (choice == 2) {
        printf("Enter the Grade: ");
        scanf(" %c",&student.grade);
        printf("The grade is: %c\n",student.grade);
    } else {
        printf("Invalid choice!\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>



typedef union {
    float usd;  // To store value in USD
    float eur;  // To store value in EUR
} Currency;


int main() {
    Currency currency;
    int choice;
    const float conversionRate = 0.85; //1 USD = 0.85 EUR


    printf("Currency Conversion\n");
    printf("1. Enter value in USD\n");
    printf("2. Enter value in EUR\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);


    if (choice == 1) {
        printf("Enter value in USD: ");
        scanf("%f", &currency.usd);
        // Convert USD to EUR
        printf("The value in EUR is: %.2f\n", currency.usd * conversionRate);
    } else if (choice == 2) {
        printf("Enter value in EUR: ");
```

```
        scanf("%f", &currency.eur);
        // Convert EUR to USD
        printf("The value in USD is: %.2f\n", currency.eur / conversionRate);
    } else {
        printf("Invalid choice!\n");
    }

    return 0;
}
```

set 3 programs

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 MAX_AIRCRAFT 5
```

```
typedef struct {
    int aircraftID;
    char model[50];
    int capacity;
    char status[20];
} Aircraft;

static int totalAircraft = 0;

void addAircraft(Aircraft *fleet, int *total) {
    printf("Enter aircraft ID: ");
    scanf("%d", &fleet[*total].aircraftID);
    printf("Enter aircraft model: ");
    scanf("%s", fleet[*total].model);
    printf("Enter aircraft capacity: ");
    scanf("%d", &fleet[*total].capacity);
    printf("Enter aircraft status: ");
    scanf("%s", fleet[*total].status);
    (*total)++;
}

void updateStatus(Aircraft *fleet, int total) {
    int id;
    printf("Enter aircraft ID to update status: ");
    scanf("%d", &id);
    for (int i = 0; i < total; i++) {
        if (fleet[i].aircraftID == id) {
            printf("Enter new status: ");
            scanf("%s", fleet[i].status);
```

```
            break;
        }
    }
}

void displayFleet(Aircraft *fleet, int total) {
    for (int i = 0; i < total; i++) {
        printf("ID: %d, Model: %s, Capacity: %d, Status: %s\n",
               fleet[i].aircraftID, fleet[i].model, fleet[i].capacity, fleet[i].status);
    }
}

int main() {
    Aircraft fleet[MAX_AIRCRAFT];
    int choice;

    do {
        printf("\n1. Add Aircraft\n2. Update Status\n3. Display Fleet\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addAircraft(fleet, &totalAircraft);
                break;
            case 2:
                updateStatus(fleet, totalAircraft);
                break;
            case 3:
```

```
            displayFleet(fleet, totalAircraft);
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }
} while (choice != 4);

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>
#include <string.h>

#define MAX_DATA_POINTS 5
```

```
typedef union {
    char imageData[100];
    struct {
        float temperature;
        float velocity;
        float altitude;
    } telemetryData;
} SatelliteData;

void processImage(SatelliteData *data) {
    printf("Processing image data: %s\n", data->imageData);
}

void processTelemetry(SatelliteData *data) {
    printf("Temperature: %.2f, Velocity: %.2f, Altitude: %.2f\n",
           data->telemetryData.temperature, data->telemetryData.velocity, data->telemetryData.altitude);
}

int main() {
    SatelliteData data[MAX_DATA_POINTS];
    int choice, i;

    for (i = 0; i < MAX_DATA_POINTS; i++) {
        printf("1. Enter image data\n2. Enter telemetry data\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter image data: ");
                scanf("%s", data[i].imageData);
                processImage(&data[i]);
                break;
            case 2:
                printf("Enter temperature: ");
                scanf("%f", &data[i].telemetryData.temperature);
                printf("Enter velocity: ");
                scanf("%f", &data[i].telemetryData.velocity);
                printf("Enter altitude: ");
                scanf("%f", &data[i].telemetryData.altitude);
                processTelemetry(&data[i]);
                break;
            default:
                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.

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

#define MAX_MISSIONS 3

typedef union {
    char crewDetails[100];
    char cargoDetails[100];
} Payload;

typedef struct {
    int missionID;
    char name[50];
    int duration; // in days
    Payload payload;
    int isCrew; // 1 for crew, 0 for cargo
} Mission;

static int totalMissions = 0;

void addMission(Mission *missions) {
    printf("Enter mission ID: ");
    scanf("%d", &missions[totalMissions].missionID);
```

```
    printf("Enter mission name: ");
    scanf("%s", missions[totalMissions].name);
    printf("Enter mission duration: ");
    scanf("%d", &missions[totalMissions].duration);

    printf("Enter payload details: ");
    if (missions[totalMissions].isCrew) {
        printf("Enter crew details: ");
        scanf("%s", missions[totalMissions].payload.crewDetails);
    } else {
        printf("Enter cargo details: ");
        scanf("%s", missions[totalMissions].payload.cargoDetails);
    }

    totalMissions++;
}

void displayMissionSummary(Mission *missions) {
    for (int i = 0; i < totalMissions; i++) {
        printf("Mission ID: %d, Name: %s, Duration: %d days\n",
               missions[i].missionID, missions[i].name, missions[i].duration);
        if (missions[i].isCrew) {
            printf("Payload: Crew - %s\n", missions[i].payload.crewDetails);
        } else {
            printf("Payload: Cargo - %s\n", missions[i].payload.cargoDetails);
        }
    }
}
```

```
int main() {
    Mission missions[MAX_MISSIONS];
    int choice;

    do {
        printf("\n1. Add Mission\n2. Display Mission Summary\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addMission(missions);
                break;
            case 2:
                displayMissionSummary(missions);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);

    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 5

typedef union {
    char routine[50];
    char emergency[50];
} MaintenanceType;

typedef struct {
    int logID;
    int aircraftID;
    char date[20];
    MaintenanceType maintenanceType;
    int isRoutine; // 1 for routine, 0 for emergency
} MaintenanceLog;

static int totalLogs = 0;
```

```
void addMaintenanceLog(MaintenanceLog *logs) {
    printf("Enter log ID: ");
    scanf("%d", &logs[totalLogs].logID);
    printf("Enter aircraft ID: ");
    scanf("%d", &logs[totalLogs].aircraftID);
    printf("Enter maintenance date (YYYY-MM-DD): ");
    scanf("%s", logs[totalLogs].date);

    printf("Enter maintenance type (1 for routine, 0 for emergency): ");
    scanf("%d", &logs[totalLogs].isRoutine);

    if (logs[totalLogs].isRoutine) {
        printf("Enter routine maintenance details: ");
        scanf("%s", logs[totalLogs].maintenanceType.routine);
    } else {
        printf("Enter emergency maintenance details: ");
        scanf("%s", logs[totalLogs].maintenanceType.emergency);
    }

    totalLogs++;
}

void displayMaintenanceLogs(MaintenanceLog *logs) {
    for (int i = 0; i < totalLogs; i++) {
        printf("Log ID: %d, Aircraft ID: %d, Date: %s\n", logs[i].logID, logs[i].aircraftID,
logs[i].date);
        if (logs[i].isRoutine) {
            printf("Routine Maintenance: %s\n", logs[i].maintenanceType.routine);
        } else {
```

```
            printf("Emergency Maintenance: %s\n",
logs[i].maintenanceType.emergency);
        }
    }
}

int main() {
    MaintenanceLog logs[MAX_LOGS];
    int choice;

    do {
        printf("\n1. Add Maintenance Log\n2. Display Maintenance Logs\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addMaintenanceLog(logs);
                break;
            case 2:
                displayMaintenanceLogs(logs);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);
```

```
    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>
#include <string.h>

#define MAX_UPDATES 5

typedef union {
    char manual[50];
    char automatic[50];
} NavigationMode;

typedef struct {
    int updateID;
    float position[3]; // x, y, z coordinates
```

```
    float velocity;
    NavigationMode mode;
    int isManual; // 1 for manual, 0 for automatic
} NavigationData;

static int totalUpdates = 0;

void updateNavigationData(NavigationData *updates) {
    printf("Enter update ID: ");
    scanf("%d", &updates[totalUpdates].updateID);
    printf("Enter position (x, y, z): ");
    scanf("%f %f %f", &updates[totalUpdates].position[0],
&updates[totalUpdates].position[1], &updates[totalUpdates].position[2]);
    printf("Enter velocity: ");
    scanf("%f", &updates[totalUpdates].velocity);

    printf("Enter navigation mode (1 for manual, 0 for automatic): ");
    scanf("%d", &updates[totalUpdates].isManual);

    if (updates[totalUpdates].isManual) {
        printf("Enter manual navigation mode details: ");
        scanf("%s", updates[totalUpdates].mode.manual);
    } else {
        printf("Enter automatic navigation mode details: ");
        scanf("%s", updates[totalUpdates].mode.automatic);
    }

    totalUpdates++;
}
```

```
void displayNavigationStatus(NavigationData *updates) {
    for (int i = 0; i < totalUpdates; i++) {
        printf("Update ID: %d, Position: (%.2f, %.2f, %.2f), Velocity: %.2f\n",
               updates[i].updateID, updates[i].position[0], updates[i].position[1],
updates[i].position[2], updates[i].velocity);
        if (updates[i].isManual) {
            printf("Navigation Mode: Manual - %s\n", updates[i].mode.manual);
        } else {
            printf("Navigation Mode: Automatic - %s\n", updates[i].mode.automatic);
        }
    }
}

int main() {
    NavigationData updates[MAX_UPDATES];
    int choice;

    do {
        printf("\n1. Update Navigation Data\n2. Display Navigation Status\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                updateNavigationData(updates);
                break;
            case 2:
                displayNavigationStatus(updates);
```

```
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }
} while (choice != 3);

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
```

```
#define SIMULATION_DURATION 60 // in minutes, fixed duration

// Union for control settings: manual or automated
typedef union {
    char manualControl[50];
    char automatedControl[50];
} ControlSettings;

// Struct for simulation data
typedef struct {
    int simulationID;
    char aircraftModel[50];
    int duration; // simulation duration in minutes
    ControlSettings controlSettings;
    int isManual; // 1 for manual, 0 for automated
} Simulation;

static int totalSimulations = 0;

// Function to start a new simulation
void startSimulation(Simulation *simulations) {
    printf("Enter simulation ID: ");
    scanf("%d", &simulations[totalSimulations].simulationID);

    printf("Enter aircraft model: ");
    scanf("%s", simulations[totalSimulations].aircraftModel);

    // Set fixed simulation duration
    simulations[totalSimulations].duration = SIMULATION_DURATION;
```

```
    printf("Enter control type (1 for Manual, 0 for Automated): ");
    scanf("%d", &simulations[totalSimulations].isManual);

    if (simulations[totalSimulations].isManual) {
        printf("Enter manual control settings: ");
        scanf(" %[^\n]", simulations[totalSimulations].controlSettings.manualControl);
    } else {
        printf("Enter automated control settings: ");
        scanf(" %[^\n]", simulations[totalSimulations].controlSettings.automatedControl);
    }

    totalSimulations++;
}

// Function to display the simulation results
void displaySimulationResults(Simulation *simulations) {
    for (int i = 0; i < totalSimulations; i++) {
        printf("\nSimulation ID: %d\n", simulations[i].simulationID);
        printf("Aircraft Model: %s\n", simulations[i].aircraftModel);
        printf("Simulation Duration: %d minutes\n", simulations[i].duration);

        // Display control settings based on the type
        if (simulations[i].isManual) {
            printf("Control Type: Manual\n");
            printf("Manual Control Settings: %s\n",
simulations[i].controlSettings.manualControl);
        } else {
            printf("Control Type: Automated\n");
```

```
            printf("Automated Control Settings: %s\n",
simulations[i].controlSettings.automatedControl);
        }
    }
}

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

    do {
        printf("\n1. Start New Simulation\n2. Display Simulation Results\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                startSimulation(simulations);
                break;
            case 2:
                displaySimulationResults(simulations);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 3);
```

```
    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_TESTS 5

// Union for test data: physical or software
typedef union {
    float physicalData;   // e.g., weight, size, etc.
    char softwareData[100]; // software version or logs
} TestData;

// Struct for component test details
typedef struct {
    int testID;
```

```
    char componentName[50];
    TestData testData;
    int isPhysical; // 1 for physical test, 0 for software test
} ComponentTest;

static int totalTests = 0;

// Function to record test results
void recordTestResults(ComponentTest *test) {
    printf("Enter test ID: ");
    scanf("%d", &test->testID);

    printf("Enter component name: ");
    scanf(" %[^\n]", test->componentName);

    printf("Enter test type (1 for Physical, 0 for Software): ");
    scanf("%d", &test->isPhysical);

    if (test->isPhysical) {
        printf("Enter physical test data (e.g., weight or size): ");
        scanf("%f", &test->testData.physicalData);
    } else {
        printf("Enter software test data (e.g., software version or logs): ");
        scanf(" %[^\n]", test->testData.softwareData);
    }

    totalTests++;
}
```

```
// Function to display test results
void displayTestResults(ComponentTest *tests) {
    for (int i = 0; i < totalTests; i++) {
        printf("\nTest ID: %d\n", tests[i].testID);
        printf("Component Name: %s\n", tests[i].componentName);

        // Display based on test type
        if (tests[i].isPhysical) {
            printf("Test Type: Physical\n");
            printf("Physical Test Data: %.2f\n", tests[i].testData.physicalData);
        } else {
            printf("Test Type: Software\n");
            printf("Software Test Data: %s\n", tests[i].testData.softwareData);
        }
    }
}

int main() {
    ComponentTest tests[MAX_TESTS];
    int choice;

    do {
        printf("\n1. Record New Test Results\n2. Display Test Summaries\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (totalTests < MAX_TESTS) {
```

```
                recordTestResults(&tests[totalTests]);
            } else {
                printf("Maximum number of tests reached.\n");
            }
            break;
        case 2:
            if (totalTests == 0) {
                printf("No tests recorded yet.\n");
            } else {
                displayTestResults(tests);
            }
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 3);

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 5

typedef union {
    char engineerDetails[50];
    char scientistDetails[50];
} RoleDetails;

typedef struct {
    int crewID;
    char name[50];
    char role[20];
    RoleDetails roleDetails;
    int isEngineer; // 1 for engineer, 0 for scientist
} CrewMember;

static int totalCrew = 0;

void addCrewMember(CrewMember *crew) {
    printf("Enter crew ID: ");
    scanf("%d", &crew[totalCrew].crewID);
```

```
    printf("Enter crew name: ");
    scanf("%s", crew[totalCrew].name);
    printf("Enter role (Engineer/Scientist): ");
    scanf("%s", crew[totalCrew].role);

    if (strcmp(crew[totalCrew].role, "Engineer") == 0) {
        crew[totalCrew].isEngineer = 1;
        printf("Enter engineer details: ");
        scanf("%s", crew[totalCrew].roleDetails.engineerDetails);
    } else {
        crew[totalCrew].isEngineer = 0;
        printf("Enter scientist details: ");
        scanf("%s", crew[totalCrew].roleDetails.scientistDetails);
    }

    totalCrew++;
}

void displayCrewDetails(CrewMember *crew) {
    for (int i = 0; i < totalCrew; i++) {
        printf("Crew ID: %d, Name: %s, Role: %s\n", crew[i].crewID, crew[i].name,
crew[i].role);
        if (crew[i].isEngineer) {
            printf("Role Details (Engineer): %s\n", crew[i].roleDetails.engineerDetails);
        } else {
            printf("Role Details (Scientist): %s\n", crew[i].roleDetails.scientistDetails);
        }
    }
}
```

```
int main() {
    CrewMember crew[MAX_CREW];
    int choice;

    do {
        printf("\n1. Add Crew Member\n2. Display Crew Details\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addCrewMember(crew);
                break;
            case 2:
                displayCrewDetails(crew);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);

    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_ANALYSES 5

typedef union {
    int numericalData[50];
    char qualitativeData[50];
} DataType;

typedef struct {
    int experimentID;
    char description[100];
    DataType data;
    int isNumerical; // 1 for numerical data, 0 for qualitative data
} ResearchData;

static int totalAnalyses = 0;
```

```
void analyzeData(ResearchData *data) {
    printf("Enter experiment ID: ");
    scanf("%d", &data[totalAnalyses].experimentID);
    printf("Enter description of experiment: ");
    scanf(" %[^\n]", data[totalAnalyses].description);

    printf("Enter data type (1 for numerical, 0 for qualitative): ");
    scanf("%d", &data[totalAnalyses].isNumerical);

    if (data[totalAnalyses].isNumerical) {
        printf("Enter numerical data (comma separated): ");
        for (int i = 0; i < 5; i++) {
            scanf("%d", &data[totalAnalyses].data.numericalData[i]);
        }
    } else {
        printf("Enter qualitative data: ");
        scanf(" %[^\n]", data[totalAnalyses].data.qualitativeData);
    }

    totalAnalyses++;
}

void generateReport(ResearchData *data) {
    for (int i = 0; i < totalAnalyses; i++) {
        printf("Experiment ID: %d, Description: %s\n", data[i].experimentID,
data[i].description);
        if (data[i].isNumerical) {
            printf("Numerical Data: ");
```

```
            for (int j = 0; j < 5; j++) {
                printf("%d ", data[i].data.numericalData[j]);
            }
            printf("\n");
        } else {
            printf("Qualitative Data: %s\n", data[i].data.qualitativeData);
        }
    }
}

int main() {
    ResearchData data[MAX_ANALYSES];
    int choice;

    do {
        printf("\n1. Analyze Data\n2. Generate Report\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                analyzeData(data);
                break;
            case 2:
                generateReport(data);
                break;
            case 3:
                printf("Exiting...\n");
                break;
```

```
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);

    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

typedef union {
    char scheduled[50];
    char completed[50];
} LaunchStatus;
```

```
typedef struct {
    int launchID;
    char rocketName[50];
    char date[20];
    LaunchStatus status;
    int isScheduled; // 1 for scheduled, 0 for completed
} Launch;

static int totalLaunches = 0;

void scheduleLaunch(Launch *launches) {
    printf("Enter launch ID: ");
    scanf("%d", &launches[totalLaunches].launchID);
    printf("Enter rocket name: ");
    scanf("%s", launches[totalLaunches].rocketName);
    printf("Enter launch date (YYYY-MM-DD): ");
    scanf("%s", launches[totalLaunches].date);

    printf("Enter launch status (1 for scheduled, 0 for completed): ");
    scanf("%d", &launches[totalLaunches].isScheduled);

    if (launches[totalLaunches].isScheduled) {
        printf("Enter scheduled launch details: ");
        scanf("%s", launches[totalLaunches].status.scheduled);
    } else {
        printf("Enter completed launch details: ");
        scanf("%s", launches[totalLaunches].status.completed);
    }
```

```
    totalLaunches++;
}

void displayLaunchSchedules(Launch *launches) {
    for (int i = 0; i < totalLaunches; i++) {
        printf("Launch ID: %d, Rocket Name: %s, Date: %s\n",
               launches[i].launchID, launches[i].rocketName, launches[i].date);
        if (launches[i].isScheduled) {
            printf("Status: Scheduled - %s\n", launches[i].status.scheduled);
        } else {
            printf("Status: Completed - %s\n", launches[i].status.completed);
        }
    }
}

int main() {
    Launch launches[MAX_LAUNCHES];
    int choice;

    do {
        printf("\n1. Schedule Launch\n2. Display Launch Schedules\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleLaunch(launches);
                break;
```

```
            case 2:
                displayLaunchSchedules(launches);
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);

    return 0;
}
```