SET OF PROGRAMS

1.Inventory Update System

Input: An array of integers representing inventory levels and an array of changes in stock.

Process: Pass the arrays to a function by reference to update inventory levels.

Output: Print the updated inventory levels and flag items below the restocking threshold.

Concepts: Arrays, functions, pass by reference, decision-making (if-else).

```
#include <stdio.h>

// Function to update inventory levels
void updateInventory(int inventory[], int changes[], int size, int restockThreshold) {
    for (int i = 0; i < size; i++) {
        inventory[i] += changes[i];  // Apply the changes to inventory levels

        // Print updated inventory and check if it’s below restocking threshold
        printf("Item %d: Updated inventory = %d", i + 1, inventory[i]);

        // Check if inventory is below restocking threshold
        if (inventory[i] < restockThreshold) {
            printf(" [Below Restocking Threshold]\n");
        } else {
            printf("\n");
        }
    }
}

int main() {
    int inventory[] = {100, 50, 30, 200, 10};  // Initial inventory levels
    int changes[] = {-20, 10, -5, -30, 15};    // Stock changes (can be positive or negative)
    int size = sizeof(inventory) / sizeof(inventory[0]); // Get the size of the arrays
    int restockThreshold = 25; // Define the restocking threshold

    // Call the function to update inventory
    updateInventory(inventory, changes, size, restockThreshold);

    return 0;
}
```

2.Product Price Adjustment

Input: An array of demand levels (constant) and an array of product prices.

Process: Use a function to calculate new prices based on demand levels. The function should return a pointer to an array of adjusted prices.

Output: Display the original and adjusted prices.

Concepts: Passing constant data, functions, pointers, arrays.

```
#include <stdio.h>

// Function to adjust product prices based on demand level
void adjustPrices(const int *demandLevels, int *prices, int length) {
    for (int i = 0; i < length; i++) {
        // Adjust price based on demand (simple logic for illustration)
        // For example: if demand level is higher, increase the price by 10%.
        if (demandLevels[i] > 50) {
            prices[i] += prices[i] * 0.10;  // Increase price by 10%
```

```
        } else {
            prices[i] -= prices[i] * 0.05;  // Decrease price by 5%
        }
    }
}

int main() {
    // Example input arrays for demand levels and prices
    int demandLevels[] = {30, 70, 50, 80, 40}; // Demand levels for 5 products
    int prices[] = {100, 200, 150, 250, 120}; // Original prices for the same products
    int length = sizeof(demandLevels) / sizeof(demandLevels[0]); // Calculate number of products

    printf("Original Prices: \n");
    for (int i = 0; i < length; i++) {
        printf("Product %d: $%d\n", i+1, prices[i]);
    }

    // Adjust prices based on demand levels
    adjustPrices(demandLevels, prices, length);

    printf("\nAdjusted Prices: \n");
    for (int i = 0; i < length; i++) {
        printf("Product %d: $%d\n", i+1, prices[i]);
    }

    return 0;
}
```

3.Daily Sales Tracker
Input: Array of daily sales amounts.
Process: Use do-while to validate sales data input. Use a function to calculate total sales using pointers.
Output: Display total sales for the day.
Concepts: Loops, arrays, pointers, functions.

```
#include <stdio.h>

// Function to calculate total sales using pointers
void calculateTotalSales(float *sales, int numDays, float *totalSales) {
    *totalSales = 0.0;
    for (int i = 0; i < numDays; i++) {
        *totalSales += *(sales + i);  // Add sales amount using pointer arithmetic
    }
}

int main() {
    int numDays;
    float totalSales;

    // Get number of days for tracking sales
    printf("Enter the number of days to track sales: ");
    scanf("%d", &numDays);

    // Create an array to store sales data
    float sales[numDays];
```

```
    // Input daily sales amounts using a do-while loop for validation
    int i = 0;
    do {
        printf("Enter sales for day %d (positive value): ", i + 1);
        scanf("%f", &sales[i]);

        if (sales[i] < 0) {
            printf("Invalid input! Please enter a non-negative sales amount.\n");
        }

        i++;
    } while (i < numDays);

    // Calculate total sales using pointers
    calculateTotalSales(sales, numDays, &totalSales);

    // Display total sales for the day
    printf("Total sales for the day: %.2f\n", totalSales);

    return 0;
}
```

4.Discount Decision System
Input: Array of sales volumes.
Process: Pass the sales volume array by reference to a function. Use a switch statement to assign discount rates.
Output: Print discount rates for each product.
Concepts: Decision-making (switch), arrays, pass by reference, functions.

```
#include <stdio.h>

// Function to assign discount rates based on sales volume
void assignDiscountRate(int sales[], int n) {
    // Loop through each sales volume to assign a discount
    for (int i = 0; i < n; i++) {
        int discountRate;

        // Using switch to assign discount based on sales volume
        switch (sales[i]) {
            case 0 ... 10:    // sales between 0 and 10 units
                discountRate = 5; // 5% discount
                break;
            case 11 ... 50:   // sales between 11 and 50 units
                discountRate = 10; // 10% discount
                break;
            case 51 ... 100:  // sales between 51 and 100 units
                discountRate = 15; // 15% discount
                break;
            case 101 ... 200: // sales between 101 and 200 units
                discountRate = 20; // 20% discount
                break;
            default:          // sales above 200 units
                discountRate = 25; // 25% discount
                break;
        }
```

```
        // Output the discount for the current product
        printf("Product %d with sales volume %d gets a discount of %d%%.\n", i + 1, sales[i], discountRate);
    }
}

int main() {
    // Define an array of sales volumes
    int sales[] = {5, 30, 75, 150, 220};

    // Get the number of products (size of the array)
    int n = sizeof(sales) / sizeof(sales[0]);

    // Call the function to assign discount rates
    assignDiscountRate(sales, n);

    return 0;
}
```

5.Transaction Anomaly Detector
Input: Array of transaction amounts.
Process: Use pointers to traverse the array. Classify transactions as "Normal" or "Suspicious" based on thresholds using if-else.
Output: Print classification for each transaction.
Concepts: Arrays, pointers, loops, decision-making.

```
#include <stdio.h>

#define NORMAL_THRESHOLD 1000
#define SUSPICIOUS_THRESHOLD 5000

void classify_transactions(double *transactions, int size) {
    for (int i = 0; i < size; i++) {
        if (*(transactions + i) <= NORMAL_THRESHOLD) {
            printf("Transaction %d: Normal\n", i + 1);
        } else if (*(transactions + i) > NORMAL_THRESHOLD && *(transactions + i) <=
SUSPICIOUS_THRESHOLD) {
            printf("Transaction %d: Suspicious\n", i + 1);
        } else {
            printf("Transaction %d: Very Suspicious\n", i + 1);
        }
    }
}

int main() {
    double transactions[] = {200, 1500, 3000, 7000, 450};
    int size = sizeof(transactions) / sizeof(transactions[0]);

    printf("Classifying Transactions:\n");
    classify_transactions(transactions, size);

    return 0;
}
```

6.Account Balance Operations

Input: Array of account balances.
Process: Pass the balances array to a function that calculates interest. Return a pointer to the updated balances array.
Output: Display updated balances.
Concepts: Functions, arrays, pointers, loops.

```
#include <stdio.h>

// Function to calculate interest on account balances
void calculateInterest(float *balances, int size, float interestRate) {
    for (int i = 0; i < size; i++) {
        balances[i] = balances[i] + (balances[i] * interestRate / 100); // Update balance with interest
    }
}

// Function to display the account balances
void displayBalances(float *balances, int size) {
    printf("Updated Account Balances:\n");
    for (int i = 0; i < size; i++) {
        printf("Account %d: $%.2f\n", i + 1, balances[i]);
    }
}

int main() {
    // Example array of account balances
    float balances[] = {1000.50, 2500.75, 3500.20, 4500.10};
    int size = sizeof(balances) / sizeof(balances[0]); // Get the number of accounts

    // Example interest rate
    float interestRate = 5.0; // 5% interest rate

    // Display original balances
    printf("Original Account Balances:\n");
    displayBalances(balances, size);

    // Calculate interest and update the balances array
    calculateInterest(balances, size, interestRate);

    // Display updated balances
    displayBalances(balances, size);

    return 0;
}
```

7.Bank Statement Generator
Input: Array of transaction types (e.g., 1 for Deposit, 2 for Withdrawal) and amounts.
Process: Use a switch statement to classify transactions. Pass the array as a constant parameter to a function.
Output: Summarize total deposits and withdrawals.
Concepts: Decision-making, passing constant data, arrays, functions.

```
#include <stdio.h>

// Function to process transactions and summarize deposits and withdrawals
void generateBankStatement(const int transactionTypes[], const float amounts[], int size) {
```

```
    float totalDeposits = 0.0;
    float totalWithdrawals = 0.0;

    // Process each transaction
    for (int i = 0; i < size; i++) {
        switch (transactionTypes[i]) {
            case 1: // Deposit
                totalDeposits += amounts[i];
                break;
            case 2: // Withdrawal
                totalWithdrawals += amounts[i];
                break;
            default:
                printf("Invalid transaction type at index %d\n", i);
        }
    }

    // Output the summary
    printf("Total Deposits: $%.2f\n", totalDeposits);
    printf("Total Withdrawals: $%.2f\n", totalWithdrawals);
}

int main() {
    // Example transactions
    int transactionTypes[] = {1, 2, 1, 2, 1}; // 1 = Deposit, 2 = Withdrawal
    float amounts[] = {500.0, 200.0, 300.0, 150.0, 1000.0};

    int size = sizeof(transactionTypes) / sizeof(transactionTypes[0]);

    // Call the function to generate the bank statement
    generateBankStatement(transactionTypes, amounts, size);

    return 0;
}
```

8.Loan Eligibility Check
Input: Array of customer credit scores.
Process: Use if-else to check eligibility criteria. Use pointers to update eligibility status.
Output: Print customer eligibility statuses.
Concepts: Decision-making, arrays, pointers, functions.

```
#include <stdio.h>

void checkEligibility(int *creditScores, char *status, int numCustomers) {
    for (int i = 0; i < numCustomers; i++) {
        // Eligibility Criteria: Credit score >= 650
        if (*(creditScores + i) >= 650) {
            *(status + i) = 'Y';  // Eligible
        } else {
            *(status + i) = 'N';  // Not eligible
        }
    }
}

void printEligibility(char *status, int numCustomers) {
```

```
    printf("Customer Eligibility Status:\n");
    for (int i = 0; i < numCustomers; i++) {
        printf("Customer %d: %c\n", i + 1, *(status + i));
    }
}

int main() {
    int numCustomers;

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

    // Declare and input the credit scores
    int creditScores[numCustomers];
    for (int i = 0; i < numCustomers; i++) {
        printf("Enter credit score for customer %d: ", i + 1);
        scanf("%d", &creditScores[i]);
    }

    // Declare an array to store the eligibility status
    char eligibilityStatus[numCustomers];

    // Check eligibility
    checkEligibility(creditScores, eligibilityStatus, numCustomers);

    // Print eligibility results
    printEligibility(eligibilityStatus, numCustomers);

    return 0;
}
```

9.Order Total Calculator
Input: Array of item prices.
Process: Pass the array to a function. Use pointers to calculate the total cost.
Output: Display the total order value.
Concepts: Arrays, pointers, functions, loops.

```
#include <stdio.h>

// Function to calculate the total cost using pointers
float calculate_total(float *prices, int size) {
    float total = 0.0;

    // Loop through the array using pointer arithmetic to calculate the sum
    for (int i = 0; i < size; i++) {
        total += *(prices + i);  // Pointer arithmetic to access array elements
    }

    return total;
}

int main() {
    int n;
```

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

    float prices[n];

    // Input prices of each item
    printf("Enter the prices of %d items:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Price of item %d: ", i + 1);
        scanf("%f", &prices[i]);
    }

    // Call the function to calculate the total
    float total = calculate_total(prices, n);

    // Display the total order value
    printf("The total order value is: $%.2f\n", total);

    return 0;
}
```

10.Stock Replenishment Alert
Input: Array of inventory levels.
Process: Use a function to flag products below a threshold. Return a pointer to flagged indices.
Output: Display flagged product indices.
Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>

// Function to flag products with inventory levels below the threshold
// It returns a pointer to the flagged indices
int* flag_low_inventory(int inventory[], int size, int threshold, int* flaggedCount) {
    static int flaggedIndices[100]; // Static array to hold flagged indices
    *flaggedCount = 0; // Initialize the count of flagged products

    for (int i = 0; i < size; i++) {
        if (inventory[i] < threshold) {
            flaggedIndices[*flaggedCount] = i; // Store the index
            (*flaggedCount)++; // Increment the flagged count
        }
    }

    return flaggedIndices; // Return the pointer to the flagged indices array
}

int main() {
    int inventory[] = {150, 30, 200, 75, 20, 10, 90}; // Inventory levels of products
    int size = sizeof(inventory) / sizeof(inventory[0]); // Array size
    int threshold = 50; // Threshold for flagging low inventory
    int flaggedCount = 0; // Variable to hold the number of flagged products

    // Call function to get the flagged product indices
    int* flaggedIndices = flag_low_inventory(inventory, size, threshold, &flaggedCount);
```

```
    // Display the flagged product indices
    printf("Flagged product indices (below threshold %d):\n", threshold);
    for (int i = 0; i < flaggedCount; i++) {
        printf("Product at index %d\n", flaggedIndices[i]);
    }

    return 0;
}
```

11.Customer Reward Points
Input: Array of customer purchase amounts.
Process: Pass the purchase array by reference to a function that calculates reward points using if-else.
Output: Display reward points for each customer.
Concepts: Arrays, functions, pass by reference, decision-making.

```
#include <stdio.h>

void calculateRewardPoints(int purchases[], int numCustomers) {
    for(int i = 0; i < numCustomers; i++) {
        int points = 0;

        // Reward points calculation based on purchase amount
        if(purchases[i] > 100) {
            points = purchases[i] * 0.1;  // 10% reward points for purchases above 100
        }
        else if(purchases[i] > 50) {
            points = purchases[i] * 0.05;  // 5% reward points for purchases between 51 and 100
        }
        else {
            points = 0;  // No reward points for purchases 50 or below
        }

        // Output reward points for the current customer
        printf("Customer %d spent $%d, earned %d reward points.\n", i + 1, purchases[i], points);
    }
}

int main() {
    // Example customer purchase amounts
    int purchases[] = {120, 80, 45, 200, 55};
    int numCustomers = sizeof(purchases) / sizeof(purchases[0]);

    // Call the function to calculate and display reward points
    calculateRewardPoints(purchases, numCustomers);

    return 0;
}
```

12.Shipping Cost Estimator
Input: Array of order weights and shipping zones.
Process: Use a switch statement to calculate shipping costs based on zones. Pass the weight array as a constant parameter.
Output: Print the shipping cost for each order.
Concepts: Decision-making, passing constant data, arrays, functions.

```c
#include <stdio.h>

// Function to calculate shipping cost based on weight and shipping zone
void calculateShippingCost(const int weights[], int numOrders, int zone) {
    int shippingCost;

    for (int i = 0; i < numOrders; i++) {
        int weight = weights[i];

        // Use switch to calculate shipping cost based on zone
        switch (zone) {
            case 1:
                // Zone 1: $5 for up to 10kg, $10 for 10-20kg, $15 for 20kg+
                if (weight <= 10) {
                    shippingCost = 5;
                } else if (weight <= 20) {
                    shippingCost = 10;
                } else {
                    shippingCost = 15;
                }
                break;
            case 2:
                // Zone 2: $8 for up to 10kg, $15 for 10-20kg, $20 for 20kg+
                if (weight <= 10) {
                    shippingCost = 8;
                } else if (weight <= 20) {
                    shippingCost = 15;
                } else {
                    shippingCost = 20;
                }
                break;
            case 3:
                // Zone 3: $10 for up to 10kg, $20 for 10-20kg, $30 for 20kg+
                if (weight <= 10) {
                    shippingCost = 10;
                } else if (weight <= 20) {
                    shippingCost = 20;
                } else {
                    shippingCost = 30;
                }
                break;
            default:
                printf("Invalid shipping zone.\n");
                return;
        }

        // Output the shipping cost for the current order
        printf("Order %d (Weight: %dkg) Shipping Cost: $%d\n", i + 1, weight, shippingCost);
    }
}

int main() {
    // Define an array of weights for the orders
    int weights[] = {5, 12, 25, 8, 18};
    int numOrders = sizeof(weights) / sizeof(weights[0]);
```

```
    // Define the shipping zone
    int shippingZone = 2;  // Change this value for different zones (1, 2, 3)

    // Calculate and print the shipping costs
    calculateShippingCost(weights, numOrders, shippingZone);

    return 0;
}
```

13.Missile Trajectory Analysis
Input: Array of trajectory data points.
Process: Use functions to find maximum and minimum altitudes. Use pointers to access data.
Output: Display maximum and minimum altitudes.
Concepts: Arrays, pointers, functions.

```
#include <stdio.h>

// Function to find the maximum altitude
int find_max_altitude(int *data, int size) {
    int max = *data;  // Initialize with the first element
    for (int i = 1; i < size; i++) {
        if (*(data + i) > max) {  // Using pointer arithmetic
            max = *(data + i);
        }
    }
    return max;
}

// Function to find the minimum altitude
int find_min_altitude(int *data, int size) {
    int min = *data;  // Initialize with the first element
    for (int i = 1; i < size; i++) {
        if (*(data + i) < min) {  // Using pointer arithmetic
            min = *(data + i);
        }
    }
    return min;
}

int main() {
    // Example trajectory data (altitudes)
    int trajectory[] = {100, 250, 150, 300, 50, 200, 400};
    int size = sizeof(trajectory) / sizeof(trajectory[0]);  // Size of the array

    // Using functions to find maximum and minimum altitudes
    int max_altitude = find_max_altitude(trajectory, size);
    int min_altitude = find_min_altitude(trajectory, size);

    // Display the results
    printf("Maximum Altitude: %d\n", max_altitude);
    printf("Minimum Altitude: %d\n", min_altitude);

    return 0;
}
```

14.Target Identification System
Input: Array of radar signal intensities.
Process: Classify signals into categories using a switch statement. Return a pointer to the array of classifications.
Output: Display classified signal types.
Concepts: Decision-making, functions returning pointers, arrays.

```
#include <stdio.h>

#define NUM_SIGNALS 5 // Number of signals to classify

// Signal category constants
#define WEAK 0
#define MODERATE 1
#define STRONG 2
#define UNKNOWN 3

// Function to classify radar signal intensities
int* classify_signals(int signals[], int num_signals) {
    // Array to store classifications
    static int classifications[NUM_SIGNALS];

    // Classifying each signal intensity
    for (int i = 0; i < num_signals; i++) {
        if (signals[i] >= 0 && signals[i] <= 50) {
            classifications[i] = WEAK;
        } else if (signals[i] >= 51 && signals[i] <= 100) {
            classifications[i] = MODERATE;
        } else if (signals[i] >= 101 && signals[i] <= 200) {
            classifications[i] = STRONG;
        } else {
            classifications[i] = UNKNOWN;
        }
    }

    return classifications;
}

// Function to display the classification
void display_classifications(int* classifications, int num_signals) {
    printf("Radar Signal Classifications:\n");
    for (int i = 0; i < num_signals; i++) {
        switch (classifications[i]) {
            case WEAK:
                printf("Signal %d: WEAK\n", i+1);
                break;
            case MODERATE:
                printf("Signal %d: MODERATE\n", i+1);
                break;
            case STRONG:
                printf("Signal %d: STRONG\n", i+1);
                break;
            case UNKNOWN:
                printf("Signal %d: UNKNOWN\n", i+1);
```

```
            break;
        }
    }
}

int main() {
    // Array of radar signal intensities
    int signals[NUM_SIGNALS] = {30, 75, 150, 220, 60};

    // Get classified signal categories
    int* classifications = classify_signals(signals, NUM_SIGNALS);

    // Display the classifications
    display_classifications(classifications, NUM_SIGNALS);

    return 0;
}
```

15.Threat Level Assessment
Input: Array of sensor readings.
Process: Pass the array by reference to a function that uses if-else to categorize threats.
Output: Display categorized threat levels.
Concepts: Arrays, functions, pass by reference, decision-making.

```
#include <stdio.h>

// Function to categorize threat levels based on sensor readings
void categorizeThreats(int readings[], int size) {
    // Loop through each sensor reading
    for (int i = 0; i < size; i++) {
        if (readings[i] < 20) {
            printf("Sensor %d: Low Threat\n", i + 1);
        } else if (readings[i] >= 20 && readings[i] <= 50) {
            printf("Sensor %d: Moderate Threat\n", i + 1);
        } else if (readings[i] > 50) {
            printf("Sensor %d: High Threat\n", i + 1);
        }
    }
}

int main() {
    // Example array of sensor readings
    int sensorReadings[] = {10, 30, 60, 40, 55};
    int size = sizeof(sensorReadings) / sizeof(sensorReadings[0]);

    // Pass the array by reference to the function
    categorizeThreats(sensorReadings, size);

    return 0;
}
```

16.Signal Calibration
Input: Array of raw signal data.
Process: Use a function to adjust signal values by reference. Use pointers for data traversal.
Output: Print calibrated signal values.

Concepts: Arrays, pointers, functions, loops.

```
#include <stdio.h>

// Function to calibrate the signal values
void calibrate_signal(int *signal, int length, int reference) {
    for (int i = 0; i < length; i++) {
        // Adjust each signal value by subtracting the reference value
        signal[i] -= reference;
    }
}

// Function to print the calibrated signal values
void print_signal(int *signal, int length) {
    printf("Calibrated Signal Values:\n");
    for (int i = 0; i < length; i++) {
        printf("%d ", signal[i]);
    }
    printf("\n");
}

int main() {
    // Example raw signal data (array)
    int raw_signal[] = {100, 150, 200, 250, 300};
    int length = sizeof(raw_signal) / sizeof(raw_signal[0]); // Calculate the array length

    // Reference value for calibration (you can adjust this based on your need)
    int reference_value = 50;

    // Call the function to calibrate the signal values
    calibrate_signal(raw_signal, length, reference_value);

    // Call the function to print the calibrated signal values
    print_signal(raw_signal, length);

    return 0;
}
```

17.Matrix Row Sum
Input: 2D array representing a matrix.
Process: Write a function that calculates the sum of each row. The function returns a pointer to an array of row sums.
Output: Display the row sums.
Concepts: Arrays, functions returning pointers, loops.

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

// Function to calculate the sum of each row of the matrix
int* rowSum(int rows, int cols, int matrix[rows][cols]) {
    // Allocate memory for row sums
    int *row_sums = (int *)malloc(rows * sizeof(int));

    if (row_sums == NULL) {
        printf("Memory allocation failed.\n");
```

```
        return NULL;
    }

    // Calculate the sum of each row
    for (int i = 0; i < rows; i++) {
        row_sums[i] = 0;  // Initialize the sum for the current row
        for (int j = 0; j < cols; j++) {
            row_sums[i] += matrix[i][j];  // Add the element to the row sum
        }
    }

    return row_sums;  // Return the pointer to the row sums array
}

int main() {
    int rows = 3, cols = 4;

    // Example matrix: 3x4 matrix
    int matrix[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // Get the row sums
    int *row_sums = rowSum(rows, cols, matrix);

    if (row_sums != NULL) {
        // Print the row sums
        printf("Row sums: \n");
        for (int i = 0; i < rows; i++) {
            printf("Row %d sum: %d\n", i + 1, row_sums[i]);
        }

        // Free the allocated memory for row sums
        free(row_sums);
    }

    return 0;
}
```

18.Statistical Mean Calculator
Input: Array of data points.
Process: Pass the data array as a constant parameter. Use pointers to calculate the mean.
Output: Print the mean value.
Concepts: Passing constant data, pointers, functions.

```
#include <stdio.h>

float calculate_mean(const int* data, int size) {
    int sum = 0;

    // Loop through the array using a pointer and accumulate the sum.
    for (int i = 0; i < size; i++) {
        sum += *(data + i);  // Using pointer arithmetic to access elements
```

```
    }

    // Calculate and return the mean.
    return (float)sum / size;
}

int main() {
    // Define an array of data points
    int data[] = {10, 20, 30, 40, 50};
    int size = sizeof(data) / sizeof(data[0]);  // Calculate the size of the array

    // Calculate the mean using the calculate_mean function
    float mean = calculate_mean(data, size);

    // Print the mean value
    printf("The mean is: %.2f\n", mean);

    return 0;
}
```

19.Temperature Gradient Analysis
Input: Array of temperature readings.
Process: Compute the gradient using a function that returns a pointer to the array of gradients.
Output: Display temperature gradients.
Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>

// Function that computes the gradient and returns a pointer to the gradients array
float* computeTemperatureGradient(float temperatures[], int size) {
    static float gradients[100]; // Static array to hold gradients (size can be changed as needed)

    for (int i = 0; i < size - 1; i++) {
        gradients[i] = temperatures[i + 1] - temperatures[i];  // Calculate gradient (difference)
    }

    return gradients;  // Return pointer to gradients array
}

int main() {
    // Sample array of temperature readings
    float temperatures[] = {21.5, 22.8, 23.1, 21.9, 20.7};
    int size = sizeof(temperatures) / sizeof(temperatures[0]);

    // Call function to compute temperature gradients
    float* gradients = computeTemperatureGradient(temperatures, size);

    // Display temperature gradients
    printf("Temperature Gradients:\n");
    for (int i = 0; i < size - 1; i++) {
        printf("Gradient between T[%d] and T[%d]: %.2f\n", i, i + 1, gradients[i]);
    }

    return 0;
}
```

20.Data Normalization
Input: Array of data points.
Process: Pass the array by reference to a function that normalizes values to a range of 0–1 using pointers.
Output: Display normalized values.
Concepts: Arrays, pointers, pass by reference, functions.\

```
#include <stdio.h>

// Function to normalize the array values to range [0, 1]
void normalizeArray(float *arr, int size) {
    float min = arr[0];
    float max = arr[0];

    // Find the minimum and maximum values in the array
    for (int i = 1; i < size; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    // Normalize the array elements
    for (int i = 0; i < size; i++) {
        arr[i] = (arr[i] - min) / (max - min);
    }
}

// Function to print the array
void printArray(float *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%f ", arr[i]);
    }
    printf("\n");
}

int main() {
    // Array of data points
    float data[] = {23.4, 12.1, 44.7, 9.3, 38.6};
    int size = sizeof(data) / sizeof(data[0]);

    printf("Original Array:\n");
    printArray(data, size);

    // Normalize the array
    normalizeArray(data, size);

    printf("Normalized Array:\n");
    printArray(data, size);

    return 0;
}
```

21.Exam Score Analysis
Input: Array of student scores.
Process: Write a function that returns a pointer to the highest score. Use loops to calculate the average score.
Output: Display the highest and average scores.
Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>

// Function to find the highest score
int* highest_score(int scores[], int size) {
    int* max_score = &scores[0];  // Initialize the pointer to the first element

    // Loop through the array to find the highest score
    for (int i = 1; i < size; i++) {
        if (scores[i] > *max_score) {
            max_score = &scores[i];  // Update the pointer to the new highest score
        }
    }

    return max_score;
}

// Function to calculate the average score
float calculate_average(int scores[], int size) {
    int sum = 0;

    // Loop through the array to calculate the sum of scores
    for (int i = 0; i < size; i++) {
        sum += scores[i];
    }

    // Calculate and return the average
    return (float)sum / size;
}

int main() {
    int scores[] = {85, 90, 78, 92, 88, 76};  // Example array of student scores
    int size = sizeof(scores) / sizeof(scores[0]);  // Calculate the size of the array

    // Get the highest score by calling the highest_score function
    int* max_score = highest_score(scores, size);

    // Calculate the average score
    float average = calculate_average(scores, size);

    // Display the highest and average scores
    printf("Highest Score: %d\n", *max_score);
    printf("Average Score: %.2f\n", average);

    return 0;
}
```

22.Grade Assignment

Input: Array of student marks.
Process: Pass the marks array by reference to a function. Use a switch statement to assign grades.
Output: Display grades for each student.
Concepts: Arrays, decision-making, pass by reference, functions.

```
#include <stdio.h>

// Function prototype
void gradeAssignments(int marks[], int n);

int main() {
    int n;

    // Get the number of students
    printf("Enter the number of students: ");
    scanf("%d", &n);

    int marks[n];

    // Input marks for each student
    printf("Enter the marks of %d students:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Student %d: ", i + 1);
        scanf("%d", &marks[i]);
    }

    // Pass the marks array by reference to the gradeAssignments function
    gradeAssignments(marks, n);

    return 0;
}

void gradeAssignments(int marks[], int n) {
    // Loop through each student's marks and assign grades
    for (int i = 0; i < n; i++) {
        printf("Student %d: ", i + 1);

        // Switch statement for grading
        switch (marks[i] / 10) {
            case 10:
            case 9:
                printf("Grade A\n");
                break;
            case 8:
                printf("Grade B\n");
                break;
            case 7:
                printf("Grade C\n");
                break;
            case 6:
                printf("Grade D\n");
                break;
            default:
                printf("Grade F\n");
        }
```

```
    }
}
```

23.Student Attendance Tracker
Input: Array of attendance percentages.
Process: Use pointers to traverse the array. Return a pointer to an array of defaulters.
Output: Display defaulters' indices.
Concepts: Arrays, pointers, functions returning pointers.

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

#define THRESHOLD 75

// Function to find the defaulters (students with attendance below 75%)
int* findDefaulters(float* attendance, int totalStudents, int* defaulterCount) {
    // Dynamically allocate memory for a maximum of totalStudents (since all could be defaulters)
    int* defaulters = (int*)malloc(totalStudents * sizeof(int));
    *defaulterCount = 0;  // Initialize defaulter count

    // Traverse the array of attendance percentages using pointers
    for (int i = 0; i < totalStudents; i++) {
        if (*(attendance + i) < THRESHOLD) {  // Check if attendance is below threshold
            defaulters[*defaulterCount] = i;   // Store the index of the defaulter
            (*defaulterCount)++;  // Increment the defaulter count
        }
    }

    // Return pointer to the defaulters' indices array
    return defaulters;
}

int main() {
    int totalStudents = 10;
    float attendance[] = {80.5, 70.0, 90.0, 60.0, 65.5, 85.0, 77.5, 72.0, 88.5, 69.5};

    int defaulterCount = 0;
    // Call the function to find defaulters
    int* defaulters = findDefaulters(attendance, totalStudents, &defaulterCount);

    // Output the defaulters' indices
    if (defaulterCount > 0) {
        printf("Defaulters' indices (attendance below %d%%):\n", THRESHOLD);
        for (int i = 0; i < defaulterCount; i++) {
            printf("%d ", defaulters[i]);
        }
    } else {
        printf("No defaulters found (all students have sufficient attendance).\n");
    }

    // Free the dynamically allocated memory
    free(defaulters);

    return 0;
}
```

24.Quiz Performance Analyzer
Input: Array of quiz scores.
Process: Pass the array as a constant parameter to a function that uses if-else for performance categorization.
Output: Print categorized performance.
Concepts: Arrays, passing constant data, functions, decision-making.

```
#include <stdio.h>

// Function to categorize performance based on score
void categorize_performance(const int scores[], int size) {
    for (int i = 0; i < size; i++) {
        if (scores[i] >= 90) {
            printf("Score: %d - Excellent\n", scores[i]);
        } else if (scores[i] >= 75) {
            printf("Score: %d - Good\n", scores[i]);
        } else if (scores[i] >= 50) {
            printf("Score: %d - Average\n", scores[i]);
        } else {
            printf("Score: %d - Needs Improvement\n", scores[i]);
        }
    }
}

int main() {
    // Example array of quiz scores
    const int quiz_scores[] = {95, 82, 74, 60, 48, 91, 85};
    int size = sizeof(quiz_scores) / sizeof(quiz_scores[0]); // Calculate the size of the array

    // Call the function to categorize performance
    categorize_performance(quiz_scores, size);

    return 0;
}
```