Weekend 2 task(4th &5thjan)

---------------------------

## 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>
#define SIZE 5
#define THRESHOLD 10

void updateInventory(int *inventory, int *changes, int size) {
    for (int i = 0; i < size; i++) {
        inventory[i] += changes[i];
        if (inventory[i] < THRESHOLD) {
            printf("Item %d below threshold! Inventory: %d\n", i, inventory[i]);
        }
    }
}

int main() {
    int inventory[SIZE] = {20, 15, 8, 25, 10};
    int changes[SIZE] = {-5, -3, -10, -2, -8};

    updateInventory(inventory, changes, SIZE);

    printf("Updated inventory levels:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Item %d: %d\n", i, inventory[i]);
    }
    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>
#define SIZE 5

float *adjustPrices(const int *demand, float *prices, int size) {
    static float adjustedPrices[SIZE];
    for (int i = 0; i < size; i++) {
        adjustedPrices[i] = prices[i] * (1 + (demand[i] / 100.0));
    }
    return adjustedPrices;
}

int main() {
    const int demand[SIZE] = {10, 20, -5, 15, -10};
    float prices[SIZE] = {100.0, 200.0, 150.0, 120.0, 180.0};

    float *newPrices = adjustPrices(demand, prices, SIZE);
    printf("Original and Adjusted Prices:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Product %d: Original: %.2f, Adjusted: %.2f\n", i, prices[i], newPrices[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>
#define SIZE 5

void calculateTotalSales(float *sales, int size, float *total) {
    *total = 0;
    for (int i = 0; i < size; i++) {
        *total += sales[i];
    }
}

int main() {
    float sales[SIZE];
    float total = 0;
    int i = 0;
    do {
        printf("Enter sales for item %d (must be >= 0): ", i + 1);
        scanf("%f", &sales[i]);
    } while (++i < SIZE);

    calculateTotalSales(sales, SIZE, &total);

    printf("Total sales for the day: %.2f\n", total);
    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>
#define SIZE 5

void assignDiscount(int *sales, float *discounts, int size) {
    for (int i = 0; i < size; i++) {
        switch (sales[i] / 10) {
            case 0: case 1: discounts[i] = 0.05; break; // 0-19 sales: 5% discount
            case 2: case 3: discounts[i] = 0.10; break; // 20-39 sales: 10% discount
            default: discounts[i] = 0.15;               // 40+ sales: 15% discount
        }
    }
}

int main() {
    int sales[SIZE] = {15, 25, 5, 45, 30};
    float discounts[SIZE];

    assignDiscount(sales, discounts, SIZE);

    printf("Discount rates:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Product %d: %.2f%%\n", i, discounts[i] * 100);
    }
    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 SIZE 5
#define THRESHOLD 1000

void classifyTransactions(float *transactions, int size) {
    for (int i = 0; i < size; i++) {
        if (transactions[i] > THRESHOLD) {
            printf("Transaction %d: Suspicious (%.2f)\n", i, transactions[i]);
        } else {
            printf("Transaction %d: Normal (%.2f)\n", i, transactions[i]);
        }
    }
}

int main() {
    float transactions[SIZE] = {500, 1200, 300, 1500, 800};

    classifyTransactions(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>
#define SIZE 5
#define INTEREST_RATE 0.05

float *calculateInterest(float *balances, int size) {
    static float updatedBalances[SIZE];
    for (int i = 0; i < size; i++) {
        updatedBalances[i] = balances[i] + (balances[i] * INTEREST_RATE);
    }
    return updatedBalances;
}

int main() {
    float balances[SIZE] = {1000, 2000, 1500, 500, 800};

    float *newBalances = calculateInterest(balances, SIZE);

    printf("Updated account balances:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Account %d: %.2f\n", i + 1, newBalances[i]);
    }

    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>
#define SIZE 5

void summarizeTransactions(const int *types, const float *amounts, int size) {
    float totalDeposits = 0, totalWithdrawals = 0;
    for (int i = 0; i < size; i++) {
        switch (types[i]) {
            case 1: totalDeposits += amounts[i]; break;
            case 2: totalWithdrawals += amounts[i]; break;
        }
    }
    printf("Total Deposits: %.2f\n", totalDeposits);
    printf("Total Withdrawals: %.2f\n", totalWithdrawals);
}

int main() {
    const int transactionTypes[SIZE] = {1, 2, 1, 1, 2}; // 1: Deposit, 2: Withdrawal
    const float transactionAmounts[SIZE] = {500, 200, 300, 400, 100};

    summarizeTransactions(transactionTypes, transactionAmounts, 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>
#define SIZE 5
```

```
#define ELIGIBILITY_SCORE 600

void checkEligibility(const int *scores, char *statuses, int size) {
    for (int i = 0; i < size; i++) {
        statuses[i] = (scores[i] >= ELIGIBILITY_SCORE) ? 'Y' : 'N';
    }
}

int main() {
    const int creditScores[SIZE] = {650, 550, 700, 400, 750};
    char eligibilityStatuses[SIZE];

    checkEligibility(creditScores, eligibilityStatuses, SIZE);

    printf("Loan Eligibility Statuses:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Customer %d: %c\n", i + 1, eligibilityStatuses[i]);
    }

    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>
```

```
#define SIZE 5


float calculateTotal(const float *prices, int size) {
    float total = 0;
    for (int i = 0; i < size; i++) {
        total += prices[i];
    }
    return total;
}


int main() {
    const float itemPrices[SIZE] = {19.99, 45.50, 10.25, 15.00, 60.75};


    float total = calculateTotal(itemPrices, SIZE);


    printf("Total Order Value: %.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>
#define SIZE 5
#define THRESHOLD 10
```

```
int *flagLowStock(const int *inventory, int size) {
    static int flaggedIndices[SIZE];
    int count = 0;

    for (int i = 0; i < size; i++) {
        if (inventory[i] < THRESHOLD) {
            flaggedIndices[count++] = i;
        }
    }
    flaggedIndices[count] = -1; // Sentinel value
    return flaggedIndices;
}

int main() {
    const int inventoryLevels[SIZE] = {5, 20, 8, 12, 3};

    int *flagged = flagLowStock(inventoryLevels, SIZE);

    printf("Products below threshold:\n");
    for (int i = 0; flagged[i] != -1; i++) {
        printf("Product %d\n", flagged[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>
#define SIZE 5

void calculateRewards(const float *purchases, int *rewards, int size) {
    for (int i = 0; i < size; i++) {
        rewards[i] = (purchases[i] > 100) ? 10 : 5;
    }
}

int main() {
    const float purchaseAmounts[SIZE] = {150.0, 50.0, 200.0, 90.0, 120.0};
    int rewardPoints[SIZE];

    calculateRewards(purchaseAmounts, rewardPoints, SIZE);

    printf("Customer Reward Points:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Customer %d: %d points\n", i + 1, rewardPoints[i]);
    }
    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.

```
#include <stdio.h>
#define SIZE 5

void calculateShippingCosts(const float *weights, const int *zones, float *costs, int size) {
    for (int i = 0; i < size; i++) {
        switch (zones[i]) {
            case 1: costs[i] = weights[i] * 5; break;
            case 2: costs[i] = weights[i] * 7; break;
            case 3: costs[i] = weights[i] * 10; break;
            default: costs[i] = weights[i] * 12; break;
        }
    }
}

int main() {
    const float orderWeights[SIZE] = {10.5, 5.0, 8.2, 12.0, 7.5};
    const int shippingZones[SIZE] = {1, 2, 3, 1, 2};
    float shippingCosts[SIZE];

    calculateShippingCosts(orderWeights, shippingZones, shippingCosts, SIZE);

    printf("Shipping Costs:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Order %d: %.2f\n", i + 1, shippingCosts[i]);
    }
    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>
#define SIZE 5

void findMinMax(const float *trajectory, int size, float *min, float *max) {
    *min = *max = trajectory[0];
    for (int i = 1; i < size; i++) {
        if (trajectory[i] < *min) *min = trajectory[i];
        if (trajectory[i] > *max) *max = trajectory[i];
    }
}

int main() {
    const float trajectoryData[SIZE] = {120.5, 150.0, 90.2, 200.0, 170.5};
    float minAltitude, maxAltitude;

    findMinMax(trajectoryData, SIZE, &minAltitude, &maxAltitude);

    printf("Minimum Altitude: %.2f\n", minAltitude);
    printf("Maximum Altitude: %.2f\n", maxAltitude);
    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 SIZE 5

int *classifySignals(const float *intensities, int size) {
    static int classifications[SIZE];
    for (int i = 0; i < size; i++) {
        classifications[i] = (intensities[i] > 75.0) ? 2 : (intensities[i] > 50.0 ? 1 : 0);
    }
    return classifications;
}

int main() {
    const float signalIntensities[SIZE] = {60.5, 80.0, 45.0, 90.0, 55.5};

    int *classifiedSignals = classifySignals(signalIntensities, SIZE);

    printf("Signal Classifications:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Signal %d: Type %d\n", i + 1, classifiedSignals[i]);
    }

    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>
#define SIZE 5

void assessThreats(float *readings, char *categories, int size) {
    for (int i = 0; i < size; i++) {
        categories[i] = (readings[i] > 100.0) ? 'H' : (readings[i] > 50.0 ? 'M' : 'L');
    }
}

int main() {
    float sensorReadings[SIZE] = {120.0, 60.5, 40.0, 110.0, 75.0};
    char threatLevels[SIZE];

    assessThreats(sensorReadings, threatLevels, SIZE);

    printf("Threat Levels:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Sensor %d: %c\n", i + 1, threatLevels[i]);
    }

    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>
#define SIZE 5

void calibrateSignals(float *signals, int size) {
    for (int i = 0; i < size; i++) {
        signals[i] *= 1.1; // Example: Increase signal by 10%
    }
}

int main() {
    float rawSignals[SIZE] = {100.0, 150.0, 80.0, 200.0, 120.0};

    calibrateSignals(rawSignals, SIZE);

    printf("Calibrated Signals:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Signal %d: %.2f\n", i + 1, rawSignals[i]);
    }

    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>
#define ROWS 3
#define COLS 4

int *calculateRowSums(const int matrix[ROWS][COLS]) {
    static int rowSums[ROWS];
    for (int i = 0; i < ROWS; i++) {
        rowSums[i] = 0;
        for (int j = 0; j < COLS; j++) {
            rowSums[i] += matrix[i][j];
        }
    }
    return rowSums;
}
```

```
int main() {
    const int matrix[ROWS][COLS] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};

    int *rowSums = calculateRowSums(matrix);

    printf("Row Sums:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Row %d: %d\n", i + 1, rowSums[i]);
    }

    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>
#define SIZE 5

float calculateMean(const float *data, int size) {
    float sum = 0;
    for (int i = 0; i < size; i++) {
        sum += data[i];
    }
    return sum / size;
}


int main() {
    const float dataPoints[SIZE] = {10.0, 20.0, 30.0, 40.0, 50.0};


    float mean = calculateMean(dataPoints, SIZE);


    printf("Mean value: %.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>
#define SIZE 5


float *computeGradient(const float *temperatures, int size) {
    static float gradients[SIZE - 1];
    for (int i = 0; i < size - 1; i++) {
        gradients[i] = temperatures[i + 1] - temperatures[i];
    }
    return gradients;
}
```

```
int main() {
    const float temperatureReadings[SIZE] = {10.0, 15.0, 20.0, 25.0, 30.0};

    float *gradients = computeGradient(temperatureReadings, SIZE);

    printf("Temperature Gradients:\n");
    for (int i = 0; i < SIZE - 1; i++) {
        printf("Gradient %d: %.2f\n", 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>
#define SIZE 5

void normalizeData(float *data, int size) {
    float min = data[0], max = data[0];
    for (int i = 1; i < size; i++) {
        if (data[i] < min) min = data[i];
        if (data[i] > max) max = data[i];
    }
    for (int i = 0; i < size; i++) {
        data[i] = (data[i] - min) / (max - min);
    }
}

int main() {
    float data[SIZE] = {10.0, 20.0, 30.0, 40.0, 50.0};

    normalizeData(data, SIZE);

    printf("Normalized Data:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Data %d: %.2f\n", i + 1, data[i]);
    }

    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>
#define SIZE 5
```

```
float *findHighestScore(const float *scores, int size) {
    static float result[2]; // [highest, average]
    float total = 0;
    result[0] = scores[0];

    for (int i = 0; i < size; i++) {
        if (scores[i] > result[0]) result[0] = scores[i];
        total += scores[i];
    }
    result[1] = total / size;
    return result;
}

int main() {
    const float scores[SIZE] = {85.5, 90.0, 78.0, 88.5, 92.0};

    float *results = findHighestScore(scores, SIZE);

    printf("Highest Score: %.2f\n", results[0]);
    printf("Average Score: %.2f\n", results[1]);

    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>
#define SIZE 5

void assignGrades(const float *marks, char *grades, int size) {
    for (int i = 0; i < size; i++) {
        switch ((int)(marks[i] / 10)) {
            case 10: case 9: grades[i] = 'A'; break;
            case 8: grades[i] = 'B'; break;
            case 7: grades[i] = 'C'; break;
            case 6: grades[i] = 'D'; break;
            default: grades[i] = 'F'; break;
        }
    }
}

int main() {
    const float marks[SIZE] = {85.0, 92.0, 75.5, 60.0, 45.0};
    char grades[SIZE];

    assignGrades(marks, grades, SIZE);

    printf("Grades:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Student %d: %c\n", i + 1, grades[i]);
    }

    return 0;
}
```

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>
#define SIZE 5
#define THRESHOLD 75.0

int *findDefaulters(const float *attendance, int size) {
    static int defaulters[SIZE];
    int count = 0;

    for (int i = 0; i < size; i++) {
        if (attendance[i] < THRESHOLD) {
            defaulters[count++] = i;
        }
    }
    defaulters[count] = -1; // Sentinel value
    return defaulters;
}

int main() {
    const float attendance[SIZE] = {80.0, 70.0, 85.0, 65.0, 90.0};
    int *defaulters = findDefaulters(attendance, SIZE);


    printf("Defaulters:\n");
    for (int i = 0; defaulters[i] != -1; i++) {
        printf("Student %d\n", defaulters[i] + 1);
    }


    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>
#define SIZE 5


void analyzePerformance(const float *scores, char *categories, int size) {
    for (int i = 0; i < size; i++) {
        if (scores[i] >= 90) categories[i] = 'E'; // Excellent
        else if (scores[i] >= 75) categories[i] = 'G'; // Good
        else if (scores[i] >= 50) categories[i] = 'A'; // Average
        else categories[i] = 'P'; // Poor
    }
}
```

```
int main() {
    const float scores[SIZE] = {95.0, 82.5, 70.0, 48.5, 89.0};
    char performance[SIZE];

    analyzePerformance(scores, performance, SIZE);

    printf("Performance Categories:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Quiz %d: %c\n", i + 1, performance[i]);
    }

    return 0;
}
```