1. Find Maximum and Minimum in an Array Problem Statement: Write a program to find the maximum and minimum values in a single-dimensional array of integers. Use: A const variable for the array size. A static variable to keep track of the maximum difference between the maximum and minimum values. if statements within a for loop to determine the maximum and minimum values.

#include <stdio.h>

// Function to find maximum and minimum in an array

void findMaxMin(int arr[], int size) {

int max, min;

static int maxDifference = 0; // Static variable to track maximum difference

// Initialize max and min with the first element of the array

max = min = arr[0];

// Traverse the array to find max and min

for (int i = 1; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

if (arr[i] < min) {

min = arr[i];

}

}

// Calculate the maximum difference

maxDifference = max - min;

// Output the results

printf("Maximum value: %d\n", max);

printf("Minimum value: %d\n", min);

printf("Maximum difference (max - min): %d\n", maxDifference);

}

int main() {

const int ARRAY\_SIZE = 5; // Constant array size

int arr[ARRAY\_SIZE] = {12, 7, 5, 20, 3}; // Example array

findMaxMin(arr, ARRAY\_SIZE);

return 0;

}

2. Array Element Categorization

Problem Statement: Categorize elements of a single-dimensional array into positive, negative, and zero values. Use:

A const variable to define the size of the array.

A for loop for traversal.

if-else statements to classify each element into separate arrays using static storage.

#include <stdio.h>

// Function to categorize array elements

void categorizeElements(int arr[], int size) {

static int positives[100], negatives[100], zeros[100]; // Static arrays for storage

int posCount = 0, negCount = 0, zeroCount = 0;

// Traverse the array and classify elements

for (int i = 0; i < size; i++) {

if (arr[i] > 0) {

positives[posCount++] = arr[i];

} else if (arr[i] < 0) {

negatives[negCount++] = arr[i];

} else {

zeros[zeroCount++] = arr[i];

}

}

// Output the results

printf("Positive numbers: ");

for (int i = 0; i < posCount; i++) {

printf("%d ", positives[i]);

}

printf("\n");

printf("Negative numbers: ");

for (int i = 0; i < negCount; i++) {

printf("%d ", negatives[i]);

}

printf("\n");

printf("Zeros: ");

for (int i = 0; i < zeroCount; i++) {

printf("%d ", zeros[i]);

}

printf("\n");

}

int main() {

const int ARRAY\_SIZE = 10; // Constant array size

int arr[ARRAY\_SIZE] = {12, -5, 0, 23, -7, 0, 15, -2, 0, 4}; // Example array

categorizeElements(arr, ARRAY\_SIZE);

return 0;

}

1. Cumulative Sum of Array Elements Problem Statement: Calculate the cumulative sum of elements in a single-dimensional array. Use: A static variable to hold the running total. A for loop to iterate through the array and update the cumulative sum. A const variable to set the array size.

#include <stdio.h>

// Function to calculate the cumulative sum of array elements

void calculateCumulativeSum(int arr[], int size) {

static int runningTotal = 0; // Static variable to hold the running total

printf("Cumulative Sum: ");

for (int i = 0; i < size; i++) {

runningTotal += arr[i]; // Update the running total

printf("%d ", runningTotal); // Print the current cumulative sum

}

printf("\n");

}

int main() {

const int ARRAY\_SIZE = 6; // Constant array size

int arr[ARRAY\_SIZE] = {5, 3, 8, -2, 7, 4}; // Example array

calculateCumulativeSum(arr, ARRAY\_SIZE);

return 0;

}

4. Check Prime Numbers in an Array

Problem Statement: Identify which elements in a single-dimensional array are prime numbers. Use:

A for loop to iterate through the array and check each element.

A nested for loop to determine if a number is prime.

if statements for decision-making.

A const variable to define the size of the array.

#include <stdio.h>

#include <stdbool.h>

// Function to check if a number is prime

bool isPrime(int num) {

if (num <= 1) return false; // Numbers <= 1 are not prime

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

return false; // Found a divisor, not prime

}

}

return true; // No divisors found, number is prime

}

// Function to identify prime numbers in an array

void checkPrimes(int arr[], int size) {

printf("Prime numbers in the array: ");

for (int i = 0; i < size; i++) {

if (isPrime(arr[i])) {

printf("%d ", arr[i]); // Print the prime number

}

}

printf("\n");

}

int main() {

const int ARRAY\_SIZE = 10; // Constant array size

int arr[ARRAY\_SIZE] = {2, 3, 4, 5, 10, 11, 13, 14, 15, 17}; // Example array

checkPrimes(arr, ARRAY\_SIZE);

return 0;

}

5. Array Rotation by N Positions

Problem Statement: Rotate the elements of a single-dimensional array to the left by N positions. Use:

A const variable for the rotation count.

A static array to store the rotated values.

A while loop for performing the rotation.

#include <stdio.h>

void rotateArrayLeft(int arr[], int size, int n) {

static int rotated[100]; // Static array to store the rotated values

int i, newIndex;

// Adjust rotation count if greater than array size

n = n % size;

// Perform rotation using a while loop

i = 0;

while (i < size) {

newIndex = (i - n + size) % size; // Calculate new index (cyclic rotation)

rotated[newIndex] = arr[i];

i++;

}

// Print the rotated array

printf("Rotated Array: ");

for (i = 0; i < size; i++) {

printf("%d ", rotated[i]);

}

printf("\n");

}

int main() {

const int ARRAY\_SIZE = 8; // Constant array size

const int ROTATION\_COUNT = 3; // Constant rotation count

int arr[ARRAY\_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8}; // Example array

rotateArrayLeft(arr, ARRAY\_SIZE, ROTATION\_COUNT);

return 0;

}

6. Count Frequency of Each Element

Problem Statement: Count the frequency of each unique element in a single-dimensional array. Use:

A const variable for the size of the array.

A nested for loop to compare each element with the rest.

A static array to store the frequency count.

#include <stdio.h>

#include <stdbool.h>

void countFrequency(int arr[], int size) {

static int frequency[100]; // Static array to store frequency count

bool visited[100] = {false}; // Array to track visited elements

printf("Element | Frequency\n");

printf("-------------------\n");

for (int i = 0; i < size; i++) {

if (visited[i]) continue; // Skip if element is already processed

int count = 1; // Start counting from 1

for (int j = i + 1; j < size; j++) {

if (arr[i] == arr[j]) {

count++;

visited[j] = true; // Mark duplicate element as visited

}

}

frequency[i] = count; // Store frequency of the current element

// Print element and its frequency

printf(" %d | %d\n", arr[i], frequency[i]);

}

}

int main() {

const int ARRAY\_SIZE = 10; // Constant array size

int arr[ARRAY\_SIZE] = {3, 5, 3, 7, 9, 5, 7, 3, 5, 7}; // Example array

countFrequency(arr, ARRAY\_SIZE);

return 0;

}

7. Sort Array in Descending Order

Problem Statement: Sort a single-dimensional array in descending order using bubble sort. Use:

A const variable for the size of the array.

A nested for loop for sorting.

if statements for comparing and swapping elements.

#include <stdio.h>

void sortDescending(int arr[], int size) {

// Bubble sort to arrange elements in descending order

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] < arr[j + 1]) { // Compare and swap if necessary

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

const int ARRAY\_SIZE = 6; // Constant array size

int arr[ARRAY\_SIZE] = {12, 45, 7, 23, 5, 89}; // Example array

printf("Original Array: ");

for (int i = 0; i < ARRAY\_SIZE; i++) {

printf("%d ", arr[i]);

}

printf("\n");

sortDescending(arr, ARRAY\_SIZE);

printf("Sorted Array in Descending Order: ");

for (int i = 0; i < ARRAY\_SIZE; i++) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

8. Find the Second Largest Element

Problem Statement: Find the second largest element in a single-dimensional array. Use:

A const variable for the array size.

A static variable to store the second largest element.

if statements and a single for loop to compare elements.

#include <stdio.h>

void findSecondLargest(int arr[], int size) {

if (size < 2) { // Check for sufficient elements

printf("Array must have at least two elements.\n");

return;

}

static int secondLargest; // Static variable to store the second largest element

int largest = arr[0]; // Initialize largest element

secondLargest = -2147483648; // Minimum integer value (for edge cases)

for (int i = 1; i < size; i++) {

if (arr[i] > largest) {

// Update both largest and second largest

secondLargest = largest;

largest = arr[i];

} else if (arr[i] > secondLargest && arr[i] != largest) {

// Update only second largest

secondLargest = arr[i];

}

}

if (secondLargest == -2147483648) {

printf("No distinct second largest element found.\n");

} else {

printf("Second Largest Element: %d\n", secondLargest);

}

}

int main() {

const int ARRAY\_SIZE = 7; // Constant array size

int arr[ARRAY\_SIZE] = {12, 35, 1, 10, 34, 35, 10}; // Example array

findSecondLargest(arr, ARRAY\_SIZE);

return 0;

}

9. Odd and Even Number Separation

Problem Statement: Separate the odd and even numbers from a single-dimensional array into two separate arrays. Use:

A const variable for the size of the array.

if-else statements to classify elements.

A for loop for traversal and separation.

#include <stdio.h>

void separateOddEven(int arr[], int size) {

int odd[size], even[size]; // Arrays to store odd and even numbers

int oddIndex = 0, evenIndex = 0; // Indices for odd and even arrays

for (int i = 0; i < size; i++) {

if (arr[i] % 2 == 0) {

even[evenIndex++] = arr[i]; // Add to even array

} else {

odd[oddIndex++] = arr[i]; // Add to odd array

}

}

// Print even numbers

printf("Even Numbers: ");

for (int i = 0; i < evenIndex; i++) {

printf("%d ", even[i]);

}

printf("\n");

// Print odd numbers

printf("Odd Numbers: ");

for (int i = 0; i < oddIndex; i++) {

printf("%d ", odd[i]);

}

printf("\n");

}

int main() {

const int ARRAY\_SIZE = 10; // Constant array size

int arr[ARRAY\_SIZE] = {12, 35, 7, 10, 23, 5, 46, 8, 3, 19}; // Example array

separateOddEven(arr, ARRAY\_SIZE);

return 0;

}

10. Cyclically Shift Array Elements

Problem Statement: Shift all elements of a single-dimensional array cyclically to the right by one position. Use:

A const variable for the array size.

A static variable to temporarily store the last element during shifting.

A for loop for the shifting operation.

has context menu

#include <stdio.h>

void cyclicShiftRight(int arr[], int size) {

static int lastElement; // Static variable to temporarily store the last element

lastElement = arr[size - 1]; // Store the last element

// Shift all elements to the right by one position

for (int i = size - 2; i >= 0; i--) {

arr[i + 1] = arr[i];

}

// Place the stored last element in the first position

arr[0] = lastElement;

}

int main() {

const int ARRAY\_SIZE = 6; // Constant array size

int arr[ARRAY\_SIZE] = {1, 2, 3, 4, 5, 6}; // Example array

printf("Original Array: ");

for (int i = 0; i < ARRAY\_SIZE; i++) {

printf("%d ", arr[i]);

}

printf("\n");

cyclicShiftRight(arr, ARRAY\_SIZE);

printf("Array After Cyclic Right Shift: ");

for (int i = 0; i < ARRAY\_SIZE; i++) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**1. Engine Temperature Monitoring System**

Write a program to monitor engine temperatures at 10 different time intervals in degrees Celsius. Use:

* Proper variable declarations with const to ensure fixed limits like maximum temperature.
* Storage classes (static for counters and extern for shared variables).
* Decision-making statements to alert if the temperature exceeds a safe threshold.
* A loop to take 10 temperature readings into a single-dimensional array and check each value.

#include <stdio.h>

#define MAX\_TEMPERATURE 100 // Constant for maximum safe temperature

// Static variable to count the total number of alerts

static int alertCount = 0;

void monitorTemperatures() {

const int NUM\_READINGS = 10; // Constant for the number of temperature readings

int temperatures[NUM\_READINGS]; // Array to store temperature readings

// Loop to take 10 temperature readings

printf("Enter 10 engine temperature readings in degrees Celsius:\n");

for (int i = 0; i < NUM\_READINGS; i++) {

printf("Reading %d: ", i + 1);

scanf("%d", &temperatures[i]);

}

// Check each reading against the maximum safe temperature

printf("\nTemperature Monitoring Report:\n");

for (int i = 0; i < NUM\_READINGS; i++) {

if (temperatures[i] > MAX\_TEMPERATURE) {

alertCount++; // Increment alert count

printf("ALERT: Reading %d (Temperature: %d°C) exceeds maximum safe temperature (%d°C)!\n",

i + 1, temperatures[i], MAX\_TEMPERATURE);

} else {

printf("Reading %d (Temperature: %d°C) is within the safe range.\n", i + 1, temperatures[i]);

}

}

// Display the total number of alerts

printf("\nTotal Alerts: %d\n", alertCount);

}

int main() {

monitorTemperatures();

return 0;

}

**2. Fuel Efficiency Calculator**

Develop a program that calculates and displays fuel efficiency based on distances covered in 10 different trips.

* Use an array to store distances.
* Implement a loop to take inputs and calculate efficiency for each trip using a predefined fuel consumption value.
* Use volatile for sensor data inputs and conditionals to check for low efficiency (< 10 km/L).

#include <stdio.h>

#define FUEL\_CONSUMPTION 5.0 // Fuel consumed per trip in liters (constant)

void calculateEfficiency() {

const int NUM\_TRIPS = 10; // Constant for the number of trips

float distances[NUM\_TRIPS]; // Array to store distances covered

volatile float fuelConsumption = FUEL\_CONSUMPTION; // Volatile for sensor-based input (fixed here)

float efficiency[NUM\_TRIPS]; // Array to store efficiency values

// Take distance inputs for 10 trips

printf("Enter distances covered in 10 trips (in km):\n");

for (int i = 0; i < NUM\_TRIPS; i++) {

printf("Trip %d: ", i + 1);

scanf("%f", &distances[i]);

}

// Calculate and display fuel efficiency for each trip

printf("\nFuel Efficiency Report:\n");

for (int i = 0; i < NUM\_TRIPS; i++) {

efficiency[i] = distances[i] / fuelConsumption; // Calculate efficiency (km/L)

// Check for low efficiency (< 10 km/L)

if (efficiency[i] < 10.0) {

printf("Trip %d: Efficiency = %.2f km/L - ALERT: Low efficiency!\n", i + 1, efficiency[i]);

} else {

printf("Trip %d: Efficiency = %.2f km/L - OK\n", i + 1, efficiency[i]);

}

}

}

int main() {

calculateEfficiency();

return 0;

}

**3. Altitude Monitoring for Aircraft**

Create a program to store altitude readings (in meters) from a sensor over 10 seconds.

* Use a register variable for fast access to the current altitude.
* Store the readings in a single-dimensional array.
* Implement logic to identify if the altitude deviates by more than ±50 meters between consecutive readings.

#include <stdio.h>

#define MAX\_DEVIATION 50 // Maximum allowable deviation in meters

void monitorAltitude() {

const int NUM\_READINGS = 10; // Number of altitude readings

int altitudes[NUM\_READINGS]; // Array to store altitude readings

register int currentAltitude; // Register variable for fast access to the current altitude

// Take altitude readings for 10 seconds

printf("Enter altitude readings (in meters) for 10 seconds:\n");

for (int i = 0; i < NUM\_READINGS; i++) {

printf("Reading %d: ", i + 1);

scanf("%d", &altitudes[i]);

}

// Check for deviations between consecutive readings

printf("\nAltitude Monitoring Report:\n");

for (int i = 1; i < NUM\_READINGS; i++) {

currentAltitude = altitudes[i]; // Current altitude using register variable

int deviation = currentAltitude - altitudes[i - 1]; // Calculatedeviation

printf("%d\n",deviation);

// Check if deviation exceeds ±50 meters

if (deviation > MAX\_DEVIATION || deviation < -MAX\_DEVIATION) {

printf("ALERT: Deviation of %d meters between reading %d and reading %d!\n",

deviation, i, i + 1);

} else {

printf("Reading %d to Reading %d: Stable (Deviation: %d meters)\n",

i, i + 1, deviation);

}

}

}

int main() {

monitorAltitude();

return 0;

}

**4. Satellite Orbit Analyzer**

Design a program to analyze the position of a satellite based on 10 periodic readings.

* Use const for defining the orbit radius and limits.
* Store position data in an array and calculate deviations using loops.
* Alert the user with a decision-making statement if deviations exceed specified bounds.

#include <stdio.h>

#define MAX\_DEVIATION 20.0 // Maximum allowable deviation from the orbit radius in kilometers

void analyzeOrbit() {

const float ORBIT\_RADIUS = 1000.0; // Standard orbit radius in kilometers

const int NUM\_READINGS = 10; // Number of periodic readings

float positions[NUM\_READINGS]; // Array to store position data

float deviations[NUM\_READINGS]; // Array to store calculated deviations

// Take position readings for the satellite

printf("Enter satellite positions (in kilometers) for 10 periodic readings:\n");

for (int i = 0; i < NUM\_READINGS; i++) {

printf("Reading %d: ", i + 1);

scanf("%f", &positions[i]);

}

// Analyze deviations from the orbit radius

printf("\nSatellite Orbit Analysis Report:\n");

for (int i = 0; i < NUM\_READINGS; i++) {

deviations[i] = positions[i] - ORBIT\_RADIUS; // Calculate deviation

// Check if the deviation exceeds allowable bounds

if (deviations[i] > MAX\_DEVIATION || deviations[i] < -MAX\_DEVIATION) {

printf("Reading %d: ALERT! Deviation = %.2f km exceeds allowable bounds.\n",

i + 1, deviations[i]);

} else {

printf("Reading %d: OK. Deviation = %.2f km within bounds.\n",

i + 1, deviations[i]);

}

}

}

int main() {

analyzeOrbit();

return 0;

}

**5. Heart Rate Monitor**

Write a program to record and analyze heart rates from a patient during 10 sessions.

* Use an array to store the heart rates.
* Include static variables to count abnormal readings (below 60 or above 100 BPM).
* Loop through the array to calculate average heart rate and display results.

#include <stdio.h>

#define NUM\_SESSIONS 10 // Number of sessions

#define MIN\_HEART\_RATE 60 // Minimum normal heart rate in BPM

#define MAX\_HEART\_RATE 100 // Maximum normal heart rate in BPM

void analyzeHeartRates() {

int heartRates[NUM\_SESSIONS]; // Array to store heart rates

static int abnormalCount = 0; // Static variable to count abnormal readings

int totalHeartRate = 0; // Variable to calculate the total for average

float averageHeartRate; // Variable to store average heart rate

// Input heart rates for each session

printf("Enter heart rates (in BPM) for 10 sessions:\n");

for (int i = 0; i < NUM\_SESSIONS; i++) {

printf("Session %d: ", i + 1);

scanf("%d", &heartRates[i]);

// Check if the reading is abnormal

if (heartRates[i] < MIN\_HEART\_RATE || heartRates[i] > MAX\_HEART\_RATE) {

abnormalCount++;

}

// Add the reading to the total for average calculation

totalHeartRate += heartRates[i];

}

// Calculate average heart rate

averageHeartRate = (float)totalHeartRate / NUM\_SESSIONS;

// Display results

printf("\nHeart Rate Analysis Report:\n");

printf("Average Heart Rate: %.2f BPM\n", averageHeartRate);

printf("Number of Abnormal Readings: %d\n", abnormalCount);

// Display individual readings and classify them

for (int i = 0; i < NUM\_SESSIONS; i++) {

if (heartRates[i] < MIN\_HEART\_RATE) {

printf("Session %d: %d BPM - BELOW NORMAL\n", i + 1, heartRates[i]);

} else if (heartRates[i] > MAX\_HEART\_RATE) {

printf("Session %d: %d BPM - ABOVE NORMAL\n", i + 1, heartRates[i]);

} else {

printf("Session %d: %d BPM - NORMAL\n", i + 1, heartRates[i]);

}

}

}

int main() {

analyzeHeartRates();

return 0;

}

**6. Medicine Dosage Validator**

Create a program to validate medicine dosage for 10 patients based on weight and age.

* Use decision-making statements to determine if the dosage is within safe limits.
* Use volatile for real-time input of weight and age, and store results in an array.
* Loop through the array to display valid/invalid statuses for each patient.

#include <stdio.h>

#define NUM\_PATIENTS 10 // Number of patients

#define MIN\_DOSAGE 5.0 // Minimum safe dosage in mg

#define MAX\_DOSAGE 20.0 // Maximum safe dosage in mg

void validateDosage() {

volatile float weight[NUM\_PATIENTS]; // Real-time input for weights

volatile int age[NUM\_PATIENTS]; // Real-time input for ages

float dosage[NUM\_PATIENTS]; // Calculated dosages

int results[NUM\_PATIENTS]; // Stores 1 for valid, 0 for invalid

// Input data for each patient

printf("Enter weight (kg) and age (years) for %d patients:\n", NUM\_PATIENTS);

for (int i = 0; i < NUM\_PATIENTS; i++) {

printf("Patient %d - Weight (kg): ", i + 1);

scanf("%f", &weight[i]);

printf("Patient %d - Age (years): ", i + 1);

scanf("%d", &age[i]);

// Calculate dosage: Dosage formula (example: weight \* age / 10)

dosage[i] = (weight[i] \* age[i]) / 10.0;

// Validate dosage

if (dosage[i] >= MIN\_DOSAGE && dosage[i] <= MAX\_DOSAGE) {

results[i] = 1; // Valid

} else {

results[i] = 0; // Invalid

}

}

// Display results

printf("\nDosage Validation Report:\n");

for (int i = 0; i < NUM\_PATIENTS; i++) {

printf("Patient %d: Weight = %.2f kg, Age = %d years, Dosage = %.2f mg - ",

i + 1, weight[i], age[i], dosage[i]);

if (results[i]) {

printf("VALID\n");

} else {

printf("INVALID\n");

}

}

}

int main() {

validateDosage();

return 0;

}

**7. Warehouse Inventory Tracker**

Develop a program to manage the inventory levels of 10 products.

* Store inventory levels in an array.
* Use a loop to update levels and a static variable to track items below reorder threshold.
* Use decision-making statements to suggest reorder actions.

#include <stdio.h>

#define NUM\_PRODUCTS 10 // Number of products

#define REORDER\_THRESHOLD 50 // Minimum inventory level to trigger reorder

void trackInventory() {

int inventory[NUM\_PRODUCTS]; // Array to store inventory levels

static int reorderCount = 0; // Static variable to track items below reorder threshold

int reorderList[NUM\_PRODUCTS]; // Array to track which items need reordering

// Input inventory levels for each product

printf("Enter the inventory levels for %d products:\n", NUM\_PRODUCTS);

for (int i = 0; i < NUM\_PRODUCTS; i++) {

printf("Product %d: ", i + 1);

scanf("%d", &inventory[i]);

// Check if the product needs to be reordered

if (inventory[i] < REORDER\_THRESHOLD) {

reorderList[i] = 1; // Mark for reorder

reorderCount++;

} else {

reorderList[i] = 0; // No reorder needed

}

}

// Display Inventory Report

printf("\nInventory Report:\n");

for (int i = 0; i < NUM\_PRODUCTS; i++) {

printf("Product %d: Inventory = %d - ", i + 1, inventory[i]);

if (reorderList[i]) {

printf("NEEDS REORDER\n");

} else {

printf("SUFFICIENT\n");

}

}

// Display reorder summary

printf("\nSummary:\n");

printf("Total Products Below Reorder Threshold: %d\n", reorderCount);

}

int main() {

trackInventory();

return 0;

}

**8. Missile Launch Codes Validator**

Develop a program to validate 10 missile launch codes.

* Use an array to store the codes.
* Use const for defining valid code lengths and formats.
* Implement decision-making statements to mark invalid codes and count them using a static variable.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define NUM\_CODES 10 // Number of missile codes

#define VALID\_CODE\_LENGTH 8 // Fixed length of a valid code

void validateLaunchCodes() {

char codes[NUM\_CODES][VALID\_CODE\_LENGTH + 1]; // Array to store codes (+1 for null terminator)

static int invalidCount = 0; // Static variable to count invalid codes

int results[NUM\_CODES]; // Stores 1 for valid, 0 for invalid

// Input missile codes

printf("Enter %d missile launch codes (8 characters each):\n", NUM\_CODES);

for (int i = 0; i < NUM\_CODES; i++) {

printf("Code %d: ", i + 1);

scanf("%s", codes[i]);

// Validation

if (strlen(codes[i]) != VALID\_CODE\_LENGTH) {

results[i] = 0; // Invalid if length is incorrect

invalidCount++;

} else {

// Check if all characters are alphanumeric

int isValid = 1;

for (int j = 0; j < VALID\_CODE\_LENGTH; j++) {

if (!isalnum(codes[i][j])) {

isValid = 0;

break;

}

}

if (isValid) {

results[i] = 1; // Valid

} else {

results[i] = 0; // Invalid

invalidCount++;

}

}

}

// Display Validation Report

printf("\nMissile Launch Codes Validation Report:\n");

for (int i = 0; i < NUM\_CODES; i++) {

printf("Code %d: %s - ", i + 1, codes[i]);

if (results[i]) {

printf("VALID\n");

} else {

printf("INVALID\n");

}

}

// Summary

printf("\nSummary:\n");

printf("Total Invalid Codes: %d\n", invalidCount);

}

int main() {

validateLaunchCodes();

return 0;

}

**9. Target Tracking System**

Write a program to track 10 target positions (x-coordinates) and categorize them as friendly or hostile.

* Use an array to store positions.
* Use a loop to process each position and conditionals to classify targets based on predefined criteria (e.g., distance from the base).
* Use register for frequently accessed decision thresholds.

#include <stdio.h>

#include <stdlib.h>

#define NUM\_TARGETS 10 // Number of targets

#define BASE\_POSITION 0 // Base position at x = 0

#define HOSTILE\_THRESHOLD 50 // Threshold distance to classify as hostile

void trackTargets() {

int positions[NUM\_TARGETS]; // Array to store target positions

char classifications[NUM\_TARGETS]; // Array to store target classification ('F' or 'H')

register int threshold = HOSTILE\_THRESHOLD; // Frequently used decision threshold

// Input target positions

printf("Enter the x-coordinates of %d targets:\n", NUM\_TARGETS);

for (int i = 0; i < NUM\_TARGETS; i++) {

printf("Target %d: ", i + 1);

scanf("%d", &positions[i]);

// Classify target

if (abs(positions[i] - BASE\_POSITION) > threshold) {

classifications[i] = 'H'; // Hostile

} else {

classifications[i] = 'F'; // Friendly

}

}

// Display results

printf("\nTarget Tracking Report:\n");

for (int i = 0; i < NUM\_TARGETS; i++) {

printf("Target %d: Position = %d - %s\n",

i + 1,

positions[i],

(classifications[i] == 'H') ? "HOSTILE" : "FRIENDLY");

}

}

int main() {

trackTargets();

return 0;

}

1. Matrix Addition

Problem Statement: Write a program to perform the addition of two matrices. The program should:

Take two matrices as input, each of size M x N, where M and N are defined using const variables.

Use a static two-dimensional array to store the resulting matrix.

Use nested for loops to perform element-wise addition.

Use if statements to validate that the matrices have the same dimensions before proceeding with the addition.

Requirements:

Declare matrix dimensions as const variables.

Use decision-making constructs to handle invalid dimensions.

Print the resulting matrix after addition.

#include <stdio.h>

#define M 3 // Number of rows

#define N 3 // Number of columns

void printMatrix(int matrix[M][N]);

int main() {

// Declare and initialize two matrices

int matrix1[M][N] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int matrix2[M][N] = {

{9, 8, 7},

{6, 5, 4},

{3, 2, 1}

};

// Declare a matrix to store the result

int result[M][N] ;

// Perform matrix addition

for (int i = 0; i < M; i++) {

for (int j = 0; j < N; j++) {

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

// Print the resulting matrix

printf("Resulting Matrix after Addition:\n");

printMatrix(result);

return 0;

}

// Function to print a matrix

void printMatrix(int matrix[M][N]) {

for (int i = 0; i < M; i++) {

for (int j = 0; j < N; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}