```
///----------printing memory location of array elements
#include <stdio.h>

int main()
{
   char a[10];

   for(int i=0;i<10;i++)
   {
      printf("address of a[%d] is %p\n",i,&a[i]);
   }

   return 0;
}

#include <stdio.h>

int main()
{
   int a[10];
   printf("size of int= %d\n",sizeof(int));

   printf("size of array a = %d \n",sizeof(a));

   for(int i=0;i<10;i++)
   {
      printf("address of a[%d] is %p\n",i,&a[i]);
   }

   return 0;
}
```

```c
#include <stdio.h>

int main()
{
   int a[10];
   printf("Size of int = %d \n",sizeof(int));
   printf("Size of array a = %d \n",sizeof(a));
   printf("Address of a is %p \n",a);
   for(int i = 0; i < 10; i++){
      printf("Address of a[%d] is %p \n",i, &a[i]);
   }


   return 0;
}


////---------- Array of bounds

#include <stdio.h>

int main()
{
   int a[10];

   printf("%d \n",a[14]);

   return 0;
}
```

```
///----------------Average of marks using Array---------
#include <stdio.h>

int main()
{
   int grades[10];
   int count = 10;
   long sum = 0;
   float average = 0.0f;

   printf("\nEnter 10 gardes :\n");
   for(int i=0;i<count;i++)
   {
      printf("%2u>",i+1);
      scanf("%d",&grades);
      sum+=grades[i];

   }
   average = (float)sum/count;
   printf("average of the 10 grades = %.2f\n",average);


   return 0;
}


////---------- declaring an array of size 10, initializing the first 2 elemnts

#include <stdio.h>
```

```
int main()
{
   int arr[10] = { 10, 20};
   for(int i=0;i<10;i++)
   {
      printf("arr[%d] = %d\n",i,arr[i]);
   }

   return 0;
}
```

///----------using designated initializer , initializung values to 7th and 9th index elemnts

```
#include <stdio.h>

int main()
{
   int arr[10] = { 10 , 20 , [7]=1 , [9]=50};
   for(int i=0;i<10;i++)
   {
      printf("arr[%d] = %d\n",i,arr[i]);
   }

   return 0;
}
```

```
////----------------no of days in a month
#include <stdio.h>

int main()
{
   int arr[10] = { 10 , 20 , [7]=1 , [9]=50};
   for(int i=0;i<10;i++)
   {
      printf("arr[%d] = %d\n",i,arr[i]);
   }

   return 0;
}

////----------------Prime numbers upto 100

#include <stdio.h>

int main() {
   int prime[100] ;
   int index = 0;

   for (int n = 3; n<= 100; n++) {
      int isPrime = 1;

      for (int i = 2; i * i <= n; i++) {
         if (n % i == 0) {
            isPrime = 0;
            break;
         }
      }
```

```
        if (isPrime) {
            prime[index++] = n;
        }
    }

    printf("Prime numbers between 3 and 100 are:\n");
    for (int i = 0; i < index; i++) {
        printf("%d ", prime[i]);
    }
    printf("\n");

    return 0;
}
```

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>
#define ARRAY_SIZE 10
void findMaxMin(int arr[],int size, int *max, int *min)
{

*max= arr[0];
*min= arr[0];
    for( int i=1;i< 10; i++)
```

```
    {
        if(arr[i]> *max)
        {
            *max = arr[i];
        }
        if(arr[i]< *min)
        {
            *min = arr[i];
        }

    }
}


int main()

{
    const int size = ARRAY_SIZE;

    int arr[ARRAY_SIZE] = {12, 3, 45, 7, 9, 23, 56, 1, 34, 8};
    int max, min;
    static int maxDifference = 0;

    findMaxMin(arr, size, &max, &min);
    maxDifference = max - min;

    printf("Maximum value: %d\n", max);
    printf("Minimum value: %d\n", min);
    printf("Maximum difference between max and min values: %d\n", maxDifference);

    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>

#define SIZE 10

void categorizeElements(int arr[], int size, int pos[], int neg[], int zero[], int *posCount, int *negCount, int *zeroCount)
{
   *posCount = 0;
   *negCount = 0;
   *zeroCount = 0;

   for (int i = 0; i < size; i++)
   {
      if (arr[i] > 0)
      {
         pos[(*posCount)++] = arr[i];
      } else if (arr[i] < 0)
      {
         neg[(*negCount)++] = arr[i];
      } else
      {
         zero[(*zeroCount)++] = arr[i];
      }
   }
```

```
}

int main()

{
    int arr[SIZE] = {1, -2, 0, 4, -5, 6, 0, -8, 9, 0};

    static int pos[SIZE];
    static int neg[SIZE];
    static int zero[SIZE];

    int posCount, negCount, zeroCount;

    categorizeElements(arr, SIZE, pos, neg, zero, &posCount, &negCount, &zeroCount);

    printf("Positive numbers: ");
    for (int i = 0; i < posCount; i++) {
        printf("%d ", pos[i]);
    }
    printf("\n");

    printf("Negative numbers: ");
    for (int i = 0; i < negCount; i++) {
        printf("%d ", neg[i]);
    }
    printf("\n");

    printf("Zero values: ");
    for (int i = 0; i < zeroCount; i++) {
        printf("%d ", zero[i]);
    }
```

```
    printf("\n");


    return 0;
}
```

3. 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>


#define SIZE 10


void sum( int arr[] , int size)
{
   static int cumulativeSum = 0;


    for (int i = 0; i < size; i++)
    {
        cumulativeSum += arr[i];
        printf("Cumulative sum after %d : %d\n",i, cumulativeSum);
    }
}


int main()
{
    const int size = SIZE;
    int arr[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9,};
```

```
    sum(arr, 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>
#define SIZE 10

int main() {

    int array[SIZE] = {2, 3, 4, 5, 6, 7, 8, 9, 10,11};
    int i, j, isPrime;

    printf("Prime numbers in the array are: \n");
    for (i = 0; i < SIZE; i++)
    {
        if (array[i] < 2)
        {
            continue;
        }
        isPrime = 1;
        for (j = 2; j <= array[i] / 2; j++)
```

```
        {
            if (array[i] % j == 0)
            {
                isPrime = 0;
                break;
            }
        }

        if (isPrime)
        {
            printf("%d ", array[i]);
        }
    }
    printf("\n");

    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>

#define SIZE 10
```

```
void rotateLeft(int arr[], int n, int size)
{
   static int rotated[SIZE];
   int i = 0;
   int j = n % size;

   while (i < size)
   {
      rotated[i] = arr[j];
      i++;
      j = (j + 1) % size;
   }


   for (i = 0; i < size; i++) {
      arr[i] = rotated[i];
   }
}

int main()

{
   const int n = 3;

   int arr[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   printf("Original array:\n");

   for (int i = 0; i < SIZE; i++)
   {
      printf("%d ", arr[i]);
```

```
    }
    printf("\n");

    rotateLeft(arr, n, SIZE);

    printf("Array after rotating left by %d positions:\n", n);
    for (int i = 0; i < SIZE; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    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>
#define SIZE 10

int main() {
    int a[SIZE] = {1, 2, 2, 3, 3, 3, 4, 4, 4, 4};
    int f[SIZE] = {0};
    for (int i = 0; i < SIZE; i++)
    {
```

```
        if (f[i] != 0)
        {
            continue;
        }

        int count = 1;

        for (int j = i + 1; j < SIZE; j++)
        {
            if (a[i] == a[j]) {
                count++;
                f[j] = -1;
            }
        }

        f[i] = count;
    }

    printf("Element Frequency\n");
    for (int i = 0; i < SIZE; i++)
    {
        if (f[i] > 0)
        {
            printf("%d      %d\n", a[i], f[i]);
        }
    }

    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>

#define SIZE 5

void bubbleSort(int arr[], const int size)
{
   for (int i = 0; i < size - 1; i++)
   {
      for (int j = 0; j < size - i - 1; j++)
      {
         if (arr[j] < arr[j + 1])
         {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

int main()

{
```

```
    int arr[SIZE] = {6, 34, 5, 12, 2};

    printf("Original array: ");
    for (int i = 0; i < SIZE; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    bubbleSort(arr, SIZE);

    printf("Sorted array in descending order: ");

    for (int i = 0; i < 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>
#define SIZE 10
```

```
int main()
{
   int a[SIZE] = {10, 20, 30, 50, 8, 15, 70, 80, 90, 100};
   static int secondLargest = -1;
   int largest = 1;

   if (SIZE < 2)
   {
      printf("Array must have atleast two elements\n");
      return 0;
   }

   for (int i = 0; i < SIZE; i++)
   {
      if (a[i] > largest)
      {
         secondLargest = largest;
         largest = a[i];
      } else if (a[i] > secondLargest && a[i] != largest) {
         secondLargest = a[i];
      }
   }

   printf("The second largest element is: %d\n", secondLargest);

   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>


#define SIZE 10


int main()
{
   int arr[SIZE] = {1,2,3,4,5,6,7,8,9,10};
   int even[SIZE], odd[SIZE];
   int evenCount = 0, oddCount = 0;



   for (int i = 0; i < SIZE; i++)
   {
      if (arr[i] % 2 == 0)
      {
         even[evenCount++] = arr[i];
      } else
```

```
        {
            odd[oddCount++] = arr[i];
        }
    }


    printf("Even numbers: ");
    for (int i = 0; i < evenCount; i++)
    {
        printf("%d ", even[i]);
    }
    printf("\n");


    printf("Odd numbers: ");
    for (int i = 0; i < oddCount; i++)
    {
        printf("%d ", odd[i]);
    }
    printf("\n");

    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.

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

void cyclicShift(int arr[])

{
   static int lastElement;
   lastElement = arr[SIZE - 1];

   for (int i = SIZE - 1; i > 0; i--)
 {
      arr[i] = arr[i - 1];
   }



   arr[0] = lastElement;
}

int main()

{
   int arr[SIZE] = {1, 2, 3, 4, 5};

   printf("Original array: ");

   for (int i = 0; i < SIZE; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");
```

```
    cyclicShift(arr);

    printf("Array after cyclic shift to the right: ");

    for (int i = 0; i < 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.

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 10.0

int main()
{
   volatile float distances[10];
   float efficiencies[10];
   int i;



   printf("Enter distances covered in 10 trips (in km):\n");
   for (i = 0; i < 10; i++)
   {
      printf("Trip %d: ", i + 1);
      scanf("%f", &distances[i]);
   }

   printf("\nFuel efficiency for each trip:\n");
   for (i = 0; i < 10; i++)
   {
      efficiencies[i] = distances[i] / FUEL_CONSUMPTION;
      printf("Trip %d: %.2f km/L", i + 1, efficiencies[i]);

      if (efficiencies[i] < 10.0)
      {
         printf(" (Low Efficiency!)");
      }
      printf("\n");
   }

   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 READINGS 10
#define DEVIATION_THRESHOLD 50

int main() {
   int altitudes[READINGS];
   register int current_altitude;
   int i;
   printf("Enter altitude readings (in meters) for 10 seconds:\n");
   for (i = 0; i < READINGS; i++)
   {
      printf("Second %d: ", i + 1);
      scanf("%d", &altitudes[i]);
   }


   printf("\nChecking for deviations greater than ±%d meters:\n", DEVIATION_THRESHOLD);
   for (i = 1; i < READINGS; i++)
   {
      current_altitude = altitudes[i];
      int difference = current_altitude - altitudes[i - 1];
```

```
        if (difference > DEVIATION_THRESHOLD || difference < -DEVIATION_THRESHOLD)
        {
            printf("Deviation detected between second %d and %d: %d meters\n", i, i + 1, difference);
        }
    }

    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 READINGS 10
#define MAX_DEVIATION 100

int main()
{
    const int ORBIT_RADIUS = 20000;
    int positions[READINGS];
    int i;

    printf("Enter satellite position readings (in km) relative to the Earth's center:\n");
    for (i = 0; i < READINGS; i++)
    {
        printf("Reading %d: ", i + 1);
        scanf("%d", &positions[i]);
    }
```

```
    printf("\n deviations from orbit radius (%d km):\n", ORBIT_RADIUS);

    for (i = 0; i < READINGS; i++)
    {
        int deviation = positions[i] - ORBIT_RADIUS;
        if (deviation > MAX_DEVIATION || deviation < -MAX_DEVIATION)
        {
            printf("Alert: Reading %d exceeds the bound Deviation = %d km\n", i + 1, deviation);
        } else {
            printf("Reading %d is within bounds. Deviation = %d km\n", i + 1, deviation);
        }
    }

    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
#define LOWER_LIMIT 60
#define UPPER_LIMIT 100
void heartRate(int heartRates[], int size)
{
    static int abnormalCount = 0;
```

```
    int sum = 0;
    int i;



    for (i = 0; i < size; i++)
    {
        if (heartRates[i] < LOWER_LIMIT || heartRates[i] > UPPER_LIMIT)
        {
            abnormalCount++;
        }
        sum += heartRates[i];
    }



    float average = (float)sum / size;
    printf("\nResults:\n");
    printf("Average Heart Rate: %.2f\n", average);
    printf("Number of Abnormal Readings: %d\n", abnormalCount);
    if (abnormalCount > 0) {
        printf("Warning: Some readings are abnormal\n");

    } else {
        printf("All readings are within the normal range.\n");
    }
}
```

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>



int validate_dosage(volatile int weight, volatile int age)
{
   if (age < 12) {
      if (weight < 30) {
         return 1;
      } else {
         return 0;
      }
   } else {
      if (weight < 50) {
         return 1;
      } else {
         return 0;
      }
   }
}

int main() {
   volatile int weight[10];
   volatile int age[10];
   int results[10];



   for (int i = 0; i < 10; i++) {
      printf("Enter weight (kg) for patient %d: ", i + 1);
      scanf("%d", &weight[i]);
      printf("Enter age (years) for patient %d: ", i + 1);
      scanf("%d", &age[i]);
```

```
    }


    for (int i = 0; i < 10; i++) {
        results[i] = validate_dosage(weight[i], age[i]);
    }


    for (int i = 0; i < 10; i++) {
        if (results[i] == 1) {
            printf("Patient %d: Valid dosage\n", i + 1);
        } else {
            printf("Patient %d: Invalid dosage\n", i + 1);
        }
    }

    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 PRODUCTS 10
#define THRESHOLD 50

void manageInventory(int inventory[], int size)
```

```
{
   static int belowThresholdCount = 0;
   int i;
   printf("\nInventory Analysis:\n");
   for (i = 0; i < size; i++)
   {
      printf("Product %d:  Level = %d ", i + 1, inventory[i]);
      if (inventory[i] <THRESHOLD)
      {
         belowThresholdCount++;
         printf("(Reorder Suggested)");
      }
      printf("\n");
   }


   printf("\nSummary:\n");
   printf("Number of Products Below Reorder Threshold: %d\n", belowThresholdCount);

   if (belowThresholdCount > 0)
   {
      printf("Please reorder items with low stock.\n");
   } else
   {
      printf("All inventory levels are on stock\n");
   }
}

int main()

{
```

```
    int inventory[PRODUCTS];
    int i;
    printf("Enter inventory levels for %d products:\n", PRODUCTS);
    for (i = 0; i < PRODUCTS; i++)
    {
        printf("Product %d: ", i + 1);
        scanf("%d", &inventory[i]);
    }
    manageInventory(inventory, PRODUCTS);

    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>
#define CODES 10
#define MLENGTH 6

static int invalidCodeCount = 0;

int isValidCode(char code[]) {
    int length = 0;
    while (code[length] != '\0') {
        length++;
```

```
    }

    if (length != MLENGTH) {
        return 0;
    }

    for (int i = 0; i < 3; i++) {
        if (code[i] < 'A' || code[i] > 'Z') {
            return 0;
        }
    }

    for (int i = 3; i < 6; i++) {
        if (code[i] < '0' || code[i] > '9') {
            return 0;
        }
    }

    return 1;
}

int main() {
    char codes[CODES][MLENGTH + 1];
    int validCodeCount = 0;

    for (int i = 0; i < CODES; i++) {
        printf("Enter missile launch code %d: ", i + 1);
        scanf("%s", codes[i]);

        if (isValidCode(codes[i])) {
            validCodeCount++;
```

```
        } else {
            invalidCodeCount++;
        }
    }

    printf("\nValidation Summary:\n");
    printf("Valid codes: %d\n", validCodeCount);
    printf("Invalid codes: %d\n", invalidCodeCount);

    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>

#define TARGET 10

int main()
{
    int positions[TARGET] = {6, 12, -12, 9, 2, -9, 8, -6, -2, 10};

    register int threshold = 5;
```

```
    for (int i = 0; i < TARGET; i++) {
        int position = positions[i];

        if (position >= -threshold && position <= threshold) {
            printf("Target at position %d: Friendly\n", position);
        } else {
            printf("Target at position %d: Hostile\n", position);
        }
    }

    return 0;
}
```

Problem Statements on 2 Dimensional Arrays

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
#define N 3

int main()

{
   int A1[M][N], A2[M][N], result[M][N];

   printf("Enter elements of the first matrix (%d x %d):\n", M, N);
   for (int i = 0; i < M; i++)
   {
      for (int j = 0; j < N; j++)
      {
         printf("Enter element at position [%d][%d]: ", i + 1, j + 1);
         scanf("%d", &A1[i][j]);
      }
   }

   printf("Enter elements of the second matrix (%d x %d):\n", M, N);
   for (int i = 0; i < M; i++)
   {
      for (int j = 0; j < N; j++)
      {
         printf("Enter element at position [%d][%d]: ", i + 1, j + 1);
         scanf("%d", &A2[i][j]);
```

```
        }
    }

    if (M != M || N != N)
    {
        printf("Matrix dimensions do not match.\n");
        return 1;
    }

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            result[i][j] = A1[i][j] + A2[i][j];
        }
    }

    printf("Resulting matrix :\n");
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

2. Transpose of a Matrix

- **Problem Statement**: Write a program to compute the transpose of a matrix. The program should:
    - Take a matrix of size M x N as input, where M and N are declared as const variables.
    - Use a static two-dimensional array to store the transposed matrix.
    - Use nested for loops to swap rows and columns.
    - Validate the matrix size using if statements before transposing.
- **Requirements**:
    - Print the original and transposed matrices.
    - Use a type qualifier (const) to ensure the matrix size is not modified during execution.

```
#include <stdio.h>

#define M 3
#define N 4

void printMatrix(const int matrix[M][N], int rows, int cols)
{
   for (int i = 0; i < rows; i++)
   {
      for (int j = 0; j < cols; j++)
      {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
}

void transposeMatrix(const int original[M][N], int transposed[N][M])
{
   for (int i = 0; i < M; i++)
```

```
    {
        for (int j = 0; j < N; j++)
        {
            transposed[j][i] = original[i][j];
        }
    }
}

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

    if (M <= 0 || N <= 0)
    {
        printf("Invalid matrix size.\n");
        return 1;
    }

    int transposed[N][M];

    printf("Original Matrix:\n");
    printMatrix(matrix, M, N);

    transposeMatrix(matrix, transposed);

    printf("\nTransposed Matrix:\n");
    for (int i = 0; i < N; i++) {
```

```
        for (int j = 0; j < M; j++) {
            printf("%d ", transposed[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

3. Find the Maximum Element in Each Row

- **Problem Statement**: Write a program to find the maximum element in each row of a two-dimensional array. The program should:
    - Take a matrix of size M x N as input, with dimensions defined using const variables.
    - Use a static array to store the maximum value of each row.
    - Use nested for loops to traverse each row and find the maximum element.
    - Use if statements to compare and update the maximum value.
- **Requirements**:
    - Print the maximum value of each row after processing the matrix.
    - Handle edge cases where rows might be empty using decision-making statements.

```
#include <stdio.h>

#define M 3
#define N 4

int main() {
    int m[M][N] = {
        {0, 2, 4, 6},
        {8, 10, 12, 14},
        {1, 2, 3, 4}
```

```
    };

    int maxRow[M];

    for (int i = 0; i < M; i++) {
        int maxVal = m[i][0];

        for (int j = 1; j < N; j++) {
            if (m[i][j] > maxVal) {
                maxVal = m[i][j];
            }
        }

        maxRow[i] = maxVal;
    }
    printf("Maximum values in each row:\n");
    for (int i = 0; i < M; i++) {
        printf("Row %d: %d\n", i + 1, maxRow[i]);
    }

    return 0;
}
```

4. Matrix Multiplication

- **Problem Statement**: Write a program to multiply two matrices. The program should:
    - Take two matrices as input:
        - Matrix A of size M x N
        - Matrix B of size N x P
    - Use const variables to define the dimensions M, N, and P.
    - Use nested for loops to calculate the product of the matrices.

- Use a static two-dimensional array to store the resulting matrix.
- Use if statements to validate that the matrices can be multiplied (N in Matrix A must equal M in Matrix B).

- **Requirements**:
  - Print both input matrices and the resulting matrix.
  - Handle cases where multiplication is invalid using decision-making constructs.

```
#include <stdio.h>

int main() {
   const int M = 2;
   const int N = 3;
   const int P = 2;

   int A[M][N], B[N][P], result[M][P];

   printf("Enter the elements of Matrix A (size %dx%d):\n", M, N);
   for (int i = 0; i < M; i++) {
      for (int j = 0; j < N; j++) {
         scanf("%d", &A[i][j]);
      }
   }

   printf("Enter the elements of Matrix B (size %dx%d):\n", N, P);
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < P; j++) {
         scanf("%d", &B[i][j]);
      }
   }
```

```
if (N != N) {
    printf("Matrix multiplication is not possible.\n");
    return 1;
}

for (int i = 0; i < M; i++) {
    for (int j = 0; j < P; j++) {
        result[i][j] = 0;
    }
}



for (int i = 0; i < M; i++) {
    for (int j = 0; j < P; j++) {
        for (int k = 0; k < N; k++) {
            result[i][j] += A[i][k] * B[k][j];
        }
    }
}

printf("\nMatrix A:\n");
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        printf("%d ", A[i][j]);
    }
    printf("\n");
}

printf("\nMatrix B:\n");
for (int i = 0; i < N; i++) {
    for (int j = 0; j < P; j++) {
```

```
            printf("%d ", B[i][j]);
        }
        printf("\n");
    }

    printf("\nResulting Matrix (A x B):\n");
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < P; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

5. Count Zeros in a Sparse Matrix

- **Problem Statement**: Write a program to determine if a given matrix is sparse. A matrix is sparse if most of its elements are zero. The program should:
    - Take a matrix of size M x N as input, with dimensions defined using const variables.
    - Use nested for loops to count the number of zero elements.
    - Use if statements to compare the count of zeros with the total number of elements.
    - Use a static variable to store the count of zeros.
- **Requirements**:
    - Print whether the matrix is sparse or not.
    - Use decision-making statements to handle matrices with no zero elements.
    - Validate matrix dimensions before processing.

```
#include <stdio.h>
#define M 3
#define N 3
```

```
void checkSparse(int matrix[M][N]) {
    static int zeroCount = 0;

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (matrix[i][j] == 0) {
                zeroCount++;
            }
        }
    }

    int totalElements = M * N;

    if (zeroCount == 0) {
        printf("The matrix has no zero elements.\n");
    } else if (zeroCount > totalElements / 2) {
        printf("The matrix is sparse.\n");
    } else {
        printf("The matrix is not sparse.\n");
    }
}

int main() {
    int matrix[M][N];

    printf("Enter the elements of the matrix (%dx%d):\n", M, N);
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
```

```
    if (M <= 0 || N <= 0) {
        printf("Invalid matrix dimensions.\n");
        return 1;
    }

    checkSparse(matrix);

    return 0;
}
```

Problem Statements on 3 Dimensional Arrays

1. 3D Matrix Addition

Problem Statement: Write a program to perform element-wise addition of two three-dimensional matrices. The program should:

Take two matrices as input, each of size X x Y x Z, where X, Y, and Z are defined using const variables.

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

Use nested for loops to iterate through the elements of the matrices.

Use if statements to validate that the dimensions of both matrices are the same before performing addition.

Requirements:

Declare matrix dimensions as const variables.

Use decision-making statements to handle mismatched dimensions.

Print the resulting matrix after addition.

```
#include <stdio.h>

#define X 3
#define Y 3
#define Z 3
```

```
void inputMatrix(int matrix[X][Y][Z], const char *name) {
    printf("Enter elements for matrix %s:\n", name);
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("Element [%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrix[i][j][k]);
            }
        }
    }
}

void addMatrices(int matrix1[X][Y][Z], int matrix2[X][Y][Z], int result[X][Y][Z]) {
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                result[i][j][k] = matrix1[i][j][k] + matrix2[i][j][k];
            }
        }
    }
}

void printMatrix(int matrix[X][Y][Z], const char *name) {
    printf("Matrix %s:\n", name);
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%d ", matrix[i][j][k]);
            }
            printf("\n");
```

```
        }
        printf("\n");
    }
}

int main() {
    int matrix1[X][Y][Z];
    int matrix2[X][Y][Z];
    static int result[X][Y][Z];

    inputMatrix(matrix1, "A");
    inputMatrix(matrix2, "B");



    if (X == X && Y == Y && Z == Z) {
        addMatrices(matrix1, matrix2, result);
        printMatrix(result, "Result");
    } else {
        printf("Matrices dimensions do not match.\n");
    }

    return 0;
}
```

2. Find the Maximum Element in a 3D Array

Problem Statement: Write a program to find the maximum element in a three-dimensional matrix. The program should:

Take a matrix of size X x Y x Z as input, where X, Y, and Z are declared as const variables.

Use a static variable to store the maximum value found.

Use nested for loops to traverse all elements of the matrix.

Use if statements to compare and update the maximum value.

Requirements:

Print the maximum value found in the matrix.

Handle edge cases where the matrix might contain all negative numbers or zeros using decision-making statements.

```
#include <stdio.h>

#define X 3
#define Y 3
#define Z 3

void findMaxMatrix(int matrix[X][Y][Z]) {
   static int maxValue;
   int initialized = 0;

   for (int i = 0; i < X; i++) {
      for (int j = 0; j < Y; j++) {
         for (int k = 0; k < Z; k++) {
            if (!initialized) {
               maxValue = matrix[i][j][k];
               initialized = 1;
            } else if (matrix[i][j][k] > maxValue) {
               maxValue = matrix[i][j][k];
            }
         }
      }
   }
```

```
    printf("The maximum value in the matrix is: %d\n", maxValue);
}

int main() {
    int matrix[X][Y][Z] = {
        {
            {1, -2, 3},
            {4, 5, -6},
            {7, 8, 9}
        },
        {
            {-10, 11, 12},
            {13, 14, 15},
            {16, 17, 18}
        },
        {
            {19, 20, 21},
            {22, 23, 24},
            {25, 26, 27}
        }
    };

    findMaxMatrix(matrix);

    return 0;
}
```

3. 3D Matrix Scalar Multiplication

Problem Statement: Write a program to perform scalar multiplication on a three-dimensional matrix. The program should:

Take a matrix of size X x Y x Z and a scalar value as input, where X, Y, and Z are declared as const variables.

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

Use nested for loops to multiply each element of the matrix by the scalar.

Requirements:

Print the original matrix and the resulting matrix after scalar multiplication.

Use decision-making statements to handle invalid scalar values (e.g., zero or negative scalars) if necessary.

```
#include <stdio.h>

#define X 2
#define Y 3
#define Z 4

void printMatrix(int matrix[X][Y][Z]) {
   for (int i = 0; i < X; i++) {
      for (int j = 0; j < Y; j++) {
         for (int k = 0; k < Z; k++) {
            printf("%d ", matrix[i][j][k]);
         }
         printf("\n");
      }
      printf("\n");
   }
}

int main() {
   int matrix[X][Y][Z] = {
      {
         {1, 2, 3, 4},
         {5, 6, 7, 8},
         {9, 10, 11, 12}
```

```
    },
    {
        {13, 14, 15, 16},
        {17, 18, 19, 20},
        {21, 22, 23, 24}
    }
};
int result[X][Y][Z];
int scalar;

printf("Enter a scalar value: ");
scanf("%d", &scalar);

if (scalar <= 0) {
    printf("Invalid scalar value. Please enter a positive non-zero scalar.\n");
    return 1;
}

printf("Original Matrix:\n");
printMatrix(matrix);



for (int i = 0; i < X; i++) {
    for (int j = 0; j < Y; j++) {
        for (int k = 0; k < Z; k++) {
            result[i][j][k] = matrix[i][j][k] * scalar;
        }
    }
}

printf("Resulting Matrix after scalar multiplication:\n");
```

```
    printMatrix(result);

    return 0;
}
```

4. Count Positive, Negative, and Zero Elements in a 3D Array

Problem Statement: Write a program to count the number of positive, negative, and zero elements in a three-dimensional matrix. The program should:

Take a matrix of size X x Y x Z as input, where X, Y, and Z are defined using const variables.

Use three static variables to store the counts of positive, negative, and zero elements, respectively.

Use nested for loops to traverse the matrix.

Use if-else statements to classify each element.

Requirements:

Print the counts of positive, negative, and zero elements.

Ensure edge cases (e.g., all zeros or all negatives) are handled correctly.

```
#include <stdio.h>

#define X 3
#define Y 3
#define Z 3

void countElements(int matrix[X][Y][Z], int *positiveCount, int *negativeCount, int *zeroCount) {
    static int posCount = 0;
    static int negCount = 0;
    static int zeroCountStatic = 0;

    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
```

```
            if (matrix[i][j][k] > 0) {
                posCount++;
            } else if (matrix[i][j][k] < 0) {
                negCount++;
            } else {
                zeroCountStatic++;
            }
        }
    }

    *positiveCount = posCount;
    *negativeCount = negCount;
    *zeroCount = zeroCountStatic;
}

int main() {
    int matrix[X][Y][Z] = {
        {
            {1, -2, 0},
            {4, 5, -6},
            {0, 0, 0}
        },
        {
            {-1, 2, 3},
            {0, -5, 6},
            {7, 8, -9}
        },
        {
            {0, 0, 0},
            {1, -1, 1},
```

```
        {-1, 1, -1}
      }
    };

    int positiveCount, negativeCount, zeroCount;

    countElements(matrix, &positiveCount, &negativeCount, &zeroCount);

    printf("Positive elements: %d\n", positiveCount);
    printf("Negative elements: %d\n", negativeCount);
    printf("Zero elements: %d\n", zeroCount);

    return 0;
}
```

5. Transpose of a 3D Matrix Along a Specific Axis

Problem Statement: Write a program to compute the transpose of a three-dimensional matrix along a specific axis (e.g., swap rows and columns for a specific depth). The program should:

Take a matrix of size X x Y x Z as input, where X, Y, and Z are defined using const variables.

Use a static three-dimensional array to store the transposed matrix.

Use nested for loops to perform the transpose operation along the specified axis.

Use if statements to validate the chosen axis for transposition.

Requirements:

Print the original matrix and the transposed matrix.

Ensure invalid axis values are handled using decision-making constructs.

```
#include <stdio.h>

#define X 3
#define Y 3
#define Z 3
```

```
void printMatrix(int matrix[X][Y][Z], const char *name) {
    printf("%s:\n", name);
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%d ", matrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

void transposeAlongAxis(int matrix[X][Y][Z], int transposed[X][Y][Z], int axis) {
    if (axis == 0) {
        for (int i = 0; i < X; i++) {
            for (int j = 0; j < Y; j++) {
                for (int k = 0; k < Z; k++) {
                    transposed[j][i][k] = matrix[i][j][k];
                }
            }
        }
    } else if (axis == 1) {
        for (int i = 0; i < X; i++) {
            for (int j = 0; j < Y; j++) {
                for (int k = 0; k < Z; k++) {
                    transposed[k][j][i] = matrix[i][j][k];
                }
            }
        }
```

```
    } else if (axis == 2) {
        for (int i = 0; i < X; i++) {
            for (int j = 0; j < Y; j++) {
                for (int k = 0; k < Z; k++) {
                    transposed[i][k][j] = matrix[i][j][k];
                }
            }
        }
    } else {
        printf("Invalid axis value. Please choose 0, 1, or 2.\n");
    }
}

int main() {
    int matrix[X][Y][Z] = {
        {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
        {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}},
        {{19, 20, 21}, {22, 23, 24}, {25, 26, 27}}
    };

    int transposed[X][Y][Z] = {0};
    int axis;

    printf("Enter the axis for transposition (0, 1, or 2): ");
    scanf("%d", &axis);

    printMatrix(matrix, "Original Matrix");

    transposeAlongAxis(matrix, transposed, axis);

    if (axis >= 0 && axis <= 2) {
```

```
        printMatrix(transposed, "Transposed Matrix");
    }

    return 0;
}
```