----
1. 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 SIZE 4  static

int max_diff=0; int

main()

{

  int a[SIZE]={15,30,45,60};

  int max=a[0]; int

  min=a[0];

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

      if(a[i]>max){

          max=a[i];

      }

      if(a[i]<min){

          min=a[i];

      }

  }

  max_diff=max-min;

  printf("Maximum Element is %d\n",max);

  printf("Minimum Element is %d\n",min);

  printf("Max_Diff is %d\n",max_diff);
  return 0;
}
```

Output

```
-------
Maximum  Element  is  60
Minimum  Element  is  15
Max_Diff is 45
```

2. 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 4

int main() {
    int arr[SIZE] = {1,-3,0,6};
    int positive[SIZE], negative[SIZE],zero[SIZE];
    int positiveIndex = 0, negativeIndex = 0,zeroIndex = 0;

    for (int i = 0; i < SIZE; i++) { if
        (arr[i]>0) {
            positive[positiveIndex++] = arr[i];
        } else if(arr[i]<0) { negative[negativeIndex++]
            = arr[i];
        }
        else{
            zero[zeroIndex++] = arr[i];
        }
    }

    printf("Positive numbers: ");
    for (int i = 0; i < positiveIndex; i++) {
        printf("%d ", positive[i]);
    }

    printf("\nNegative numbers: ");
    for (int i = 0; i < negativeIndex; i++) { printf("%d
        ", negative[i]);
    }

    printf("\nLess than 0 numbers: "); for
    (int i = 0; i < zeroIndex; i++) {
        printf("%d ", zero[i]);
    }

    return 0;
}
```

Output

```
Positive numbers:1 6

Negative numbers: -3

Less than 0 numbers: 0
```

3. 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>
static int cumulative_sum=0;
const int a[10]={1,2,3,4,5,6,7,8,9,10};
int main()
{
    for(int i=0;i<10;i++){
        cumulative_sum=cumulative_sum+a[i];
    }
    printf("Cumulative Sum is %d\n",cumulative_sum);
    return 0;
}
```

Output

```
– – – –
Cumulative Sum is 55
```

4. 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>


int main()

{

  const int primes[12]={2,3,4,5,6,7,8,9,10,11,12}; for(int

  num=2;num<12;num++){

      int prime=1;

      for(int i=2;i<num;i++){ if(num%i==0){

              prime=0;

              break;

          }

      }

      if(prime){

          printf("%d ",num);

      }

  }

  return 0;


}
```

Output

```
-------

2 3 5 7 11
```

5. 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 5
#define N 2

int main() {
    int arr[SIZE] = {1, 2, 3, 4, 5};
    static int rotated[SIZE]; int
    i = 0;

    while (i < SIZE) {
        rotated[i] = arr[(i + N) % SIZE]; i++;
    }

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

    printf("Array after rotation: ");
    for (i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

```
Output
-------
Array after rotation:3 4 5 1 2
```

6. 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 5

int main() {
    int arr[SIZE] = {1, 2, 3, 2, 1};
    static int freq[SIZE]; int
    i, j;

    freq[i] = 0;

    for (i = 0; i < SIZE; i++) { if
        (freq[i] == 0) {
            for (j = 0; j < SIZE; j++) { if
                (arr[i] == arr[j]) {
                    freq[i]++;
                }
            }
        }
    }

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

    return 0;
}
```

```
Output
-------
1 appears 2 times
2 appears 2 times
3 appears 1 times
2 appears 2 times
1 appears 2 times
```

7. 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

int main() {
    int arr[SIZE] = {5, 2, 9, 1, 5};
    int temp, i, j;

    for (i = 0; i < SIZE - 1; i++) {
        for (j = 0; j < SIZE - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {

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

    printf("Array in descending order: "); for
    (i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output

```
-------
Array in descending order:9 5 5 2 1
```

8. 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 5

int main() {
    int arr[SIZE] = {1, 2, 3, 4, 5};
    static int sec_largest = -1; int
    largest = arr[0];



    for (int i = 1; i < SIZE; i++) { if
        (arr[i] > largest) {
            sec_largest = largest;
            largest = arr[i];
        } else if (arr[i] > sec_largest && arr[i] != largest) {
            sec_largest = arr[i];
        }
    }

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

    return 0;
}
```

Output

```
--------
The second largest element is:4
```

9. 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 evenIndex = 0, oddIndex = 0;

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

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

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

    return 0;
}
```

```
Output
-------
Even numbers:2 4 6 8 10
Odd numbers:1 3 5 7 9
```

10. 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

int main() {
    int arr[SIZE] = {10, 20, 30, 40, 50};
    static int temp;
    int i;
    temp = arr[SIZE - 1];

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

    arr[0] = temp;

    printf("Array after cyclic shift to the right: "); for
    (i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output

```
--------
Array after cyclic shift to the right:50 10 20 30 40
```

11. 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_TEMP 100
```

```
#define INTERVALS 10

extern int temperatureReadings[INTERVALS];

int temperatureReadings[INTERVALS];

int main() {
    static int alertCount = 0;
    int i;

    printf("Enter the engine temperatures for 10 intervals (in degrees Celsius):\n");

    for (i = 0; i < INTERVALS; i++) {
        printf("Temperature at interval %d: ", i + 1);
        scanf("%d", &temperatureReadings[i]);
    }

    for (i = 0; i < INTERVALS; i++) {
        if (temperatureReadings[i] > MAX_TEMP) {
            printf("Alert: Temperature exceeded safe limit at interval %d! (Temp:
%d°C)\n", i + 1, temperatureReadings[i]);
            alertCount++;
        }
    }
    printf("\nNumber of alerts: %d\n", alertCount);

    if (alertCount == 0) {
        printf("All temperatures are within the safe limit.\n");
    }
    return 0;
}
```

12. 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_USED 6.0
#define TRIPS 10



int main() {
    float distances[TRIPS];
    volatile float fuel = FUEL_USED; float
    efficiency;
    int i;

    printf("Enter the distnces for 10 trips(in kms):\n");

    for(i=0;i<TRIPS;i++){
        printf("Distance for trip %d:",i+1);
        scanf("%f",&distances[i]);
    }

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

    for (i = 0; i < TRIPS; i++) {
        efficiency = distances[i] / fuel;
        printf("Trip %d: %.2f km/L", i + 1, efficiency);

        if (efficiency < 10.0) { printf("Low
            efficiency");
        }
        printf("\n");
    }
}
```

13. 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_LIMIT 50

int main() {
    int altitude[READINGS];
    register int currentAltitude;
    int i;

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



    for (i = 0; i < READINGS; i++) {
        printf("Reading %d: ", i + 1); scanf("%d",
        &altitude[i]);
    }

    printf("\nChecking for altitude deviations greater than ±%d meters:\n",
DEVIATION_LIMIT);

    for (i = 1; i < READINGS; i++) {
        currentAltitude = altitude[i];
        int previousAltitude = altitude[i - 1];
        int deviation = currentAltitude - previousAltitude;

        if (deviation > DEVIATION_LIMIT || deviation < -DEVIATION_LIMIT) { printf("Deviation
            detected between Reading %d and Reading %d.\n", i, i + 1); printf("Previous
            Altitude: %d meters\n", previousAltitude);
            printf("Current Altitude: %d meters\n", currentAltitude);
            printf("Deviation: %d meters\n\n", deviation);
        }
    }

    return 0;
}
```

14. 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 ORBIT_RADIUS 1000

#define DEVIATION_LIMIT 50


int main() {

    int positions[READINGS]; int

    i;


    printf("Enter satellite position readings:\n");


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

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

        &positions[i]);

    }


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

        int deviation = positions[i] - ORBIT_RADIUS;

        if (deviation > DEVIATION_LIMIT || deviation < -DEVIATION_LIMIT) { printf("Significant

            deviation detected at Reading %d:\n", i + 1);
            printf("Position: %d km\n", positions[i]);
            printf("Deviation: %d km\n\n", deviation);
        }
    }



    return 0;
}
```

15. 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 SESSIONS 10


int main() {
    int heartrates[SESSIONS]; static
    int abnormalcount = 0; int sum =
    0;
    float avg;
    int i;
    printf("Enter heart rates for 10 sessions:\n"); for
    (i = 0; i < SESSIONS; i++) {
        printf("Session %d: ", i + 1);
        scanf("%d", &heartrates[i]);
        if (heartrates[i] < 60 || heartrates[i] > 100) {
            abnormalcount++;
        }

        sum += heartrates[i];
    }

    avg = (float)sum / SESSIONS;

    printf("\nAverage Heart Rate: %.2f BPM\n", avg);
    printf("Abnormal Count: %d\n", abnormalcount);

    return 0;
}
```

16. 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 PATIENTS 10
#define MIN_AGE 15
#define MAX_AGE 55
```

```
#define MIN_WEIGHT 45.0
#define MAX_WEIGHT 100.0

int main() { volatile
   int age;
   volatile float weight; int
   status[PATIENTS]; int i;

   printf("Enter age and weight of 10 patients:\n");
   for(i=1;i<=PATIENTS;i++){
       printf("Patient %d is:\n",i);
       printf("Age:");
       scanf("%d",&age);
       printf("Weight:");
       scanf("%f",&weight);

       if(age>=MIN_AGE && age<=MAX_AGE && weight>=MIN_WEIGHT &&
weight<=MAX_WEIGHT){
           printf("Safe Dosage\n");
       }
       else{
           printf("Not safe dosage\n");
       }
   }
   return 0;
}
```

17. 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 REORDER_THRESHOLD 20


int main() {

    int inventory[PRODUCTS]; static

    int belowThreshold = 0; int i;


    printf("Enter the current inventory levels for 10 products:\n");


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

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

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

    }


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


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

        if (inventory[i] < REORDER_THRESHOLD) {

            belowThreshold++;

            printf("Product %d: Low inventory (%d units). Suggest reorder.\n", i + 1,
inventory[i]);
```

```
        } else {
            printf("Product %d: Inventory sufficient (%d units).\n", i + 1, inventory[i]);
        }
    }

    printf("\nTotal products below reorder threshold: %d\n", belowThreshold);

    return 0;
}
```

18. 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 CODE_LENGTH 6

int main() {
    char codes[CODES][CODE_LENGTH + 1]; static
    int invalidCount = 0;
    int i, j;



    printf("Enter 10 missile launch codes (6 characters each):\n");
```

```
        for (i = 0; i < CODES; i++) {
          printf("Code %d: ", i + 1);
      scanf("%s", codes[i]);
  }



  for (i = 0; i < CODES; i++) {
      int valid = 1;


      for (j = 0; codes[i][j] != '\0'; j++) {}


      if (j != CODE_LENGTH) {
          valid = 0;
      }



      for (j = 0; codes[i][j] != '\0'; j++) {
          if (!((codes[i][j] >= '0' && codes[i][j] <= '9') ||
                (codes[i][j] >= 'A' && codes[i][j] <= 'Z') ||
                (codes[i][j] >= 'a' && codes[i][j] <= 'z'))) {
              valid = 0;
              break;
          }
      }



      if (valid) {
          printf("Code %d: Valid\n", i + 1);
      } else {
            printf("Code %d: Invalid\n", i + 1);

            invalidCount++;

        }

    }



    printf("\nTotal invalid codes: %d\n", invalidCount);


    return 0;

}
```

19. 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. has

context menu

```
#include <stdio.h>


#define TARGETS 10

#define BASE_POSITION 50

#define HOSTILE_THRESHOLD 30


int main() {

    int targets[TARGETS];

    register int i;

    register int hostile_count = 0;
    printf("Enter the x-coordinates of 10 targets:\n");


    for (i = 0; i < TARGETS; i++) {
        printf("Target %d: ", i + 1);
        scanf("%d", &targets[i]);
    }



    for (i = 0; i < TARGETS; i++) {
        int distance = targets[i] - BASE_POSITION;


        if (distance < -HOSTILE_THRESHOLD || distance > HOSTILE_THRESHOLD) {
            printf("Target %d (Position: %d) is Hostile\n", i + 1, targets[i]); hostile_count++;
        } else {
            printf("Target %d (Position: %d) is Friendly\n", i + 1, targets[i]);
        }
    }

    printf("\nTotal hostile targets: %d\n", hostile_count);


    return 0;
}
```

2D 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  // Number of rows in Matrices

#define N 3  // Number of columns in Matrices



int main() {

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

    int i, j;


    // Input Matrix A

    printf("Enter elements of Matrix A (%d x %d):\n", M, N); for

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

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

            printf("A[%d][%d]: ", i + 1, j + 1);

            scanf("%d", &A[i][j]);
        }
    }

    // Input Matrix B
    printf("Enter elements of Matrix B (%d x %d):\n", M,N); for (i
    = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            printf("B[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &B[i][j]);
        }
    }

    if (M != M || N != N) {
        printf("Matrix addition is not possible. Matrices must have the same dimensions.\n");
        return 1;
    }

    // Initialize result matrix to zero for
    (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            result[i][j] = 0;
        }
    }

    // Matrix addition logic
    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
                result[i][j] += A[i][j] + B[i][j];
        }
    }

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

    // Print Matrix B
    printf("\nMatrix B (%d x %d):\n", M,N); for
    (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            printf("%d ", B[i][j]);
        }
        printf("\n");
    }

    // Print the resulting matrix
    printf("\nResulting Matrix (A + B):\n"); for
    (i = 0; i < M; i++) {
        for (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  // Number of rows in the matrix

#define N 2  // Number of columns in the matrix


int main() {

    const int rows = M, cols = N;

    int matrix[M][N], transpose[N][M]; int

    i, j;


    // Input matrix elements

    printf("Enter elements of the matrix (%d x %d):\n", rows, cols); for (i =

    0; i < rows; i++) {

        for (j = 0; j < cols; j++) {
            printf("matrix[%d][%d]: ", i + 1, j + 1);

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

        }

    }


    if (rows <= 0 || cols <= 0) {

        printf("Invalid matrix size.\n");

        return 1;

    }


    // Compute the transpose

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

        for (j = 0; j < cols; j++) {

            transpose[j][i] = matrix[i][j];

        }

    }


    // Print the original matrix

    printf("\nOriginal Matrix (%d x %d):\n", rows, cols); for

    (i = 0; i < rows; i++) {

        for (j = 0; j < cols; j++) { printf("%d

            ", matrix[i][j]);

        }

        printf("\n");

    }


    // Print the transposed matrix

    printf("\nTransposed Matrix (%d x %d):\n", cols, rows); for (i

    = 0; i < cols; i++) {
        for (j = 0; j < rows; j++) { printf("%d
            ", transpose[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  // Number of rows in the matrix
#define N 4  // Number of columns in the matrix

int main() {
    const int rows = M, cols = N;
    int matrix[M][N], maxx[M];
    int i, j;
```

```
// Input matrix elements

printf("Enter elements of the matrix (%d x %d):\n", rows, cols); for (i =

0; i < rows; i++) {

    for (j = 0; j < cols; j++) {

        printf("matrix[%d][%d]: ", i + 1, j + 1);

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

    }

}


// Find the maximum element in each row

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

    int max = matrix[i][0];


    for (j = 1; j < cols; j++) { if

        (matrix[i][j] > max) {

            max = matrix[i][j];

        }

    }


    maxx[i] = max;

}


// Print the maximum values of each row

printf("\nMaximum element in each row:\n"); for

(i = 0; i < rows; i++) {

    printf("Max in Row %d: %d\n", i + 1, maxx[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>


#define M 2  // Number of rows in Matrix A

#define N 3  // Number of columns in Matrix A and rows in Matrix B #define

P 2  // Number of columns in Matrix B


int main() {
    int A[M][N], B[N][P], result[M][P];
    int i, j, k;


    // Input Matrix A
    printf("Enter elements of Matrix A (%d x %d):\n", M, N);
    for (i = 0; i < M; i++) { for
        (j = 0; j < N; j++) {
            printf("A[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &A[i][j]);
        }
    }

    // Input Matrix B
    printf("Enter elements of Matrix B (%d x %d):\n", N, P); for (i
    = 0; i < N; i++) {
        for (j = 0; j < P; j++) {
            printf("B[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &B[i][j]);
        }
    }

    // Check if multiplication is possible: N (columns of A) must equal M (rows of B) if (N !=
    M) {
        printf("Matrix multiplication is not possible. The number of columns of Matrix A must
be equal to the number of rows of Matrix B.\n");
        return 1;
    }

    // Initialize result matrix to zero for
    (i = 0; i < M; i++) {
        for (j = 0; j < P; j++) {
            result[i][j] = 0;
        }
    }
```

```
// Matrix multiplication logic
for (i = 0; i < M; i++) {
    for (j = 0; j < P; j++) {
        for (k = 0; k < N; k++) {
            result[i][j] += A[i][k] * B[k][j];  // Multiply and add to the result
        }
    }
}

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

// Print Matrix B
printf("\nMatrix B (%d x %d):\n", N, P); for (i
= 0; i < N; i++) {
    for (j = 0; j < P; j++) {
        printf("%d ", B[i][j]);
    }
    printf("\n");
}

// Print the resulting matrix
```

```
    printf("\nResulting Matrix (A * B):\n"); for
    (i = 0; i < M; i++) {
        for (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  // Number of rows in the matrix
#define N 4  // Number of columns in the matrix

int main() {
    const int rows = M, cols = N;
```

```
int matrix[M][N];
static int zeroCount = 0; int
i, j;

// Input matrix elements
printf("Enter elements of the matrix (%d x %d):\n", rows, cols); for (i =
0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        printf("matrix[%d][%d]: ", i + 1, j + 1);
        scanf("%d", &matrix[i][j]);
    }
}

// Count the number of zeros in the matrix for
(i = 0; i < rows; i++) {
     for (j = 0; j < cols; j++) { if
          (matrix[i][j] == 0) {
              zeroCount++;
        }
    }
}

int totalElements = rows * cols;

if (zeroCount > totalElements / 2) {
    printf("\nThe matrix is sparse.\n");
} else {
    printf("\nThe matrix is not sparse.\n");
}
```

```
    return 0;
}
```

3D Array

----

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 2  // Number of rows (depth) in the matrix

#define Y 2  // Number of rows in each matrix #define Z

3  // Number of columns in each matrix


int main() {

    const int rows = X, cols = Y, depth = Z;

    int matrix1[X][Y][Z], matrix2[X][Y][Z], result[X][Y][Z]; int i,

    j, k;
```

```
// Input matrix1 elements

printf("Enter elements for the first matrix (%d x %d x %d):\n", X, Y, Z); for (i =

0; i < rows; i++) {

    for (j = 0; j < cols; j++) {

        for (k = 0; k < depth; k++) {

            printf("matrix1[%d][%d][%d]: ", i + 1, j + 1, k + 1);

            scanf("%d", &matrix1[i][j][k]);

        }

    }

}


// Input matrix2 elements

printf("Enter elements for the second matrix (%d x %d x %d):\n", X, Y, Z); for (i = 0;

i < rows; i++) {

    for (j = 0; j < cols; j++) {

        for (k = 0; k < depth; k++) {

            printf("matrix2[%d][%d][%d]: ", i + 1, j + 1, k + 1);

            scanf("%d", &matrix2[i][j][k]);

        }

    }

}


// Check if both matrices have the same dimensions if

(rows != X || cols != Y || depth != Z) {

    printf("Error: Matrices have mismatched dimensions.\n"); return 1;

}
```

```
    // Perform element-wise addition of matrix1 and matrix2 for
    (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                result[i][j][k] = matrix1[i][j][k] + matrix2[i][j][k];
            }
        }
    }

    // Print the resulting matrix after addition
    printf("\nThe resulting matrix after addition is:\n"); for
    (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                    printf("result[%d][%d][%d] = %d\n", i + 1, j + 1, k + 1, result[i][j][k]);
            }
        }
    }

    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 2  // Number of rows (depth) in the matrix

#define Y 3  // Number of rows in each matrix #define Z

2  // Number of columns in each matrix


int main() {

    const int rows = X, cols = Y, depth = Z; int

    matrix[X][Y][Z];

    int max_value;

    int i, j, k;


    // Input elements of the matrix

    printf("Enter elements for the matrix (%d x %d x %d):\n", X, Y, Z); for (i =

    0; i < rows; i++) {

        for (j = 0; j < cols; j++) {

            for (k = 0; k < depth; k++) { printf("matrix[%d][%d][%d]:

                ", i + 1, j + 1, k + 1);

                scanf("%d", &matrix[i][j][k]);

            }

        }

    }


    max_value = matrix[0][0][0];
    // Traverse the matrix to find the maximum value for (i
    = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                if (matrix[i][j][k] > max_value) {
                    max_value = matrix[i][j][k];
                }
            }
        }
    }

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

    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  // Number of rows (depth) in the matrix

#define Y 3  // Number of rows in each matrix #define Z

2  // Number of columns in each matrix


int main() {

    const int rows = X, cols = Y, depth = Z; int

    matrix[X][Y][Z];

    int result[X][Y][Z];

    int scalar;

    int i, j, k;


    // Input elements of the matrix

    printf("Enter elements for the matrix (%d x %d x %d):\n", X, Y, Z); for (i =

    0; i < rows; i++) {

        for (j = 0; j < cols; j++) {

            for (k = 0; k < depth; k++) { printf("matrix[%d][%d][%d]:

                ", i + 1, j + 1, k + 1);

                scanf("%d", &matrix[i][j][k]);

            }

        }

    }


    printf("Enter a scalar value: ");

    scanf("%d", &scalar);


    // Check for invalid scalar if

    (scalar <= 0) {

        printf("Invalid scalar value! Scalar must be a positive integer.\n"); return 1;
```

```
}

// Print the original matrix
printf("\nOriginal Matrix:\n");
for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { printf("%d ",
            matrix[i][j][k]);
        }
        printf("\n");
    }
    printf("\n");
}

// Perform scalar multiplication for
(i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { result[i][j][k] =
            matrix[i][j][k] * scalar;
        }
    }
}

// Print the resulting matrix after scalar multiplication printf("\nResulting
Matrix after Scalar Multiplication by %d:\n", scalar); for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { printf("%d ",
            result[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    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 2

#define Y 3

#define Z 2


int main() {
    const int rows = X, cols = Y, depth = Z; int
    matrix[X][Y][Z];
```

```
static int positive_count = 0, negative_count = 0, zero_count = 0; int i, j,
k;

printf("Enter elements for the matrix (%d x %d x %d):\n", X, Y, Z); for (i =
0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { printf("matrix[%d][%d][%d]:
            ", i + 1, j + 1, k + 1);
            scanf("%d", &matrix[i][j][k]);
        }
    }
}

for (i = 0; i < rows; i++) { for
    (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { if
            (matrix[i][j][k] > 0) {
                positive_count++;
            } else if (matrix[i][j][k] < 0) {
                negative_count++;
            } else {
                zero_count++;
            }
        }
    }
}

printf("\nPositive count: %d\n", positive_count);
printf("Negative count: %d\n", negative_count);
```

```
    printf("Zero count: %d\n", zero_count);


    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 2

#define Y 3

#define Z 2


int main() {
    const int rows = X, cols = Y, depth = Z; int
    matrix[X][Y][Z];
    static int transposed[X][Y][Z]; int
    i, j, k, axis;


    printf("Enter elements for the matrix (%d x %d x %d):\n", X, Y, Z);
```

```
for (i = 0; i < rows; i++) { for
    (j = 0; j < cols; j++) {
        for (k = 0; k < depth; k++) { printf("matrix[%d][%d][%d]:
            ", i + 1, j + 1, k + 1);
            scanf("%d", &matrix[i][j][k]);
        }
    }
}

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

if (axis == 1) {
    for (i = 0; i < rows; i++) { for
        (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) { transposed[i][j][k] =
                matrix[j][i][k];
            }
        }
    }
} else if (axis == 2) {
    for (i = 0; i < rows; i++) { for
        (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                transposed[i][j][k] = matrix[i][k][j];
            }
        }
    }
} else {
        printf("Invalid axis value.\n");
        return 0;
    }

    printf("Original Matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                    printf("matrix[%d][%d][%d] = %d\n", i + 1, j + 1, k + 1, matrix[i][j][k]);
            }
        }
    }
    printf("Transposed Matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < depth; k++) {
                printf("transposed[%d][%d][%d] = %d\n", i + 1, j + 1, k + 1, transposed[i][j][k]);
            }
        }
    }

    return 0;
}
```