Array- Questions: 27-12-2024

Q1 Print prime numbers between 1 and 100:

```
#include <stdio.h>

int main() {

   int prime[100]={[0]=2,[1]=3};

   int count = 2;

   int i, j;


   for (i = 4; i <= 100; i++) {


      for (j = 2; j <= i / 2; j++) {

         if (i % j == 0) {

            break;

         }

      }

      if (j>i/2) {

         prime[count++] = i;

      }

   }

   printf("Prime numbers between 2 and 100:\n");

   for (i = 0; i < count; 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>

int main() {

    int const arr[10] = {12, 5, 8,23,45,3,87,30,20, 2};

static int maxDifference = 0;

    int 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];

        }

    }

    maxDifference = max - min;

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

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

    printf("Maximum Difference (Max - Min): %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>

int main() {

    static int max_size=10;

    int const array[10] = {10, -5, 0, 34, -15, 0, 7, -2, 0, 3};

    int positive[max_size], negative[max_size], zero[max_size];

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

    for (int i = 0; i < 10; i++) {

        if (array[i] > 0) {

            positive[posCount++] = array[i];

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

            negative[negCount++] = array[i];

        } else {

            zero[zeroCount++] = array[i];

        }

    }

    printf("Positive numbers: ");

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

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

    }

    printf("\n");



    printf("Negative numbers: ");

    for (int i = 0; i < negCount; i++) {
        printf("%d ", negative[i]);

    }

    printf("\n");

    printf("Zeros: ");

    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 5

int main()

{

    int const a[SIZE] = {2, 3, 4, 5, 6};

    int static sum_total = 0;

    for (int i = 0; i < SIZE; i++)

    {

        sum_total += a[i];

    }
    printf("Sum of elements of array = %d\n", sum_total);

    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>

int isPrime(int num);

int main()

{

    int size=5;

    int const arr[size];

    printf("Enter array elements\n");

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

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

    }

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

        if (isPrime(arr[i])) {

            printf("%d is prime.\n", arr[i]);

        } else {

            printf("%d is not prime.\n", arr[i]);

        }

    }

    return 0;
}

int isPrime(int num) {

    if (num <= 1) {

        return 0;

    }

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

        if (num % i == 0) {

            return 0;

        }

    }

    return 1;

}
```

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 5

#define N 2

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

    int temp[size]; // Static array to store rotated values

    int i, j = 0;

    // Perform the rotation using a while loop
    while (j < size) {
        temp[j] = arr[(j + n) % size]; // Rotate the elements to the left
        j++;
    }
    // Copy the rotated values back to the original array
    for (i = 0; i < size; i++) {
        arr[i] = temp[i];
    }
}
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");
    rotateArray(arr, SIZE, N);
    // Print the rotated array
    printf("Array after rotating by %d positions: ", 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 arr[SIZE] = {1, 2, 3, 2, 1, 5, 1, 5, 2, 3};

    int freq[SIZE]; // Array to store frequency counts

    int i, j;

    // Initialize the frequency array to -1

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

        freq[i] = -1;

    }

    // Count frequencies using nested loops

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

        int count = 1; // Initialize count for the current element

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

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

                count++;          // Increment count for matching elements

                freq[j] = 0;      // Mark as counted

            }

        }

        if (freq[i] != 0) {        // If not already counted
```

```
            freq[i] = count;
        }
    }
    printf("Element\tFrequency\n");
    for (i = 0; i < SIZE; i++) {
        if (freq[i] != 0) {
            printf("%d\t%d\n", arr[i], freq[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
int main() {
    int arr[SIZE] = {12, 7, 15, 3, 10};
    int i, j, temp;
    printf("Original array:\n");
    for (i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    // Bubble sort

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

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

            if (arr[j] < arr[j + 1]) {



                temp = arr[j];

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

                arr[j + 1] = temp;

            }

        }

    }

    printf("Array sorted in descending order:\n");

    for (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>

// #include <limits.h> for finding the minimum elemnt in array INT_MIN
```

```
#define SIZE 5

int main() {

    const int arr[SIZE] = {0, 20, 90, -5, 15};

    int largest = arr[0];

    int second_largest = arr[1];

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

        if (arr[i] > largest) {

            second_largest = largest;

            largest = arr[i];

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

            second_largest = arr[i];

        }

    }

    printf("Largest element: %d\n", largest);

    printf("Second largest element: %d\n", second_largest);

 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() {
    const int arr[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int odd[SIZE], even[SIZE];
    int odd_count = 0, even_count = 0;
    for (int i = 0; i < SIZE; i++) {
        if (arr[i] % 2 == 0) {
            even[even_count++] = arr[i];//After even[0]=arr[0], counter is incremented by 1
        } else {
            odd[odd_count++] = arr[i];
        }
    }
    printf("Even numbers:\n");
    for (int i = 0; i < even_count; i++) {
        printf("%d ", even[i]);
    }
    printf("\n");
    printf("Odd numbers:\n");
    for (int i = 0; i < odd_count; i++) {
        printf("%d ", odd[i]);
    }
    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

int main() {

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

    int temp;

    temp = arr[SIZE - 1];

    for (int i = SIZE - 1; i > 0; i--) {

        arr[i] = arr[i - 1];

    }

    arr[0] = temp;

    printf("Array after cyclic shift:\n");

    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.

```
#include <stdio.h>

#define size 10

int max_threshold = 50;
int main() {
   float temperature_reading[size];
   static float maximum_temp = 0;
   int count=1;
   for(int i=0;i<size;i++){
      printf("Temperature %d reading: ",count);
      scanf("%f",&temperature_reading[i]);
      if (temperature_reading[i] > max_threshold) {
         printf("Warning: Temperature exceeds maximum threshold\n");
         maximum_temp = temperature_reading[i];
      } else {
         printf("Temperature: %f\n", temperature_reading[i]);
      }
      count++;
   }
   printf("Maximum Temperature Recorded: %f\n", maximum_temp);
  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 NUM_TRIPS 10

#define FUEL_CONSUMPTION_RATE 10


int main() {

    volatile float distances[NUM_TRIPS];

    float fuel_efficiency;

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

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

        printf("Distance for trip %d: ", i + 1);

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


        fuel_efficiency = distances[i] / FUEL_CONSUMPTION_RATE;


        printf("Fuel Efficiency for trip %d: %.2f km/L\n", i + 1, fuel_efficiency);

        if (fuel_efficiency < 10) {

            printf("Warning: Fuel efficiency is below 10 km/L for trip %d!\n", i + 1);

        }

    }

    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 NUM_READINGS 10
#define THRESHOLD 50
int main() {
    int altitude[NUM_READINGS];
    register int current_altitude;
    printf("Enter altitude readings for 10 seconds:\n");
    for (int i = 0; i < NUM_READINGS; i++) {
        printf("Altitude at second %d: ", i + 1);
        scanf("%d", &altitude[i]);
        int deviation = altitude[i] - altitude[i +1];
        if (deviation > THRESHOLD) {
            printf("Warning: Altitude increased by more than 50 meters at second %d\n", i + 1);
        } else if (deviation < -THRESHOLD) {
            printf("Warning: Altitude decreased by more than 50 meters at second %d\n", i + 1);
        }
    }
    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.

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

int main() {

    int heart_rates[NUM_SESSIONS];

    static int abnormal_count = 0;

    int total_heart_rate = 0;

    float average_heart_rate;

    printf("Enter the heart rate readings for %d sessions:\n", NUM_SESSIONS);

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

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

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

        if (heart_rates[i] < 60 || heart_rates[i] > 100) {

            abnormal_count++;

        }

        total_heart_rate += heart_rates[i];

    }


    average_heart_rate = total_heart_rate / NUM_SESSIONS;

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

    printf("Total sessions: %d\n", NUM_SESSIONS);
```

```
          printf("Abnormal readings (below 60 or above 100 BPM): %d\n", abnormal_count);
          printf("Average heart rate: %.2f BPM\n", average_heart_rate);

          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
#define SAFE_DOSAGE_MIN 10.0
#define SAFE_DOSAGE_MAX 50.0


float calculateDosage(float weight, int age) {
    if (age > 12) {
        return weight * 0.8;
    }
}


int main() {
    volatile float weights[NUM_PATIENTS];
    volatile int ages[NUM_PATIENTS];
    int results[NUM_PATIENTS];
    printf("Enter weight (kg) and age (years) for %d patients:\n", NUM_PATIENTS);

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

        printf("Patient %d:\n", i + 1);

        printf("  Weight (kg): ");

        scanf("%f", (float*)&weights[i]);

        printf("  Age (years): ");

        scanf("%d", (int*)&ages[i]);

         float dosage = calculateDosage(weights[i], ages[i]);

         if (dosage >= SAFE_DOSAGE_MIN && dosage <= SAFE_DOSAGE_MAX) {

            results[i] = 1;

        } else {

            results[i] = 0;

        }

         printf("Patient %d: Dosage is %s\n",

               i + 1,

               results[i] ? "Valid" : "Invalid");

    }

    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
#define REORDER_THRESHOLD 20
int main() {
    int inventory[NUM_PRODUCTS];
    static int below_threshold_count = 0;
    printf("Enter 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]);
        if (inventory[i] < REORDER_THRESHOLD) {
            below_threshold_count++;
        }
    }
    printf("\nInventory Status:\n");
    for (int i = 0; i < NUM_PRODUCTS; i++) {
        printf("Product %d: %d units", i + 1, inventory[i]);
        if (inventory[i] < REORDER_THRESHOLD) {
            printf(" (Reorder suggested!)");
        }
        printf("\n");
    }

    printf("Total products below reorder threshold: %d\n", below_threshold_count);
```

```
        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.

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 MAX_TARGETS 10

#define BASE_POSITION 0

#define FRIENDLY_THRESHOLD 50

int main() {

    int target_positions[MAX_TARGETS];

    char F_H;

    register int friendly_threshold = FRIENDLY_THRESHOLD;

    int friendly_count = 0, hostile_count = 0;

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

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

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

        scanf("%d", &target_positions[i]);
    }
    for (int i = 0; i < MAX_TARGETS; i++) {
        int distance = target_positions[i] - BASE_POSITION;
        if (distance < 0) {
            distance = -distance;
        }
        if (distance <= friendly_threshold) {
            target_classifications = 'F';
            friendly_count++;
        } else {
            target_classifications= 'H';
            hostile_count++;
        }
    }
    printf("\nTarget Classifications:\n");
    for (int i = 0; i < MAX_TARGETS; i++) {
        printf("Target %d : Position: %d\n",
               i + 1, target_positions[i]);
    }
    printf("Friendly Targets: %d\n", friendly_count);
    printf("Hostile Targets: %d\n", hostile_count);
    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 2

#define N 3

int main()

{

    int a[M][N];

    int b[M][N];

    int sum[M][N];

    printf("Enter array elements of Matrix A\n");

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

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

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

        }

    }

    printf("Enter array elements of Matrix B\n");

for(int i=0;i<M;i++){
        for(int j=0;j<N;j++){
            scanf("%d",&b[i][j]);
        }
    }
    for(int i=0;i<M;i++){
        for(int j=0;j<N;j++){
            sum[i][j]=a[i][j]+b[i][j];
        }
    }
    for(int i=0;i<M;i++){
        for(int j=0;j<N;j++){
          printf("A+B=%d\n",sum[i][j]);
        }
    }
    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 2

int main(){

    int A[M][N];

    int transpose_A[N][M];

    printf("Enter array A\n");

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

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

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

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

        }

    }


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

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

            transpose_A[j][i]=A[i][j];

        }

    }

    printf("Original matrix\n");

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

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

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

        }

        printf("\n");

    }
    printf("Transpose of A\n");
       for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            printf("%d  ",transpose_A[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 2
int main() {
    const int matrix[M][N];
    int static max_row[M];
    printf("Enter the elements of the matrix (%d x %d):\n", M, N);
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("Enter element [%d][%d]: ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d  ",matrix[i][j]);
        }
        printf("\n");
    }
    for(int i=0;i<M;i++){
        int maxVal=matrix[i][0];
        for(int j=0;j<N;j++){
            if(matrix[i][j]>maxVal)
            maxVal=matrix[i][j];
        }
        max_row[i]=maxVal;
    }
printf("Maximum elemnts in each row:\n");
for(int i=0;i<M;i++){
    printf("%d\n",max_row[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() {

   int const M1=1,N1=2,M2=2,N2=3;

   int A[M1][N1];

   int B[M2][N2];

   int result[M1][N2];

   printf("Enter the elements of the matrix A (%d x %d):\n", M1, N1);

   for (int i = 0; i < M1; i++) {

       for (int j = 0; j < N1; j++) {

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

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

       }

   }

    printf("Enter the elements of the matrix B (%d x %d):\n", M2, N2);

   for (int i = 0; i < M2; i++) {
       for (int j = 0; j < N2; j++) {
           printf("Enter element [%d][%d]: ", i + 1, j + 1);
           scanf("%d", &B[i][j]);
       }
   }
   if(N1==M2)
   {
       for(int i=0;i<M1;i++){
           for(int j=0;j<N2;j++){
               for(int k=0;k<N2;k++){
                   result[i][j]+=A[i][k]*B[k][j];
               }
       }
       }
   for (int i = 0; i < M1; i++) {
       for (int j = 0; j < N2; j++) {
           printf("%d  ",result[i][j]);
       }
       printf("\n");
       }
   }
   // else if(N1!=M2)
   // printf("Invalid Dimensions\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>

int main() {
    const int M = 3, N = 3;
    int matrix[M][N];
    int zero_count = 0;


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


    int total_elements = M * N;
    if (zero_count > total_elements / 2) {

        printf("\nThe matrix is sparse.\n");

    } else {

        printf("\nThe matrix is not sparse.\n");

    }

    printf("Number of zeros in the matrix: %d\n", zero_count);



    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>



int main() {

    const int X = 2, Y = 2, Z = 2;

    int A[X][Y][Z], B[X][Y][Z], result[X][Y][Z];



    printf("Enter the elements of matrix A (%d x %d x %d):\n", 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("Enter element A[%d][%d][%d]: ", i + 1, j + 1, k + 1);
            scanf("%d", &A[i][j][k]);
        }
    }
}

printf("Enter the elements of matrix B (%d x %d x %d):\n", 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("Enter element B[%d][%d][%d]: ", i + 1, j + 1, k + 1);
            scanf("%d", &B[i][j][k]);
        }
    }
}


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] = A[i][j][k] + B[i][j][k];
        }
        }

    }



    printf("\nResulting matrix after addition:\n");

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

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

            for (int k = 0; k < Z; 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>



int main() {

    const int X = 2,Y = 2,Z=3;
    int matrix[X][Y][Z];

    int static maxValue=0;



    printf("Enter the elements of the matrix (%d x %d x %d ):\n", 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("Enter element [%d][%d][%d]\n",i+1,j+1,k+1);

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

                if(matrix[i][j][k]>maxValue){

                   maxValue=matrix[i][j][k];

                }

                }



            }

        }



    printf("Maximum array value =%d\n",maxValue);



    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>

int main() {

    const int X = 2,Y = 2,Z=3;

    int matrix[X][Y][Z];

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

    int scalar;

    printf("Enter the elements of the matrix (%d x %d x %d ):\n", 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("Enter element [%d][%d][%d]\n",i+1,j+1,k+1);

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

                }

            }

        }

    printf("Enter Scalar: ");

    scanf("%d",&scalar);

    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]=scalar*matrix[i][j][k];

                }

        }
    }

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

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

            for(int k=0;k<Z;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>



int main() {

    const int X = 2,Y = 2,Z=3;
```

```
   int matrix[X][Y][Z];

   int static maxValue=0;

   int count_zero=0,count_positive=0,count_negative=0;

   printf("Enter the elements of the matrix (%d x %d x %d ):\n", 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("Enter element [%d][%d][%d]\n",i+1,j+1,k+1);

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

               if(matrix[i][j][k]>0){

                   count_positive++;


               }

               else if(matrix[i][j][k]==0)

                   count_zero++;

               else

                   count_negative++;


               }

           }

       }


   printf("Positive elements=%d\n",count_positive);

printf("Zeros=%d\n",count_zero);

 printf("Negative elements=%d\n",count_negative);
```

```
    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>


int main() {
    const int X = 2, Y = 3, Z = 2;
    int matrix[X][Y][Z];
    int transposedMatrix[X][Z][Y];
    int axis;


    printf("Enter the elements of the 3D matrix (%d x %d x %d):\n", 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("Enter element [%d][%d][%d]: ", i + 1, j + 1, k + 1);
                scanf("%d", &matrix[i][j][k]);
            }
        }
    }

    printf("Choose the axis for transposition:\n");
    printf("1: Transpose rows and columns (swap Y and Z)\n");
    printf("Enter your choice: ");
    scanf("%d", &axis);

    if (axis != 1) {
        printf("Invalid axis choice. Only '1' (rows and columns) is supported in this program.\n");
        return 1;
    }


    if (axis == 1) {
        for (int i = 0; i < X; i++) {
            for (int j = 0; j < Y; j++) {
                for (int k = 0; k < Z; k++) {
                    transposedMatrix[i][k][j] = matrix[i][j][k];
                }
            }
        }
    }

    printf("\nOriginal Matrix:\n");
    for (int i = 0; i < X; i++) {
        printf("Depth %d:\n", i + 1);
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("%d ", matrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    printf("Transposed Matrix (rows and columns swapped):\n");
    for (int i = 0; i < X; i++) {
        printf("Depth %d:\n", i + 1);
        for (int j = 0; j < Z; j++) {
            for (int k = 0; k < Y; k++) {
                printf("%d ", transposedMatrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

Weekend Task

1. Temperature Data Logger (2D Array)

- **Problem Statement**: Design a program to log temperature readings from multiple sensors for 24 hours, sampled every hour.
- **Requirements**:
    - Use a 2D array of size [N][24] to store temperature data, where N is the number of sensors (defined as a const variable).
    - Use static variables to calculate and store the daily average temperature for each sensor.
    - Use nested for loops to populate and analyze the array.
    - Use if statements to identify sensors exceeding a critical threshold temperature.

2. LED Matrix Control (2D Array)

- **Problem Statement**: Simulate the control of an LED matrix of size 8x8. Each cell in the matrix can be ON (1) or OFF (0).
- **Requirements**:
    - Use a 2D array to represent the LED matrix.
    - Use static variables to count the number of ON LEDs.
    - Use nested for loops to toggle the state of specific LEDs based on input commands.
    - Use if statements to validate commands (e.g., row and column indices).

3. Robot Path Mapping (2D Array)

- **Problem Statement**: Track the movement of a robot on a grid of size M x N.
- **Requirements**:
    - Use a 2D array to store visited positions (1 for visited, 0 otherwise).
    - Declare grid dimensions using const variables.
    - Use a while loop to update the robot’s position based on input directions (e.g., UP, DOWN, LEFT, RIGHT).
    - Use if statements to ensure the robot stays within bounds.

4. Sensor Data Aggregation (3D Array)

- **Problem Statement**: Store and analyze data from multiple sensors placed in a 3D grid (e.g., environmental sensors in a greenhouse).
- **Requirements**:
    - Use a 3D array of size [X][Y][Z] to store data, where dimensions are defined using const variables.

    - Use nested for loops to populate the array with sensor readings.
    - Use if statements to find and count sensors reporting critical values (e.g., temperature > 50°C).
    - Use static variables to store aggregated results (e.g., average readings per layer).

5. Image Processing (2D Array)

- **Problem Statement**: Perform edge detection on a grayscale image represented as a 2D array.
- **Requirements**:
    - Use a 2D array of size [H][W] to store pixel intensity values (defined using const variables).
    - Use nested for loops to apply a basic filter (e.g., Sobel filter) on the matrix.
    - Use decision-making statements to identify and highlight edge pixels (threshold-based).
    - Store the output image in a static 2D array.

6. Traffic Light Controller (State Management with 2D Array)

- **Problem Statement**: Manage the states of traffic lights at an intersection with four roads, each having three lights (red, yellow, green).
- **Requirements**:
    - Use a 2D array of size [4][3] to store the state of each light (1 for ON, 0 for OFF).
    - Use nested for loops to toggle light states based on time intervals.
    - Use static variables to keep track of the current state cycle.
    - Use if statements to validate light transitions (e.g., green should not overlap with red).

7. 3D LED Cube Animation (3D Array)

- **Problem Statement**: Simulate an animation on an LED cube of size 4x4x4.
- **Requirements**:
    - Use a 3D array to represent the LED cube's state.
    - Use nested for loops to turn ON/OFF LEDs in a predefined pattern.
    - Use static variables to store animation progress and frame counters.
    - Use if-else statements to create transitions between animation frames.

8. Warehouse Inventory Tracking (3D Array)

- **Problem Statement**: Track inventory levels for multiple products stored in a 3D warehouse (e.g., rows, columns, and levels).
- **Requirements**:
  - Use a 3D array of size [P][R][C] to represent the inventory of P products in a grid.
  - Use nested for loops to update inventory levels based on shipments.
  - Use if statements to detect low-stock levels in any location.
  - Use a static variable to store total inventory counts for each product.

9. Signal Processing on a 3D Matrix

- **Problem Statement**: Apply a basic signal filter to a 3D matrix representing sampled signals over time.
- **Requirements**:
  - Use a 3D array of size [X][Y][Z] to store signal data.
  - Use nested for loops to apply a filter that smoothens the signal values.
  - Use if statements to handle boundary conditions while processing the matrix.
  - Store the filtered results in a static 3D array.

10. Weather Data Analysis (3D Array)

- **Problem Statement**: Analyze weather data recorded over multiple locations and days, with hourly samples for each day.
- **Requirements**:
  - Use a 3D array of size [D][L][H] to store temperature readings (D days, L locations, H hours per day).
  - Use nested for loops to calculate the average daily temperature for each location.
  - Use if statements to find the location and day with the highest temperature.
  - Use static variables to store results for each location.