```
#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;
}
```
Same value for a and a[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));
  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;
}
```
values

=----------

```
#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(" a[%d] = %d \n",i,a[i]);
  }
    return 0;
}
```
Array out of bond

```
#include <stdio.h>

int main()
{
  int a[10];
  printf("%d \n",a[14]);

    return 0;
}
```

CALCULATING AVERAGE OF 10 GRADES USING ARRAY

```
#include <stdio.h>

int main()
{
  int grades[10];
  int count=10;
  long sum=0;
  float average=0.0f;
  printf("enter the 10 grades:\n");
  for(int i=0;i<count;++i){
   printf("%2u>",i+1);
   scanf("%d",&grades[i]);
   sum+=grades[i];
  }
  average=(float)sum/count;
  printf("\nAverage of 10 grades is:%2f \n",average);


    return 0;
}


#include <stdio.h>

int main()
{
  int a[10] = {10,20};

  for(int i=0;i<=10;++i){
    printf("enter the values are: %d \n",a[i]);
}

    return 0;
}
```

DESIGNATED

```
#include <stdio.h>

int main()
{
  int a[10] = {10,20,[7]=1,[9]=50};

  for(int i=0;i<=10;++i){
    printf("enter the values are: %d \n",a[i]);
}

    return 0;
}
```

Questsio---------

```
#include <stdio.h>
```

```
#define months 12
int main()
{
  int days[months]={31,28,31,30,31,30,31,31,30,31,30,31};
  int index;

  for(int index=0;index<months;++index){
    printf("Month %d has %2d days.\n",index+1,days[index]);
}

    return 0;
}
```

designated same question

```
#include <stdio.h>
#define months 12
int main()
{
  int days[months]={31,28,[4]=31,30,31,[1]=29};
  int index;

  for(int index=0;index<months;++index){
    printf("Month %d has %2d days.\n",index+1,days[index]);
}

    return 0;
}
```

PROBLEM STATEMENTS

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 10

int main() {
    int a[size] = {12, 45, 3, 7, 19, 31, 25, 60, 2, 9};
    int max, min;
    static int maxDiff = 0;

    max = min = a[0];

    for (int i = 1; i < size; i++) {
        if (a[i] > max) {
            max = a[i];
        }
        if (a[i] < min) {
            min = a[i];         }
    }
```

```
    int diff = max - min;

    if (diff > maxDiff) {
        maxDiff = diff;
    }

    printf("Maximum value: %d\n", max);
    printf("Minimum value: %d\n", min);
    printf("Maximum difference: %d\n", maxDiff);

    return 0;
}
```

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 10

int main() {
    int a[size] = {5, -3, 0, 12, -7, 0, 8, -2, 0, 6};

    int p[size], n[size], z[size];
    int posIndex = 0, negIndex = 0, zeroIndex = 0;
    for (int i = 0; i < size; i++) {
        if (a[i] > 0) {
            p[posIndex++] = a[i];
        } else if (a[i] < 0) {
            n[negIndex++] = a[i];
        } else {
            z[zeroIndex++] = a[i];
        }
    }

    printf("Positive values: ");
    for (int i = 0; i < posIndex; i++) {
        printf("%d ", p[i]);
    }
    printf("\n");

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

    printf("Zero values: ");
    for (int i = 0; i < zeroIndex; i++) {
        printf("%d ", z[i]);
    }
    printf("\n");
```

```
    return 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>

#define SIZE 5

int main() {
    int a[SIZE] = {1, 2, 3, 4, 5};


    static int cumulativeSum = 0;

    for (int i = 0; i < SIZE; i++) {
        cumulativeSum += a[i];
        printf("Cumulative sum after element %d: %d\n", i + 1, cumulativeSum);
    }

    return 0;
}
```

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>

#define SIZE 10

int isPrime(int n) {
    if (n <= 1) {
        return 0;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return 0;
        }
    }
    return 1;
}

int main() {

    int a[SIZE] = {11, 20, 5, 8, 7, 4, 13, 17, 18, 10};

    printf("Prime numbers in the array are:\n");

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

```
        }
    }

    return 0;
}
```

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 7
const int N = 3;

void rotateLeft(int arr[], int size, int n) {
    static int rotatedArray[SIZE];

    int i = 0;

    while (i < size) {
        rotatedArray[i] = arr[(i + n) % size];
        i++;
    }

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

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

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

    printf("Original array:\n");
    printArray(arr, SIZE);

    rotateLeft(arr, SIZE, N);

    printf("Array after rotating left by %d positions:\n", N);
    printArray(arr, SIZE);

    return 0;
}
```

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 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 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 10
int main() {
    int b[SIZE] = {23, 45, 12, 56, 78, 89, 34, 90, 56, 67};

    for (int i = 0; i < SIZE - 1; i++) {
        for (int j = 0; j < SIZE - 1 - i; j++) {
            if (b[j] < b[j + 1]) {
                int temp = b[j];
                b[j] = b[j + 1];
                b[j + 1] = temp;
            }
        }
    }
```

```
    printf("Sorted array in descending order:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", b[i]);
    }

    return 0;
}
```

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 10

int main() {
    int a[SIZE] = {10, 20, 4, 45, 99, 101, 150, 80, 60, 120};
    static int secondLargest = -1;
    int largest = -1;

    if (SIZE < 2) {
        printf("Array must have at least two elements to find the second largest.\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.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 a[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int e[SIZE], o[SIZE];
    int eCount = 0, oCount = 0;

    for (int i = 0; i < SIZE; i++) {
        if (a[i] % 2 == 0) {
            e[eCount++] = a[i];
        } else {
            o[oCount++] = a[i];
```

```
        }
    }

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

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

    return 0;
}
```

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

void cyclicShiftRight(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");

    cyclicShiftRight(arr);

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

```
    return 0;
}
```
SECOND SET OF PROGRAMS

1.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>

const int MAX_SAFE_TEMP = 100;

extern int engineTemperatures[10];

void inputTemperatures(void);

void monitorTemperatures(void);

void monitorCounter(void);

int main() {
    inputTemperatures();

    monitorTemperatures();

    monitorCounter();

    return 0;
}

int engineTemperatures[10] = {0};

void inputTemperatures(void) {
    for (int i = 0; i < 10; i++) {
        printf("Enter temperature reading for time %d (in degrees Celsius): ", i + 1);
        scanf("%d", &engineTemperatures[i]);
    }
}

void monitorTemperatures(void) {
    static int counter = 0;

    for (int i = 0; i < 10; i++) {
        if (engineTemperatures[i] > MAX_SAFE_TEMP) {
            printf("Alert: Temperature at time %d (%.2d°C) exceeds safe threshold!\n", i + 1,
engineTemperatures[i]);
            counter++;
        }
    }

    if (counter == 0) {
        printf("All temperatures are within safe limits.\n");
    }
```

```
}

void monitorCounter(void) {
    static int checkedReadings = 0;
    checkedReadings++;
    printf("Total number of temperature readings checked: %d\n", checkedReadings);
}
```

2.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 N_TRIPS 10
#define FUEL_CONSUMPTION 60

float cal_eff(float distance) {
    return distance / FUEL_CONSUMPTION;
}

int main() {
    float distances[N_TRIPS];
    volatile float sensor_input;
    float eff;
    int i;

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


    for (i = 0; i < N_TRIPS; i++) {
        printf("Enter distance for trip %d: ", i + 1);

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

        eff = cal_eff(distances[i]);

        printf("Trip %d - Distance: %.2f km, Fuel Efficiency: %.2f km/L\n", i + 1, distances[i], eff);

        if (eff < 10.0) {
            printf("Warning: Low fuel efficiency detected for trip %d!\n", i + 1);
        }
    }

    return 0;
}
```

3.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 N_READINGS 10
#define DEV_THRESHOLD 50

int main() {
    int aReadings[N_READINGS];

    register int currentAltitude;

    printf("Enter altitude readings for 10 seconds:\n");
    for (int i = 0; i < N_READINGS; i++) {
        printf("Reading %d: ", i + 1);
        scanf("%d", &aReadings[i]);
    }

    for (int i = 1; i < N_READINGS; i++) {
        currentAltitude = aReadings[i];
        int difference = aReadings[i] - aReadings[i - 1];

        if (difference > DEV_THRESHOLD || difference < -DEV_THRESHOLD) {
            printf("Warning: Altitude deviation of more than %d meters detected between reading %d and
%d.\n",
                    DEV_THRESHOLD, i, i + 1);
        }
    }

    return 0;
}
```

4.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 R 1000.0
#define DEV_LIMIT 50.0
#define N_READINGS 10

double cal_deviation(double position) {
    if (position < R) {
        return R - position;
    } else {
        return position - R;
    }
}

int main() {
    double positions[N_READINGS];
    double deviation;
    int i;
```

```
    printf("Enter the positions of the satellite (in km) for %d readings:\n", N_READINGS);
    for (i = 0; i < N_READINGS; i++) {
        printf("Reading %d: ", i + 1);
        scanf("%lf", &positions[i]);
    }

    for (i = 0; i < N_READINGS; i++) {
        deviation = cal_deviation(positions[i]);

        if (deviation > DEV_LIMIT) {
            printf("Warning: Deviation for reading %d (%.2f km) exceeds limit! Deviation = %.2f km\n",
                   i + 1, positions[i], deviation);
        } else {
            printf("Reading %d (%.2f km) is within acceptable limits. Deviation = %.2f km\n",
                   i + 1, positions[i], deviation);
        }
    }

    return 0;
}
```

5.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 N_SESSIONS 10

void recordHeartRates(int heartRates[]);
void analyzeHeartRates(int heartRates[], int *abnormalCount, int *belowCount, int *aboveCount, double
*average);

int main() {
    int heartRates[N_SESSIONS];
    int abnormalCount = 0, belowCount = 0, aboveCount = 0;
    double average = 0.0;

    recordHeartRates(heartRates);

    analyzeHeartRates(heartRates, &abnormalCount, &belowCount, &aboveCount, &average);

    printf("\nHeart Rate Analysis:\n");
    printf("Total sessions: %d\n", N_SESSIONS);
    printf("Abnormal readings (below 60 or above 100): %d\n", abnormalCount);
    printf("Readings below 60 BPM: %d\n", belowCount);
    printf("Readings above 100 BPM: %d\n", aboveCount);
    printf("Average heart rate: %.2f BPM\n", average);

    return 0;
}

void recordHeartRates(int heartRates[]) {
    printf("Enter the heart rates for %d sessions:\n", N_SESSIONS);
    for (int i = 0; i < N_SESSIONS; i++) {
```

```
        printf("Session %d: ", i + 1);
        scanf("%d", &heartRates[i]);
    }
}

void analyzeHeartRates(int heartRates[], int *abnormalCount, int *belowCount, int *aboveCount, double
*average) {
    int sum = 0;

    for (int i = 0; i < N_SESSIONS; i++) {
        sum += heartRates[i];

        if (heartRates[i] < 60 || heartRates[i] > 100) {
            (*abnormalCount)++;
            if (heartRates[i] < 60) {
                (*belowCount)++;
            } else if (heartRates[i] > 100) {
                (*aboveCount)++;
            }
        }
    }

    *average = (double)sum / N_SESSIONS;
}
```

6.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 MIN_DOSAGE 10
#define MAX_DOSAGE 100

void validateDosage(int age, float weight, char status[]) {
    float dosage;

    if (age >= 18) {
        dosage = weight * 0.05;
    } else {
        dosage = weight * 0.03;
    }

    if (dosage >= MIN_DOSAGE && dosage <= MAX_DOSAGE) {
        status[0] = 'V';
        status[1] = 'a';
        status[2] = 'l';
        status[3] = 'i';
        status[4] = 'd';
        status[5] = '\0';
    } else {
        status[0] = 'I';
        status[1] = 'n';
        status[2] = 'v';
```

```
        status[3] = 'a';
        status[4] = 'l';
        status[5] = 'i';
        status[6] = 'd';
        status[7] = '\0';
    }
}

int main() {
    int age[10];
    float weight[10];
    char status[10][10];
    int i;

    for (i = 0; i < 10; i++) {
        printf("Enter age and weight for patient %d:\n", i + 1);

        volatile int inputAge;
        volatile float inputWeight;

        printf("Age: ");
        scanf("%d", &inputAge);
        printf("Weight: ");
        scanf("%f", &inputWeight);

        age[i] = inputAge;
        weight[i] = inputWeight;

        validateDosage(age[i], weight[i], status[i]);
    }

    printf("\nPatient Dosage Validation Results:\n");
    for (i = 0; i < 10; i++) {
        printf("Patient %d - Age: %d, Weight: %.2f, Dosage Status: %s\n",
               i + 1, age[i], weight[i], status[i]);
    }

    return 0;
}
```

7.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 N_PRODUCTS 10
#define R_THRESHOLD 5

void updateInventory(int inventory[], int productIndex, int quantity);
void checkInventory(int inventory[]);

int main() {
    int inventory[N_PRODUCTS] = {10, 8, 3, 15, 6, 7, 12, 9, 4, 11};
    static int itemsBelowThreshold = 0;
```

```
    int productIndex, quantity;

    printf("Welcome to the Inventory Management System\n");

    while (1) {
        printf("\nEnter product index (0 to 9) to update or -1 to exit: ");
        scanf("%d", &productIndex);

        if (productIndex == -1) {
            break;
        }

        if (productIndex < 0 || productIndex >= N_PRODUCTS) {
            printf("Invalid product index. Please enter a number between 0 and 9.\n");
            continue;
        }

        printf("Enter quantity to update for product %d: ", productIndex);
        scanf("%d", &quantity);

        updateInventory(inventory, productIndex, quantity);

        checkInventory(inventory);

        printf("\nCurrent Inventory Levels:\n");
        for (int i = 0; i < N_PRODUCTS; i++) {
            printf("Product %d: %d\n", i, inventory[i]);
        }

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

    printf("\nExiting Inventory Management System...\n");
    return 0;
}

void updateInventory(int inventory[], int productIndex, int quantity) {
    inventory[productIndex] += quantity;
}

void checkInventory(int inventory[]) {
    static int itemsBelowThreshold = 0;

    for (int i = 0; i < N_PRODUCTS; i++) {
        if (inventory[i] < R_THRESHOLD) {
            itemsBelowThreshold++;
            printf("Product %d is below the reorder threshold. Reorder needed.\n", i);
        }
    }

    if (itemsBelowThreshold > 0) {
        printf("Suggestion: Reorder %d product(s) to restock.\n", itemsBelowThreshold);
    } else {
        printf("All products are above the reorder threshold.\n");
    }
```

}
8.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 M_CODES 10
#define V_C_LENGTH 6

static int invalidCodeCount = 0;

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

    if (length != V_C_LENGTH) {
        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[M_CODES][V_C_LENGTH + 1];
    int validCodeCount = 0;

    for (int i = 0; i < M_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.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 N_TARGETS 10

int main() {
    int t_positions[N_TARGETS] = {5, -10, 12, 7, 2, -15, 0, 6, -3, 8};

    register int d_threshold = 5;

    for (int i = 0; i < N_TARGETS; i++) {
        int t_position = t_positions[i];

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

    return 0;
}
```

TWO DIMENSIONAL ARRAY

```
#include <stdio.h>

int main()
{
  int a[3][3];
  for (int i = 0; i < 3 ; i++){
      for(int j = 0; j < 3; j++){
          printf("Address of a[%d][%d] = %p \n",i,j,&a[i][j]);
      }
  }
    return 0;
}
```

THREE DIMENSIONAL ARRAY

```
#include <stdio.h>

int main()
{
  int a[2][3][4];
  for (int i = 0; i <2 ; i++){
```

```
    for(int j = 0; j < 3; j++){
        {
            for(int k=0;k<4;k++){

        printf("Address of a[%d][%d][%d] = %p \n",i,j,k,&a[i][j][k]);
            }
    }
 }
   return 0;
}}
```

TWO DIMENSIONAL ARRAY PROGRAMS

1.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], B2[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", &B2[i][j]);
        }
    }

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

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

```
            result[i][j] = A1[i][j] + B2[i][j];
        }
    }

    printf("Resulting matrix after addition:\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.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 3

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

int main() {
    int originalMatrix[M][N] = {
        {1, 2},
        {3, 4},
        {5, 6}
    };

    int transposedMatrix[N][M];

    if (M > 0 && N > 0) {
        printf("Original Matrix (%dx%d):\n", M, N);
        printMatrix(originalMatrix);

        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                transposedMatrix[j][i] = originalMatrix[i][j];
            }
        }
```

```
        printf("\nTransposed Matrix (%dx%d):\n", N, M);
        printMatrix(transposedMatrix);
    } else {
        printf("Matrix size is invalid.\n");
    }

    return 0;
}
```

3.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] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    int maxInRow[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];
            }
        }

        maxInRow[i] = maxVal;
    }

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

    return 0;
}
```

4.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. The number of columns in Matrix A must be equal to the
number of rows in Matrix B.\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.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;
}
```

THREE DIMENSIONAL ARRAY PROGRAMS

1.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

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

    printf("Enter elements of the first 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("matrix1[%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrix1[i][j][k]);
            }
        }
    }

    printf("Enter elements of the second 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("matrix2[%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrix2[i][j][k]);
            }
        }
    }

    if (X != X || Y != Y || Z != Z) {
        printf("Error: Matrices dimensions do not match. Exiting program.\n");
        return 1;
    }

    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];
            }
        }
    }

    printf("Resulting matrix after addition (%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("result[%d][%d][%d] = %d\n", i, j, k, result[i][j][k]);
            }
        }
    }

    return 0;
}
```

2.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

int main() {
    int matrix[X][Y][Z] = {
        {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
        {{-1, -2, -3}, {-4, -5, -6}, {-7, -8, -9}},
        {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}}
    };

    static int max_val = 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] > max_val) {
                    max_val = matrix[i][j][k];
                }
            }
        }
    }

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

    return 0;
}
```

3.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 3
#define Y 3
#define Z 3

void printMatrix(int matrix[X][Y][Z]) {
    printf("Matrix:\n");
    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 scalarMultiply(int matrix[X][Y][Z], int scalar, 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] = matrix[i][j][k] * scalar;
            }
        }
    }
}

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 scalar, result[X][Y][Z];

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

    if (scalar <= 0) {
        printf("Invalid scalar value! The scalar must be a positive integer.\n");
        return 1;
    }

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

    scalarMultiply(matrix, scalar, result);

    printf("\nResulting Matrix after scalar multiplication:\n");
    printMatrix(result);

    return 0;
}
```

4.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

static int positiveCount = 0;
static int negativeCount = 0;
static int zeroCount = 0;

void countElements(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++) {
                if (matrix[i][j][k] > 0) {
                    positiveCount++;
                } else if (matrix[i][j][k] < 0) {
                    negativeCount++;
                } else {
                    zeroCount++;
                }
            }
        }
    }
}

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

int main() {

    int matrix[X][Y][Z] = {
        {{1, -2, 0}, {3, -1, 0}, {0, 0, 2}},
        {{-3, 0, 1}, {4, 5, -6}, {0, 2, -1}},
        {{-1, 0, 0}, {0, 3, -2}, {5, 1, 0}}
    };

    countElements(matrix);

    printCounts();

    return 0;
}
```

5.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 2
#define Z 4

void printMatrix(int matrix[X][Y][Z]) {
    printf("Matrix:\n");
    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 transposeMatrix(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[i][k][j] = 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[k][i][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}}
    };

    int transposed[X][Y][Z];
    int axis;

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

    printf("Enter the axis to transpose along (0: X-axis, 1: Y-axis, 2: Z-axis): ");
    scanf("%d", &axis);

    transposeMatrix(matrix, transposed, axis);

    if (axis >= 0 && axis <= 2) {
        printf("Transposed Matrix (along axis %d):\n", axis);
        printMatrix(transposed);
    }

    return 0;
}
```