```c
/*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
void main(){
    const int size =SIZE;
    int ar[10]={6,3,5,2,1,7,4,9,10,8};
    int max=0,min=100;
    static int diff =0;
    for(int i=0;i<size;i++)
    {
        if(max < ar[i])
        {
            max = ar[i];
        }
        if(min > ar[i])
        {
            min = ar[i];
        }
        diff = max-min;
        printf("difference between max:%d and min:%d is
%d\n",max,min,diff);
    }
    printf("Max is %d and Min is %d\n",max,min);
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\
difference between max:6 and min:6 is 0
difference between max:6 and min:3 is 3
difference between max:6 and min:3 is 3
difference between max:6 and min:2 is 4
difference between max:6 and min:1 is 5
difference between max:7 and min:1 is 6
difference between max:7 and min:1 is 6
difference between max:9 and min:1 is 8
difference between max:10 and min:1 is 9
difference between max:10 and min:1 is 9
Max is 10 and Min is 1
PS D:\projects\quest\C>
```

```c
/*
Categorize elements of a single-dimensional array into positive, negative,
and zero values. Use:
A const variable to define the size of the array.
A for loop for traversal.
if-else statements to classify each element into separate arrays using
static storage.
*/
#include<stdio.h>
#define SIZE 10
void main(){
    const int size = SIZE;
    int ar[SIZE]={1,5,-7,0,3,-2,9,0,-6,3};
    static int positive[SIZE],negative[SIZE],zero[SIZE];
    int pcount=0,ncount=0,zcount=0;
    for(int i=0;i<size;i++)
    {
        if(ar[i]>0)
        {
            positive[pcount]=ar[i];
            pcount++;
        }
        else if(ar[i]<0)
        {
            negative[ncount]=ar[i];
            ncount++;
```

```
        }
        else
        {
            zero[zcount]=ar[i];
            zcount++;
        }
    }
    for(int i =0;i<pcount;i++)
    printf("%d \t",positive[i]);
    printf("\n");
    for(int i =0;i<ncount;i++)
    printf("%d \t",negative[i]);
    printf("\n");
    for(int i =0;i<zcount;i++)
    printf("%d \t",zero[i]);

}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc 13
1       5       3       9       3
-7      -2      -6
0       0
PS D:\projects\quest\C>
```

```
/*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 MAX 10
void main(){
    const int max=MAX;
    static int sum=0;
    int ar[MAX]={1,2,3,4,5,6,7,8,9,10};
    for(int i=0;i<max;i++)
    {
```

```
        sum+=ar[i];
    }
    printf("%d",sum);
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C
55
PS D:\projects\quest\C>
```

```
/*Identify which elements in a single-dimensional array are prime numbers.
Use:
A for loop to iterate through the array and check each element.
A nested for loop to determine if a number is prime.
if statements for decision-making.
A const variable to define the size of the array.*/
#include <stdio.h>
#include <math.h>
#define MAX 10
int main() {
    const int max = MAX;
    int ar[MAX] = {11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
    int prime =1,num;
    for (int i = 0; i < max; i++) {
         num = ar[i];
         prime = 1;
        if (num <= 1) {
            prime = 0;
        } else {
            for (int j = 2; j <= num/2; j++) {
                if (num % j == 0) {
                    prime = 0;
                    break;
                }
            }
        }
        if (prime == 1) {
            printf("%d \t", num);
        }
    }
```

```
    return 0;
}
```

```
PS D:\projects\quest\C> cd "d:\projects
11      13      17      19
PS D:\projects\quest\C>
```

```
/*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 RCOUNT 3;
void main()
{
     int ar[10]={1,2,3,4,5,6,7,8,9,10};
    int first;
    const int rcount =RCOUNT;
    int count =rcount;
    static int rar[10];
    for(int i=0;i<10;i++)
     rar[i]=ar[i];
    while(count>0)
    {
        first =rar[0];
        for(int i=0;i<10-1;i++)
        {
            rar[i]=rar[i+1];
        }
        rar[9]=first;
        count--;
    }
printf("\n");
for(int i=0;i<10;i++)
printf("%d \t",rar[i]);
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc 135.c -o

4       5       6       7       8       9       10      1       2       3
PS D:\projects\quest\C>
```

```c
/*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 20
void main(){
    const int size=SIZE;
    int ar[SIZE]={1,2,3,1,4,5,6,6,7,8,9,5,4,3,2,1,6,6,7,1};
    static int freq[10]={0};
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<size;j++)
        {
            if(i==ar[j])
            {
                freq[ar[j]]++;
            }
        }
    }
    for(int i=0;i<10;i++)
    printf("frequency of %d is %d\n",i,freq[i]);
}
```

```
PS D:\projects\quest\C> cc
frequency of 0 is 0
frequency of 1 is 4
frequency of 2 is 2
frequency of 3 is 2
frequency of 4 is 2
frequency of 5 is 2
frequency of 6 is 4
frequency of 7 is 2
frequency of 8 is 1
frequency of 9 is 1
PS D:\projects\quest\C>
```

```c
/*
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
void main(){
    const int size = SIZE;
    int ar[SIZE]={1,2,3,4,5,7,8,6,10,9};
    int temp;
    for(int i=0;i<size;i++)
    {
        for(int j=i+1;j<size;j++)
        {
            if(ar[i]<ar[j])
            {
                temp=ar[i];
                ar[i]=ar[j];
                ar[j]=temp;
            }
        }
    }
```

```c
    for(int i=0;i<size;i++)
    printf("%d \t",ar[i]);
}
/*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>
#define SIZE 10
int main() {
    const int size = SIZE;
    int arr[SIZE] = {34, 72, 55, 19, 46, 89, 21, 40, 68, 90};
    static int s_l = 0;
    int l = 0;
    for (int i=0;i<size;i++)
    {
        if (arr[i]>l)
        {
            s_l = l;
            l = arr[i];
        } else if (arr[i] > s_l && arr[i] != l) {
            s_l = arr[i];
        }
    }

    if (s_l == INT_MIN) {
        printf("There is no second largest element in the array.\n");
    } else {
        printf("The second largest element in the array is: %d\n", s_l);
    }

    return 0;
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc 138.
The second largest element in the array is: 89
PS D:\projects\quest\C>
```

```c
/*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;
void main()
{ const int size= SIZE;
int ar[10]={1,2,3,4,5,6,7,8,9,10};
int odd[10],even[10];
int odcount=0,evcount=0;
for(int i=0;i<size;i++)
{
    if(ar[i]%2==0)
    {
        odd[odcount]=ar[i];
        odcount++;
    }
    else
    {
        even[evcount]=ar[i];
        evcount++;
    }
}
for(int i=0;i<odcount;i++)
printf("%d \t",odd[i]);
printf("\n");
for(int i=0;i<evcount;i++)
printf("%d \t",even[i]);
}
```

```
PS D:\projects\quest\C> cd "d:\project
2       4       6       8       10
1       3       5       7       9
PS D:\projects\quest\C>
```

```c
/*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>
#include <stdlib.h>
#include <time.h>
#define MAX 60
void main(){
    const int max=MAX;
    static int alert=0;
    int temp[10],t;
    srand(time(NULL));
    for(int i=0;i<10;i++)
    {
        t=(rand()%100)+1;
        temp[i]=t;
    }
    for(int i=0;i<10;i++)
    {
        if(temp[i]>max)
        alert++;
    }
    if(alert>0)
    printf("The temperature exceeded max threshold %d times\n",alert);
    else
    printf("All readings were under the threshold\n");
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($
The temperature exceeded max threshold 5 times
PS D:\projects\quest\C>
```

```c
/*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 CONSUME 3.0
void main(){
    volatile float distance[10];
    float efficiency[10];
    for(int i=0;i<10;i++)
    {
        printf("Enter distance travelled in trip %d\n",i+1);
        scanf("%f",&distance[i]);
        efficiency[i]=distance[i]/CONSUME;
    }
    for(int i=0;i<10;i++)
    {
        printf("The fuel efficiency for trip %d is %f",i+1,efficiency[i]);
        if(efficiency[i]<10)
        printf("-Low efficiency");
        printf("\n");
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc 141
Enter distance travelled in trip 1
10
Enter distance travelled in trip 2
20
Enter distance travelled in trip 3
30
Enter distance travelled in trip 4
40
Enter distance travelled in trip 5
50
Enter distance travelled in trip 6
60
Enter distance travelled in trip 7
65
Enter distance travelled in trip 8
30
Enter distance travelled in trip 9
47
Enter distance travelled in trip 10
20
The fuel efficiency for trip 1 is 3.333333-Low efficiency
The fuel efficiency for trip 2 is 6.666667-Low efficiency
The fuel efficiency for trip 3 is 10.000000
The fuel efficiency for trip 4 is 13.333333
The fuel efficiency for trip 5 is 16.666666
The fuel efficiency for trip 6 is 20.000000
The fuel efficiency for trip 7 is 21.666666
The fuel efficiency for trip 8 is 10.000000
The fuel efficiency for trip 9 is 15.666667
The fuel efficiency for trip 10 is 6.666667-Low efficiency
PS D:\projects\quest\C>
```

```c
/*
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*/
```

```c
#include<stdio.h>
void main(){
    int altitude[10],temp;
    register int ca;
    srand(time(NULL));
    for(int i=0;i<10;i++)
    {
        altitude[i]=rand()%500;
    }
    for(int i=0;i<9;i++)
    {
        if(altitude[i]+50<altitude[i+1])
        printf("Alert current reading is %d which is %d higher than
previous reading
%d\n",altitude[i+1],altitude[i+1]-altitude[i],altitude[i]);
        else if(altitude[i]-50>altitude[i+1])
        printf("Alert current reading is %d which is %d lower than
previous reading
%d\n",altitude[i+1],altitude[i]-altitude[i+1],altitude[i]);
        else
        printf("Deviation is within expected limits\n");
    }

}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; if ($?) { gcc 141
Enter distance travelled in trip 1
10
Enter distance travelled in trip 2
20
Enter distance travelled in trip 3
30
Enter distance travelled in trip 4
40
Enter distance travelled in trip 5
50
Enter distance travelled in trip 6
60
Enter distance travelled in trip 7
65
Enter distance travelled in trip 8
30
Enter distance travelled in trip 9
47
Enter distance travelled in trip 10
20
The fuel efficiency for trip 1 is 3.333333-Low efficiency
The fuel efficiency for trip 2 is 6.666667-Low efficiency
The fuel efficiency for trip 3 is 10.000000
The fuel efficiency for trip 4 is 13.333333
The fuel efficiency for trip 5 is 16.666666
The fuel efficiency for trip 6 is 20.000000
The fuel efficiency for trip 7 is 21.666666
The fuel efficiency for trip 8 is 10.000000
The fuel efficiency for trip 9 is 15.666667
The fuel efficiency for trip 10 is 6.666667-Low efficiency
PS D:\projects\quest\C>
```

```c
/*
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*/
```

```c
#include<stdio.h>
void main(){
    int altitude[10],temp;
    register int ca;
    srand(time(NULL));
    for(int i=0;i<10;i++)
    {
        altitude[i]=rand()%500;
    }
    for(int i=0;i<9;i++)
    {
        if(altitude[i]+50<altitude[i+1])
        printf("Alert current reading is %d which is %d higher than
previous reading
%d\n",altitude[i+1],altitude[i+1]-altitude[i],altitude[i]);
        else if(altitude[i]-50>altitude[i+1])
        printf("Alert current reading is %d which is %d lower than
previous reading
%d\n",altitude[i+1],altitude[i]-altitude[i+1],altitude[i]);
        else
        printf("Deviation is within expected limits\n");
    }

}
```

```
Deviation is within expected limits
Alert current reading is 484 which is 126 higher than previous reading 358
Deviation is within expected limits
Alert current reading is 288 which is 167 lower than previous reading 455
Alert current reading is 449 which is 161 higher than previous reading 288
Alert current reading is 194 which is 255 lower than previous reading 449
Alert current reading is 448 which is 254 higher than previous reading 194
Alert current reading is 145 which is 303 lower than previous reading 448
Alert current reading is 455 which is 310 higher than previous reading 145
PS D:\projects\quest\C>
```

```c
/*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.
```

```c
Alert the user with a decision-making statement if deviations exceed
specified bounds.*/
#include<stdio.h>
#include<stdlib.h>

#define RADIUS 3000
#define LIMIT 100
void main()
{
    const int radius=RADIUS;
    const int limit=LIMIT;
    int position[10],deviation[10];
    for(int i=0;i<10;i++)
    {
        printf("Reading  %d ->",i+1);
        scanf("%d",&position[i]);

    }
    for(int i=0;i<10;i++)
    {
         deviation[i]=abs(position[i]-radius);
        printf("Deviation for reading %d is %d\n",i+1,deviation[i]);
        if(deviation[i]>100)
        printf("Alert deviation exceeds specified bounds\n");
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C
Reading  1 ->2000
Reading  2 ->2300
Reading  3 ->2200
Reading  4 ->2750
Reading  5 ->2650
Reading  6 ->1500
Reading  7 ->1450
Reading  8 ->1370
Reading  9 ->1300
Reading  10 ->1200
Deviation for reading 1 is 1000
Alert deviation exceeds specified bounds
Deviation for reading 2 is 700
Alert deviation exceeds specified bounds
Deviation for reading 3 is 800
Alert deviation exceeds specified bounds
Deviation for reading 4 is 250
Alert deviation exceeds specified bounds
Deviation for reading 5 is 350
Alert deviation exceeds specified bounds
Deviation for reading 6 is 1500
Alert deviation exceeds specified bounds
Deviation for reading 7 is 1550
Alert deviation exceeds specified bounds
Deviation for reading 8 is 1630
Alert deviation exceeds specified bounds
Deviation for reading 9 is 1700
Alert deviation exceeds specified bounds
Deviation for reading 10 is 1800
Alert deviation exceeds specified bounds
PS D:\projects\quest\C>
```

```
/*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).
```

```c
Loop through the array to calculate average heart rate and display
results.*/
#include<stdio.h>
void main(){
    int heart[10],avg,sum=0;
    static int alert=0;
    srand(time(NULL));
    for(int i=0;i<10;i++)
    {
        heart[i]=(rand()%150)+1;
        if(heart[i]>100 || heart[i]<60)
        alert++;

    }
    for(int i=0;i<10;i++)
    {
        sum+=heart[i];
        printf("%d\n",heart[i]);
    }
    avg=sum/10;
    printf("avg heart beat is %d\n",avg);
    printf("Number of abnormal readings %d \n",alert);
}
```

```
8
61
128
119
3
23
73
74
118
26
avg heart beat is 63
Number of abnormal readings 7
```

```c
/* 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.
```

```c
/* 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 MAX 10
#define DOSE 3
void main(){
    volatile int weight,age;
    const int max=MAX;
    const int dose=DOSE;
    int patient[max];
    for(int i=0;i<max;i++)
    {
        printf("Enter the weight and age for patient %d\n",i+1);
        scanf("%d %d",&weight,&age);
        patient[i]=weight/age;
    }
    for(int i=0;i<max;i++)
    {
        if(patient[i]>=dose)
        printf("The dose is valid for patient %d\n",i+1);
        else
        printf("The dose is not valid for patient %d\n",i+1);
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest\C\" ; 
Enter the weight and age for patient 1
100 25
Enter the weight and age for patient 2
70 30
Enter the weight and age for patient 3
40 10
Enter the weight and age for patient 4
50 15
Enter the weight and age for patient 5
76 23
Enter the weight and age for patient 6
10 3
Enter the weight and age for patient 7
15 4
Enter the weight and age for patient 8
59 10
Enter the weight and age for patient 9
130 56
Enter the weight and age for patient 10
28 6
The dose is valid for patient 1
The dose is not valid for patient 2
The dose is valid for patient 3
The dose is valid for patient 4
The dose is valid for patient 5
The dose is valid for patient 6
The dose is valid for patient 7
The dose is valid for patient 8
The dose is not valid for patient 9
The dose is valid for patient 10
```

```c
/*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 SIZE 10
```

```c
#define T 50
void main(){
    const int size =SIZE;
    const int t=T;
    int stock[size],c;
    static int track=0;
    printf("Initial stock level\n");
    for(int i=0;i<size;i++)
    {
        printf("Enter stock level for product %d\n",i+1);
        scanf("%d",&stock[i]);
    }
    for(int i=0;i<size;i++)
    {
        printf("Enter change in stock level\n");
        scanf("%d",&c);
        stock[i]=stock[i]+c;
    }
    for(int i=0;i<size;i++)
    {
        if(stock[i]<t)
        track++;
    }
    if(track>0)
    printf("Stock resupply is to be done\n");
    else
    printf("Stock resupply not required\n");
}
```

```
Enter change in stock level
50
Enter change in stock level
120
Enter change in stock level
-20
Enter change in stock level
40
Enter change in stock level
20
Enter change in stock level
10
Stock resupply is to be done
PS D:\projects\quest\C>
```

```c
/*Develop a program to validate 10 missile launch codes.
Use an array to store the codes.
Use const for defining valid code lengths and formats.
Implement decision-making statements to mark invalid codes and count them
using a static variable*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX 8
void main()
 {
    char codes[10][MAX+1];
    const int max = MAX;
    static int invalid=0;
    int valid=1;
    for (int i=0;i<10;i++)
    {
        printf("Enter launch code %d: ", i + 1);
        scanf("%s", codes[i]);
    }
    for (int i=0;i<10;i++) {
        valid = 1;
        if (strlen(codes[i]) != max) {
            valid = 0;
```

```
        } else {
            int has_upper = 0;
            for (int j=0;j<max; j++) {
                if (isupper(codes[i][j])) {
                    has_upper = 1;
                    break;
                }
            }
            if (!has_upper) {
                valid=0;
            }
        }
        if (valid) {
            printf("Code %d: %s - Valid\n", i + 1,codes[i]);
        } else {
            printf("Code %d: %s - Invalid\n", i + 1,codes[i]);
            invalid++;
        }
    }
    printf("\nTotal number of invalid codes: %d\n",invalid);
}
```

```
PS D:\projects\quest\C> cd "d:\projects\
Enter launch code 1: Balanced
Enter launch code 2: Password
Enter launch code 3: password
Enter launch code 4: hello
Enter launch code 5: live
Enter launch code 6: project
Enter launch code 7: proJect
Enter launch code 8: passWord
Enter launch code 9: HELLO
Enter launch code 10: PASSWORD
Code 1: Balanced - Valid
Code 2: Password - Valid
Code 3: password - Invalid
Code 4: hello - Invalid
Code 5: live - Invalid
Code 6: project - Invalid
Code 7: proJect - Invalid
Code 8: passWord - Valid
Code 9: HELLO - Invalid
Code 10: PASSWORD - Valid

Total number of invalid codes: 6
PS D:\projects\quest\C>
```

```c
/*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 SIZE 10
#define THRESHOLD 100
void main()
{
    const int size=SIZE;
    register int threshold =THRESHOLD;
```

```
    int position[size];
    char category[size];
    for(int i=0;i<size;i++)
    {
    printf("Enter the position for %d :",i+1);
    scanf("%d",&position[i]);
    }
    for(int i=0;i<size;i++)
    {
        if(position[i]>threshold)
        category[i]='H';//hostile
        else
        category[i]='F';//friendly
    }

    for(int i=0;i<size;i++)
    printf("Target at %d is %c\n",position[i],category[i]);
}
```

```
PS D:\projects\quest\C> cd
Enter the position for 1 :50
Enter the position for 2 :20
Enter the position for 3 :10
Enter the position for 4 :40
Enter the position for 5 :50
Enter the position for 6 :60
Enter the position for 7 :97
Enter the position for 8 :99
Enter the position for 9 :25
Enter the position for 10 :1
Target at 500 is H
Target at 200 is H
Target at 100 is F
Target at 40 is F
Target at 50 is F
Target at 60 is F
Target at 97 is F
Target at 99 is F
Target at 250 is H
Target at 15 is F
PS D:\projects\quest\C>
```

```c
/*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

void main()
{
    const int rows = M;
    const int cols = N;
    int matrix1[M][N], matrix2[M][N];
    static int result[M][N];
    printf("Enter elements of the first %dx%d matrix:\n", rows, cols);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix1[i][j]);
        }
    }
    printf("Enter elements of the second %dx%d matrix:\n", rows, cols);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix2[i][j]);
        }
    }
    if (rows == M && cols == N)
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }
        printf("Resulting matrix after addition:\n");
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
```

```
                printf("%d ", result[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Matrix dimensions do not match. Addition cannot be
performed.\n");
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\que
Enter elements of the first 3x3 matrix:
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]: 3
Element [1][0]: 4
Element [1][1]: 5
Element [1][2]: 6
Element [2][0]: 7
Element [2][1]: 8
Element [2][2]: 9
Enter elements of the second 3x3 matrix:
Element [0][0]: 9
Element [0][1]: 8
Element [0][2]: 7
Element [1][0]: 6
Element [1][1]: 5
Element [1][2]: 4
Element [2][0]: 3
Element [2][1]: 2
Element [2][2]: 1
Resulting matrix after addition:
10 10 10
10 10 10
10 10 10
PS D:\projects\quest\C>
```

```
/*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:
```

```c
/*
Print the original and transposed matrices.
Use a type qualifier (const) to ensure the matrix size is not modified
during execution.
*/
#include <stdio.h>

#define M 3
#define N 4

void main()
 {
    const int rows=M;
    const int cols=N;
    int matrix[M][N];
    static int transposed[N][M];
    printf("Enter elements of the %dx%d matrix:\n",rows,cols);
    for (int i=0;i<rows;i++) {
        for (int j=0;j<cols;j++)
        {
            printf("Element [%d][%d]: ",i,j);
            scanf("%d", &matrix[i][j]);
        }
    }
    if (rows==M && cols==N) {
        for (int i=0;i<rows;i++)
        {
            for (int j=0;j<cols;j++) {
                transposed[j][i] = matrix[i][j];
            }
        }
        printf("\nOriginal matrix:\n");
        for (int i=0;i<rows;i++)
         {
            for (int j=0;j<cols;j++) {
                printf("%d ",matrix[i][j]);
            }
            printf("\n");
        }
        printf("\nTransposed matrix:\n");
        for (int i=0;i<cols;i++)
```

```
        {
            for (int j=0;j<rows; j++) {
                printf("%d ",transposed[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Matrix dimensions do not match the defined constants.
Transposition cannot be performed.\n");
    }
}
```

```
Enter elements of the 3x4 matrix:
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]: 3
Element [0][3]: 4
Element [1][0]: 5
Element [1][1]: 6
Element [1][2]: 7
Element [1][3]: 8
Element [2][0]: 9
Element [2][1]: 10
Element [2][2]: 11
Element [2][3]: 12

Original matrix:
1 2 3 4
5 6 7 8
9 10 11 12

Transposed matrix:
1 5 9
2 6 10
3 7 11
4 8 12
PS D:\projects\quest\C>
```

```c
/*Find the Maximum Element in Each Row
Problem Statement: Write a program to find the maximum element in each row
of a two-dimensional array. The program should:
Take a matrix of size M x N as input, with dimensions defined using const
variables.
Use a static array to store the maximum value of each row.
Use nested for loops to traverse each row and find the maximum element.
Use if statements to compare and update the maximum value.
Requirements:
Print the maximum value of each row after processing the matrix.
Handle edge cases where rows might be empty using decision-making
statements.
*/
#include <stdio.h>
#define M 3
#define N 4

void main() {
    const int rows =M;
    const int cols =N;
    int matrix[M][N];
    static int max_in_row[M];
    printf("Enter elements of the %dx%d matrix:\n",rows,cols);
    for (int i=0;i<rows;i++)
    {
        for (int j=0;j<cols;j++)
         {
            printf("Element [%d][%d]: ",i,j);
            scanf("%d",&matrix[i][j]);
        }
    }
    for (int i=0;i<rows;i++)
    {
        if (cols > 0)
        {
            int max_value = matrix[i][0];
            for (int j = 1; j < cols; j++)
            {
                if (matrix[i][j] > max_value)
                {
```

```
                    max_value = matrix[i][j];
                }
            }
            max_in_row[i] = max_value;
        } else
         {
            max_in_row[i] = -1;
        }
    }
    printf("\nMaximum value in each row:\n");
    for (int i = 0; i < rows; i++)
    {
        if (cols > 0)
         {
            printf("Row %d: %d\n", i + 1, max_in_row[i]);
        } else
        {
            printf("Row %d: Empty row\n", i + 1);
        }
    }
}
```

```
PS D:\projects\quest\C> cd "d:\projects\quest
Enter elements of the 3x4 matrix:
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]: 3
Element [0][3]: 4
Element [1][0]: 5
Element [1][1]: 6
Element [1][2]: 7
Element [1][3]: 8
Element [2][0]: 9
Element [2][1]: 10
Element [2][2]: 11
Element [2][3]: 12

Maximum value in each row:
Row 1: 4
Row 2: 8
Row 3: 12
PS D:\projects\quest\C>
```

```c
/*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
#define N 3
```

```c
#define P 2
void main()
{
    const int rowsA=M;
    const int colsA=N;
    const int rowsB=N;
    const int colsB=P;
    int matrixA[M][N];
    int matrixB[N][P];
    static int result[M][P]={0};


    printf("Enter elements of the %dx%d matrix A:\n",rowsA,colsA);
    for (int i = 0; i < rowsA; i++)
     {
        for (int j = 0; j < colsA; j++)
         {
            printf("Element A[%d][%d]: ", i, j);
            scanf("%d", &matrixA[i][j]);
        }
    }
    printf("Enter elements of the %dx%d matrix B:\n", rowsB, colsB);
    for (int i = 0; i < rowsB; i++)
    {
        for (int j = 0; j < colsB; j++)
        {
            printf("Element B[%d][%d]: ",i,j);
            scanf("%d", &matrixB[i][j]);
        }
    }
    if (colsA == rowsB)
    {
        for (int i = 0; i < rowsA; i++)
        {
            for (int j = 0; j < colsB; j++)
            {
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += matrixA[i][k]*matrixB[k][j];
                }
            }
```

```
        }
        printf("\nMatrix A:\n");
        for (int i = 0; i < rowsA; i++)
        {
            for (int j = 0; j < colsA; j++)
            {
                printf("%d ",matrixA[i][j]);
            }
            printf("\n");
        }
        printf("\nMatrix B:\n");
        for (int i = 0; i < rowsB; i++)
        {
            for (int j = 0; j < colsB; j++)
            {
                printf("%d ",matrixB[i][j]);
            }
            printf("\n");
        }
        printf("\nResultant Matrix (AxB):\n");
        for (int i = 0; i < rowsA; i++)
        {
            for (int j = 0; j < colsB; j++)
            {
                printf("%d ",result[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Matrix multiplication is not possible. Number of columns
in Matrix A must equal number of rows in Matrix B.\n");
    }
}
```

```
Enter elements of the 2x3 matrix A:
Element A[0][0]: 1
Element A[0][1]: 2
Element A[0][2]: 3
Element A[1][0]: 4
Element A[1][1]: 5
Element A[1][2]: 6
Enter elements of the 3x2 matrix B:
Element B[0][0]: 7
Element B[0][1]: 8
Element B[1][0]: 9
Element B[1][1]: 10
Element B[2][0]: 11
Element B[2][1]: 12

Matrix A:
1 2 3
4 5 6

Matrix B:
7 8
9 10
11 12

Resultant Matrix (AxB):
58 64
139 154
```

```
/*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.
```

```c
/*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
int main() {
    const int rows = M;
    const int cols = N;
    int matrix[M][N];
    static int zero_count = 0;
    int total_elements = rows * cols;
    printf("Enter elements of the %dx%d matrix:\n", rows, cols);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }
    if (rows <= 0 || cols <= 0)
    {
        printf("Invalid matrix dimensions.\n");
        return 1;
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (matrix[i][j] == 0) {
                zero_count++;
            }
        }
    }
    if (zero_count > total_elements / 2)
    {
        printf("The matrix is sparse.\n");
    } else {
        printf("The matrix is not sparse.\n");
```

```c
    }
    if (zero_count == 0) {
        printf("The matrix has no zero elements.\n");
    }
    printf("\nThe matrix is:\n");
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
         {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
PS D:\projects\quest\C> cd "d:\projects
Enter elements of the 3x3 matrix:
Element [0][0]: 1
Element [0][1]: 0
Element [0][2]: 0
Element [1][0]: 0
Element [1][1]: 1
Element [1][2]: 0
Element [2][0]: 0
Element [2][1]: 0
Element [2][2]: 1
The matrix is sparse.

The matrix is:
1 0 0
0 1 0
0 0 1
PS D:\projects\quest\C>
```

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

```c
/*
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
#define Y 3
#define Z 3
int main() {
    const int x = X;
    const int y = Y;
    const int z = Z;
    int matrixA[X][Y][Z], matrixB[X][Y][Z];
    static int result[X][Y][Z];
    printf("Enter elements of the first %dx%dx%d matrix:\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("Element A[%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrixA[i][j][k]);
            }
        }
    }
    printf("Enter elements of the second %dx%dx%d matrix:\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("Element B[%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrixB[i][j][k]);
            }
        }
    }
    if (x <= 0 || y <= 0 || z <= 0)
     {
        printf("Invalid matrix dimensions.\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] = matrixA[i][j][k] + matrixB[i][j][k];
            }
        }
    }
    printf("Resulting 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("Element result[%d][%d][%d] = %d\n", i, j, k,
result[i][j][k]);
            }
        }
    }
}
```

```
Enter elements of the first 2x3x3 matrix:
Element A[0][0][0]: 1
Element A[0][0][1]: 2
Element A[0][0][2]: 3
Element A[0][1][0]: 4
Element A[0][1][1]: 5
Element A[0][1][2]: 6
Element A[0][2][0]: 7
Element A[0][2][1]: 8
Element A[0][2][2]: 9
Element A[1][0][0]: 10
Element A[1][0][1]: 11
Element A[1][0][2]: 12
Element A[1][1][0]: 13
Element A[1][1][1]: 14
Element A[1][1][2]: 15
Element A[1][2][0]: 16
Element A[1][2][1]: 17
Element A[1][2][2]: 18
Enter elements of the second 2x3x3 matrix:
Element B[0][0][0]: 19
Element B[0][0][1]: 18
Element B[0][0][2]: 17
Element B[0][1][0]: 16
Element B[0][1][1]: 15
Element B[0][1][2]: 14
Element B[0][2][0]: 13
Element B[0][2][1]: 12
Element B[0][2][2]: 11
Element B[1][0][0]: 10
Element B[1][0][1]: 9
Element B[1][0][2]: 8
Element B[1][1][0]: 7
Element B[1][1][1]: 6
Element B[1][1][2]: 5
```

```
Element B[1][2][0]: 4
Element B[1][2][1]: 3
Element B[1][2][2]: 2
Resulting matrix after addition:
Element result[0][0][0] = 20
Element result[0][0][1] = 20
Element result[0][0][2] = 20
Element result[0][1][0] = 20
Element result[0][1][1] = 20
Element result[0][1][2] = 20
Element result[0][2][0] = 20
Element result[0][2][1] = 20
Element result[0][2][2] = 20
Element result[1][0][0] = 20
Element result[1][0][1] = 20
Element result[1][0][2] = 20
Element result[1][1][0] = 20
Element result[1][1][1] = 20
Element result[1][1][2] = 20
Element result[1][2][0] = 20
Element result[1][2][1] = 20
Element result[1][2][2] = 20
PS D:\projects\quest\C>
```

```
/*
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.
*/
```

```c
#include <stdio.h>
#define X 2
#define Y 3
#define Z 4
#define MIN -1
void main() {
    const int x = X;
    const int y = Y;
    const int z = Z;
    int matrix[X][Y][Z];
    static int max_value = MIN;
    printf("Enter elements of the %dx%dx%d matrix:\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("Element [%d][%d][%d]: ", i, j, k);
                scanf("%d", &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++) {
                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);
}
```

```
Element [0][0][1]: 2
Element [0][0][2]: 3
Element [0][0][3]: 4
Element [0][1][0]: 5
Element [0][1][1]: 6
Element [0][1][2]: 7
Element [0][1][3]: 8
Element [0][1][0]: 5
Element [0][1][1]: 6
Element [0][1][2]: 7
Element [0][1][0]: 5
Element [0][1][1]: 6
Element [0][1][0]: 5
Element [0][1][0]: 5
Element [0][1][0]: 5
Element [0][1][1]: 6
Element [0][1][2]: 7
Element [0][1][3]: 8
Element [0][2][0]: 12
Element [0][2][1]: 14
Element [0][2][2]: 2
Element [0][2][3]: 3
Element [1][0][0]: 9
Element [1][0][1]: 18
Element [1][0][2]: 2
Element [1][0][3]: 5
Element [1][1][0]: 3
Element [1][1][1]: 2
Element [1][1][2]: 1
Element [1][1][3]: 11
Element [1][2][0]: 12
Element [1][2][1]: 14
Element [1][2][2]: 15
Element [1][2][3]: 13
The maximum value in the matrix is: 18
PS D:\projects\quest\C>
```

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

```c
/*
Take a matrix of size X x Y x Z and a scalar value as input, where X, Y, and Z are declared as const variables.
Use a static three-dimensional array to store the resulting matrix.
Use nested for loops to multiply each element of the matrix by the scalar.
Requirements:
Print the original matrix and the resulting matrix after scalar multiplication.
Use decision-making statements to handle invalid scalar values (e.g., zero or negative scalars) if necessary.
*/
#include <stdio.h>

#define X 2
#define Y 3
#define Z 4
void main() {
    const int x = X;
    const int y = Y;
    const int z = Z;
    int matrix[X][Y][Z];
    static int result[X][Y][Z];
    int scalar;
    printf("Enter elements of the %dx%dx%d matrix:\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("Element [%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrix[i][j][k]);
            }
        }
    }
    printf("Enter scalar value for multiplication: ");
    scanf("%d", &scalar);
    if (scalar == 0)
        printf("Scalar value is zero. All elements will be zero after multiplication.\n");
    else if (scalar < 0)
```

```
        printf("Warning: Scalar value is negative. Proceeding with
multiplication.\n");

    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            for (int k = 0; k < z; k++)
            {
                result[i][j][k] = matrix[i][j][k] * scalar;
            }
        }
    }
    printf("Original 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");
    }
    printf("Resulting matrix after scalar multiplication:\n");
    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");
    }
}
```

```
Element [1][1][1]: 18
Element [1][1][2]: 19
Element [1][1][3]: 20
Element [1][2][0]: 21
Element [1][2][1]: 22
Element [1][2][2]: 23
Element [1][2][3]: 24
Enter scalar value for multiplication: 2
Original matrix:
1 2 3 4
5 6 7 8
9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24

Resulting matrix after scalar multiplication:
2 4 6 8
10 12 14 16
18 20 22 24

26 28 30 32
34 36 38 40
42 44 46 48

PS D:\projects\guest\C>
```

```
/*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.
```

```c
Ensure edge cases (e.g., all zeros or all negatives) are handled
correctly.
*/
#include <stdio.h>
#define X 2
#define Y 3
#define Z 4
void main() {
    const int x = X;
    const int y = Y;
    const int z = Z;
    int matrix[X][Y][Z];
    static int countPositive = 0;
    static int countNegative = 0;
    static int countZero = 0;
    printf("Enter elements of the %dx%dx%d matrix:\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("Element [%d][%d][%d]: ", i, j, k);
                scanf("%d", &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++)
            {
                if (matrix[i][j][k] > 0)
                {
                    countPositive++;
                } else if (matrix[i][j][k] < 0)
                 {
                    countNegative++;
                } else {
```

```
                    countZero++;
                }
            }
        }
    }
    printf("Number of positive elements: %d\n", countPositive);
    printf("Number of negative elements: %d\n", countNegative);
    printf("Number of zero elements: %d\n", countZero);
}
```

```
Element [0][0][3]: 2
Element [0][1][0]: 0
Element [0][1][1]: 1
Element [0][1][2]: 4
Element [0][1][3]: 5
Element [0][2][0]: 6
Element [0][2][1]: -2
Element [0][2][2]: -3
Element [0][2][3]:
-5
Element [1][0][0]: 3
Element [1][0][1]: 9
Element [1][0][2]: -8
Element [1][0][3]: 1
Element [1][1][0]: 2
Element [1][1][1]: 3
Element [1][1][2]: -5
Element [1][1][3]: 2
Element [1][2][0]: -4
Element [1][2][1]: 0
Element [1][2][2]: 0
Element [1][2][3]: 0
Number of positive elements: 12
Number of negative elements: 7
Number of zero elements: 5
```

```
/*Transpose of a 3D Matrix Along a Specific Axis
```

```c
/*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.
has context menu*/
#include <stdio.h>
#define X 2
#define Y 3
#define Z 4
int main() {
    const int x = X;
    const int y = Y;
    const int z = Z;
    int matrix[X][Y][Z]={0};
    static int transposed[X][Y][Z];
    int axis;
    printf("Enter elements of the %dx%dx%d matrix:\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("Element [%d][%d][%d]: ", i, j, k);
                scanf("%d", &matrix[i][j][k]);
            }
        }
    }
    printf("Enter the axis of transposition (0 for depth, 1 for rows, 2
for columns): ");
    scanf("%d", &axis);
    if (axis < 0 || axis > 2)
```

```
    {
        printf("Invalid axis! Please choose 0, 1, or 2.\n");
        return 1;
    }
    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][j][k] = matrix[j][i][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[i][j][k] = matrix[i][k][j];
                }
            }
        }
    } else if (axis == 2) {
        for (int i = 0; i < x; i++)
        {
            for (int j = 0; j < y; j++)
            {
                for (int k = 0; k < z; k++)
                {
                    transposed[i][j][k] = matrix[k][j][i];
                }
            }
        }
    }
    printf("Original 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");
    }
    printf("Transposed matrix along axis %d:\n", axis);
    for (int i = 0; i < x; i++)
     {
        for (int j = 0; j < y; j++)
        {
            for (int k = 0; k < z; k++)
             {
                printf("%d ", transposed[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}
```

```
Enter the axis of transposition (0 for depth, 1 for rows, 2 for columns):
Original matrix:
1 2 3 4
5 6 7 8
9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24

Transposed matrix along axis 1:
1 5 9 13
2 6 10 14
3 7 11 15

13 17 21 4
14 18 22 3
15 19 23 2

PS D:\projects\guest\C>
```