1.Find Maximum and Minimum in an Array

- Problem Statement: Write a program to find the maximum and minimum values in a single-dimensional array of integers. Use:
  - A const variable for the array size.
  - A static variable to keep track of the maximum difference between the maximum and minimum values.
  - if statements within a for loop to determine the maximum and minimum values.

```
#include <stdio.h>
#define SIZE 10

void main()
{
   const int a[SIZE]={6, 8, 13, 18, 15, 1, 9, 24, 19, 2};
   int max,min,i;
   static int d=0;
   max=min=a[0];
   for(i=0;i<SIZE;i++)
   {
      if(a[i]>max)
         max=a[i];
      else
         min=a[i];
   }
   d=max-min;
```

```
    printf("Maximum Value: %d\n",max);
    printf("Minimum Value: %d\n",min);
    printf("Maximum Difference: %d\n",d);
}
```

O/P:

Maximum Value: 24

Minimum Value: 2

Maximum Difference: 22

2. Array Element Categorization

- Problem Statement: Categorize elements of a single-dimensional array into positive, negative, and zero values. Use:
    - A const variable to define the size of the array.
    - A for loop for traversal.
    - if-else statements to classify each element into separate arrays using static storage.

```
#include <stdio.h>
#define SIZE 10

void main()
{
    const int a[SIZE]={8, 0, 19, 15, -18, -13, 0, -1, 9, 24};
    static int p[SIZE], n[SIZE], z[SIZE];
    int cp=0, cn=0, cz=0,i;
    for(i=0;i<SIZE;i++)
```

```
{
  if(a[i]>0)
   {
      p[cp]=a[i];
      cp++;
   }
   else if(a[i]<0)
   {
      n[cn]=a[i];
      cn++;
   }
   else
   {
      z[cz]=a[i];
      cz++;
   }
}
printf("Positive Numbers: ");
for(i=0;i<cp;i++)
   printf("%d ",p[i]);
printf("\n");
printf("Negative Numbers: ");
for(i=0;i<cn;i++)
   printf("%d ",n[i]);
printf("\n");
printf("Zeroes: ");
for(i=0;i<cz;i++)
```

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

O/P:

Positive Numbers: 8 19 15 9 24

Negative Numbers: -18 -13 -1

Zeroes: 0 0

3. Cumulative Sum of Array Elements

- Problem Statement: Calculate the cumulative sum of elements in a single-dimensional array. Use:
  - A static variable to hold the running total.
  - A for loop to iterate through the array and update the cumulative sum.
  - A const variable to set the array size.

```
#include <stdio.h>
#define SIZE 10

void main()
{
    const int a[SIZE]={10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    static int cs=0;
    int i;
    for(i = 0; i < SIZE; i++)
    {
```

```
        cs += a[i];
        printf("Cumulative Sum = %d \n", cs);
    }
}
```

O/P:

Cumulative Sum = 10

Cumulative Sum = 30

Cumulative Sum = 60

Cumulative Sum = 100

Cumulative Sum = 150

Cumulative Sum = 210

Cumulative Sum = 280

Cumulative Sum = 360

Cumulative Sum = 450

Cumulative Sum = 550

4. Check Prime Numbers in an Array

- Problem Statement: Identify which elements in a single-dimensional array are prime numbers. Use:
    - A for loop to iterate through the array and check each element.
    - A nested for loop to determine if a number is prime.
    - if statements for decision-making.
    - A const variable to define the size of the array.

```
#include <stdio.h>
#define SIZE 10
```

```
int main()
{
   const int a[SIZE] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
   int i,j,p;
   printf("The prime numbers in the given array are:\n");
   for(i=0;i<SIZE;i++)
   {
      p=1;
      if(a[i]<2)
         p=0;
      else
      {
         for(j=2;j<=a[i]/2;j++)
         {
            if(a[i]%j==0)
            {
               p=0;
               break;
            }
         }
      }
      if(p==1)
         printf("%d ", a[i]);
   }
   printf("\n");
   return 0;
```

```
}
```

O/P:

The prime numbers in the given array are:

2 3 5 7 11

5. Array Rotation by N Positions

- Problem Statement: Rotate the elements of a single-dimensional array to the left by N positions. Use:
  - A const variable for the rotation count.
  - A static array to store the rotated values.
  - A while loop for performing the rotation.

```
#include <stdio.h>
#define SIZE 10
#define COUNT 4

void main()
{
   const int a[SIZE]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   static int r[SIZE];
   int i,n;
   printf("Original Array:\n");
   for(i=0;i<SIZE;i++)
      printf("%d ",a[i]);
   printf("\n");
   i=0;
   while(i<SIZE)
```

```
    {
        n=(i-COUNT+SIZE)%SIZE;
        r[n]=a[i];
        i++;
    }
    printf("Array after rotating %d positions to the left:\n",COUNT);
    for(i=0;i<SIZE;i++)
        printf("%d ",r[i]);
    printf("\n");
}
```

O/P:

Original Array:

1 2 3 4 5 6 7 8 9 10

Array after rotating 4 positions to the left:

5 6 7 8 9 10 1 2 3 4

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
void main()
```

```
{
   const int a[SIZE]={1, 2, 3, 3, 4, 4, 4, 5, 5, 6};
   static int f[SIZE];
   int i,j,c;
   printf("Array elements and their frequencies:\n");
   for(i=0;i<SIZE;i++)
   {
      c=1;
      if(f[i]!=0)
         continue;
      for(j=i+1;j<SIZE;j++)
      {
         if(a[i]==a[j])
         {
            c++;
            f[j]=-1;
         }
      }
      f[i]=c;
      printf("Element %d appears %d times\n",a[i],f[i]);
   }
}
```

O/P:

```
Array elements and their frequencies:
Element 1 appears 1 times
Element 2 appears 1 times
```

Element 3 appears 2 times

Element 4 appears 3 times

Element 5 appears 2 times

Element 6 appears 1 times

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 10
void main()
{
   const int a[SIZE]={1, 9, 3, 19, 13, 8, 24, 6, 15, 18};
   int s[SIZE];
   int i,j,t;
   printf("Original Array:\n");
   for(i=0;i<SIZE;i++)
      printf("%d ",a[i]);
   printf("\n");
   for(i=0;i<SIZE;i++)
      s[i]=a[i];
   for(i=0;i<SIZE-1;i++)
   {
```

```
        for(j=0;j<SIZE-i-1;j++)
        {
            if(s[j]<s[j+1])
            {
                t=s[j];
                s[j]=s[j+1];
                s[j+1]=t;
            }
        }
    }
    printf("Sorted array in descending order:\n");
    for(i=0;i<SIZE;i++)
        printf("%d ",s[i]);
    printf("\n");
}
```

O/P:

Original Array:

1 9 3 19 13 8 24 6 15 18

Sorted array in descending order:

24 19 18 15 13 9 8 6 3 1

8. Find the Second Largest Element

- Problem Statement: Find the second largest element in a single-dimensional array. Use:
    - A const variable for the array size.
    - A static variable to store the second largest element.

- if statements and a single for loop to compare elements.

```
#include <stdio.h>
#define SIZE 10
void main()
{
   const int a[SIZE]={1, 9, 3, 19, 13, 8, 24, 6, 15, 18};
   static int sl,l;
   int i;
   printf("Array:\n");
   for(i=0;i<SIZE;i++)
      printf("%d ",a[i]);
   printf("\n");
   for(i=2;i<SIZE;i++)
   {
      if(a[i]>l)
      {
         sl=l;
         l=a[i];
      }
      else if(a[i]>sl && a[i]!=l)
         sl=a[i];
   }
   printf("SL: %d \n",sl);
}
```

O/P:

Array:

1 9 3 19 13 8 24 6 15 18

SL: 19

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

void main()
{
   const int a[SIZE]={1, 9, 3, 19, 13, 8, 24, 6, 15, 18};
   int e[SIZE], o[SIZE], ec=0, oc=0, i;
   for(i=0;i<SIZE;i++)
   {
      if(a[i]%2==0)
      {
         e[ec]=a[i];
         ec++;
      }
      else
```

```
        {
            o[oc]=a[i];
            oc++;
        }
    }
    printf("Even numbers:\n");
    for(i=0;i<ec;i++)
        printf("%d ",e[i]);
    printf("\n");
    printf("Odd numbers:\n");
    for(i=0;i<oc;i++)
        printf("%d ",o[i]);
    printf("\n");
}
```

O/P:

Even numbers:

8 24 6 18

Odd numbers:

1 9 3 19 13 15

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 10
void main()
{
   const int a[SIZE]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   static int s[SIZE];
   int i,l;
   printf("Original Array:\n");
   for(i=0;i<SIZE;i++)
      printf("%d ",a[i]);
   printf("\n");
   l=a[SIZE-1];
   for(i=SIZE-1;i>0;i--)
      s[i]=a[i-1];
   s[0]=l;
   printf("Array after cyclic right shift by one position:\n");
   for(i=0;i<SIZE;i++)
      printf("%d ",s[i]);
   printf("\n");
}
```

O/P:

```
Original Array:
1 2 3 4 5 6 7 8 9 10
Array after cyclic right shift by one position:
10 1 2 3 4 5 6 7 8 9
```

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 MT 100
#define S 10

static int c=0;
extern int f=0;

int main()
{
   const int st=70;
   int t[S], i;
   for(i=0;i<S;i++)
   {
      printf("Enter the temperature at time interval %d:\n",i+1);
      scanf("%d", &t[i]);
      if(t[i]>st)
      {
         f=1;
```

```
            c++;
                printf("Temperature exceeds the safe threshold at interval %d\n",i+1);
        }
        if(t[i]>MT)
          printf("Temperature exceeded max safe limit at interval %d\n",i+1);
    }
    if(c>0)
        printf("\nTotal no. of times temp exceeded safe threshold: %d\n",c);
    else
        printf("\nTemperature was in the given range\n");
    return 0;
}
```

O/P:

```
Enter the temperature at time interval 1:
75
Temperature exceeds the safe threshold at interval 1
Enter the temperature at time interval 2:
80
Temperature exceeds the safe threshold at interval 2
Enter the temperature at time interval 3:
50
Enter the temperature at time interval 4:
40
Enter the temperature at time interval 5:
33
Enter the temperature at time interval 6:
```

95

Temperature exceeds the safe threshold at interval 6

Enter the temperature at time interval 7:

102

Temperature exceeds the safe threshold at interval 7

Temperature exceeded max safe limit at interval 7

Enter the temperature at time interval 8:

99

Temperature exceeds the safe threshold at interval 8

Enter the temperature at time interval 9:

54

Enter the temperature at time interval 10:

101

Temperature exceeds the safe threshold at interval 10

Temperature exceeded max safe limit at interval 10

Total no. of times temp exceeded safe threshold: 6

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 S 10
#define F 5.0

volatile float d[S];
volatile float fe;

void main()
{
   int i;
   float e;
   printf("Enter the distances covered in 10 different trips in km\n");
   for(i=0;i<S;i++)
   {
      printf("Enter distance for trip %d: ", i+1);
      scanf("%f",&d[i]);
   }
   printf("\nFuel Efficiency in km/L for each trip:\n");
   for(i=0;i<S;i++)
   {
      e=d[i]/F;
      fe=e;
      printf("Trip %d: %.2f km/L\n",i+1,fe);
      if(fe<10)
         printf("Low fuel efficiency for trip %d(< 10 km/L)\n",i+1);
   }
}
```

O/P:

```
Enter the distances covered in 10 different trips in km
Enter distance for trip 1: 10
Enter distance for trip 2: 50
Enter distance for trip 3: 70
Enter distance for trip 4: 90
Enter distance for trip 5: 88
Enter distance for trip 6: 45
Enter distance for trip 7: 95
Enter distance for trip 8: 85
Enter distance for trip 9: 158
Enter distance for trip 10: 924

Fuel Efficiency in km/L for each trip:
Trip 1: 2.00 km/L
Low fuel efficiency for trip 1(< 10 km/L)
Trip 2: 10.00 km/L
Trip 3: 14.00 km/L
Trip 4: 18.00 km/L
Trip 5: 17.60 km/L
Trip 6: 9.00 km/L
Low fuel efficiency for trip 6(< 10 km/L)
Trip 7: 19.00 km/L
Trip 8: 17.00 km/L
Trip 9: 31.60 km/L
Trip 10: 184.80 km/L
```

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>
#include<stdlib.h>

#define S 10
#define D 50

int main()
{
   register int ca;
   int a[S],i;
   printf("Enter altitude readings (in meters) for 10 seconds\n");
   for(i=0;i<S;i++)
   {
      printf("Reading %d: ",i+1);
      scanf("%d",&a[i]);
   }
   for (i=1;i<S;i++)
   {
      ca= a[i];
      if(abs(ca-a[i-1]) > D)
```

```
            printf("Altitude deviation detected between readings %d and %d\n",i,i+1);
        }
        return 0;
    }
```

O/P:

Enter altitude readings (in meters) for 10 seconds

Reading 1: 150

Reading 2: 175

Reading 3: 225

Reading 4: 300

Reading 5: 325

Reading 6: 350

Reading 7: 25

Reading 8: 110

Reading 9: 150

Reading 10: 160

Altitude deviation detected between readings 3 and 4

Altitude deviation detected between readings 6 and 7

Altitude deviation detected between readings 7 and 8

4. Satellite Orbit Analyzer

Design a program to analyze the position of a satellite based on 10 periodic readings.

- Use const for defining the orbit radius and limits.
- Store position data in an array and calculate deviations using loops.

- Alert the user with a decision-making statement if deviations exceed specified bounds.

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

#define S 10
#define R 10000
#define D 500

void main()
{
   const int or=R;
   const int d=D;
   int p[S], i, dev;
     printf("Enter satellite position readings in km for 10 periodic intervals\n");
   for(i=0;i<S;i++)
   {
      printf("Reading %d: ",i+1);
      scanf("%d", &p[i]);
   }
   for(i=1;i<S;i++)
   {
      dev=abs(p[i] - p[i-1]);
      printf("Deviation between reading %d and %d: %d km\n",i,i+1,dev);
      if(dev>d)
```

```
            printf("Deviation exceeds the maximum allowed deviation of %d
km.\n",d);
        }
    }
```

O/P:

```
Enter satellite position readings in km for 10 periodic intervals
Reading 1: 15000
Reading 2: 15300
Reading 3: 14900
Reading 4: 15800
Reading 5: 16000
Reading 6: 16450
Reading 7: 17000
Reading 8: 17356
Reading 9: 17550
Reading 10: 18000
Deviation between reading 1 and 2: 300 km
Deviation between reading 2 and 3: 400 km
Deviation between reading 3 and 4: 900 km
Alert: Deviation exceeds the maximum allowed deviation of 500 km.
Deviation between reading 4 and 5: 200 km
Deviation between reading 5 and 6: 450 km
Deviation between reading 6 and 7: 550 km
Alert: Deviation exceeds the maximum allowed deviation of 500 km.
Deviation between reading 7 and 8: 356 km
Deviation between reading 8 and 9: 194 km
Deviation between reading 9 and 10: 450 km
```

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 S 10

int main()
{
   int hr[S], i, t=0;
   static int a=0;
   double avg;
   printf("Enter heart rate readings in BPM for 10 sessions\n");
   for(i=0;i<S;i++)
   {
      printf("Session %d: ",i+1);
      scanf("%d",&hr[i]);
      if (hr[i]<60 || hr[i]>100)
         a++;
      t+=hr[i];
   }
   avg=(double)t/S;
   printf("Total number of abnormal readings: %d\n",a);
   printf("Average heart rate: %.2f BPM\n",avg);
```

```
    if(a>0)
        printf("Abnormal heart rate readings were observed\n");
    else
        printf("Normal rate readings were observed\n");
    return 0;
}
```

O/P:

```
Enter heart rate readings in BPM for 10 sessions
Session 1: 70
Session 2: 75
Session 3: 80
Session 4: 65
Session 5: 90
Session 6: 102
Session 7: 50
Session 8: 99
Session 9: 72
Session 10: 69
Total number of abnormal readings: 2
Average heart rate: 77.20 BPM
Abnormal heart rate readings were observed
```

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 S 10
#define MIW 30
#define MAW 120
#define MIA 18
#define MAA 80

int dosage(float w,int a)
{
   if (w>=MIW && w<=MAW && a>=MIA && a<=MAA)
      return 1;
   else
      return 0;
}

int main()
{
   volatile float w[S];
   volatile int a[S];
   char r[S];
   int i;
   printf("Enter the weight in kg and age in years for 10 patients\n");
```

```
    for(i=0;i<S;i++)
    {
        printf("Patient %d - Weight: ",i+1);
        scanf("%f",&w[i]);
        printf("Patient %d - Age: ",i+1);
        scanf("%d",&a[i]);
        if (dosage(w[i],a[i]))
            r[i]='V';
        else
            r[i]='I';
    }
    for(i=0;i<S;i++)
            printf("Patient %d - Status: %s\n",i+1, r[i] == 'V' ? "Valid" :
"Invalid");
    return 0;
}
```

O/P:

```
Enter the weight in kg and age in years for 10 patients
Patient 1 - Weight: 50
Patient 1 - Age: 25
Patient 2 - Weight: 55
Patient 2 - Age: 24
Patient 3 - Weight: 60
Patient 3 - Age: 74
Patient 4 - Weight: 69
Patient 4 - Age: 45
Patient 5 - Weight: 78
```

Patient 5 - Age: 80

Patient 6 - Weight: 86

Patient 6 - Age: 85

Patient 7 - Weight: 44

Patient 7 - Age: 20

Patient 8 - Weight: 40

Patient 8 - Age: 14

Patient 9 - Weight: 22

Patient 9 - Age: 8

Patient 10 - Weight: 59

Patient 10 - Age: 29

Patient 1 - Status: Valid

Patient 2 - Status: Valid

Patient 3 - Status: Valid

Patient 4 - Status: Valid

Patient 5 - Status: Valid

Patient 6 - Status: Invalid

Patient 7 - Status: Valid

Patient 8 - Status: Invalid

Patient 9 - Status: Invalid

Patient 10 - Status: Valid

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 S 10
#define R 30

int main()
{
   int a[S] ,i;
   static int rc=0;
   printf("Enter the inventory levels for 10 products\n");
   for(i=0;i<S;i++)
   {
      printf("Product %d - Inventory level: ",i+1);
      scanf("%d",&a[i]);
      if(a[i]<R)
      {
         rc++;
         printf("--- Re-order\n");
      }
      else
         printf("---Sufficient stock\n");

   }
   printf("\nTotal number of products below reorder threshold: %d\n",rc);
   return 0;
}
```

O/P:

```
Enter the inventory levels for 10 products
Product 1 - Inventory level: 20
--- Re-order
Product 2 - Inventory level: 12
--- Re-order
Product 3 - Inventory level: 17
--- Re-order
Product 4 - Inventory level: 26
--- Re-order
Product 5 - Inventory level: 18
--- Re-order
Product 6 - Inventory level: 35
---Sufficient stock
Product 7 - Inventory level: 45
---Sufficient stock
Product 8 - Inventory level: 65
---Sufficient stock
Product 9 - Inventory level: 30
---Sufficient stock
Product 10 - Inventory level: 54
---Sufficient stock

Total number of products below reorder threshold: 5
```

8. Missile Launch Codes Validator

Develop a program to validate 10 missile launch codes.

- Use an array to store the codes.
- Use const for defining valid code lengths and formats.
- Implement decision-making statements to mark invalid codes and count them using a static variable.

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

#define S 10
#define L 6
#define F "XXXXXX"

int ancode(char c[])
{
   if(strlen(c) != L)
      return 0;
   for(int i=0;i<L;i++)
   {
      if(!isalnum(c[i]))
         return 0;
   }
   return 1;
}

int main()
```

```
{
   const int l=L;
   const char f[]=F;
   char c1[S][l+1];
   static int c=0;
   int i;
     printf("Enter 10 missile launch codes each of 6 alphanumeric characters:\n");
   for(i=0;i<S;i++)
   {
      printf("Enter code %d: ",i+1);
      scanf("%s",c1[i]);
      if (!ancode(c1[i]))
      {
         c++;
         printf("Invalid code %d: %s\n",i+1,c1[i]);
      }
   }
   printf("\nValidation completed.\n");
   printf("Total number of invalid codes: %d\n",c);
   return 0;
}
```

O/P:

```
Enter 10 missile launch codes each of 6 alphanumeric characters:
Enter code 1: ABC123
Enter code 2: DEF456
Enter code 3: WE5784
```

Enter code 4: QWE34@

Invalid code 4: QWE34@

Enter code 5: FG45RT

Enter code 6: 65TY7U

Enter code 7: aqws45

Enter code 8: GTY#@6

Invalid code 8: GTY#@6

Enter code 9: kjuy78

Enter code 10: MNJH255

Invalid code 10: MNJH255

Validation completed.

Total number of invalid codes: 3

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 S 10

#define P 0

#define T 50
```

```
int main()
{
   int p[S], i;
   register int t=T;
   printf("Enter the x-coordinates of 10 targets\n");
   for(i=0;i<S;i++)
   {
      printf("Target %d - Position: ",i+1);
      scanf("%d",&p[i]);
   }
   printf("\nTarget classification:\n");
   for(i=0;i<S;i++)
   {
      if(p[i] < -t || p[i] > t)
         printf("Target %d (Position: %d) - Hostile\n", i+1,p[i]);
      else
         printf("Target %d (Position: %d) - Friendly\n", i+1,p[i]);
   }
   return 0;
}
```

O/P:

```
Enter the x-coordinates of 10 targets
Target 1 - Position: 30
Target 2 - Position: -52
Target 3 - Position: 15
Target 4 - Position: 48
```

Target 5 - Position: 56

Target 6 - Position: 87

Target 7 - Position: -86

Target 8 - Position: 55

Target 9 - Position: -99

Target 10 - Position: 21

Target classification:

Target 1 (Position: 30) - Friendly

Target 2 (Position: -52) - Hostile

Target 3 (Position: 15) - Friendly

Target 4 (Position: 48) - Friendly

Target 5 (Position: 56) - Hostile

Target 6 (Position: 87) - Hostile

Target 7 (Position: -86) - Hostile

Target 8 (Position: 55) - Hostile

Target 9 (Position: -99) - Hostile

Target 10 (Position: 21) - Friendly

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 S 6
static int res[S][S];

void add(int r, int c, int m1[S][S], int m2[S][S])
{
   int i,j;
   printf("\nAddition of 2 matrices:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         res[i][j]=m1[i][j]+m2[i][j];
```

```
    }
    printf("Resultant Matrix:\n");
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
            printf("%d ",res[i][j]);
        printf("\n");
    }
}

void main()
{
    int r1, c1, r2, c2, m1[S][S],  m2[S][S], i, j;
    printf("Enter the number of rows and columns of first matrix:\n");
    scanf("%d %d",&r1,&c1);
      printf("Enter the number of rows and columns of second matrix:\n");
    scanf("%d %d",&r2,&c2);
    if(r1!=r2 || c1!=c2)
        printf("Addition of 2 matrices is not possible\n");
    else
    {
        printf("Enter the elements of first matrix:\n");
        for(i=0;i<r1;i++)
        {
            for(j=0;j<c1;j++)
                scanf("%d",&m1[i][j]);
        }
```

```
            printf("Enter the elements of second matrix:\n");
            for(i=0;i<r2;i++)
            {
               for(j=0;j<c2;j++)
                  scanf("%d",&m2[i][j]);
            }
            add(r1,c1,m1,m2);
         }
      }
```

O/P:

```
Enter the number of rows and columns of first matrix:
3 3
Enter the number of rows and columns of second matrix:
3 3
Enter the elements of first matrix:
1 3 2
2 1 3
3 2 1
Enter the elements of second matrix:
1 2 3
3 1 2
2 3 1

Addition of 2 matrices:
Resultant Matrix:
2 5 5
```

5 2 5

5 5 2

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 S 6
static int t[S][S];

void main()
{
   int r, c, m[S][S], i, j;
   printf("Enter the number of rows and columns of the matrix:\n");
   scanf("%d %d",&r,&c);
   printf("Enter the elements of matrix:\n");
   for(i=0;i<r;i++)
```

```
{
   for(j=0;j<c;j++)
      scanf("%d",&m[i][j]);
}
if(r>0 && c>0)
{
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         t[j][i]=m[i][j];
   }
   printf("\nOriginal matrix (%d x %d):\n", r, c);
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         printf("%d ",m[i][j]);
      printf("\n");
   }
   printf("\nTransposed matrix (%d x %d):\n", c, r);
   for(i=0;i<c;i++)
   {
      for(j=0;j<r;j++)
         printf("%d ",t[i][j]);
      printf("\n");
   }
}
else
```

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

O/P:

Enter the number of rows and columns of the matrix:

3 2

Enter the elements of matrix:

1 2

3 4

5 6

Original matrix (3 x 2):

1 2

3 4

5 6

Transposed matrix (2 x 3):

1 3 5

2 4 6

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 S 6
static int t[S][S];

void main()
{
   int r, c, m[S][S], i, j, m1;
   printf("Enter the number of rows and columns of the matrix:\n");
   scanf("%d %d",&r,&c);
   int mr[r];
   printf("Enter the elements of matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         scanf("%d",&m[i][j]);
   }
   for(i=0;i<r;i++)
   {
      m1=m[i][0];
      for(j=1;j<c;j++)
```

```
        {
            if(m[i][j]>m1)
                m1=m[i][j];
        }
        mr[i]=m1;
    }
    printf("\nMaximum element in each row:\n");
    for(i=0;i<r;i++)
        printf("Row %d: %d\n", i+1,mr[i]);
}
```

O/P:

Enter the number of rows and columns of the matrix:

3 3

Enter the elements of matrix:

1 2 5

7 5 4

5 8 6

Maximum element in each row:

Row 1: 5

Row 2: 7

Row 3: 8

4. Matrix Multiplication

- Problem Statement: Write a program to multiply two matrices. The program should:

- Take two matrices as input:
  - Matrix A of size M x N
  - Matrix B of size N x P
- Use const variables to define the dimensions M, N, and P.
- Use nested for loops to calculate the product of the matrices.
- Use a static two-dimensional array to store the resulting matrix.
- Use if statements to validate that the matrices can be multiplied (N in Matrix A must equal M in Matrix B).

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

```
#include <stdio.h>
#define S 6
static int r[S][S];

void mul(int r1, int c1, int r2, int c2, int m1[S][S], int m2[S][S])
{
   int i,j,k;
   printf("\nMultiplication of 2 matrices:\n");
   for(i=0;i<r1;i++)
   {
      for(j=0;j<c2;j++)
      {
         r[i][j]=0;
         for(k=0;k<c1;k++)
            r[i][j]+=m1[i][k]*m2[k][j];
```

```
        }
    }
    printf("Resultant Matrix:\n");
    for(i=0;i<r1;i++)
    {
        for(j=0;j<c2;j++)
            printf("%d ",r[i][j]);
        printf("\n");
    }
}

void main()
{
    int r1,c1,r2,c2,m1[S][S],m2[S][S],i,j;
    printf("Enter the number of rows and columns of first matrix:\n");
    scanf("%d %d",&r1,&c1);
      printf("Enter the number of rows and columns of second matrix:\n");
    scanf("%d %d",&r2,&c2);
    if(c1!=r2)
    {
        printf("Multiplication of 2 matrices is not possible\n");
        return;
    }
    else
    {
        printf("Enter the elements of first matrix:\n");
        for(i=0;i<r1;i++)
```

```
        {
            for(j=0;j<c1;j++)
                scanf("%d",&m1[i][j]);
        }
        printf("Enter the elements of second matrix:\n");
        for(i=0;i<r2;i++)
        {
            for(j=0;j<c2;j++)
                scanf("%d",&m2[i][j]);
        }
    }
    mul(r1,c1,r2,c2,m1,m2);
}
```

O/P:

```
Enter the number of rows and columns of first matrix:
3 3
Enter the number of rows and columns of second matrix:
3 3
Enter the elements of first matrix:
1 3 2
2 1 3
3 2 1
Enter the elements of second matrix:
1 2 3
3 1 2
2 3 1
```

Multiplication of 2 matrices:

Resultant Matrix:

14 11 11

11 14 11

11 11 14

5. Count Zeros in a Sparse Matrix

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

```
#include <stdio.h>
#define S 6
static int r[S][S];

void main()
```

```
{
   int r, c, i, j, zc=0, t;
   const int m[S][S];
   printf("Enter the number of rows and columns of the matrix:\n");
   scanf("%d %d",&r,&c);
   printf("Enter the elements of  matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         scanf("%d",&m[i][j]);
   }
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
         if(m[i][j]==0)
            zc++;
      }
   }
   t=r*c;
   if(zc>t/2)
      printf("Sparse matrix\n");
   else
      printf("Not a sparse matrix\n");
}
```

O/P:

Enter the number of rows and columns of the matrix:

3 3

Enter the elements of matrix:

1 0 0

0 0 0

5 0 4

Sparse matrix

Enter the number of rows and columns of the matrix:

3 3

Enter the elements of matrix:

4 5 7

1 0 5

0 2 0

Not a sparse matrix

Problem Statements on 3 Dimensional Arrays

1. 3D Matrix Addition

- Problem Statement: Write a program to perform element-wise addition of two three-dimensional matrices. The program should:
    - Take two matrices as input, each of size X x Y x Z, where X, Y, and Z are defined using const variables.
    - Use a static three-dimensional array to store the resulting matrix.
    - Use nested for loops to iterate through the elements of the matrices.

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

- Requirements:
  - Declare matrix dimensions as const variables.
  - Use decision-making statements to handle mismatched dimensions.
  - Print the resulting matrix after addition.

```
#include <stdio.h>
#define S 6
static int r[S][S][S];

void main()
{
   int r1, c1, r2, c2, i, j, k, d1, d2;
   const int m1[S][S][S], m2[S][S][S];
   printf("Enter the depth, number of rows and columns of the first matrix:\n");
   scanf("%d %d %d",&d1,&r1,&c1);
  printf("Enter the depth, number of rows and columns of the second matrix:\n");
   scanf("%d %d %d",&d2,&r2,&c2);
   printf("Enter the elements of first matrix:\n");
   for(k=0;k<d1;k++)
   {
      for(i=0;i<r1;i++)
      {
         for(j=0;j<c1;j++)
            scanf("%d",&m1[k][i][j]);
```

```
    }
}
printf("Enter the elements of second matrix:\n");
for(k=0;k<d2;k++)
{
    for(i=0;i<r2;i++)
    {
        for(j=0;j<c2;j++)
            scanf("%d",&m2[k][i][j]);
    }
}
if(d1==d2 && r1==r2 && c1==c2)
{
    for(k=0;k<d1;k++)
    {
        for(i=0;i<r1;i++)
        {
            for(j=0;j<c1;j++)
                r[k][i][j]=m1[k][i][j]+m2[k][i][j];
        }
    }
    printf("Resultant Matrix:\n");
    for(k=0;k<d1;k++)
    {
        for(i=0;i<r1;i++)
        {
            for(j=0;j<c1;j++)
```

```
                printf("%d ",r[k][i][j]);
            printf("\n");
        }
    }
}
else
    printf("Addition of matrices is not possible\n");
}
```

O/P:

Enter the depth, number of rows and columns of the first matrix:

2 3 2

Enter the depth, number of rows and columns of the second matrix:

2 3 2

Enter the elements of first matrix:

1 2

3 4

5 6

7 8

9 10

11 12

Enter the elements of second matrix:

1 1

1 1

1 1

1 1

1 1

1 1

Resultant Matrix:

2 3

4 5

6 7

8 9

10 11

12 13

2. Find the Maximum Element in a 3D Array

- Problem Statement: Write a program to find the maximum element in a three-dimensional matrix. The program should:
    - Take a matrix of size X x Y x Z as input, where X, Y, and Z are declared as const variables.
    - Use a static variable to store the maximum value found.
    - Use nested for loops to traverse all elements of the matrix.
    - Use if statements to compare and update the maximum value.
- Requirements:
    - Print the maximum value found in the matrix.
    - Handle edge cases where the matrix might contain all negative numbers or zeros using decision-making statements.

```
#include <stdio.h>

#define S 6

static int r[S][S][S];


void main()
```

```
{
   int r, c, i, j, k, d;
   const int m[S][S][S];
   static int m1;
     printf("Enter the depth, number of rows and columns of matrix:\n");
   scanf("%d %d %d",&d,&r,&c);
   printf("Enter the elements of matrix:\n");
   for(k=0;k<d;k++)
   {
      for(i=0;i<r;i++)
      {
         for(j=0;j<c;j++)
            scanf("%d",&m[k][i][j]);
      }
   }
   m1=m[0][0][0];
   for(k=0;k<d;k++)
   {
      for(i=0;i<r;i++)
      {
         for(j=0;j<c;j++)
         {
            if(m[k][i][j] > m1)
               m1=m[k][i][j];
         }
      }
   }
```

```
        printf("The maximum value in the matrix is: %d\n",m1);
    }
```

O/P:

Enter the depth, number of rows and columns of matrix:

3 2 2

Enter the elements of matrix:

1

-2

3

4

-5

6

-7

8

0

9

-10

11

The maximum value in the matrix is: 11

3. 3D Matrix Scalar Multiplication

- Problem Statement: Write a program to perform scalar multiplication on a three-dimensional matrix. The program should:
    - Take a matrix of size X x Y x Z and a scalar value as input, where X, Y, and Z are declared as const variables.
    - Use a static three-dimensional array to store the resulting matrix.

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

- Requirements:
  - Print the original matrix and the resulting matrix after scalar multiplication.
  - Use decision-making statements to handle invalid scalar values (e.g., zero or negative scalars) if necessary.

```
#include <stdio.h>
#define S 6
static int r1[S][S][S];

void main()
{
   int r, c, i, j, k, d, s;
   const int m[S][S][S];
    printf("Enter the number of rows columns and depth of matrix:\n");
   scanf("%d %d %d",&r,&c,&d);
   printf("Enter the elements of matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
         for(k=0;k<d;k++)
            scanf("%d",&m[i][j][k]);
      }
   }
   printf("Enter the scalar:\n");
```

```
scanf("%d",&s);
if(s<=0)
{
   printf("Scalar value must be positive or non zero\n");
   return;
}
for(i=0;i<r;i++)
{
   for(j=0;j<c;j++)
   {
      for(k=0;k<d;k++)
         r1[i][j][k]=m[i][j][k]*s;
   }
}
printf("Original matrix:\n");
for(i=0;i<r;i++)
{
   for(j=0;j<c;j++)
   {
      for(k=0;k<d;k++)
         printf("%d  ",m[i][j][k]);
      printf("\n");
   }
   printf("\n");
}
printf("Resultant matrix:\n");
for(i=0;i<r;i++)
```

```
        {
            for(j=0;j<c;j++)
            {
                for(k=0;k<d;k++)
                    printf("%d  ",r1[i][j][k]);
                printf("\n");
            }
            printf("\n");
        }
    }
```

O/P:

Enter the number of rows columns and depth of matrix:

2 2 3

Enter the elements of matrix:

1

2

3

4

5

6

7

8

9

10

11

12

Enter the scalar:

2

Original matrix:

1 2 3

4 5 6

7 8 9

10 11 12

Resultant matrix:

2 4 6

8 10 12

14 16 18

20 22 24

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

- Problem Statement: Write a program to count the number of positive, negative, and zero elements in a three-dimensional matrix. The program should:
    - Take a matrix of size X x Y x Z as input, where X, Y, and Z are defined using const variables.
    - Use three static variables to store the counts of positive, negative, and zero elements, respectively.
    - Use nested for loops to traverse the matrix.
    - Use if-else statements to classify each element.
- Requirements:
    - Print the counts of positive, negative, and zero elements.

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

```
#include <stdio.h>
#define S 6
static int r1[S][S][S];

void main()
{
   int r, c, i, j, k, d;
   const int m[S][S][S];
   static int pc=0, nc=0, zc=0;
     printf("Enter the number of rows columns and depth of matrix:\n");
   scanf("%d %d %d",&r,&c,&d);
   printf("Enter the elements of matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
         for(k=0;k<d;k++)
            scanf("%d",&m[i][j][k]);
      }
   }
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
```

```
                for(k=0;k<d;k++)
                {
                    if(m[i][j][k] > 0)
                        pc++;
                    else if(m[i][j][k] < 0)
                        nc++;
                    else
                        zc++;
                }
            }
        }
        printf("Positive elements: %d\n", pc);
        printf("Negative elements: %d\n", nc);
        printf("Zero elements: %d\n", zc);
    }
```

O/P:

```
Enter the number of rows columns and depth of matrix:
2 2 3
Enter the elements of matrix:
1
-2
4
-5
0
0
-8
```

4

9

0

-7

6

Positive elements: 5

Negative elements: 4

Zero elements: 3

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

- Problem Statement: Write a program to compute the transpose of a three-dimensional matrix along a specific axis (e.g., swap rows and columns for a specific depth). The program should:
    - Take a matrix of size X x Y x Z as input, where X, Y, and Z are defined using const variables.
    - Use a static three-dimensional array to store the transposed matrix.
    - Use nested for loops to perform the transpose operation along the specified axis.
    - Use if statements to validate the chosen axis for transposition.
- Requirements:
    - Print the original matrix and the transposed matrix.
    - Ensure invalid axis values are handled using decision-making constructs.

```
#include <stdio.h>

#define S 6


void main()
```

```
{
   int r, c, i, j, k, d, a;
   const int m[S][S][S];
   static int t[S][S][S];
     printf("Enter the number of rows columns and depth of matrix:\n");
   scanf("%d %d %d",&r,&c,&d);
   printf("Enter the elements of matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
         for(k=0;k<d;k++)
            scanf("%d",&m[i][j][k]);
      }
   }
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
      {
         for(k=0;k<d;k++)
            t[j][i][k]=m[i][j][k];
      }
   }
   printf("Original matrix:\n");
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
```

```
            {
                for(k=0;k<d;k++)
                    printf("%d  ",m[i][j][k]);
                printf("\n");
            }
            printf("\n");
        }
        printf("Transposed matrix:\n");
        for(i=0;i<c;i++)
        {
            for(j=0;j<r;j++)
            {
                for(k=0;k<d;k++)
                    printf("%d  ",t[i][j][k]);
                printf("\n");
            }
            printf("\n");
        }
    }
```

O/P:

Enter the number of rows columns and depth of matrix:

2 3 2

Enter the elements of matrix:

1

2

3

4

5

6

7

8

9

10

11

12

Original matrix:

1 2

3 4

5 6

7 8

9 10

11 12

Transposed matrix:

1 2

7 8

3 4

9 10

5 6

11 12