Assignment – 7

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.

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

void findMaxMin(int arr[], int size);

int main() {

    const int ARRAY_SIZE = 5;

    int numbers[ARRAY_SIZE];

    printf("Enter %d integers:\n", ARRAY_SIZE);

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

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

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

    }

    findMaxMin(numbers, ARRAY_SIZE);



    return 0;
}

void findMaxMin(int arr[], int size) {
    int max = arr[0], min = arr[0];
    static int maxDifference = 0;
        for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    maxDifference = max - min;
    printf("Maximum value: %d\n", max);
    printf("Minimum value: %d\n", min);
    printf("Maximum difference (max - min): %d\n", maxDifference);
}
```

```
O/p: Enter 5 integers:
Element 1: 10
Element 2: 35

Element 3: 20

Element 4: 40

Element 5: 12

Maximum value: 40

Minimum value: 10

Maximum difference (max - min): 30
```

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.

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

void categorizeElements(int arr[], int size);

int main() {

    const int ARRAY_SIZE = 10;

    int numbers[ARRAY_SIZE];

    printf("Enter %d integers:\n", ARRAY_SIZE);

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

        printf("Element %d: ", i + 1);
        scanf("%d", &numbers[i]);
    }
    categorizeElements(numbers, ARRAY_SIZE);

    return 0;
}

void categorizeElements(int arr[], int size) {
    static int positives[100], negatives[100], zeros[100];
    int posCount = 0, negCount = 0, zeroCount = 0;
    for (int i = 0; i < size; i++) {
        if (arr[i] > 0) {
            positives[posCount++] = arr[i];
        } else if (arr[i] < 0) {
            negatives[negCount++] = arr[i];
        } else {
            zeros[zeroCount++] = arr[i];
        }
    }
    printf("\nPositive elements: ");
    for (int i = 0; i < posCount; i++) {
        printf("%d ", positives[i]);
    }

    printf("\nNegative elements: ");
    for (int i = 0; i < negCount; i++) {
        printf("%d ", negatives[i]);
    }

    printf("\nZero elements: ");
    for (int i = 0; i < zeroCount; i++) {
        printf("%d ", zeros[i]);
    }
    printf("\n");
}
```

O/p:
```
Enter 10 integers:
Element 1: 10
Element 2: -2
Element 3: -6

Element 4: 53

Element 5: 25

Element 6: 0

Element 7: 34

Element 8: -9

Element 9: 23

Element 10: 34


Positive elements: 10 53 25 34 23 34

Negative elements: -2 -6 -9

Zero elements: 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.

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

void calculateCumulativeSum(int arr[], int size);
```

```
int main() {
    const int ARRAY_SIZE = 5;
    int numbers[ARRAY_SIZE];
    printf("Enter %d integers:\n", ARRAY_SIZE);
    for (int i = 0; i < ARRAY_SIZE; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &numbers[i]);
    }
    calculateCumulativeSum(numbers, ARRAY_SIZE);

    return 0;
}

void calculateCumulativeSum(int arr[], int size) {
    static int cumulativeSum = 0; // Static variable for running total

    printf("\nCumulative Sum:\n");
    for (int i = 0; i < size; i++) {
        cumulativeSum += arr[i];
        printf("After element %d (%d): %d\n", i + 1, arr[i],
cumulativeSum);
    }
}
```

O/p:

```
Enter 5 integers:
Element 1: 23
Element 2: 12
Element 3: -7
Element 4: 12
Element 5: 65

Cumulative Sum:
After element 1 (23): 23
After element 2 (12): 35
After element 3 (-7): 28
After element 4 (12): 40
After element 5 (65): 105
```

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.

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

#include <stdbool.h>

#define SIZE 10

int main() {

    int arr[SIZE] = {2, 3, 4, 5, 6,11, 23, 9, 17,67};

    const int size = SIZE;

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

        bool is_prime = true;

        if (arr[i] <= 1) is_prime = false;

        for (int j = 2; j * j <= arr[i]; j++) {

            if (arr[i] % j == 0) {

                is_prime = false;

                break;

            }

        }

        if (is_prime) printf("%d ", arr[i]);

    }
    printf("\n");

    return 0;

}
```

O/p:

```
2 3 5 11 23 17 67
```

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.

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

  #define SIZE 5

#define N 2

int main() {

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

    const int size = SIZE;

    static int rotated[SIZE];

    int i = 0;

  while (i < size) {

        rotated[i] = arr[(i + N) % size];

        i++;
    }
for (int i = 0; i < size; i++) printf("%d ", rotated[i]);
    printf("\n");
    return 0;
}
```

O/p: 3 4 5 1 2

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.

Sol:
```
#include <stdio.h>
#define SIZE 10
int main() {
    int arr[SIZE] = {1, 2, 2, 3, 3, 3,4,5,5,6,};
    const int size = SIZE;
    static int freq[SIZE];
    int i, j, count;
 for (i = 0; i < size; i++) {
        count = 1;
        if (freq[i] == 0) {
            for (j = i + 1; j < size; j++) {
                if (arr[i] == arr[j]) {
                    count++;
                    freq[j] = -1;
                }
            }
            freq[i] = count;
        }
    }

    for (i = 0; i < size; i++) {
        if (freq[i] != -1) {
            printf("%d occurs %d times\n", arr[i], freq[i]);
        }
    }
    return 0;
}
```

O/p:
```
1 occurs 1 times
2 occurs 2 times
```

```
3 occurs 3 times

4 occurs 1 times

5 occurs 2 times

6 occurs 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.

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

 #define SIZE 5

int main() {

   int arr[SIZE] = {3, 10, 4, 2, 5};

   const int size = SIZE;

   int temp;



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

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

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

               temp = arr[j];
```

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

                arr[j + 1] = temp;

            }

        }

    }


    for (int i = 0; i < size; i++) printf("%d ", arr[i]);

    printf("\n");

    return 0;

}
```

O/p: 10 5 4 3 2

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.

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

 #define SIZE 10

static int secondLargest = -1;

int main() {

    int arr[SIZE] = {12, 34, 54, 2, 3, 45, 6, 7, 8, 9};
```

```
   int largest = arr[0];
    for (int i = 1; i < SIZE; i++) {
        if (arr[i] > largest) {
            secondLargest = largest;
            largest = arr[i];
        } else if (arr[i] > secondLargest && arr[i] != largest) {
            secondLargest = arr[i];
        }
    }

    if (secondLargest == -1) {
        printf("No second largest element found.\n");
    } else {
        printf("Second largest element: %d\n", secondLargest);
    }

    return 0;
}
```

```
O/p: Second largest element: 45
```

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.

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

 #define SIZE 6

int main() {

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

    const int size = SIZE;

    static int odd[SIZE], even[SIZE];

    int odd_idx = 0, even_idx = 0;


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

        if (arr[i] % 2 == 0) even[even_idx++] = arr[i];

        else odd[odd_idx++] = arr[i];

    }


    printf("Odd: ");

    for (int i = 0; i < odd_idx; i++) printf("%d ", odd[i]);

    printf("\nEven: ");

    for (int i = 0; i < even_idx; i++) printf("%d ", even[i]);
    printf("\n");

    return 0;

}
```

O/p: Odd: 1 3 5

Even: 2 4 6

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

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

#define SIZE 5

int main() {

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

    const int size = SIZE;

    static int temp, i;

  temp = arr[size - 1];

    for (i = size - 1; i > 0; i--) {

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

    }
```

```
   arr[0] = temp;


   for (i = 0; i < size; i++) printf("%d ", arr[i]);

   printf("\n");

   return 0;

}
```

O/p: 5 1 2 3 4

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.

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

  #define MAX_TEMP 100

extern int alertTriggered;


void checkTemperature(int temp);
```

```
int alertTriggered = 0;
int main() {
    int temperatures[10];
        static int readingCount = 0;
printf("Enter engine temperatures for 10 intervals (in °C):\n");
     for (int i = 0; i < 10; i++) {
        printf("Temperature reading %d: ", i + 1);
        scanf("%d", &temperatures[i]);
        readingCount++;
    }

    printf("\nChecking temperatures against the safe threshold...\n");
    for (int i = 0; i < 10; i++) {
        printf("Checking reading %d: %d°C\n", i + 1, temperatures[i]);
        checkTemperature(temperatures[i]);
    }

    if (alertTriggered) {
        printf("\nALERT: One or more temperatures exceeded the safe
threshold of %d°C!\n", MAX_TEMP);

    } else {

        printf("\nAll temperatures are within the safe threshold of
%d°C.\n", MAX_TEMP);

    }

     printf("\nTotal readings processed: %d\n", readingCount);



    return 0;

}

void checkTemperature(int temp) {

    if (temp > MAX_TEMP) {

        alertTriggered = 1;

        printf("ALERT: Temperature %d°C exceeds the safe limit of
%d°C!\n", temp, MAX_TEMP);

    }

}
```

O/p:
```
Enter engine temperatures for 10 intervals (in °C):

Temperature reading 1: 101

Temperature reading 2: 35

Temperature reading 3: 42
Temperature reading 4: 10

Temperature reading 5: 55

Temperature reading 6: 26

Temperature reading 7: 20

Temperature reading 8: 12

Temperature reading 9: -2

Temperature reading 10: 37


Checking temperatures against the safe threshold...

Checking reading 1: 101°C

ALERT: Temperature 101°C exceeds the safe limit of 100°C!

Checking reading 2: 35°C

Checking reading 3: 42°C

Checking reading 4: 10°C

Checking reading 5: 55°C

Checking reading 6: 26°C

Checking reading 7: 20°C

Checking reading 8: 12°C

Checking reading 9: -2°C

Checking reading 10: 37°C
```

```
ALERT: One or more temperatures exceeded the safe threshold of
100°C!


Total readings processed: 10
```

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).

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

   #define FUEL_CONSUMPTION 8

    volatile float distance;

   void calculateEfficiency(float distance);

  int main() {

      float distances[10];

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


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

      printf("Trip %d distance (in km): ", i + 1);
```

```
        scanf("%f", &distances[i]);
    }
    printf("\nFuel Efficiency for each trip:\n");
    for (int i = 0; i < 10; i++) {
        printf("Trip %d: Distance = %.2f km, ", i + 1, distances[i]);
        calculateEfficiency(distances[i]);
    }
   return 0;
}
void calculateEfficiency(float distance) {
    float efficiency = distance / (FUEL_CONSUMPTION / 100.0);
    printf("Fuel Efficiency = %.2f km/L", efficiency);
    if (efficiency < 10.0) {
        printf(" (Low Efficiency Alert!)\n");
    } else {
        printf("\n");
    }
}
```

O/p:

```
Enter the distances covered in 10 different trips (in km):
```

```
Trip 1 distance (in km): 100

Trip 2 distance (in km): 40

Trip 3 distance (in km): 150

Trip 4 distance (in km): 35

Trip 5 distance (in km): 41

Trip 6 distance (in km): 200

Trip 7 distance (in km): 150

Trip 8 distance (in km): 78

Trip 9 distance (in km): 90

Trip 10 distance (in km): 2


Fuel Efficiency for each trip:

Trip 1: Distance = 100.00 km, Fuel Efficiency = 1250.00 km/L

Trip 2: Distance = 40.00 km, Fuel Efficiency = 500.00 km/L

Trip 3: Distance = 150.00 km, Fuel Efficiency = 1875.00 km/L

Trip 4: Distance = 35.00 km, Fuel Efficiency = 437.50 km/L

Trip 5: Distance = 41.00 km, Fuel Efficiency = 512.50 km/L

Trip 6: Distance = 200.00 km, Fuel Efficiency = 2500.00 km/L

Trip 7: Distance = 150.00 km, Fuel Efficiency = 1875.00 km/L

Trip 8: Distance = 78.00 km, Fuel Efficiency = 975.00 km/L
Trip 9: Distance = 90.00 km, Fuel Efficiency = 1125.00 km/L

Trip 10: Distance = 2.00 km, Fuel Efficiency = 25.00 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.

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

 #define SIZE 10

#define DEVIATION_THRESHOLD 50

int altitudes[SIZE];

int main() {

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

        printf("Enter altitude reading %d (meters): ", i + 1);

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



        if (i > 0 && (altitudes[i] - altitudes[i - 1]) >
DEVIATION_THRESHOLD) {

            printf("Warning: Altitude deviation exceeds %d meters!\n",
DEVIATION_THRESHOLD);

        }
    }

   return 0;
}
```

```
O/p: Enter altitude reading 1 (meters): 1000
Enter altitude reading 2 (meters): 200
Enter altitude reading 3 (meters): 1045
Warning: Altitude deviation exceeds 50 meters!
Enter altitude reading 4 (meters): 344
Enter altitude reading 5 (meters): 1500
Warning: Altitude deviation exceeds 50 meters!
Enter altitude reading 6 (meters): 500
Enter altitude reading 7 (meters): 600
Warning: Altitude deviation exceeds 50 meters!
Enter altitude reading 8 (meters): 900
Warning: Altitude deviation exceeds 50 meters!
Enter altitude reading 9 (meters): 1203
Warning: Altitude deviation exceeds 50 meters!
Enter altitude reading 10 (meters): 1000
```

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.

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

#define SIZE 10

#define ORBIT_RADIUS 30000

#define DEVIATION_LIMIT 1000

const int orbit_radius = ORBIT_RADIUS;

int positions[SIZE];

int main() {

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

        printf("Enter satellite position reading %d (km): ", i + 1);

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



        if (i > 0 && (positions[i] - positions[i - 1]) > DEVIATION_LIMIT)
{

            printf("Warning: Satellite position deviation exceeds %d km!\n",
DEVIATION_LIMIT);

        }
    }

    return 0;
}
```

```
O/p: Enter satellite position reading 1 (km): 30000
Enter satellite position reading 2 (km): 29000
Enter satellite position reading 3 (km): 35000
Warning: Satellite position deviation exceeds 1000 km!
Enter satellite position reading 4 (km): 33000
Enter satellite position reading 5 (km): 32000
Enter satellite position reading 6 (km): 40900
Warning: Satellite position deviation exceeds 1000 km!
Enter satellite position reading 7 (km): 50000
Warning: Satellite position deviation exceeds 1000 km!
Enter satellite position reading 8 (km): 34000
Enter satellite position reading 9 (km): 31000
Enter satellite position reading 10 (km): 6 49000
Warning: Satellite position deviation exceeds 1000 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.

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

#define SIZE 10

static int abnormal_count = 0;

int heart_rates[SIZE];



int main() {

    int sum = 0;

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

        printf("Enter heart rate for session %d (BPM): ", i + 1);

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



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

            abnormal_count++;

        }

        sum += heart_rates[i];
    }


    float average = sum / (float)SIZE;
    printf("Average Heart Rate: %.2f BPM\n", average);
    printf("Number of abnormal readings: %d\n", abnormal_count);


    return 0;
}
```

O/p:
```
Enter heart rate for session 1 (BPM): 88
Enter heart rate for session 2 (BPM): 80
Enter heart rate for session 3 (BPM): 120
Enter heart rate for session 4 (BPM): 100
Enter heart rate for session 5 (BPM): 60
Enter heart rate for session 6 (BPM): 65
Enter heart rate for session 7 (BPM): 110
Enter heart rate for session 8 (BPM): 34
Enter heart rate for session 9 (BPM): 86
Enter heart rate for session 10 (BPM): 50
Average Heart Rate: 79.30 BPM
Number of abnormal readings: 4
```

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

Sol:
```
#include <stdio.h>
#define SIZE 10
volatile int weight[SIZE], age[SIZE];
int dosages[SIZE];
int main() {
    for (int i = 0; i < SIZE; i++) {
        printf("Enter weight and age for patient %d (kg, years): ", i + 1);
        scanf("%d %d", &weight[i], &age[i]);

        if (weight[i] < 50 || age[i] < 18 || weight[i] > 100) {
            printf("Invalid dosage for patient %d\n", i + 1);
        } else {
            printf("Valid dosage for patient %d\n", i + 1);
        }
    }
     return 0;
}
```

```
O/p: Enter weight and age for patient 1 (kg, years): 78 56
Valid dosage for patient 1
Enter weight and age for patient 2 (kg, years): 55 35
Valid dosage for patient 2
Enter weight and age for patient 3 (kg, years): 90 40
Valid dosage for patient 3
Enter weight and age for patient 4 (kg, years): 77 39
Valid dosage for patient 4
Enter weight and age for patient 5 (kg, years): 35 18
Invalid dosage for patient 5
Enter weight and age for patient 6 (kg, years): 18 45
Invalid dosage for patient 6
Enter weight and age for patient 7 (kg, years): 80 66
Valid dosage for patient 7
Enter weight and age for patient 8 (kg, years): 90 18
Valid dosage for patient 8
Enter weight and age for patient 9 (kg, years): 35 16
```

```
Invalid dosage for patient 9

Enter weight and age for patient 10 (kg, years): 23 8

Invalid dosage for patient 10
```

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.

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

  #define SIZE 10

#define REORDER_THRESHOLD 5

static int low_inventory_count = 0;

int inventory[SIZE];

int main() {

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

        printf("Enter inventory level for product %d: ", i + 1);

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


        if (inventory[i] < REORDER_THRESHOLD) {

            printf("Reorder needed for product %d!\n", i + 1);
            low_inventory_count++;
        }
    }

    printf("Total products needing reorder: %d\n", low_inventory_count);

    return 0;
}
```

```
O/p: Enter inventory level for product 1: 1
Reorder needed for product 1!
Enter inventory level for product 2: 8
Enter inventory level for product 3: 7
Enter inventory level for product 4: 9
Enter inventory level for product 5: 3
Reorder needed for product 5!
Enter inventory level for product 6: 2
Reorder needed for product 6!
Enter inventory level for product 7: 1
Reorder needed for product 7!
Enter inventory level for product 8: 0
```

```
Reorder needed for product 8!
Enter inventory level for product 9: 8
Enter inventory level for product 10: 10
Total products needing reorder: 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.

Sol:
```
#include <stdio.h>
#include <string.h>
#define SIZE 10
#define VALID_CODE_LENGTH 6

const int valid_code_length = VALID_CODE_LENGTH;
char codes[SIZE][VALID_CODE_LENGTH + 1];
static int invalid_codes = 0;

int main() {
    for (int i = 0; i < SIZE; i++) {
        printf("Enter missile launch code %d: ", i + 1);
        scanf("%s", codes[i]);

        if (strlen(codes[i]) != valid_code_length) {
            invalid_codes++;
            printf("Invalid code length for code %d\n", i + 1);
        }
    }

    printf("Number of invalid codes: %d\n", invalid_codes);

    return 0;
}
```

O/p:
```
Enter missile launch code 1: ABCDE
Invalid code length for code 1
Enter missile launch code 2: 12AB123A
Invalid code length for code 2
Enter missile launch code 3: ABCDEF
Enter missile launch code 4: 125ADE
Enter missile launch code 5: H67123
Enter missile launch code 6: 123AD

Invalid code length for code 6

Enter missile launch code 7: 432

Invalid code length for code 7

Enter missile launch code 8: 1230AD

Enter missile launch code 9: ABCD12

Enter missile launch code 10: AZXS

Invalid code length for code 10

Number of invalid codes: 5
```

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.

Sol:
```c
#include <stdio.h>

#define SIZE 10

#define FRIENDLY_DISTANCE_THRESHOLD 100



int distance_threshold = FRIENDLY_DISTANCE_THRESHOLD;

int target_positions[SIZE];
int main() {
    for (int i = 0; i < SIZE; i++) {
        printf("Enter target position %d : ", i + 1);
        scanf("%d", &target_positions[i]);

        if (target_positions[i] < distance_threshold) {
            printf("Target %d is friendly.\n", i + 1);
        } else {
            printf("Target %d is hostile.\n", i + 1);
        }
    }

    return 0;
}
```

O/p:
```
Enter target position 1 : 100
Target 1 is hostile.
Enter target position 2 : 50
Target 2 is friendly.
Enter target position 3 : 67
```

```
Target 3 is friendly.

Enter target position 4 : 45

Target 4 is friendly.

Enter target position 5 : 120

Target 5 is hostile.

Enter target position 6 : 150

Target 6 is hostile.

Enter target position 7 : 67

Target 7 is friendly.

Enter target position 8 : 43

Target 8 is friendly.

Enter target position 9 : 87

Target 9 is friendly.

Enter target position 10 : 9

Target 10 is friendly.
```

Problem Statements on 2 Dimensional Arrays

1. Matrix Addition

- **Problem Statement**: Write a program to perform the addition of two matrices. The program should:

    - Take two matrices as input, each of size M x N, where M and N are defined using const variables.
    - Use a static two-dimensional array to store the resulting matrix.
    - Use nested for loops to perform element-wise addition.
    - Use if statements to validate that the matrices have the same dimensions before proceeding with the addition.
- **Requirements**:
    - Declare matrix dimensions as const variables.
    - Use decision-making constructs to handle invalid dimensions.
    - Print the resulting matrix after addition.

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

#define M 3

#define N 3

int matrixA[M][N], matrixB[M][N], result[M][N];

int main() {

  printf("Enter elements for Matrix A:\n");

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

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

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

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

    }

  }

  printf("Enter elements for Matrix B:\n");
```

```
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        printf("Matrix B[%d][%d]: ", i + 1, j + 1);
        scanf("%d", &matrixB[i][j]);
    }
}
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        result[i][j] = matrixA[i][j] + matrixB[i][j];
    }
}
printf("Resulting Matrix after Addition:\n");
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        printf("%d ", result[i][j]);
    }
    printf("\n");
}

return 0;
```

}

O/p:

```
Enter elements for Matrix A:

Matrix A[1][1]: 1

Matrix A[1][2]: 3

Matrix A[1][3]: 4

Matrix A[2][1]: 5

Matrix A[2][2]: 6

Matrix A[2][3]: 7

Matrix A[3][1]: 8

Matrix A[3][2]: 4

Matrix A[3][3]: 12

Enter elements for Matrix B:

Matrix B[1][1]: 1

Matrix B[1][2]: 9

Matrix B[1][3]: 7

Matrix B[2][1]: 8

Matrix B[2][2]: 6

Matrix B[2][3]: 5

Matrix B[3][1]: 32
```

Matrix B[3][2]: 1

Matrix B[3][3]: 22

Resulting Matrix after Addition:

2 12 11

13 12 12

40 5 34

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.

Sol: #include <stdio.h>

#define M 3

#define N 3

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

```
int main() {
    printf("Enter elements for the Matrix:\n");
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("Matrix[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
    printf("Original Matrix:\n");
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    printf("Transposed Matrix:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

O/p: Enter elements for the Matrix:

```
Matrix[1][1]: 23
Matrix[1][2]: 2
Matrix[1][3]: 4
Matrix[2][1]: 5
Matrix[2][2]: 6
Matrix[2][3]: 7
Matrix[3][1]: 4
Matrix[3][2]: 5
```

Matrix[3][3]: 7

Original Matrix:

23 2 4

5 6 7

4 5 7

Transposed Matrix:

23 5 4

2 6 5

4 7 7

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.

Sol: #include <stdio.h>

#define M 3

```
#define N 3

int matrix[M][N];

int main() {

    printf("Enter elements for the Matrix:\n");

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

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

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

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

        }

    }

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

        int max = matrix[i][0];

        for (int j = 1; j < N; j++) {

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

                max = matrix[i][j];

            }

        }

        printf("Maximum in row %d: %d\n", i + 1, max);

    }
    return 0;
}
```

```
O/p: Enter elements for the Matrix:
Matrix[1][1]: 1
Matrix[1][2]: 5
Matrix[1][3]: 12
Matrix[2][1]: 22
Matrix[2][2]: 3
Matrix[2][3]: 4
Matrix[3][1]: 7
Matrix[3][2]: 6
Matrix[3][3]: 8
Maximum in row 1: 12
Maximum in row 2: 22
Maximum in 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.

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

#define M 3

#define N 3

#define P 3

int matrixA[M][N], matrixB[N][P], result[M][P];

int main() {

    printf("Enter elements for Matrix A:\n");

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

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

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

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

        }

    }
```

```
printf("Enter elements for Matrix B:\n");
for (int i = 0; i < N; i++) {
    for (int j = 0; j < P; j++) {
        printf("Matrix B[%d][%d]: ", i + 1, j + 1);
        scanf("%d", &matrixB[i][j]);
    }
}
for (int i = 0; i < M; i++) {
    for (int j = 0; j < P; j++) {
        result[i][j] = 0;
        for (int k = 0; k < N; k++) {
            result[i][j] += matrixA[i][k] * matrixB[k][j];
        }
    }
}
printf("Matrix A:\n");
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        printf("%d ", matrixA[i][j]);
    }
    printf("\n");
}

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

printf("Resulting Matrix after Multiplication:\n");
for (int i = 0; i < M; i++) {
    for (int j = 0; j < P; j++) {
        printf("%d ", result[i][j]);
    }
    printf("\n");
}

return 0;
```

```
}
```

O/p:
```
Enter elements for Matrix A:

Matrix A[1][1]: 2

Matrix A[1][2]: 3

Matrix A[1][3]: 1

Matrix A[2][1]: 11

Matrix A[2][2]: 6

Matrix A[2][3]: 7

Matrix A[3][1]: 4

Matrix A[3][2]: 5

Matrix A[3][3]: 8

Enter elements for Matrix B:

Matrix B[1][1]: 34

Matrix B[1][2]: 21

Matrix B[1][3]: 2

Matrix B[2][1]: 3

Matrix B[2][2]: 12

Matrix B[2][3]: 4

Matrix B[3][1]: 5

Matrix B[3][2]: 6
```

Matrix B[3][3]: 7

Matrix A:

2 3 1

11 6 7

4 5 8

Matrix B:

34 21 2

3 12 4

5 6 7

Resulting Matrix after Multiplication:

82 84 23

427 345 95

191 192 84

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.

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

  #define M 3

#define N 3

int matrix[M][N];



int main() {

    printf("Enter elements for the Matrix:\n");

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

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

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

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

        }

    }



    int zeroCount = 0;

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

        for (int j = 0; j < N; j++) {
            if (matrix[i][j] == 0) {
                zeroCount++;
            }
        }
    }
    if (zeroCount > (M * N) / 2) {
        printf("The matrix is sparse.\n");
    } else {
        printf("The matrix is not sparse.\n");
    }

    return 0;
}
```

O/p:

```
Enter elements for the Matrix:
Matrix[1][1]: 0
Matrix[1][2]: 0
Matrix[1][3]: 1
Matrix[2][1]: 0
Matrix[2][2]: 0
```

```
Matrix[2][3]: 0

Matrix[3][1]: 0

Matrix[3][2]: 2

Matrix[3][3]: 0

The matrix is sparse.
```

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.

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

#define X 2

#define Y 2
```

```
#define Z 2

int matrixA[X][Y][Z], matrixB[X][Y][Z], result[X][Y][Z];

int main() {

    printf("Enter elements for Matrix A:\n");

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

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

            for (int k = 0; k < Z; k++) {

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

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

            }

        }

    }

   printf("Enter elements for Matrix B:\n");

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

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

            for (int k = 0; k < Z; k++) {

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

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

            }

        }
    }
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                result[i][j][k] = 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("%d ", result[i][j][k]);
            }
            printf("\n");
        }
    }
   return 0;
}
```

O/p:

```
Enter elements for Matrix A:

Matrix A[1][1][1]: 1

Matrix A[1][1][2]: 2

Matrix A[1][2][1]: 3

Matrix A[1][2][2]: 6

Matrix A[2][1][1]: 7

Matrix A[2][1][2]: 8

Matrix A[2][2][1]: 9

Matrix A[2][2][2]: 12

Enter elements for Matrix B:

Matrix B[1][1][1]: 34

Matrix B[1][1][2]: 32

Matrix B[1][2][1]: 12

Matrix B[1][2][2]: 21

Matrix B[2][1][1]: 6

Matrix B[2][1][2]: 9

Matrix B[2][2][1]: 6

Matrix B[2][2][2]: 5

Resulting Matrix after Addition:

35 34
```

```
15 27

13 17

15 17
```

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.

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

#define X 2

#define Y 2

#define Z 2

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

int main() {

    printf("Enter elements for the 3D Matrix:\n");

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

        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("Matrix[%d][%d][%d]: ", i + 1, j + 1, k + 1);
                scanf("%d", &matrix[i][j][k]);
            }
        }
    }
 int max = matrix[0][0][0];
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                if (matrix[i][j][k] > max) {
                    max = matrix[i][j][k];
                }
            }
        }
    }
 printf("Maximum element in the 3D matrix: %d\n", max);
    return 0;
}
```

O/p:

```
Enter elements for the 3D Matrix:

Matrix[1][1][1]: 1

Matrix[1][1][2]: 25

Matrix[1][2][1]: 6

Matrix[1][2][2]: 7

Matrix[2][1][1]: 4

Matrix[2][1][2]: 8

Matrix[2][2][1]: 9

Matrix[2][2][2]: 3

Maximum element in the 3D matrix: 25
```

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.

```
Sol: #include <stdio.h>
#define X 2
#define Y 2
#define Z 2
int matrix[X][Y][Z], result[X][Y][Z];
int scalar;
int main() {
    printf("Enter scalar value: ");
    scanf("%d", &scalar);
   printf("Enter elements for the 3D Matrix:\n");
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("Matrix[%d][%d][%d]: ", i + 1, j + 1, k + 1);
                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++) {
            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("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");
        }
    }

    return 0;
}
```

```
O/p: Enter scalar value: 23
Enter elements for the 3D Matrix:
Matrix[1][1][1]: 2
Matrix[1][1][2]: 2
Matrix[1][2][1]: 3
Matrix[1][2][2]: 1
Matrix[2][1][1]: 4
Matrix[2][1][2]: 5
Matrix[2][2][1]: 6
Matrix[2][2][2]: 7
Original Matrix:
2 2
3 1
```

4 5

6 7

Resulting Matrix after Scalar Multiplication:

46 46

69 23

92 115

138 161

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.

Sol: #include <stdio.h>

#define X 2

#define Y 2

```
#define Z 2

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

int positive = 0, negative = 0, zero = 0;

int main() {

    printf("Enter elements for the 3D Matrix:\n");

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

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

            for (int k = 0; k < Z; k++) {

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

                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) {

                    positive++;

                } else if (matrix[i][j][k] < 0) {

                    negative++;
                } else {
                    zero++;
                }
            }
        }
    }

    printf("Positive elements: %d\n", positive);
    printf("Negative elements: %d\n", negative);
    printf("Zero elements: %d\n", zero);

    return 0;
}
```

O/p:

```
Enter elements for the 3D Matrix:
Matrix[1][1][1]: 1
Matrix[1][1][2]: 2
Matrix[1][2][1]: 3
Matrix[1][2][2]: 0
Matrix[2][1][1]: -9
```

Matrix[2][1][2]: 8

Matrix[2][2][1]: 6

Matrix[2][2][2]: 0

Positive elements: 5

Negative elements: 1

Zero elements: 2

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.

Sol: #include <stdio.h>

#define X 2

#define Y 2

#define Z 2

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


int main() {
    printf("Enter elements for the 3D Matrix:\n");
    for (int i = 0; i < X; i++) {
        for (int j = 0; j < Y; j++) {
            for (int k = 0; k < Z; k++) {
                printf("Matrix[%d][%d][%d]: ", i + 1, j + 1, k + 1);
                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++) {
                transpose[j][i][k] = matrix[i][j][k];
            }
        }
    }
```

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

```
    return 0;

}

O/p: Enter elements for the 3D Matrix:

Matrix[1][1][1]: 12

Matrix[1][1][2]: 3

Matrix[1][2][1]: 4

Matrix[1][2][2]: 5

Matrix[2][1][1]: 3

Matrix[2][1][2]: 5

Matrix[2][2][1]: 6

Matrix[2][2][2]: 7

Original Matrix:

12 3

4 5

3 5

6 7

Transposed Matrix:

12 3

3 5

4 5
```