Core Programming Solution Group A

Q1) Print the following in the console
"Code Olympiad 2081\n"

```c
#include <stdio.h>
int main()
{
    system("cls");
    printf("\"Code Olympiad 2081\"\\n");
    return 0;
}
```

```
Output : "Code Olympiad 2081\n"
```

Q2)

```c
/*
Test if a given number is an armstrong number
An Armstrong number (or Narcissistic number) is a number that is equal to the sum of its own
digits raised to the power of the number of digits.
153 = 1^3 + 5^3 + 3^3
9474 = 9^4 + 4^4 + 7^4 + 4^4

Single-digit numbers: All single-digit numbers are Armstrong numbers because d^1=d.
*/
#include <stdio.h>
#include <math.h>

int isArmstrong(int num)
{
    int originalNum = num, sum = 0, n = 0;

    while (num != 0)
    {
        n++;
        num /= 10;
    }

    num = originalNum;

    while (num != 0)
    {
        int digit = num % 10;
        sum += pow(digit, n);
        num /= 10;
    }

    return (sum == originalNum);
}

int main()
{
    system("cls");
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (isArmstrong(num))
        printf("True\n");
    else
        printf("False\n");

    return 0;
}
```

Test Cases : Input : 5   Output : True
      Input : 150 Output : False
      Input : 9474 Output : True

Q3)

Swap two char variables without using a 3rd variable.

```c
#include <stdio.h>

int main()
{
    char a = 'A'; // ASCII 65
    char b = 'B'; // ASCII 66

    printf("Before swapping: a = %c, b = %c\n", a, b);

    a = a ^ b; // Step 1: a now holds the XOR of a and b
    b = a ^ b; // Step 2: b now holds the original value of a
    a = a ^ b; // Step 3: a now holds the original value of b

    /* //using addition and substraction
    a = a + b; // Step 1: a becomes the sum of a and b
    b = a - b; // Step 2: b becomes the original value of a
    a = a - b; // Step 3: a becomes the original value of b
    */

    /*  using multiplication and division
    a = a * b; // Step 1: a becomes the product of a and b
     b = a / b; // Step 2: b becomes the original value of a
     a = a / b; // Step 3: a becomes the original value of b
    */

    printf("After swapping: a = %c, b = %c\n", a, b);

    return 0;
}
```

Test case : Input : a=B b = D   Output : a = D   b = B

Q4)

You are given an array containing n-1 integers where each integer is in the range 1 to n. Find the missing number.

for array of size n take n-1 elements in any order and find the missing one
Eg : For an array of size 5 with elements : 1,4,2,5 the missing element is 3

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

```c
int main()
{
    int n, input;

    printf("Enter n : ");
    scanf("%d", &n);

    int num[n + 1]; // +1 to handle 1 to n indexing directly

    for (int i = 1; i <= n; i++)
        num[i] = -1;

    for (int i = 1; i < n; i++)
    {
        printf("Enter number %d: ", i);
        scanf("%d", &input);
        num[input] = input;
    }

    for (int i = 1; i <= n; i++)
    {
        if (num[i] == -1)
        {
            printf("\n%d is the missing number.\n", i);
            break;
        }
    }

    return 0;
}
```

Test Case : Input : n = 7, array = 1 3 2 5 4 7   Output = 6

Input : n = 10, array = 2,4,1,3,7,8,9,510   Output = 6

Q5)

```c
Rearrange Array Alternately
Problem Statement:
Rearrange an array such that the maximum element is followed by the minimum element, then the
second maximum, and so on.
eg: For a Given array 1,2,3,4,5,6.7.8 output must be 8 1 7 2 6 3 5 4
    for array 3,2,1,5,6,8,7,4,9 output will be 9 1 8 2 7 3 6 4 5

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

int compare(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}

int main()
{
    system("cls");
    int array[] = {7, 8, 1, 2, 3, 4, 5, 6, 9};
    int n = sizeof(array) / sizeof(array[0]);
    int newArray[n];

    qsort(array, n, sizeof(int), compare);

    int l = 0, r = n - 1;
    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0)
        {
            newArray[i] = array[r--];
        }
        else
        {
            newArray[i] = array[l++];
        }
    }

    printf("Rearranged array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", newArray[i]);

    return 0;
}
```

Test Case : Input : *1,2,3,4,5,6.7.8*   *output : 8 1 7 2 6 3 5 4*

Input : *3,2,1,5,6,8,7,4,9*   *output : 9 1 8 2 7 3 6 4 5*

Q6)

```c
/*
Rotate an Array
Given an array of size N, rotate the array to the right by K positions.

Example:
Input: arr = [1, 2, 3, 4, 5], K = 2 → Output: [4, 5, 1, 2, 3]
*/
#include <stdio.h>

// Function to reverse a portion of the array
void reverse(int arr[], int start, int end)
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

// Function to rotate the array to the right by K positions
void rotateArray(int arr[], int N, int K)
{
    K = K % N; // Handle cases where K >= N
    if (K == 0)
```

```c
        return; // No rotation needed if K is 0 or multiple of N

    reverse(arr, 0, N - 1);

    reverse(arr, 0, K - 1);

    reverse(arr, K, N - 1);
}

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

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int N = sizeof(arr) / sizeof(arr[0]);
    int K;

    printf("Enter the number of positions to rotate: ");
    scanf("%d", &K);

    rotateArray(arr, N, K);

    printf("Rotated Array: ");
    printArray(arr, N);

    return 0;
}

/*
Brute force method
 Function to rotate the array one position to the right
void rotateByOne(int arr[], int N)
{
    int last = arr[N - 1]; // Store the last element
    for (int i = N - 1; i > 0; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[0] = last;
}

 Function to rotate the array K times
void rotateArray(int arr[], int N, int K)
{
    K = K % N; // Handle cases where K >= N
    for (int i = 0; i < K; i++)
    {
        rotateByOne(arr, N); // Rotate by one position K times
    }
}
```

```
*/


/*
Using a temporary array
 Function to rotate the array using an extra array
void rotateArray(int arr[], int N, int K)
{
    K = K % N; // Handle cases where K >= N
    int temp[N]; // Temporary array to store rotated elements

     Place each element at its new position
    for (int i = 0; i < N; i++)
    {
        temp[(i + K) % N] = arr[i];
    }

    Copy the temporary array back to the original array
    for (int i = 0; i < N; i++)
    {
        arr[i] = temp[i];
    }
}
*/
Test Cases : Input : array : 1, 2, 3, 4, 5  n = 7    Output : 4,5,1,2,3
```

Q7)

```
Check if Array is Sorted
#include <stdio.h>
int main()
{
    system("cls");
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    for (int i = 1; i < n; i++)
    {
        if (arr[i] < arr[i - 1])
        {
            printf("Array is not sorted\n");
            return 0;
        }
    }
    printf("Array is sorted\n");
    return 0;
}
Test Case : Input 2 3 4 5 6 7        Output : True
Input : 5 2 6 7 8         Output : False
```

Q8)

```
Print Factors
Write a pseudo-code to print all factors of a given number.

#include <stdio.h>

int main()
{
    system("cls");
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num <= 0)
    {
        printf("Factors are undefined for non-positive numbers.\n");
        return 0;
    }

    printf("Factors of %d are: ", num);

    for (int i = 1; i * i <= num; i++)
    {
        if (num % i == 0)
        {
            printf("%d ", i);
            if (i != num / i)
            {
                printf("%d ", num / i);
            }
        }
    }
    return 0;
}

Test Case : Input 23              Output : 1, 23
Input 100          Output : 1 100 2 50 4 25 5 20 10
```

Q9)

```
Dutch National Flag Problem
Sort an array containing only 0s, 1s, and 2s without using a sorting algorithm.

Example:
Input: [0, 1, 2, 1, 0, 2] → Output: [0, 0, 1, 1, 2, 2]

#include <stdio.h>

void sort012(int arr[], int n)
{
    int low = 0, mid = 0, high = n - 1, temp;

    while (mid <= high)
    {
        if (arr[mid] == 0)
        {
            temp = arr[low];
            arr[low] = arr[mid];
            arr[mid] = temp;
            low++;
            mid++;
        }
        else if (arr[mid] == 1)
        {
            mid++;
        }
        else
        {

            temp = arr[mid];
            arr[mid] = arr[high];
            arr[high] = temp;
            high--;
        }
    }
}

int main()
{
    int arr[] = {0, 1, 2, 1, 0, 2, 0, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    sort012(arr, n);

    // Print the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Test Case : Input : 1,2, 0, 0, 2, 1, 0 Output : 0,0,01,1,2,2

Input : 0,2,1,0,0,0 Output : 0,0,0,0,1,2

Q10)

```
Move Zeroes
Write a program to move all zeroes in an array to the end while maintaining the relative order of
non-zero elements.

Example:
Input: [0, 1, 0, 3, 12] → Output: [1, 3, 12, 0, 0]

#include <stdio.h>

void moveZeroes(int arr[], int n)
{
    int pos = 0; // Index to place the next non-zero element

    // Move all non-zero elements to the front
    for (int i = 0; i < n; i++)
    {
        if (arr[i] != 0)
        {
            arr[pos] = arr[i];
            pos++;
        }
    }

    // Fill the rest of the array with zeros
```

```c
    while (pos < n)
    {
        arr[pos] = 0;
        pos++;
    }
}

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

int main()
{
    system("cls");
    int arr[] = {0, 1, 0, 3, 12};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original Array: ");
    printArray(arr, n);

    moveZeroes(arr, n);

    printf("Modified Array: ");
    printArray(arr, n);

    return 0;
}
```

Test Case : *Input: [0, 1, 0, 3, 12] → Output: [1, 3, 12, 0, 0]*

Input : [3,1,5,0,2,0]   Output : [3,1,5,2,0,0]