Assignment 01

```
//Mohit Khatri
// Problem 1: Write a C program to take Input 5 integers through keyboard, and display the
// second largest number.

// O(n) & O(1) One Pass Solution

#include <stdio.h>
#include <limits.h>

int main()
{
    int a[5];
    printf("Enter 5 numbers: ");
    for (int i = 0; i < 5; i++)
    {
        scanf("%d", &a[i]);
    }

    int largest = a[0];
    int secondLargest = INT_MIN;

    for (int i = 0; i < 5; i++)
    {
        if (a[i] > largest)
        {
            secondLargest = largest;
            largest = a[i];
        }
        else if (a[i] > secondLargest && a[i] != largest)
        {
            secondLargest = a[i];
        }
    }

    if (secondLargest != INT_MIN)
    {
        printf("Second Largest: %d", secondLargest);
    }
    else
    {
        printf("Error: No second largest element present.");
        return -1;
    }

    return 0;}
```

```c
// Problem 2: Write a C program to take Input a 4-digit integer through keyboard and check if
// it’s divisible by 2, 3, 4, and 12.

// O(1) & O(1) Solution

#include <stdio.h>
#include <stdbool.h>

int main()
{
    int num;
    printf("Enter a 4-digit integer: ");
    scanf("%d", &num);

    if (num < 1000 || num > 9999)
    {
        printf("Error: Input must be a 4-digit integer.\n");
        return -1;
    }

    bool isDivisibleBy2 = (num % 2 == 0);
    bool isDivisibleBy3 = (num % 3 == 0);
    bool isDivisibleBy4 = (num % 4 == 0);
    bool isDivisibleBy12 = (num % 12 == 0);
    printf("Divisible by 2: %s\n", isDivisibleBy2 ? "Yes" : "No");
    printf("Divisible by 3: %s\n", isDivisibleBy3 ? "Yes" : "No");
    printf("Divisible by 4: %s\n", isDivisibleBy4 ? "Yes" : "No");
    printf("Divisible by 12: %s\n", isDivisibleBy12 ? "Yes" : "No");

    return 0;
}
```

```
// Problem 3: Write a C program to take Input a 4-digit integer through keyboard, print the sum
// of product of even position digits and odd position digits. For example, if the integer is 2345,
// then the sum of the product will be 2*4+3*5=23.

// O(n) & O(1) One Pass Solution

#include <stdio.h>
int main()
{
    int num;
    printf("Enter a 4-digit integer: ");
    scanf("%d", &num);

    if (num < 1000 || num > 9999)
    {
        printf("Error: Input must be a 4-digit integer.\n");
        return -1;
    }

    int sum = 0;
    int prodEvenIndex = 1;
    int prodOddIndex = 1;
    for (int i = 0; i < 4; i++)
    {
        int digit = num % 10;
        if (i % 2 == 0)
        {
            prodEvenIndex *= digit;
        }
        else
        {
            prodOddIndex *= digit;
        }
        num /= 10;
    }
    sum = prodEvenIndex + prodOddIndex;

    printf("Sum of product of even and odd position digits: %d\n", sum);

    return 0;
}
```

```
// Problem 4: Fibonacci numbers are the numbers in the following integer sequence: 0, 1, 1, 2, 3,
// 5, 8, 13, 21 ... By defintion, the first two Fibonacci numbers are 0 and 1, and each subsequent
// number is the sum of the previous two numbers. Write a program to compute nth number in
// this series for given input n

// O(n) & O(1) Solution

#include <stdio.h>

int main()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);

    if (n < 0)
    {
        printf("Error: Input must be a positive integer.\n");
        return -1;
    }

    int a = 0;
    int b = 1;
    for (int i = 0; i < n; i++)
    {
        int temp = a;
        a = b;
        b = temp + b;
    }

    printf("Fibonacci number %d: %d\n", n, a);

    return 0;
}
```

```
// Problem 5: Print a triangle of ’*’s of height ’r’ rows. Now modify your program to print it
// upside down of given size ’r’, where r represents the no. of rows in the triangle

// Assumption: Right-angled triangle.

// O(r^2) & O(1) Solution
#include <stdio.h>

int main()
{
    int r;
    printf("Enter the height of the triangle: ");
    scanf("%d", &r);

    if (r < 1)
    {
        printf("Error: Height must be a positive integer.\n");
        return -1;
    }

    for (int i = 1; i <= r; i++)
    {
        for (int j = 1; j <= i; j++)
        {
            printf("*");
        }
        printf("\n");
    }

    printf("Upside Down Version: \n");
    for (int i = r; i >= 1; i--)
    {
        for (int j = 1; j <= i; j++)
        {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```

```
// Problem 6: Implement a C program that finds all the numbers between 01 and 1000 such that
// the number itself minus the number reversed is equal to the sum of its digits. For example:
// 54 is such a number because 54-45 (which is 9) is same as the sum of its digits (5+4 = 9).

// O(nk) & O(1) where n = max number to check and k = no of digits in n

#include <stdio.h>

int reverse(int num)
{
    int rev = 0;
    while (num > 0)
    {
        rev = rev * 10 + num % 10;
        num /= 10;
    }
    return rev;
}

int sumOfDigits(int num)
{
    int sum = 0;
    while (num > 0)
    {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

int main()
{
    printf("Numbers between 1 and 1000 such that the number itself minus the number reversed is
equal to the sum of its digits:\n");
    for (int i = 1; i <= 1000; i++)
    {
        if (i - reverse(i) == sumOfDigits(i))
        {
            printf("%d\n", i);
        }
    }
    return 0;
}
```

//Problem 7 User provides two unsorted 1-D arrays of sizes m and n, write a C program that merges the two //into another 1-D array of size m + n such that this new array becomes sorted. The sizes m and n //and values in the arrays are also provided by user.

```
#include <stdio.h>

void merge(int arr[], int left[], int leftSize, int right[], int rightSize)
{
    int i = 0, j = 0, k = 0;

    while (i < leftSize && j < rightSize)
    {
        if (left[i] <= right[j])
        {
            arr[k++] = left[i++];
        }
        else
        {
            arr[k++] = right[j++];
        }
    }

    while (i < leftSize)
    {

        arr[k++] = left[i++];
    }

    while (j < rightSize)
    {
        arr[k++] = right[j++];
    }
}

void mergeSort(int arr[], int size)
{
    if (size < 2)
        return;

    int mid = size / 2;
    int left[mid], right[size - mid];

    for (int i = 0; i < mid; i++)
    {
        left[i] = arr[i];
    }
```

```
    for (int i = mid; i < size; i++)
    {
        right[i - mid] = arr[i];
    }

    mergeSort(left, mid);
    mergeSort(right, size - mid);
    merge(arr, left, mid, right, size - mid);
}

int main()
{
    int m, n;
    printf("Enter the size of first array: ");
    scanf("%d", &m);
    printf("Enter the size of second array: ");
    scanf("%d", &n);

    int arr1[m], arr2[n], mergedArr[m + n];

    printf("Enter the elements of first array: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &arr1[i]);

    printf("Enter the elements of second array: ");
    for (int j = 0; j < n; j++)
        scanf("%d", &arr2[j]);

    mergeSort(arr1, m);
    mergeSort(arr2, n);

    merge(mergedArr, arr1, m, arr2, n);
    printf("Sorted Merged Array: ");
    for (int i = 0; i < m + n; i++)
    {
        printf("%d ", mergedArr[i]);
    }

    return 0;
}
```

```
//Problem 8 Write a C program that take 2 integer sets A[] and b[] as input and prints results of
//following set operations: (a) A union B (Write function setunion()) (b) A intersection B (Write
//function setintersection()) (c) A-B and B-A (Write function setdifference())
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
```

```
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void setUnion(int a[], int b[], int m, int n)
{
    int i = 0, j = 0;
    printf("A union B: ");

    while (i < m && j < n)
    {
        if (a[i] < b[j])
            printf("%d ", a[i++]);
        else if (b[j] < a[i])
            printf("%d ", b[j++]);
        else
        {
            printf("%d ", a[i++]);
            j++;
        }
    }

    while (i < m)
        printf("%d ", a[i++]);

    while (j < n)
        printf("%d ", b[j++]);

    printf("\n");
}

void setIntersection(int a[], int b[], int m, int n)
{
```

```
    int i = 0, j = 0;
    printf("A intersection B: ");

    while (i < m && j < n)
    {
        if (a[i] < b[j])
            i++;
        else if (b[j] < a[i])
            j++;
        else
        {
            printf("%d ", a[i++]);
            j++;
        }
    }

    printf("\n");
}

void setDifference(int a[], int b[], int m, int n)
{
    int i = 0, j = 0;
    printf("A-B: ");

    while (i < m && j < n)
    {
        if (a[i] < b[j])
            printf("%d ", a[i++]);
        else if (b[j] < a[i])
            j++;
        else
        {
            i++;
            j++;
        }
    }

    while (i < m)
        printf("%d ", a[i++]);

    printf("\n");

    i = 0;
    j = 0;
    printf("B-A: ");
```

```
    while (i < m && j < n)
    {
        if (b[j] < a[i])
            printf("%d ", b[j++]);
        else if (a[i] < b[j])
            i++;
        else
        {
            i++;
            j++;
        }
    }

    while (j < n)
        printf("%d ", b[j++]);

    printf("\n");
}

int main()
{
    int m, n, i;
    printf("Enter the size of A[]: ");
    scanf("%d", &m);
    int a[m];
    printf("Enter the elements of A[]: ");
    for (i = 0; i < m; i++)
        scanf("%d", &a[i]);
    mergeSort(a, 0, m - 1);

    printf("Enter the size of B[]: ");
    scanf("%d", &n);
    int b[n];
    printf("Enter the elements of B[]: ");
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);
    mergeSort(b, 0, n - 1);

    setUnion(a, b, m, n);
    setIntersection(a, b, m, n);
    setDifference(a, b, m, n);

    return 0;
}
```

```
//Problem 9:Write a function (function name: distance) to compute the distance between two
points //and use it to develop another function (function name: area) that will compute the area of the
//triangle whose vertices are A(x1, y1), B(x2, y2), and C(x3, y3). Use these to develop a function
//functions (function name: tritest) which returns a value 1 if the point (x, y) is inside the triangle ABC,
//otherwise a value 0 for N points, where N points are entered through the keyboard.

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

typedef struct
{
   double x;
   double y;
} Point;

double distance(Point p1, Point p2)
{
   double dx = p1.x - p2.x;
   double dy = p1.y - p2.y;
   return sqrt(dx * dx + dy * dy);
}

double area(Point p1, Point p2, Point p3)
{
   double a = distance(p1, p2);
   double b = distance(p2, p3);
   double c = distance(p3, p1);
   double s = (a + b + c) / 2;
   return sqrt(s * (s - a) * (s - b) * (s - c));
}

int tritest(Point p1, Point p2, Point p3, Point test)
{
   double total_area = area(p1, p2, p3);
   double area1 = area(p1, p2, test);
   double area2 = area(p2, p3, test);
   double area3 = area(p3, p1, test);
   if (fabs(total_area - (area1 + area2 + area3)) < 0.00001)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
```

```
int main()
{
    int n, i;
    Point p1, p2, p3, test;

    printf("Enter coordinates for Point A (x,y): ");
    scanf("%lf %lf", &p1.x, &p1.y);
    printf("Enter coordinates for Point B (x,y): ");
    scanf("%lf %lf", &p2.x, &p2.y);
    printf("Enter coordinates for Point C (x,y): ");
    scanf("%lf %lf", &p3.x, &p3.y);

    printf("Enter number of test points: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
    {
        printf("Enter coordinates for test point %d (x,y): ", i + 1);
        scanf("%lf %lf", &test.x, &test.y);

        if (tritest(p1, p2, p3, test))
        {
            printf("Test point %d is inside triangle ABC\n", i + 1);
        }
        else
        {
            printf("Test point %d is NOT inside triangle ABC\n", i + 1);
        }
    }

    return 0;
}
```

//Problem 10: An array of integers is said to be a straight-K, if it contains K elements that are K //consecutive numbers. For example, the array 6, 1, 9, 5, 7, 15, 8 is a straight because it contains 5, //6, 7, 8, and 9 for K=5. Write a program to finds the maximum value of K for the given number of //integers.

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

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

int findStraightK(int arr[], int n)
{
   int maxK = 0;

   // sort the array
   qsort(arr, n, sizeof(int), cmpfunc);

   // find the length of the longest straight-K
   int k = 1;
   for (int i = 1; i < n; i++)
   {
      if (arr[i] == arr[i - 1] + 1)
      {
         k++;
      }
      else
      {
         if (k > maxK)
         {
            maxK = k;
         }
         k = 1;
      }
   }

   if (k > maxK)
   {
      maxK = k;
   }

   return maxK;
}
```

```
int main()
{
    int n;
    printf("Enter the number of integers: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the integers: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    int maxK = findStraightK(arr, n);
    printf("The maximum value of K for the given array is: %d\n", maxK);

    return 0;
}

//Problem 11: Write a program to find the factorial of the given number (1 ≤ n ≤ 10,000,000,000).

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

double factorial(long long n)
{
    double pi = 3.14159265359;
    double e = 2.71828182846;
    double x = sqrt(2 * pi * n) * pow(n / e, n);
    return x;
}

int main()
{
    long long n;
    printf("Enter a number to find its factorial: ");
    scanf("%lld", &n);
    if (n < 0)
    {
        printf("Factorial of negative numbers does not exist.\n");
    }
    else if (n == 0)
    {
        printf("Factorial of 0 is 1.\n");
    }
    else
```

```
    {
        printf("Factorial of %lld is %.0f.\n", n, factorial(n));
    }
    return 0;
}

//Problem 13:Write a C program to print the all possible circular rotation of elements of array. For
//example if input array=3,5,2,6,1 then output=52613, 26135, 61352, 13526 and 35261.

#include <stdio.h>

void rotate(int arr[], int n)
{
    int temp = arr[n - 1];
    for (int i = n - 1; i > 0; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[0] = temp;
}

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

void circularRotate(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        rotate(arr, n);
        printArray(arr, n);
    }
}

int main()
{
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];
```

```
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    circularRotate(arr, n);

    return 0;
}

//Problem 15:Consider a positive integer n of type int. The next higher permutation of n is the
//smallest integer greater than n which is formed by permuting the digits of n. For example, the
next //higher permutation for n=1209861 is 1210689, the next higher permutation for n=1421731
is //1423117. Note that next higher permutation may not exist for every number. Write a program to
find //the next higher permutation of the given number

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

void swap(char *a, char *b)
{
    char temp = *a;
    *a = *b;
    *b = temp;
}

int nextHigherPermutation(char *number, int n)
{
    int i, j;

    // Find the largest index i such that number[i] < number[i+1]
    for (i = n - 2; i >= 0; i--)
    {
        if (number[i] < number[i + 1])
        {
            break;
        }
    }

    // If no such index exists, the permutation is already the largest possible
    if (i < 0)
    {
        return 0;
    }
```

```
    // Find the largest index j such that number[j] > number[i]
    for (j = n - 1; j > i; j--)
    {
        if (number[j] > number[i])
        {
            break;
        }
    }

    // Swap number[i] and number[j]
    swap(&number[i], &number[j]);

    // Reverse the sequence from number[i+1] to the end
    for (i = i + 1, j = n - 1; i < j; i++, j--)
    {
        swap(&number[i], &number[j]);
    }

    return 1;
}

int main()
{
    char number[20];
    printf("Enter a positive integer: ");
    scanf("%s", number);
    int n = strlen(number);
    int result = nextHigherPermutation(number, n);
    if (result == 1)
    {
        printf("The next higher permutation is %s\n", number);
    }
    else
    {
        printf("No higher permutation exists, the smallest permutation is %s\n", number);
    }
    return 0;
}

//Problem 16: Write a program to display the given number after eliminating the duplicate digits
from //it. For example: for a given number 245265 display the number 2456.

#include <stdio.h>

int main()
```

```
{
    long int num, newNum = 0;
    int digit, freq[10] = {0};

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

    while (num > 0)
    {
        digit = num % 10;
        freq[digit]++;
        num /= 10;

        if (freq[digit] == 1)
        {
            newNum = (newNum * 10) + digit;
        }
    }

    printf("Number after eliminating duplicates: %ld", newNum);

    return 0;
}
```

//Problem 17: A number n is called left trunckable prime if n and all numbers obtained by //successively removing its left most digits are prime.(Similarly right truncatable prime is defined) Ex //313 is a left truncatable prime – 313 is prime, 13 is prime, 3 is prime. 313 is also right truncatable – //313 is prime, 31 is prime, 3 is prime. Write a C program using prime() function, which takes a //number n as input and then tells whether it is left truncatable, right truncatable or both.

// Not working

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

int prime(int n)
{
    if (n <= 1)
    {
        return 0;
    }
    int i;
    for (i = 2; i <= sqrt(n); i++)
    {
        if (n % i == 0)
```

```
        {
            return 0;
        }
    }
    return 1;
}

int is_left_truncatable(int n)
{
    while (n > 0)
    {
        if (!prime(n))
        {
            return 0;
        }
        n = n % (int)pow(10, (int)log10(n));
    }
    return 1;
}

int is_right_truncatable(int n)
{
    while (n > 0)
    {
        if (!prime(n))
        {
            return 0;
        }
        n /= 10;
    }
    return 1;
}

int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    int left = is_left_truncatable(n);
    int right = is_right_truncatable(n);

    if (left && right)
    {
        printf("%d is both left-truncatable and right-truncatable\n", n);
    }
```

```
    if (left)
    {
        printf("%d is left-truncatable\n", n);
    }
    if (right)
    {
        printf("%d is right-truncatable\n", n);
    }

    return 0;
}

//Problem 19:Write a function void partition (int a[], int left, int right)which selects the first element in //the array as pivot, rearranges the array elements, such that the pivot element goes to the new //position middle between left and right, so that all left side elements are less then middle element //and all right side elements are grater then middle element.

#include <stdio.h>

void partition(int a[], int left, int right)
{
    int pivot = a[left];
    int i = left, j = right;
    int temp;

    while (i < j)
    {
        while (a[i] <= pivot && i < right)
            i++;
        while (a[j] > pivot)
            j--;
        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    a[left] = a[j];
    a[j] = pivot;

    printf("After partitioning: ");
    for (int k = 0; k <= right; k++)
    {
        printf("%d ", a[k]);
```

```
    }
}

int main()
{
    int a[] = {10, 5, 6, 8, 40, 50, 70};
    int n = sizeof(a) / sizeof(a[0]);

    printf("Before partitioning: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }

    partition(a, 0, n - 1);

    return 0;
}
```

//Problem 20: A number is circular prime if it is prime and all its cyclic rotations are also prime. e.g. //The number 1193 is a circular prime number because it is prime and all its cyclic rotations 1931, //9311, 3119 are prime. Write a program that takes an integer n as input and prints whether it is //circular prime or not. Your program has to work for all values of n which can be stored in data type //int.

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

// Function to check if a number is prime
int is_prime(int n)
{
    if (n < 2)
    {
        return 0;
    }
    for (int i = 2; i <= sqrt(n); i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}
```

```
// Function to rotate a number
int rotate(int n)
{
    char str[20];
    sprintf(str, "%d", n);
    int len = strlen(str);
    char tmp = str[0];
    for (int i = 0; i < len - 1; i++)
    {
        str[i] = str[i + 1];
    }
    str[len - 1] = tmp;
    return atoi(str);
}

int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    // Check if input number is prime
    if (!is_prime(n))
    {
        printf("%d is not a circular prime.\n", n);
        return 0;
    }

    // Generate all cyclic rotations and check if they are prime
    int m = rotate(n);
    while (m != n)
    {
        if (!is_prime(m))
        {
            printf("%d is not a circular prime.\n", n);
            return 0;
        }
        m = rotate(m);
    }

    // If all cyclic rotations are prime, the number is a circular prime
    printf("%d is a circular prime.\n", n);

    return 0;
}
```