SK_CODING_CHALLENGE_4 – 29/06/2020

Activity 1

1. Your local library needs your help! Given the expected and actual return dates for a library book, create a program that calculates the fine (if any).
The fee structure is as follows:
1. If the book is returned on or before the expected return date, no fine will be charged (i.e.:fine=0).
2. If the book is returned after the expected return day but still within the same calendar month and year as the expected return date :fine=Rs.15*(No. Of Days)
3. If the book is returned after the expected return month but still within the same calendar year as the expected return date, the fine=Rs.500*(No. Of Days).
4. If the book is returned after the calendar year in which it was expected, there is a fixed fine of Rs. 10000.
Input Format
1. The first line contains 3 space-separated integers denoting the respective DD,MM, YY on which the book was actually returned.
2. The second line contains 3 space-separated integers denoting the respective DD,MM, YY on which the book was expected to be returned (due date).
Constraints
1. DD={1 to 31}
2. MM={1 to 12}
3. YY={1 to 3000}
4. Accept only valid dates
Output Format
Print a single integer denoting the library fine for the book received as input.

Solution:

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <limits.h>
#include <stdbool.h>

int main(){
    int d1;
    int m1;
    int y1;
```

```
   scanf("%d %d %d",&d1,&m1,&y1);
   int d2;
   int m2;
   int y2;
   scanf("%d %d %d",&d2,&m2,&y2);
   if(y2==y1){
      if(m2==m1){
         if(d1<d2)
         printf("0");
         else
         printf("%d",15*(d1-d2));
      }
      else{
          if(m1<m2)
         printf("0");
         else
         printf("%d",500*(m1-m2));
      }
   }
   else{
       if(y1<y2)
         printf("0");
      else
      printf("%d",10000*(y1-y2));
   }
   return 0;
}
```

****

Activity 2:

A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself. Given a number,n, determine and print whether it's Prime or Not-Prime.

Input Format

The first line contains an integer, T, the number of test cases.

Each of the T subsequent lines contains an integer, n, to be tested for primality.

Constraints

- T={1 to 30}
- n={ 1 to 2*109}
-

Output Format

For each test case, print whether n is Prime or Not-Prime.

Solution:

```
class GFG {
```

```
    static boolean isPrime(int n)
    {
        // Corner case
        if (n <= 1)
            return false;

        // Check from 2 to n-1
        for (int i = 2; i < n; i++)
            if (n % i == 0)
                return false;

        return true;
    }

    // Driver Program
    public static void main(String args[])
    {
        if (isPrime(11))
            System.out.println(" true");
        else
            System.out.println(" false");
        if (isPrime(15))
            System.out.println(" true");
        else
            System.out.println(" false");
    }
}
```

****

Activity 3:

The height of a binary search tree is the number of edges between the tree's root and its furthest leaf. You are given a pointer, root, pointing to the root of a binary search tree. Develop the getHeight function so that it returns the height of the binary search tree.

Input Format

The locked stub code in your editor reads the following inputs and assembles them into a binary search tree:

The first line contains an integer, n , denoting the number of nodes in the tree.

Each of the n subsequent lines contains an integer, data, denoting the value of an element that must be added to the BST.

Output Format

Print the integer returned by your getHeight function denoting the height of the BST.

Solution:

```
import java.util.*;
import java.io.*;
```

```
class Node{
   Node left,right;
   int data;
   Node(int data){
      this.data=data;
      left=right=null;
   }
}
class Solution{
 public static int getHeight(Node root){
     //Write your code here
      return root == null ? -1 : 1 + Math.max(getHeight(root.left), getHeight(root.right));
   }
 public static Node insert(Node root,int data){
       if(root==null){
          return new Node(data);
       }
       else{
          Node cur;
          if(data<=root.data){
             cur=insert(root.left,data);
             root.left=cur;
          }
          else{
             cur=insert(root.right,data);
             root.right=cur;
          }
          return root;
       }
   }
   public static void main(String args[]){
          Scanner sc=new Scanner(System.in);
          int T=sc.nextInt();
          Node root=null;
          while(T-->0){
             int data=sc.nextInt();
             root=insert(root,data);
          }
          int height=getHeight(root);
          System.out.println(height);
       }
}
```

****

Activity 4:

Given an array, of size distinct elements, sort the array in ascending order using the Bubble Sort algorithm above. Once sorted, print the following lines:

1. Array is sorted in numSwaps swaps. where numofswaps is the number of swaps that took place.
2. First Element: firstElement where firstElement is the first element in the sorted array.

3. Last Element: lastElement, where lastElement is the last element in the sorted array.

Input Format
The first line contains an integer, n, denoting the number of elements in array A.
The second line contains n space-separated integers describing the respective values of A={a1, a2,.....an} .
Constraints
N={2 to 600}
Ai={1 to 2*106}, where i={0 to n}.
Output Format
Print the following three lines of output:
1. Array is sorted in numSwaps swaps. where numSwaps is the number of swaps that took place.
2. First Element: firstElement, where firstElement is the first element in the sorted array.
3. Last Element: lastElement, where lastElement is the last element in the sorted array.

Solution:

```
import java.util.Scanner;

public class Solution {
   private static int[] array;

   private static void bubbleSort() {
      int n = array.length;

      // number of swaps for all array iterations
      int totalSwaps = 0;

      for (int i = 0; i < n; i++) {
         // number of swaps for current array iteration
         int numSwaps = 0;

         for (int j = 0; j < array.length - 1; j++) {
            if (array[j] > array[j + 1]) {
               int tmp = array[j];
               array[j] = array[j + 1];
               array[j + 1] = tmp;
               numSwaps++;
               totalSwaps++;
            }
         }

         if (numSwaps == 0) {
            System.out.printf("Array is sorted in %d swaps.\n", totalSwaps);
```

```
            System.out.printf("First Element: %d\n", array[0]);
            System.out.printf("Last Element: %d\n", array[n - 1]);

            break;
         }
      }
   }

   public static void main(String[] args){
      Scanner in = new Scanner(System.in);
      int n = in.nextInt();
      array = new int[n];
      for (int i = 0; i < n; i++) {
         array[i] = in.nextInt();
      }
      in.close();

      bubbleSort();
   }
}
```

****