Lab 8 : 202201477

Q.1. Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges 1 <= month <= 12, 1 <= day <= 31, 1900 <= year <= 2015.The possible output dates would be previous date or invalid date. Design the equivalence class test cases?

Equivalence class :

E1: 1<=day<=31(valid)
E2: day<1(invalid)
E3: day>31(invalid)
E4 : 1<=month<=12 (valid)
E5: month<1 (invalid)
E6: month>12 (invalid)
E7: 1900<=year<=2015(valid)
E8: year<1900 (invalid)
E9 : year>2015 (invalid)

No.	Day	Month	Year	Class Covered	Expected Output
1	1	1	1900	E1,E4,E7	Invalid date
2	31	12	2015	E1,E4,E7	30-12-2015
3	2	2	1901	E1,E4,E7	1-2-1901
4	30	11	2014	E1,E4,E7	29-11-2014
5	1	3	2001	E1,E4,E7	28-2-2001
6	0	2	2005	E2,E4,E7	Invalid date
7	32	2	2005	E3,E4,E7	Invalid date
8	20	0	2005	E5,E1,E7	Invalid date
9	21	13	2006	E6,E1,E7	Invalid date

10	5	5	1899	E1,E4,E8	Invalid date
11	5	5	2016	E1,E4,E9	Invalid date
12	1	3	2000	E1,E4,E7	29-2-2000

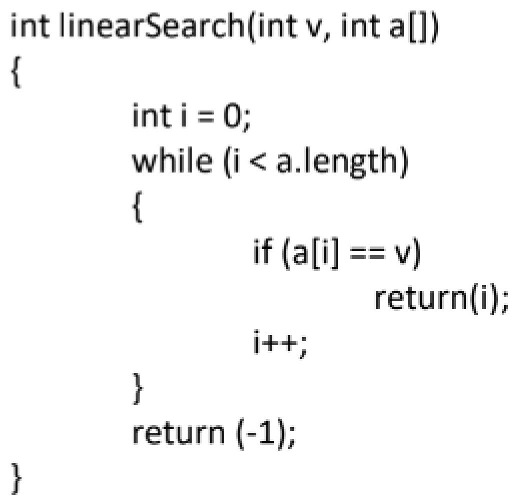
-> Here specific two test cases 5 & 12 are for the leap year test.

Q.2

1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

2. Modify your programs such that it runs, and then execute your test suites on the program.While executing your input data in a program, check whether the identified expected outcome(mentioned by you) is correct or not.

1.The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.



```
int linearSearch(int v, int a[])
{
            int i = 0;
            while (i < a.length)
            {
                        if (a[i] == v)
                                    return(i);
                        i++;
            }
            return (-1);
}
```

Example Array : [1 , 4 , -1 , 0 ,7]

E1: V present
E2: V not present
E3: V present at 0th index
E4: V present at last index of array

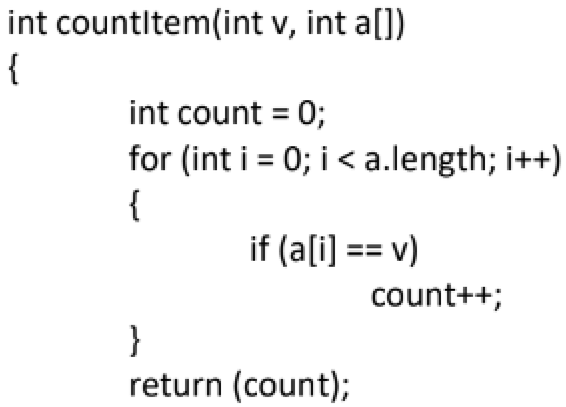
Entered value	Expected Output
4	1
6	-1
1	0
7	4

2.The function countItem returns the number of times a value v appears in an array of integers a.

E1: 0<count(v)<n

E2: count(v)=0

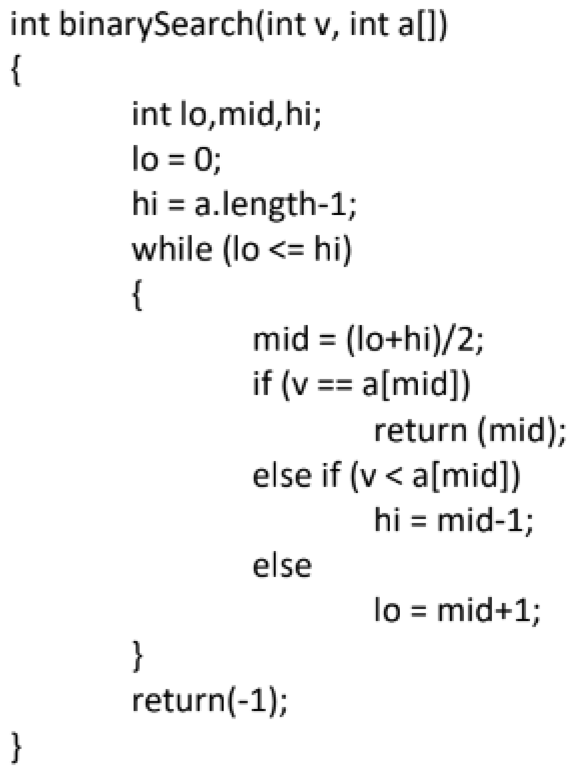
E3: count(v)=n



```
int countItem(int v, int a[])
{
        int count = 0;
        for (int i = 0; i < a.length; i++)
        {
                if (a[i] == v)
                        count++;
        }
        return (count);
```

Array	Value Entered	Output
{1 2 2 3 4}	2	2
{1 2 2 3 4}	5	0
{3 3 3 3 3}	3	5

P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in
the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

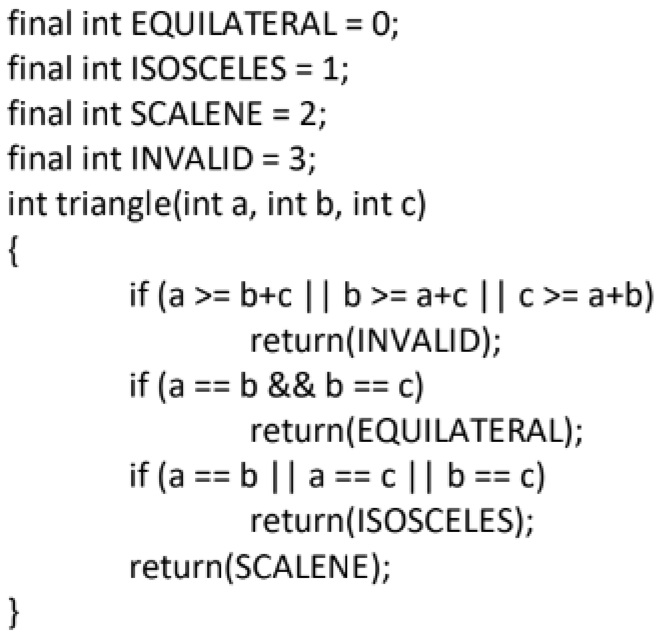


```
int binarySearch(int v, int a[])
{
        int lo,mid,hi;
        lo = 0;
        hi = a.length-1;
        while (lo <= hi)
        {
                mid = (lo+hi)/2;
                if (v == a[mid])
                        return (mid);
                else if (v < a[mid])
                        hi = mid-1;
                else
                        lo = mid+1;
        }
        return(-1);
}
```

Example Array:
[1 3 5 7 8 9]

Entered Value	Expected output
5	2
1	0
9	5
6	-1

P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979).The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

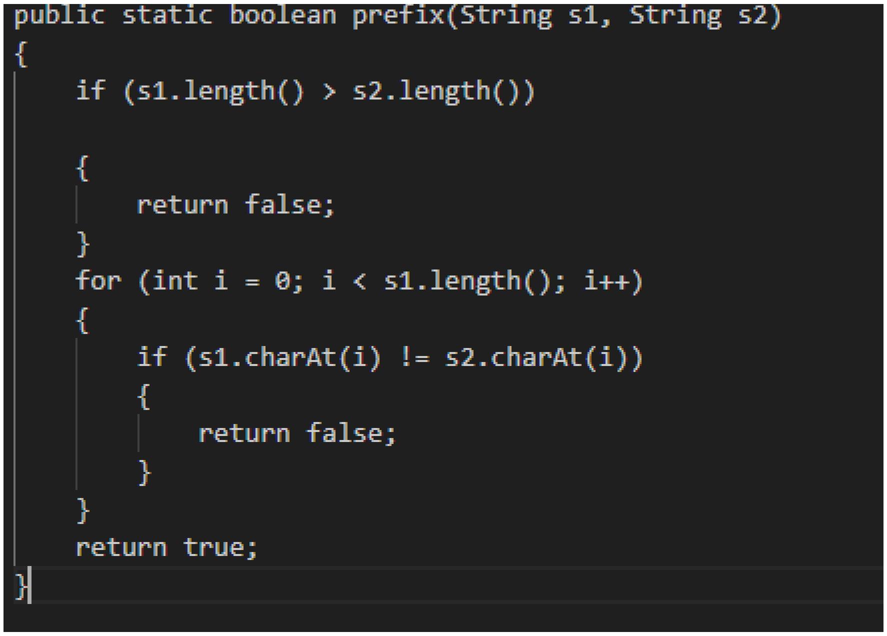


```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
        if (a >= b+c || b >= a+c || c >= a+b)
                return(INVALID);
        if (a == b && b == c)
                return(EQUILATERAL);
        if (a == b || a == c || b == c)
                return(ISOSCELES);
        return(SCALENE);
}
```

E1: all sides equal
E2: any two sides are equal
E3: all sides not equal but valid triangle
E4: a>=b+c
E5: b>=a+c
E6: c>=a+b

Input (a,b,c)	Expected output
2,2,2	Equilateral
1,2,1	isosceles
3 2 4	scalene
3,1,2	invalid
2,4,5	invalid
2,1,4	invalid

P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).



```
public static boolean prefix(String s1, String s2)
{
    if (s1.length() > s2.length())

    {
        return false;
    }
    for (int i = 0; i < s1.length(); i++)
    {
        if (s1.charAt(i) != s2.charAt(i))
        {
            return false;
        }
    }
    return true;
}
```

Entered String 2	Expected output
ab	false
abc	true
c	true
abed	false

P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral,or right angled. Determine the following for the above program:

a)Identify the equivalence classes for the system

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence classes)

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

g) For the non-triangle case, identify test cases to explore the boundary.

h) For non-positive input, identify test points.

Equivalance Class:

E1: positive sides(valid)
E2: any side <=0 (invalid)
E3: valid triangle(valid)
E4: invalid triangle(invalid)
E5: a=b=c (valid)
E6: two sides are equal (valid)
E7: all sides are different length (valid)
E8: Right angle triangle

Test case:

No	Input (a,b,c)	Expected output	Class covered

1	0,2,3	invalid	E2
2	2,-1,3	invalid	E2
3	2,2,2	Equilateral	E1,E3,E5
4	1,2,1	isosceles	E1,E3,E6
5	3 2 4	scalene	E1,E3,E7
6	3,1,2	invalid	E1,E4
7	2,4,5	invalid	E1,E4
8	2,1,4	Invalid	E1,E4
9	2,3,5	Right angle	E1,E3,E8

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

9	1,2,3	invalid	E1,E4
10	3,4,6	scalene	E1,E4,E7

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

11	5,6,5	isosceles	E1,E3,E6
12	5,7,5	isosceles	E1,E3,E6

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

13	5,5,5	equilateral	E1,E3,E5

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

14.	3,4,5	Valid Right angle	E1,E3,E8

g) For the non-triangle case, identify test cases to explore the boundary.

15	1,2,3	invalid	E1,E4
16	2,3,6	invalid	E1,E4

h) For non-positive input, identify test points.

17	0,2,3	invalid	E2
18	3,4,-1	invalid	E2