ENGINEERS WITH
SOCIAL RESPONSIBILITY

SOFTWARE ENGINEERING IT-314

LAB 09

NAME:- Jinay Vora
ID:- 202201473

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? Write a set of test cases (i.e., test suite) – specific set of data – to properly test the programs. Your test suite should include both correct and incorrect inputs.**
   1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

A1.

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning **1.** 0/25/2012 **2.** 13/2/1986 **3.** 6/0/1980 **4.** 8/32/2000 **5.** 2/30/2003 **6.** 5/13/1877 **7.** 1/7/2035 **8.** 9/9/2005	**1.** Error (Invalid Month) **2.** Error (Invalid Month) **3.** Error (Invalid Day) **4.** Error (Invalid Day) **5.** Error (Non leap year February has 28 days) **6.** Error (Invalid Year) **7.** Error (Invalid Year) **8.** Valid

9. 2/29/2012	**9.** Valid (Leap Year)
Boundary Value Analysis **1.** 1/1/1900 **2.** 12/31/2015 **3.** 0/0/1899 **4.** 13/32/2016	**1.** Valid **2.** Valid **3.** Invalid **4.** Invalid

2. **Programs:**

   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);
}
```

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning **1.** linearSearch(1, [3,1,5]) **2.** linearSearch(3, [3,1,5]) **3.** linearSearch(5, [3,1,5]) **4.** linearSearch(2, [3,1,5])	 **1.** 1 **2.** 0 **3.** 2 **4.** -1 (Is not in the array)
Boundary Value Analysis **1.** linearSearch(1, [1,2,3]) **2.** linearSearch(3, [1,2,3]) **3.** linearSearch(0, [1,2,3])	 **1.** 0 (Minimum) **2.** 2 (Maximum) **3.** -1 (Is not in the array)

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

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

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning 1. countItem(1,[1,1,2]) 2. countItem(2,[1,1,2]) 3. countItem(3,[1,1,2])	1. 2 2. 1 3. 0 (3 is not in array)
Boundary Value Analysis 1. countItem(1,[4,5,6]) 2. countItem(1,[1,1,2]) 3. countItem(1,[1,1,1])	1. 0 (Minimum) 2. 2 3. 3 (Maximum)

3. 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);
}
```

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning **1.** binarySearch(1, [3,1,5]) **2.** binarySearch(3, [3,1,5]) **3.** binarySearch(5, [3,1,5]) **4.** binarySearch(2, [3,1,5])	 **1.** 1 **2.** 0 **3.** 2 **4.** -1 (Is not in the array)

Boundary Value Analysis	
1. binarySearch(1, [1,2,3])	**1.** 0 (Minimum)
2. binarySearch(3, [1,2,3])	**2.** 2 (Maximum)
3. binarySearch(0, [1,2,3])	**3.** -1 (Is not in the array)

4. 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);

}
```

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning **1.** triangle(1,1,1) **2.** triangle(1,2,2) **3.** triangle(3,4,5) **4.** triangle(2,4,8)	**1.** Equilateral **2.** Isosceles **3.** Scalene **4.** Invalid (8>2+4)
Boundary Value Analysis **1.** a or b or c <=0	**1.** Invalid (Length can't be 0 or less)

5. 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() || 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;
}
```

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
Equivalence Partitioning **1.** prefix(abc, abc) **2.** prefix(abc, bac) **3.** prefix(ac, bac) **4.** prefix(abc, bc)	**1.** True **2.** False (a != b) **3.** False(lengths are not equal) **4.** False(lengths are not equal)
Boundary Value Analysis **1.** prefix(,)	**1.** False (lengths are NULL)

6. 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.

a) Equivalence Classes are:

i) A or B or C <= 0 (Invalid)

ii) A+B>=C or A+C>=B or B+C>=A (Invalid)

iii) A=B=C (Equilateral)

iv) A=B or B=C or A=C (Isosceles)

v) Scalene

Question c and f will have the same answer.

TESTER ACTION AND INPUT DATA	EXPECTED OUTCOME
1. triangle(1,1,1) **2.** triangle(1,2,2) **3.** triangle(3,4,5) **4.** triangle(2,4,8) **5.** triangle(0,-1,3)	**1.** Equilateral (EC 3) **2.** Isosceles (EC 4) **3.** Scalene (EC 5) **4.** Invalid (EC 2) **5.** Invalid (EC 1)