IT-314: SOFTWARE ENGINEERING

LAB-8: Functional Testing (Black-Box Testing)

Name: Harsh Lad

ID: 202201493

Group No: 6

Section: B

❖ Software Testing:-

Question 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?

Solution) The equivalence class for the required conditions as follows:

- Derived Equivalence Class:

Sr. No	Equivalence Class	Acceptance
E1	The input is numeric	Valid
E2	The input is not numeric	Invalid
E3	The day field is not a whole number.	Invalid
E4	The values in the day field are less than 1.	Invalid
E5	The values in the day field are between 1 and 31. (inclusive)	Valid
E6	The values in the day field are greater than 31.	Invalid
E7	The day field is blank	Invalid
E8	The month field is not a natural number.	Invalid
E9	The values in the month field are less than 1.	Invalid
E10	The values in the month field are between 1 and 12. (inclusive)	Valid
E11	The values in the month field are greater than 12.	Invalid
E12	The day field is blank	Invalid

E13	The year field is not a natural number.	Invalid
E14	The values in the year field are less than 1.	Invalid
E15	The values in the year field are between 1900 and 2015. (inclusive)	Valid
E16	The values in the year field are greater than 2015.	Invalid
E17	The year field is blank	Invalid

- The test cases generated from the given equivalence classes are as follows:

No.	**Test Data**	**Expected Outcome**	**Classes Covered**
1.	(7,8,2003)	T	E1,E5,E10,E15
2.	(aa,12,1924)	F	E2
3.	(5.433.., 12, 1978)	F	E3
4.	(0,4,1980)	F	E4
5.	(34,8,1970)	F	E6
6.	(_,7,1988)	F	E7
6.	(12,-0.129,1981)	F	E8
7.	(17,0,1901)	F	E9
8.	(19,901,1901)	F	E11
9.	(1,_,1971)	F	E12
10.	(6,2,infinity)	F	E13

11.	(7,8,1899)	F	E14
12.	(9,12,2030)	F	E16
13.	(19,3,_)	F	E17

Question2. 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.
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.

Programs:

Program P1. 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.

Code:

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

Solution) Equivalence partitioning for the given program are as follows

- Derived Equivalence Partitioning:

Sr No.	Equivalence Class	Acceptance
E1	Element is Present	Returns i such that a[i]==v

E2	Element is not present	Returns -1
E3	Empty array	Returns -1

- From the given set of Equivalence partitioning and Boundary Value Analysis the required set of test suit are as follows

Test case	**Tester Action**	**Input data**	**Expected Outcome**
Equivalence Partitioning			
TC1	E1	a[3]={4,5,6} , v = 5	Returns 1
TC2	E2	a[3]={7,8,9}, v=5	Returns -1
TC3	E3	a[0]={}, v=10	Returns -1
Boundary Value Analysis			
TC4	BVA1(single element in the array)	a[1]={1}, v=1	Returns 0
TC5	BVA2 (first element of the array)	a[5] = {1,2,3,4,5},v=1	Returns 0
TC6	BVA3(last element of the array)	a[5] = {1,2,3,4,5},v=5	Returns 4
TC7	BVA3(more than 1 element found in the array)	a[5] = {1,2,2,2,5},v=2	Returns 1

Program P2. The function countItem returns the number of times a value v appears in an array of integers a.

Code.

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

Solution) Equivalence partitioning and Boundary Value Analysis for the given program are as follows

- Derived Equivalence Partitioning:

Sr No.	**Equivalence Class**	**Acceptance**
E1	Element is Present	Returns count(no. Of element inside the array)
E2	Element is not present	Returns 0
E3	Empty array	Returns 0

- From the given set of Equivalence partitioning and Boundary Value Analysis the required set of test suit are as follows

Test case	**Tester Action**	**Input data**	**Expected Outcome**
Equivalence Partitioning			

TC1	E1	a[3]={4,5,6} , v = 5	Returns 1
TC2	E2	a[3]={7,8,9}, v=5	Returns 0
TC3	E3	a[0]={}, v=10	Returns 0
Boundary Value Analysis			
TC4	BVA1(Count of searched element is 1)	a[1]={1}, v=1	Returns 1
TC5	BVA2 (Count of searched element is equal to length of array)	a[5] = {2,2,2,2,2},v=2	Returns 5
TC6	BVA3(Count of searched element is more than 1 but less than length of array)	a[5] = {1,2,3,5,5},v=5	Returns 2
TC7	BVA3(Length of array is 1 but searched element is not present)	a[5] = {1},v=2	Returns 0
TC8	BVA4(Length of array is 1 and searched element is present)	a[5] = {1},v=1	Returns 1

Program 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. Assumption: the elements in the array a are sorted in non-decreasing order.

Code:

```
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);
}
```

Solution) Equivalence partitioning and Boundary Value Analysis for the given program are as follows

- Derived Equivalence Partitioning:

Sr No.	**Equivalence Class**	**Acceptance**
E1	Element is Present	Returns i(a[i]==v)
E2	Element is not present	Returns -1
E3	Empty array	Returns -1

- From the given set of Equivalence partitioning and Boundary Value Analysis the required set of test suit are as follows

Test case	Tester Action	Input data	Expected Outcome
Equivalence Partitioning			
TC1	E1	a[3]={4,5,6} , v = 5	Returns 1
TC2	E2	a[3]={7,8,9}, v=5	Returns 0
TC3	E3	a[0]={}, v=10	Returns 0
Boundary Value Analysis			
TC4	BVA1(Single element in array)	a[1]={1}, v=1	Returns 0
TC5	BVA2 (First Element in the array)	a[5] = {2,3,4,5,8,12},v= 2	Returns 0
TC6	BVA3(last element of the array)	a[5] = {1,2,3,4,5},v=5	Returns 4
TC7	BVA3(more than 1 element found in the array)	a[5] = {1,2,2,2,3},v=2	Returns 1

Program 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).

Code:

```
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);
}
```

Solution) Equivalence partitioning and Boundary Value Analysis for the given program are as follows

- Derived Equivalence Partitioning:

Sr No.	**Equivalence Class**	**Acceptance**
E1	Equilateral Triangle	Returns 0
E2	Isosceles Triangle	Returns 1
E3	Scalene Triangle	Returns 2
E4	Invalid Triangle	Returns 3

- From the given set of Equivalence partitioning and Boundary Value Analysis the required set of test suit are as follows

Test case	Tester Action	Input data	Expected Outcome
Equivalence Partitioning			
TC1	E1	Sides = {3,3,3}	Returns 0
TC2	E2	Sides = {3,4,3}	Returns 1
TC3	E3	Sides = {3,4,5}	Returns 2
TC4	E4	Sides = {3,1,5}	Returns 3
TC5	E4	Sides = {-1,-1,5}	Returns 3
Boundary Value Analysis			
TC6	BVA1(Minimum Valid triangle)	sides={1,1,1}	Returns 0
TC7	BVA2 (Minimum Invalid triangle)	sides = {1,2,3}	Returns 3
TC8	BVA3(edge case for isosceles)	sides = {1,2,1}	Returns 1
TC9	BVA3(edge case for scalene)	sides = {1,2,3}	Returns 2

Problem 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).

Code:

```
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;
}
```

Solution) Equivalence partitioning and Boundary Value Analysis for the given program are as follows

- Derived Equivalence Partitioning:

Sr No.	**Equivalence Class**	**Acceptance**
E1	s1 is a prefix od s2	Returns true
E2	s1 is not a prefix of s2	Returns false
E3	s1 is empty	Returns true(Empty string is always a prefix)
E4	s2 is empty	Returns false(if s1 is not empty)

- From the given set of Equivalence partitioning and Boundary Value Analysis the required set of test suit are as follows

Test case	Tester Action	Input data	Expected Outcome
Equivalence Partitioning			
TC1	E1	s1=’ab’, s2=’abcd’	Returns true
TC2	E2	s1=’abf’, s2=’abcd’	Returns false
TC3	E1	s1=’abcd’, s2=’abcd’	Returns true
TC4	E3	s1=’’, s2=’kdjfcd’	Returns true
TC5	E4	s1=’abcd’, s2=’’	Returns false
Boundary Value Analysis			
TC6	BVA1(length of s1 greater than length of s2)	s1=’abcdo’, s2=’abcd’	Returns false
TC7	BVA2 (length of s1 equal to length of s2)	s1=’abcd’, s2=’abcd’	Returns true
TC8	BVA3(length of s1 is one character)	s1=’a’, s2=’abcd’	Returns true
TC9	BVA3(length of s1 is one character but not a prefix of s2)	s1=’o’, s2=’abcd’	Returns false

Program 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

Solution) Equivalence partitioning for the given system as follows

- Derived Equivalence Partitioning:

Sr No.	**Equivalence Class**	**Acceptance**
E1	Equilateral Triangle	Returns 0
E2	Isosceles Triangle	Returns 1
E3	Scalene Triangle	Returns 2
E4	Right-angled Triangle	Returns 3
E5	Invalid Triangle	Returns 4
E6	Non-positive input	Returns 5

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)

Solution) Required test cases are as follows:

Test case	**Tester Action**	**Input data**	**Expected Outcome**
Equivalence Partitioning			
TC1	E1	Sides = {3,3,3}	Returns 0

TC2	E2	Sides = {3,4,3}	Returns 1
TC3	E3	Sides = {3,8,5}	Returns 2
TC4	E4	Sides = {12,5,13}	Returns 3
TC5	E5	Sides = {1,2,3}	Returns 4
TC6	E6	Sides = {1,0,11}	Returns 5
TC7	E6	Sides = {1,-1,8}	Returns 5

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

Solution) Required test cases are as follows:

Test case	Input data	Expected Outcome
TC8	Sides = {3,4,18}	Returns 2
TC9	Sides = {35,32,34}	Returns 2
TC10	Sides = {12,5,13}	Returns 2

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

Solution) Required test cases are as follows:

Test case	**Input data**	**Expected Outcome**

TC11	Sides = {3,3,8}	Returns 1
TC12	Sides = {4,4,6}	Returns 1
TC13	Sides = {1,2,1}	Returns 1

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

Solution) Required test cases are as follows:

Test case	**Input data**	**Expected Outcome**
TC14	Sides = {3,3,3}	Returns 0
TC15	Sides = {4,4,4}	Returns 0
TC16	Sides = {1,1,1}	Returns 0

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

Solution) Required test cases are as follows:

Test case	**Input data**	**Expected Outcome**
TC17	Sides = {3,4,5}	Returns 3
TC18	Sides = {12,13,5}	Returns 3
TC19	Sides = {24,7,25}	Returns 3

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

Solution) Required test cases are as follows:

Test case	**Input data**	**Expected Outcome**
TC20	Sides = {1,3,1}	Returns 4
TC21	Sides = {1,2,3}	Returns 4
TC22	Sides = {78,9,2}	Returns 4

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

Solution) Required test cases are as follows:

Test case	**Input data**	**Expected Outcome**
TC23	Sides = {3,0,1}	Returns 5
TC24	Sides = {-4,0,14}	Returns 5
TC25	Sides = {-1,-1,-11}	Returns 5