Lab 8: Software Engineering

Name: Pritish N Desai

Id: 202201312

Functional Testing (Black-Box)

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?

The equivalence class for the required conditions are as follows-

Class 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 no.	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 (both 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 natural number	Invalid
E9	The values in the month field are less than 1	Invalid
E10	The values in the month field are	Valid

	between 1 and 12 (both inclusive)	
E11	The values in the month field are greater than 12	Invalid
E12	The month field is blank	Invalid
E13	The year field is not a natural no.	Invalid
E14	The values in the year field are less than 1900	Invalid
E15	The values in the year field are between 1900 and 2015 (both 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:

TC No.	**Test case**	**Expected Outcome**	**Classes Covered**
1)	(25,9,2003)	T	E1,E5,E10,E15
2)	(ds,12,1902)	F	E2
3)	(1.4..,_,1948)	F	E3,E12
4)	(_,9,1890)	F	E7,E14
5)	(32,8,1967)	F	E6

6)	(_,7,1988)	F	E7
7)	(9,-0.75,2018)	F	E8
8)	(13,0,1901)	F	E9
9)	(19,4,_)	F	E17
10)	(25,1,2025)	T	E1,E5,E10,E15
11)	(3,1,2007)	T	E1,E5,E10,E15
12)	(12,12,2022)	T	E1,E5,E10,E15
13)	(6,2,infinity)	F	E13
14)	(19,901,2001)	F	E11
15)	(15,5,2005)	T	E1,E5,E10,E15

Q2. 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:

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.

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

Derived Equivalence Partitioning:

Sr No.	Equivalence Class	Acceptance
E1	Element is Present	Return i such that a[i]==v
E2	Element is not present	Return -1
E3	Empty array	Return -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) a[1]={1}, v=1 Returns 0
TC5 BVA2 (first element) a[5] = {1,2,3,4,5},v=1 Returns 0
TC6 BVA3(last element) a[5] = {1,2,3,4,5},v=5 Returns 4
TC7 BVA3(more than 1 element) 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

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

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	Return count(no. Of element inside the array)

E2	Element is not present	Return 0
E3	Empty array	Return 0

TC No.	**Tester Action**	**Input data**	**Expected outcome**
Equivalence Partitioning			
1)	E1	a[3]={4,5,6},v=5	Returns 1
2)	E2	a[3]={7,8,9},v=5	Return 0
3)	E3	a[0]={},v=10	Return 0
Boundary Value Analysis			
4)	BVA1(count of element is 1)	a[1]={1},v=1	Return 1
5)	BVA2(count of element is equal to length of array	a[5]={2,2,2,2,2},v=2	Return 5

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.

```
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);
}
```

Derived Equivalence Partitioning:

Sr No.	Equivalence Class	Acceptance
E1	Element is Present	Return i such that a[i]==v
E2	Element is not present	Return -1
E3	Empty array	Return -1

TC No.	**Tester Action**		**Input data**	**Expected outcome**
Equivalence Partitioning				

1)	E1	a[3]={4,5,6},v=5	Returns 1
2)	E2	a[3]={7,8,9},v=5	Return -1
3)	E3	a[0]={},v=10	Return -1
Boundary Value Analysis			
4)	BVA1(count of element is 1)	a[1]={1},v=1	Return 0
5)	BVA2(more than one element found)	a[5]={2,2,2,2,2},v=2	Return 0
6)	BVA3(first element)	a[5]={1,2,3,4,5},v=1	Return 0
7)	BVA4(last element)	a[5]={1,2,3,4,5}, v=5	Return 4

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

Equivalence partitioning for the given system as follows
Derived Equivalence Partitioning:

Sr No.	Equivalence Class	Acceptance
E1	Equilateral Triangle	Return 0
E2	Isoceles triangle	Return 1
E3	Scalene triangle	Return 2
E4	Right-angled triangle	Return 3
E5	Invalid triangle	Return 4
E6	Non positive input	Return 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)

TC No.	**Tester Action**		**Input data**	**Expected outcome**
Equivalence Partitioning				

1)	E1	sides={3,3,3}	Returns 0
2)	E2	sides={3,4,3}	Return 1
3)	E3	sides={3,8,5}	Return 2
4)	E4	sides={12.5, 5.5,13.1}	Return 3
5)	E5	sides={1.1,2.5,3.5}	Return 4
6)	E6	sides={1,0,11}	Return 5
7)	E7	sides={1,-1,8}	Return 5

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

Required test cases are as follows:

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

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

Required test cases are as follows

Test case	Input data	Expected outcome
TC11	Sides={3,3,8}	Return 1
TC12	sides={4,4,6}	Return 1
TC13	sides={1,2,1}	Return 1

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

Required test cases are as follows

Test case	Input data	Expected outcome
TC14	Sides={3,3,3}	Return 0
TC15	sides={4,4,4}	Return 0
TC16	sides={1,1,1}	Return 0

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

Required test cases are as follows

Test case	Input data	Expected outcome
TC17	Sides={3,4,5}	Return 3
TC18	sides={24,7,25}	Return 3

TC19	sides={12,13,5}	Return 3

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

Required test cases are as follows

Test case	Input data	Expected outcome
TC20	Sides={1,3,1}	Return 4
TC21	sides={1,2,3}	Return 4
TC22	sides={78,9,2}	Return 4

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

Required test cases are as follows

Test case	Input data	Expected outcome
TC23	Sides={3,0,1}	Return 5
TC24	sides={-4,0,14}	Return 5
TC25	sides={-1,-1,-11}	Return 5