Lab 8
Functional Testing (Black-Box)
Name - Desai Aditya Veeral Student ID - 202201451

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?

A.1
Equivalence Partitioning

Class Name	Equivalence Classes	Expected Outcome
E1	Day value is < 1	Invalid
E2	Day value is > 31	Invalid
E3	Day value is between 1 and 31	Valid
E4	Month value is < 1	Invalid
E5	Month value is > 12	Invalid
E6	Month value is between 1 and 12	Valid
E7	Year value is < 1900	Invalid
E8	Year value is > 2015	Invalid
E9	Year value is between 1900 and 2015	Valid

Test Case Table

Test Case	Input Data (day, month, year)	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	(0, 5, 2010)	E1	Error message: "Invalid date"	Tests invalid day (below range)

Test Case	Input Data (day, month, year)	Equivalence Classes Covered	Expected Outcome	Remarks
TC2	(32, 5, 2010)	E2	Error message: "Invalid date"	Tests invalid day (above range)
TC3	(15, 0, 2010)	E4	Error message: "Invalid date"	Tests invalid month (below range)
TC4	(15, 13, 2010)	E5	Error message: "Invalid date"	Tests invalid month (above range)
TC5	(15, 5, 1800)	E7	Error message: "Invalid date"	Tests invalid year (below range)
TC6	(15, 5, 2020)	E8	Error message: "Invalid date"	Tests invalid year (above range)
TC7	(15, 5, 2010)	E3, E6, E9	Previous date: 14-5-2010	Valid date within range
TC8	(1, 5, 2010)	E3, E6, E9	Previous date: 30-4-2010	Tests boundary day at minimum
TC9	(31, 5, 2010)	E3, E6, E9	Previous date: 30-5-2010	Tests boundary day at maximum
TC10	(15, 1, 2010)	E3, E6, E9	Previous date: 14-1-2010	Tests boundary month at minimum
TC11	(15, 12, 2010)	E3, E6, E9	Previous date: 14-12-2010	Tests boundary month at maximum
TC12	(15, 5, 1900)	E3, E6, E9	Previous date: 14-5-1900	Tests boundary year at minimum
TC13	(15, 5, 2015)	E3, E6, E9	Previous date: 14-5-2015	Tests boundary year at maximum

Test Case	Input Data (day, month, year)	Equivalence Classes Covered	Expected Outcome	Remarks
TC14	(1, 1, 2010)	Special case (first day of the year)	Previous date: 31-12-2009	Tests special case, year change
TC15	(1, 1, 1900)	Special case (first day of the first valid input year)	Error message: "Invalid date"	Tests special case
TC16	(31, 6, 2001)	Special Case	Error message: "Invalid date"	June does not have 31 days.
TC17	(29,2,2004)	Special Case: Feb in Leap year	Previous date: 28-02-2004	Feb has 29 days during Leap year
TC18	(30,2,2004)	Special Case: Feb has 28/29 days	Error message: "Invalid date"	Feb has 28/29 days only

Q.2. 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.

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.

The solution of each problem must be given in the format as follows:

Tester Action and Input Data

Equivalence Partitioning	***Expected Outcome***
a, b, c	An Error message
a-1, b, c	Yes
Boundary Value Analysis	
a, b, c-1	Yes

Q.2. Programs:

Program 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);
}
```

Solution 1:

Class Name	**Equivalence Classes**	**Expected Outcome**
E1	Element preset	Returns i such that a[i] == v
E2	Element not present	Returns -1
E3	Empty array	Returns -1

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC1	E1	Array = [4,2,3], V = 2	Return 1 (i=1)
TC2	E2	Array = [5,1,3], V = 4	Return -1
TC3	E3	Array = [], V = 2	Return -1
Boundary value analysis			
TC4	BV1(only 1 element)	Array = [1], V = 1	Return 0 (i=0)
TC5	BV2(first element of array)	Array = [1,8,3] V = 1	Return 0 (i=0)
TC6	BV3 (Last element of array)	Array = [1,9,3] V = 3	Return 2 (i=2)
TC7	BV4(More than 1 same elements present)	Array = [4,2,2,3] V = 2	Return 1 (i=1)

Program 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);
}
```

Solution 2:

Class Name	Equivalence Classes	Expected Outcome
E1	Elements present between 1 and N(length of array)	Returns count of elements
E2	Element not present	Returns 0
E3	Empty array	Returns 0

Equivalence Partitioning

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC1	E1	Array = [10,2,2], V = 2	Return 2 (count=2)
TC2	E2	Array = [9,2,3], V = 4	Return 0
TC3	E3	Array = [], V = 2	Return 0
Boundary value analysis			
TC4	BV1(only 1 element)	Array = [1], V = 1	Return 1 (count=1)
TC5	BV2(only one occurrence in array)	Array = [1,2,4,4] V = 1	Return 1 (count=1)
TC6	BV3 (all elements are same)	Array = [3,3,3,3] V = 3	Return 3 (count=3)
TC7	BV4(none elements to counted)	Array = [4,2,3], V = 5	Return 0 (count=0)

Program 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. 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);
}
```

Solution 3:

Class Name	**Equivalence Classes**	**Expected Outcome**
E1	Element preset	Returns i such that a[i] == v
E2	Element not present	Returns -1
E3	Empty array	Returns -1

Test Case	**Equivalence Classes Covered**	**Input Data**	**Expected Outcome**
Equivalence Partitioning			
TC1	E1	Array = [1,3,5,7], V = 5	Return 2 (i=2)
TC2	E2	Array = [1,2,3,5], V = 7	Return -1
TC3	E3	Array = [], V = 2	Return -1
Boundary value analysis			

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC4	BV1(only 1 element)	Array = [1], V = 1	Return 0 (i=0)
TC5	BV2(first element of array)	Array = [1,2,3] V = 1	Return 0 (i=0)
TC6	BV3 (Last element of array)	Array = [1,2,3,9] V = 9	Return 3 (i=3)
TC7	BV4(More than 1 same elements present)	Array = [1,4,4,3,5] V = 4	Return 1 (i=1)

Program 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);
}
```

Solution 4:

Class Name	Equivalence Classes	Expected Outcome
E1	a==b and b==c i.e. Triangle is Equilateral	Returns 0
E2	a==b or b==c or a==c i.e. Triangle is Isosceles	Returns 1
E3	a!=b, b!=c, c!=a i.e. Triangle is Scalene	Returns 2
E4	a>=b+c or b>=a+c or c>=a+b i.e. impossible	Returns 3
E5	One or more sides are non positive	Returns 3

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC1	E1	a = 3, b = 3, c = 3	Return 0
TC2	E2	a = 2, b = 2, c = 3	Return 1
TC3	E3	a = 3, b = 4, c = 5	Return 2
TC4	E4	a = 1, b = 1, c = 2	Return 3
TC5	E5	a = -1, b = 1, c = 3	Return 3
Boundary value analysis			
TC6	BV1(all sides same but 0)	a = 0, b = 0, c = 0	Return 3
TC7	BV2(all sides 1)	a = 1, b = 1, c = 1	Return 0
TC8	BV3(two sides same but 0)	a = 0, b = 0, c = 2	Return 3

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC9	BV4 (all sides different but one side 0)	a = 0, b = 1, c = 2	Return 3
TC10	BV5(Two sides same but 1)	a = 1, b = 1, c = 2	Return 3
TC11	BV6(All sides different and one side 1)	a = 1, b = 2, c = 3	Return 3

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

Solution 5:

Class Name	Equivalence Classes	Expected Outcome
E1	Length of s1>length of s2	Returns false
E2	s1 is not a prefix of s2	Returns false
E3	s1 is a prefix of s2	Returns true
E4	s1 is empty string	Return true
E5	s1 = s2	Return true

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC1	E1	s1 = ‘fruit’, s2 = ‘fru’	Return false
TC2	E2	s1 = ‘fruit’, s2 = ‘fried’	Return false
TC3	E3	s1 = ‘fru’, s2 = ‘fruit’	Return true
TC4	E4	s1 = ‘’, s2 = ‘fruit’	Return true
TC5	E5	s1 = ‘fruit’, s2 = ‘fruit’	Return true
Boundary value analysis			
TC6	BV1(empty string prefix)	s1 = ‘’, s2 = ‘fruit’	Return true
TC7	BV2(single char string)	s1 = ‘f’, s2 = ‘f’	Return true
TC8	BV3(equal length string)	s1 = ‘fruit’, s2 = ‘fruit’	Return true

Program 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

Class Name	**Equivalence Classes**	**Expected Outcome**
E1	a==b and b==c i.e. Triangle is Equilateral	Returns Equilateral
E2	a==b or b==c or a==c i.e. Triangle is Isosceles	Returns Isosceles
E3	a^2 +b^2 = c^2 i.e. right angled triangle	Returns Right Angled
E4	a!=b, b!=c, c!=a i.e. Triangle is Scalene	Returns Scalene
E5	a>=b+c or b>=a+c or c>=a+b i.e. impossible	Returns Invalid
E6	One or more sides are non positive	Returns Non positive values

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)

Test Case	**Equivalence Classes Covered**	**Input Data**	**Expected Outcome**
Equivalence Partitioning			
TC1	E1	a = 3.1, b = 3.1, c = 3.1	Return Equilateral

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Equivalence Partitioning			
TC2	E2	a = 2.5, b = 2.5, c = 3	Return Isosceles
TC3	E3	a = 3, b = 4, c = 5	Return Right Angled
TC4	E4	a = 5, b = 6, c = 7	Return Scalene
TC5	E5	a = 1, b = 2, c = 3	Return Invalid
TC6	E6	a = -1, b = 1, c = 3	Return Non positive values

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

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(all sides different but one side 0)	a = 0, b = 3, c = 2	Return Non positive values
TC2	BV2(all sides different but impossible)	a = 1.1, b = 2.2, c = 3.2	Return Invalid
TC3	BV3(All sides different but but one side negative)	a = -1, b = 4, c = 2	Return Non positive values
TC4	BV5(All sides different but but one side negative)	a = -1.4, b = 3, c = 2	Return Non positive values
TC5	BV6(All sides different but right angled)	a = 3, b = 4, c = 5	Return Right angled
TC6	BV7	a = 6, b = 7, c = 8	Return Scalene

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

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(two sides 0)	a = 0, b = 3, c = 0	Return Non positive values
TC2	BV2(two sides same but impossible)	a = 1.1, b = 2.2, c = 1.1	Return Invalid
TC3	BV3(Two sides same but negative)	a = -1, b = 4, c = -1	Return Non positive values
TC4	BV5(Two sides same but one side negative)	a = 2, b = -3, c = 2	Return Non positive values
TC5	BV6(All sides same)	a = 5, b = 5, c =5	Return Equilateral
TC6	BV7	a = 2, b = 3, c = 2	Return Isosceles

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

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(all sides 0)	a = 0, b = 0, c = 0	Return INon positive values
TC2	BV2(all sides negative)	a = -1.1, b = -1.1, c = -1.1	Return Non positive values
TC3	BV3	a = 3, b = 3, c = 3	Return Equilateral

f) For the boundary condition A^2 + B^2 = C^2 case (right-angle triangle), identify test cases to verify the boundary.

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(one side negative)	a = -3, b = 4, c = 5	Return Non positive values
TC2	BV2(one side zero)	a = 0, b = 4, c = 4	Return Non positive values
TC3	BV3(all sides zero)	a = 0, b = 0, c = 0	Return Non positive values
TC4	BV4	a = 3, b = 4, c = 5	Return Right Angled
TC5	BV5(all sides one)	a = 1, b = 1, c = 1.414	Return Right Angled

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

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(one side negative)	a = -3, b = 4, c = 5	Return Non positive values
TC2	BV2(one side zero)	a = 0, b = 4, c = 4	Return Non positive values
TC3	BV3(all sides zero)	a = 0, b = 0, c = 0	Return Non positive values
TC4	BV4	a = 1.5, b = 1.5, c = 3	Return Invalid

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

Test Case	Equivalence Classes Covered	Input Data	Expected Outcome
Boundary value analysis			
TC1	BV1(one side negative)	a = -3, b = 4, c = 5	Return Non positive values
TC2	BV2(one side zero)	a = 0, b = -4, c = 4	Return Non positive values
TC3	BV3(all sides zero)	a = 0, b = 0, c = 0	Return Non positive values
TC4	BV4	a = -3, b = -4, c = -5	Return Non positive values