Lab 8 IT314
Harshal Patel - 202201070

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? 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.

Ans: **Equivalence Partitioning : Test Cases**

Tester Action and Input Data Equivalence Partitioning	**Validity**	**Expected Outcome: Previous or invalid**	**Equivalence class**
Input 9, 6, 2004	Valid	9, 6, 2004	E1
Input 4, 9, 1989	Valid	3, 9, 1989	E2
Input 30 ,6 ,2004	Valid	29,6,2004	E3
Input 2, 12, 1982	Valid	1,12,1982	E4
Input 1, 3, 2016	Valid	29,2,2016	E5
Input 2, 21, 1982	Invalid [month]	Invalid date	E6
Input 76, 12, 2004	Invalid [day]	Invalid date	E7
Input 2, 21, 1882	Invalid [year]	Invalid date	E8

→Boundary Value Analysis:

Tester Action and Input Data	**Boundary conditions**	**Expected Outcome: Previous or invalid**	**Valid /Not Valid**
Input 29, 2, 2004	Leap Year	28, 2, 2004	Valid
Input 1, 1, 2004	Boundary condition	31, 12, 2003	Valid
Input 31, 12, 2015	Boundary condition	30, 12, 2015	Valid

Input 1, 2, 2016	Leap Year	29, 9, 2016	Valid
Input 0,6,2004	Boundary condition	Error	Invalid
Input 31,2,2013	No leap year	Error	Invalid
Input 29,2,2014	No leap year	Error	Invalid

Q2) Q.2. 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);
}
```

Ans: Equivalence Partitioning : Test Cases

Tester Action and Input Data Equivalence Partitioning	**Expected Outcome: Indexes or returns -1 if not found**
Input : [1, 2, 3, 4, 5] , 5	4
Input : [34, 43,215, 3], 215	2
Input : [1, 0, 2, 4], 66	-1

Boundary Value Analysis :

Tester Action and Input Data	**Expected output:**	**Reasoning**
Input : [1, 0 , 9, [empty]]	-1	Empty array element
Input : [1],1	0	Single element and present
Input : [1],2	-1	Single element but not

		present
Input : [empty]	-1	Empty input
Input : [0, 1, 2],0	0	First index
Input : [1, 8, 9] , 9	2	Last index
Input : [1,3,4],3	1	Middle index
Input : [0,3,4],0		Lowest element in size
Input : [21474836,3,4],21474836	0	Largest possible element in size

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);

}
```

Ans: Equivalence Partitioning : Test Cases

Tester Action and Input Data Equivalence Partitioning[Array,int]	**Expected Outcome: Count**	**Reasoning**	**Equivalence classes**
Input : [1, 1, 1, 4, 5] , 5	1	3 appears one time	E1
Input : [34, 43,215, 215], 215	2	215 appears once	E2
Input : [1, 0, 2, 4], 66	0	Element does not appear	E3

Input : [empty]	0	Empty array	E4
Input : [1,2,3],2	Error	Invalid size	E5
Input : [‘a’],’a’	Error	Invalid input type	E6

Boundary Value Analysis :

Tester Action and Input Data	**Reasoning**	**Expected output:**
Input : [1, 0 , 9, [empty]]	Minimum possible size of element	0
Input : [2147483647,2147483647,1, 3],2147483647	Maximum possible size of element	
Input : [0, 0, 0],0	Count of zero is 3 (Multiple appearances)	3
Input : [1, 8, 9] , 9	Last element	1
Input : [1, 8, 9] , 1	First element	1
Input : [empty]	No element to count	0
Input : [1],1	Single element	1
Input : [1],3	Single element and no element found	0

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);
}
```

Ans: Equivalence Partitioning : Test Cases

Tester Action and Input Data Equivalence Partitioning	**Expected Outcome: Returns -1 if not found**	**Valid / Invalid**	**Reasoning**
Input : [1, 2, 3, 4, 5] , 5	4	Valid	Value is present in the array
Input : [34, 43,215, 215], 215	2	Valid	Value is present at the end
Input : [34, 43,215, 215], 34	0	Valid	Value is present at the start
Input : [1, 0, 2, 4], 66	-1	Valid	Element not found
Input : [1],1	0	Valid	Single element is there and is present
Input : [1],5	-1	Valid	Single element is there and is not present
Input : [1,2,3],2	Error[-1]	Invalid	Invalid size of the array

Boundary Value Analysis :

Tester Action and Input Data	**Expected output:**	**Reasoning**
Input : [empty]	-1	No element is present
Input : [2],2	0	Single element is there in the array

Input : [2],3	-1	Single element is there and element not found
Input : [1, 0 , 9] , 9	2	Element found at last position
Input : [0, 0, 0],0	0	Element found at first position
Input : [1, 2, 8, 9, 2] , 8	2	Element in middle
Input : [0, 2, 8, 9, 2] , 0	0	Smallest possible integer
Input : [1, 2147483647, 8, 9, 2] , 2147483647	1	Largest possible integer

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);
}
```

Ans: Equivalence Partitioning : Test Cases

Tester Action and Input Data Equivalence Partitioning	**Expected Outcome: Returns the triangle or Invalid**	**Equivalence classes**	**Reasoning**
Input : 1,1,1	Equilateral	E1	All sides are equal

Input : 9,9,10	Isosceles	E2	Two sides are equal.
Input : 9,8,9	Isosceles	E3	Two sides are equal.
Input : 3,4,5	Scalene Triangle	E4	All the sides are different
Input : 2,5,10	Invalid	E5	Does not satisfy triangle inequality
Input : 10,5,1	Invalid	E6	One side is too long
Input : 0,6,3	Invalid	E7	One side is 0
Input : -1,-1,-1	Invalid	E8	Negative sides

Boundary Value Analysis :

Tester Action and Input Data	**Expected output:**	**Reasoning**
Input : 1,1,1	Equilateral Triangle	Smallest triangle
Input : 2,2,4	Invalid	Sum of 2 sides is equal to other
Input : 2,3,4	Scalene Triangle	Smallest scalene triangle
Input : 1000,1000,1000	Equilateral	Large Equilateral
Input : 1000,1000,1500	Isosceles	Two side's sum is equal to other
Input : -4,-4,-4	Invalid	Negative input sides
Input : 1000,3,2	Invalid	One side is too large

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;
}
```

Ans: Equivalence Partitioning : Test Cases

Tester Action and Input Data Equivalence Partitioning	**Expected Outcome:True or false**	**Reasoning**	**Equivalence classes**
Input : “harshal”, “harshal”	True	Prefix matches fully	E1
Input : “Samay”, “Sam”	True	Prefix match	E2
Input : “Goli”, “Martha”	False	Not a prefix	E3
Input : “beauty”, “beau”	False	Prefix match input is larger	E4
Input : “Samay”, “Sam”	True	Non empty value in input	E5
Input : “”	False	Empty input value	E6

Boundary Value Analysis :

Tester Action and Input Data	**Expected output:**	**Reasoning**
Input : “harshal”, “[Empty]”	False	No input in prefix match
Input : “[Empty]”, “[Empty]”	True	Empty string match
Input : “Me”,”Me”	True	Exact match
Input : “Hi”, “Hiiii”	True	Match
Input : “Hi”,Hello	Flase	No match

Input : “longstring”,”short” [prefix is longer]	False	Longer input prefix match string
Input : “h”, “hiiii”	True	Single character match
Input : “h”, “biiii”	False	SIngle character no match

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.
Ans: Equivalence classes for the given question:

a) **Equivalence classes:**
   1] Equilateral Triangle : If all the sides are equal.
   2] Isosceles Triangle : If two sides are equal
   3] Scalene Triangle : If all the sides are different
   4] Input Error : If Negative value is put in input, invalid
   5] Right Angled Triangle : If a^2 + b^2 = c^2 (a,b,c) are triangle side inputs

6] Invalid sided input : If inequality is not satisfied then Invalid i.e. (A + B <= C) where A,B,C are side inputs for the triangle

Ans: Test Cases

(b)

Tester Action and Input Data	**Validity**	**Expected Outcome:**	**Equivalence class**
Input 1.0, 1.0, 1.0	Valid	Equilateral	E1
Input 2.0, 3, 2.0	Valid	Isosceles	E2
Input 3.0,4.0 ,5.0	Valid	Scalene	E3
Input 3.0, 4.0, 5.0	Valid	Right Angled	E4
Input 10.0, 3.0, 12.0	Invalid	Invalid Triangle	E5
Input -2.0, 21.0, 9.0	Invalid	Invalid (Negative Side)	E6
Input 0.0, 0.0, 0.0	Invalid	Invalid (No triangle)	E7

(c)

→Boundary Condintion and test cases for inequality testing

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 3.0, 5.0, 8.0	Invalid	Inequality not satisfied
T2	Input 2.0, 3, 2.0	Valid	Satisfied
T4	Input 3.0, 4.0, 2.0	Invalid	Not satisfied

(d) Test Cases for the Boundary Condition where Two sides are equal i.e. Isosceles Triangle

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 3.0, 2.0, 3.0	Valid	A=C

(e) Test Cases for the Boundary Condition where A = B = C i.e. Equilateral triangle

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 3.0, 3.0, 3.0	Valid	Satisfied Condition
T2	Input 3.0, 3.0, 3.2	Invalid (Isosceles triangle)	Condition not Satisfied ,C should be some-what smaller
T4	Input 3.0, 3.0, 3.3	Invalid (Isosceles triangle)	C should be slightly smaller

(f) Test Cases for the Boundary Condition where there is right angled triangle : A^2 + B^2 = C^2

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 3.0, 4.0, 5.0	Valid and Right Angled Triangle	Satisfied Condition
T2	Input 3.0, 4.0, 4.0	Scalene	C should be some-what larger
T4	Input 3.0, 4.0, 5.3	Invalid	C should be slightly smaller

(g) Test Cases for the Invalid Triangle Cases : Where inequality is not satisfied

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 3.0, 4.0, 10.0	Invalid	Not Satisfied Condition
T2	Input 3.0, 4.0, 10.10	Invalid	Not Satisfied Condition

(h) Test Cases for Negative Input Sides :

Test Cases :	**Test Inputs: A,B,C**	**Validity**	**Boundary Conditions:**
T1	Input 0.0, 4.0, 1.0	Invalid	Not Satisfied Condition as one side is zero
T2	Input -3.0, 4.0, 5.0	Invalid	Not Satisfied Condition as first input side is negative