IT314 - SOFTWARE ENGINEERING

LAB - 8 FUNCTIONAL TESTING (BLACK BOX)

202201444 - VARDHMAN MEHTA

1) EQUIVALENCE CLASSES:

Equivalence class	Valid / Invalid	Description
1<= day <= 28	valid	Valid day range for all months
Day = 29	valid	Valid for all months (for february, it should be leap year)
Day = 30	valid	Valid for all months except february
Day = 31	valid	Valid for months with 31 days
Day < 1	invalid	Invalid for any month
Day > 31	Invalid	Invalid for any month
1 <= month <= 12	valid	Valdi range
Month > 12	Invalid	Invalid range
Month < 1	Invalid	Invalid range
1900 <= year <= 2015	valid	Valid year range
Year < 1900	invalid	Invalid year range
Year > 2015	invalid	Invalid year range
Year divisible by 4 but not 100, or divisible by 400	valid	Valid leap year
Year divisible by 100 but not by 400	invalid	Invalid leap year

TEST CASES :

Test Case	Day	Month	Year	Expected Output	BVA
Case 1	15	6	2010	14/06/2010	DAY : INTERMEDIATE
Case 2	1	3	2000	29/02/2000	DAY : MIN
Case 3	0	5	2020	Invalid Date	DAY : MIN-1
Case 4	30	2	2019	Invalid Date	DAY : MAX+1
Case 5	31	12	2015	30/12/2015	DAY : MAX
Case 5	1	1	1900	31/12/1899	MONTH : MIN
Case 6	19	0	2005	Invalid Date	MONTH : MIN-1
Case 7	30	6	2011	29/06/2011	MONTH : INTERMEDIATE
Case 8	28	12	2001	27/12/2001	MONTH : MAX
Case 9	29	13	2004	Invalid Date	MONTH : MAX+1
Case 10	31	4	1999	Invalid Date	EP (April with 31 Days)
Case 11	29	2	2004	28/02/2004	Leap Year
Case 12	29	2	2005	Invalid Date	Non Leap Year (February with 28 days)
Case 13	1	12	2015	30/11/2015	YEAR : MAX
Case 14	31	13	2014	Invalid Date	YEAR : MAX+1
Case 15	28	2	1990	27/02/1990	YEAR : MIN
Case 16	3	11	1989	Invalid Date	YEAR : MIN-1
Case 17	1	3	2004	29/02/2004	EP (Leap Year Transition)
Case 18	1	3	2005	28/02/2005	Non Leap Year
Case 19	31	9	2006	Invalid Date	EP (September with 31 Days)

Q-2)

P-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.

Equivalence Partitioning :

- The array includes the value v. (valid, return index)
- The array does not include the value v. (invalid, return -1)
- The array is empty.
- The array contains the value v more than once. (valid, return index of first occurrence)

Boundary Value Analysis:

- Test with an array of length 1.
- Test with an array of larger size.
- Test when v matches the first or last element.

TEST CASES :

Test Case	Input (v, a[])	Expected Outcome	Reason
Case 1	(5, [1, 2, 3, 4, 5])	4	Array contains v at index 4.
Case 2	(3, [1, 2, 3, 4, 5])	2	Array contains v at index 2.
Case 3	(6, [1, 2, 3, 4, 5])	-1	Array does not contain v.

Case 4	(3, [])	-1	Array is empty, so v cannot be found.
Case 5	(3, [3, 1, 3, 4, 5])	0	Array contains v multiple times, first occurrence at index 0.
Case 6	(5, [5])	0	Single element array where v is the first and only element.
Case 7	(10, [5, 6, 7, 8, 9, 10])	5	v is at the last index of the array.
Case 8	(1, [1, 2, 3, 4, 5])	0	v is at the first index of the array.
Case 10	(7, [-5, -3, 0, 7, 10])	3	Array contains negative, zero, and positive values; v is present.
Case 11	(-3, [-5, -3, 0, 7, 10])	1	v is a negative number in the array.
Case 12	(100, [5, 10, 20, 30, 40])	-1	v is much larger than any element in the array.
Case 13	(-100, [5, 10, 20, 30, 40])	-1	v is much smaller than any element in the array.

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

Equivalence Partitioning:

- The array does not include v.
- The array includes v exactly once.
- The array contains v multiple times.
- The array is empty.

Boundary Value Analysis:

- Array with only one element.
- Array with a larger size.
- v appears at the first and last element of the array.

TEST CASES :

Test Case	Input (v, a[])	Expected Outcome	Reason
Case 1	(5, [1, 2, 3, 4, 5])	1	Array contains v exactly once at the last index.
Case 2	(3, [1, 2, 3, 4, 5])	1	Array contains v exactly once at the middle.
Case 3	(6, [1, 2, 3, 4, 5])	0	Array does not contain v.
Case 4	(3, [])	0	Array is empty, so v cannot be found.

Case 5	(3, [3, 1, 3, 4, 5])	2	Array contains v multiple times (two occurrences).
Case 6	(5, [5])	1	Single element array where v is the first and only element.
Case 7	(3, [3, 3, 3, 3, 3])	5	Array contains v at all positions (5 occurrences).
Case 8	(10, [5, 6, 7, 8, 9, 10])	1	v is at the last index of the array.
Case 9	(1, [1, 2, 3, 1, 1])	3	v appears multiple times, including the first index.
Case 10	(-3, [-5, -3, 0, 7, 10])	1	v is a negative number present in the array.
Case 12	(100, [5, 10, 20, 30, 40])	0	v is much larger than any element in the array, not found.
Case 13	(-100, [5, 10, 20, 30, 40])	0	v is much smaller than any element in the array, not found.
Case 14	(0, [0, 1, 0, 2, 0])	3	v appears multiple times as zero (3 occurrences).

P-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.

Equivalence Partitioning:

- The array contains v.
- The array is sorted.
- The array is not sorted. (invalid)
- The array does not contain v.
- The array is empty.
- The array contains v multiple times.

Boundary Value Analysis (BVA):

- Test with an array of length 1.
- Test when v matches the first or last element.
- Test when v matches the middle element.

TEST CASES :

Test Case	**Input (v, a[])**	**Expected Outcome**	**Reason**
Case 1	(5, [1, 2, 3, 4, 5])	4	Array contains v at index 4 (last element).
Case 2	(3, [1, 2, 3, 4, 5])	2	Array contains v at index 2 (middle element).

Case 3	(6, [1, 2, 3, 4, 5])	-1	Array does not contain v.
Case 4	(3, [])	-1	Array is empty, so v cannot be found.
Case 5	(3, [3, 3, 3, 3, 3])	Any valid index (0-4)	Array contains multiple occurrences of v (all elements are v).
Case 6	(5, [5])	0	Single element array where v matches the first and only element.
Case 7	(1, [1, 2, 3, 4, 5])	0	v is the first element in the array.
Case 8	(10, [5, 6, 7, 8, 9, 10])	5	v is the last element in the array.
Case 9	(7, [5, 6, 7, 8, 9, 10])	2	v is the middle element of the array.
Case 10	(-3, [-5, -3, 0, 7, 10])	1	v is a negative number present at index 1 in the sorted array.
Case 11	(100, [10, 20, 30, 40, 50])	-1	v is much larger than any element in the array, not found.
Case 12	(-100, [-50, -30, -10, 0, 20])	-1	v is much smaller than any element in the array, not found.

Case 13	(3, [7, -2, 60, -10, 0, 3])	Invalid Input	`Reason : Input array is not sorted.`

P-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).

Suppose three sides are a, b and c.

Equivalence Partitioning:

- a+b<=c or b+c<=a or a+c<=b → invalid (not satisfying triangle property)
- a=b=c → valid → EQUILATERAL TRIANGLE
- (a=b and a!=c) or (a=c and a!=b) or (b=c & b!=a) → valid → ISOSCELES
- a!=b and b!=c and a!=c → valid → SCALENE
- a<=0 or b<=0 or c<=0 → invalid

TEST CASES :

Test Case	Input (v, a[])	Expected Outcome	Reason
Case 1	(3, 3, 3)	0	Equilateral
Case 2	(3, 3, 4)	1	Isosceles
Case 3	(3, 4, 5)	2	Scalene

Case 4	(1, 2, 3)	3	Invalid (inequality fails).
Case 5	(0, 0., 0)	3	Invalid (Side cannot be 0)
Case 6	(1, 1, 1)	0	Equilateral
Case 7	(1, 1, 2)	3	Invalid (inequality fails).
Case 8	(3, -1, 4)	3	Invalid (Negative side)
Case 9	(3, 5, -2)	3	Invalid (Negative edge)
Case 10	(2, 2, 5)	3	Invalid (inequality fails).
Case 11	(1, 2, 1)	3	Invalid (inequality fails).
Case 12	(1000000, 10000000, 10000000)	0	Equilateral

P-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).

Equivalence Classes:

- Exact match (s1 = s2) → valid
- Proper prefix (s1 matches the beginning of s2) → valid
- Empty string (s1 is "") → valid
- Single character match (s1 is a single character that matches the start of s2) → valid
- Longer proper prefix (s1 is a longer prefix of s2) → valid
- s1 is longer than s2 → invalid
- Partial match (s1 does not match the start of s2) → invalid
- Mismatch at start (s1 characters do not match the beginning of s2) → invalid

TEST CASES :

Test Case	Input (s1, s2)	Expected Outcome	Reason
Case 1	("pre", "prefix")	true	s1 is a prefix of s2.
Case 2	("pre", "postfix")	false	s1 does not match the start of s2.
Case 3	("prefix", "pre")	false	s1 is longer than s2.
Case 4	("", "hello")	true	Empty s1 is a prefix of any non-empty s2.

Case 5	("hello", "")	false	Non-empty s1 cannot be a prefix of an empty s2.
Case 6	("", "")	true	Both strings are empty.
Case 7	("test", "test")	true	s1 equals s2 (identical).
Case 8	("hello", "he")	false	s1 is longer than the start of s2.
Case 9	("pre", "prefixe")	true	s1 is a prefix of s2.
Case 10	("abc", "abcd")	true	s1 is a prefix of s2 (lengths differ by 1).
Case 11	("abc", "ab")	false	s1 is not a prefix of s2.
Case 12	("abcdefg", "abc")	false	s1 is longer than s2.
Case 13	("123", "123456")	true	s1 is a prefix of s2.
Case 14	("abc", "123abc")	false	s1 does not match the start of s2.

P-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:

- a+b<=c or b+c<=a or a+c<=b → invalid (not satisfying triangle property)
- a=b=c → valid → EQUILATERAL TRIANGLE
- (a=b and a!=c) or (a=c and a!=b) or (b=c & b!=a) → valid → ISOSCELES
- a!=b and b!=c and a!=c → valid → SCALENE
- a<=0 or b<=0 or c<=0 → invalid
- (a^2 + b^2 = c^2) or (a^2 + c^2 = b^2) or (c^2 + b^2 = a^2) → valid → right angle

b) Test Cases to Cover the Identified Equivalence Classes

Test Case	Input Values (A, B, C)	Expected Output	Covered Equivalence Class
TC1	(3.0, 3.0, 3.0)	"Equilateral"	Equilateral Triangle
TC2	(5.0, 5.0, 8.0)	"Isosceles"	Isosceles Triangle (A = B)
TC3	(7.0, 7.0, 10.0)	"Isosceles"	Isosceles Triangle (A = C)
TC4	(9.0, 12.0, 12.0)	"Isosceles"	Isosceles Triangle (B = C)
TC5	(3.0, 4.0, 5.0)	"Scalene"	Scalene Triangle
TC6	(5.0, 12.0, 13.0)	"Right Angled"	Right-Angled Triangle
TC7	(2.0, 2.0, 5.0)	"Not a triangle"	Non-Triangle
TC8	(0.0, 4.0, 5.0)	"Not a triangle"	Non-Positive Input
TC9	(-1.0, 5.0, 6.0)	"Not a triangle"	Non-Positive Input

c) Boundary Test Cases for Scalene Triangle (A + B > C)

Test Case	Input values(A, B, C)	Expected Output
TC10	(3.0, 4.0, 5.0)	"Scalene"
TC11	(4.0, 5.0, 8.0)	"Scalene"
TC12	(5.0, 7.0, 11.0)	"Not a triangle"

d) Boundary Test Cases for Isosceles Triangle (A = C)

Test Case	Input Values (A, B, C)	Expected Output
TC13	(3.0, 4.0, 3.0)	"Isosceles"
TC14	(5.0, 2.0, 5.0)	"Isosceles"

e) Boundary Test Cases for Equilateral Triangle (A = B = C)

Test Case	Input Values (A, B, C)	Expected Output
TC15	(3.0, 3.0, 3.0)	"Equilateral"
TC16	(0.0, 0.0, 0.0)	"Not a triangle"

f) Boundary Test Cases for Right-Angled Triangle ($A^2 + B^2 = C^2$)

Test Case	Input Values (A, B, C)	Expected Output
TC17	(3.0, 4.0, 5.0)	"Right Angled"

TC18	(5.0, 12.0, 13.0)	"Right Angled"

g) Boundary Test Cases for Non-Triangle

Test Case	Input Values (A, B, C)	Expected Output
TC19	(1.0, 2.0, 3.0)	"Not a triangle"
TC20	(1.0, 1.0, 3.0)	"Not a triangle"
TC21	(0.0, 0.0, 0.0)	"Not a triangle"

h) Test Points for Non-Positive Input

Test Case	Input Values (A, B, C)	Expected Output
TC22	(-1.0, 2.0, 3.0)	"Not a triangle"
TC23	(1.0, -2.0, 3.0)	"Not a triangle"
TC24	(1.0, 2.0, 0.0)	"Not a triangle"
TC25	(0.0, 0.0, 5.0)	"Not a triangle"