Lab08: Functional Testing (Black-Box)

IT314 – Software Engineering

Name: Nitin Kanzariya
ID: 202201181

Q1)

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.

Ans)

Valid Equivalence Classes:

1. EC1: Valid date (e.g., 1st of January 2015).
2. EC2: End of the month (e.g., 31st of January 2015).
3. EC3: Transition from one month to another (e.g., 1st of March to 29th of February).
4. EC4: Leap year (e.g., 29th of February in a leap year).
5. EC5: Invalid day for the month (e.g., 30th of February).
6. EC6: Day less than 1 (e.g., 0th of March).
7. EC7: Day greater than the maximum for the month (e.g., 32nd of January).
8. EC8: Year less than 1900 (e.g., 1899).
9. EC9: Year greater than 2015 (e.g., 2016)

Input Data (day, month, year)	Expected Outcome	Remark
1, 1, 2015	31/12/2014	EC1: Valid date
31, 1, 2015	30/01/2015	EC2: Valid date
1, 3, 2015	28/02/2015	EC3: Valid date
29, 2, 2016	28/02/2016	EC4: Valid leap year
30, 2, 2015	Invalid date	EC5: Invalid day for February (not a leap year).
0, 3, 2015	Invalid date	EC6: Invalid day less than 1.
32, 1, 2015	Invalid date	EC7: Invalid day greater than the maximum for January.
15, 3, 1899	Invalid date	EC8: Invalid year (less than 1900).
15, 3, 2016	Invalid date	EC9: Invalid year (greater than 2015).

Boundary Conditions:

1. BC1: Day = 1 (first day of the month).
2. BC2: Day = 31 (last day of a month with 31 days).
3. BC3: Day = 29 on a leap year (February).
4. BC4: Day = 28 on a non-leap year (February).
5. BC5: Year = 1900 (minimum valid year).

- BC6: Year = 2015 (maximum valid year).
- BC7: Invalid day (0).
- BC8: Invalid day (32).
- BC9: Invalid year (1899).
- BC10: Invalid year (2016).

Input Data (day, month, year)	Expected Outcome	Remark
1, 1, 2015	31/12/2014	BC1: First day of the year, valid previous date.
31, 1, 2015	30/01/2015	BC2: Last day of January, valid previous date.
29, 2, 2016	28/02/2016	BC3: Valid leap year, previous date is the last day of February.
28, 2, 2015	27/02/2015	BC4: Previous date in a non-leap year.
1, 3, 1900	28/02/1900	BC5: Minimum valid year, previous date is the last day of February.
31, 12, 2015	30/12/2015	BC6: Last day of the year, valid previous date.

0, 3, 2015	Invalid date	BC7: Invalid day less than 1.
32, 1, 2015	Invalid date	BC8: Invalid day greater than maximum for the month.
15, 3, 1899	Invalid date	BC9: Invalid year less than minimum valid year.
15, 3, 2016	Invalid date	BC10: Invalid year greater than maximum valid year.

Q2)

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.

Valid Equivalence Classes:

1. EC1: Value v exists in the array a (at any index).
2. EC2: Value v is the first element in the array.
3. EC3: Value v is the last element in the array.
4. EC4: Value v is not in the array (but the array is non-empty).
5. EC5: Value v is searched in an empty array.
6. EC6: Value v is of incorrect data type
7. EC7: Array has element of incorrect data type.

Input Data (v, a)	Expected Outcome	Remark

v = 3, a = [1, 2, 3, 4, 5]	2	EC1: v exists in the array; first occurrence is at index 2.
v = 1, a = [1, 2, 3, 4, 5]	0	EC2: v is the first element; returns index 0.
v = 5, a = [1, 2, 3, 4, 5]	4	EC3: v is the last element; returns index 4.
v = 6, a = [1, 2, 3, 4, 5]	-1	EC4: v is not in the array; returns -1.
v = 3, a = []	-1	EC5: Searching in an empty array; returns -1.
v= ‘3’, a=[1,2,3]	Error	EC6: v is not of integer data type
v=3, a=[1,’2’,3]	Error	EC7: array contains element of non integer type.

Boundary Conditions:

1. BC1: Array with one element where v is equal to that element.
2. BC2: Array with one element where v is not equal to that element.
3. BC3: Array with many elements; v matches the first element.
4. BC4: Array with many elements; v matches the last element.
5. BC5: Array with many elements; v does not match any element.

Input Data (v, a)	**Expected Outcome**	**Remark**
v = 1, a = [1]	0	BC1: Array has one element; v matches it, returns index 0.

v = 2, a = [1]	-1	BC2: Array has one element; v does not match, returns -1.
v = 1, a = [1, 2,3]	0	BC3: v matches the first element, returns index 0.
v = 2, a = [1, 2]	1	BC4: v matches the second element, returns index 1.
v = 3, a = [1, 2]	-1	BC5: v does not match any elements, returns -1.

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

Valid Equivalence Classes:

1. EC1: Value v exists in the array a multiple times.
2. EC2: Value v exists in the array a exactly once.
3. EC3: Value v does not exist in the array a (but the array is non-empty).
4. EC4: Array is empty.
5. EC5: Value v is of incorrect data type
6. EC6: Array contains element of incorrect data type.

Input Data (v, a)	Expected Outcome	Remark
3, [1, 2, 3, 3, 4, 5]	2	EC1: v exists multiple times (twice).
1, [1, 2, 3, 4, 5]	1	EC2: v exists once.

6, [1, 2, 3, 4, 5]	0	EC3: v does not exist in the array.
3, []	0	EC4: Searching in an empty array; returns 0.
‘3’, [1,2,3,4,5]	Error	EC5: v is not of integer type
3, [1,’2’,3,4,5]	Error	EC6: array contains an element which is not of integer type.

Boundary Conditions:

1. BC1: Array with one element where v is equal to that element.
2. BC2: Array with one element where v is not equal to that element.
3. BC3: Array with two or more elements; v matches the first element.
4. BC4: Array with two or more elements; v matches the last element.
5. BC5: Array with two or more elements; v does not match either element.

Input Data (v, a)	Expected Outcome	Remark
1, [1]	1	BC1: Array has one element; v matches it, returns 1.
2, [1]	0	BC2: Array has one element; v does not match, returns 0.
1, [1, 2]	1	BC3: v matches the first element, returns 1.
2, [1, 2]	1	BC4: v matches the last element, returns 1.
3, [1, 2]	0	BC5: v does not match any elements, returns 0.

P3) The function countItem returns the number of times a value v appears in an array of integers a.

Equivalence Classes:

1. EC1: v is present in the array.
2. EC2: v is not present in the array.
3. EC3: Array a[] is sorted and contains multiple elements.
4. EC4: Array a[] is empty.
5. EC5: v is not an integer.
6. EC6: Array a[] is not sorted.

Input Data (v, a[])	**Expected Outcome**	**Remark**
v = 3, a = [1, 2, 3, 4, 5]	2	Valid case: v = 3 found at index 2 in the sorted array.
v = 6, a = [1, 2, 3, 4, 5]	-1	Valid case: v = 6 is not in the array.
v = 1, a = [1, 2, 3, 4, 5]	0	Valid case: v = 1 is the first element (boundary).
v = 5, a = [1, 2, 3, 4, 5]	4	Valid case: v = 5 is the last element (boundary).
v = 2, a = []	-1	Edge case: Empty array, search returns -1.
v = '3', a = [1, 2, 3, 4, 5]	Error message	Invalid input: v is not an integer.
v = 3, a = [5, 3, 2, 1]	Error message	Invalid input: Array is not sorted.

Boundary Classes:

1. BC1: Array size = 0 (empty array).
2. BC2: Single-element array.
3. BC3: First element of the array (v at the first position).
4. BC4: Last element of the array (v at the last position).
5. BC5: v not found in the array, covering the upper bound.

Input Data (v, a[])	**Expected Outcome**	**Remark**
v = 1, a = []	-1	Lower boundary: Empty array, search returns -1.
v = 1, a = [1]	0	Single-element array: v = 1 is found at index 0.
v = 2, a = [1, 2, 3, 4, 5]	1	Boundary case: v = 2 is found at index 1.
v = 5, a = [1, 2, 3, 4, 5]	4	Boundary case: v = 5 is the last element.
v = 6, a = [1, 2, 3, 4, 5]	-1	Upper boundary: v = 6 is not found in the array.
v = -1, a = [-5, -3, -1, 0]	2	Negative numbers: v = -1 is found at index 2.

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).

Equivalence Classes:

1. EC1: Equilateral triangle (all sides are equal).
2. EC2: Isosceles triangle (exactly two sides are equal).
3. EC3: Scalene triangle (no sides are equal).
4. EC4: Invalid triangle (the sum of any two sides must be greater than the third).
5. EC5: Negative side lengths (invalid by definition).

Input Data (a, b, c)	**Expected Outcome**	**Remark**
a = 3, b = 3, c = 3	0	EC1: All sides are equal, so it is an equilateral triangle.
a = 4, b = 4, c = 2	1	EC2: Two sides are equal, so it is an isosceles triangle.
a = 5, b = 4, c = 3	2	EC3: All sides are different, so it is a scalene triangle.
a = 1, b = 1, c = 3	3	EC4: Invalid triangle since 1 + 1 is not greater than 3.
a = -1, b = 2, c = 3	3	EC5: Invalid triangle due to negative length.

Boundary Conditions:

1. BC1: Sides are zero (invalid).
2. BC2: Minimum positive integers for sides (1, 1, 1).
3. BC3: Sides just at the edge of being invalid (e.g., a = b + c).
4. BC4: One side being equal to the sum of the other two (invalid).

Input Data (a, b, c)	**Expected Outcome**	**Remark**
a = 0, b = 0, c = 0	3	BC1: Invalid triangle since sides cannot be zero.
a = 1, b = 1, c = 1	0	BC2: Equilateral triangle with minimum positive lengths.
a = 2, b = 2, c = 4	3	BC3: Invalid triangle since 2 + 2 is not greater than 4.
a = 2, b = 2, c = 3	1	BC4: Isosceles triangle; two sides are equal.
a = 3, b = 4, c = 7	3	BC4: Invalid triangle since 3 + 4 is not greater than 7.

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).

Equivalence Classes:

1. EC1: s1 is a prefix of s2.
2. EC2: s1 is equal to s2.
3. EC3: s1 is an empty string, and s2 is non-empty.
4. EC4: s1 is longer than s2.

5. EC5: s1 is not a prefix of s2 (i.e., the characters differ after the prefix length).

Input Data (s1, s2)	**Expected Outcome**	**Remark**
s1 = "abc", s2 = "abcdef"	true	EC1: s1 is a valid prefix of s2.
s1 = "test", s2 = "test"	true	EC2: s1 and s2 are equal, so s1 is trivially a prefix.
s1 = "", s2 = "hello"	true	EC3: An empty string is a prefix of any non-empty string.
s1 = "abc", s2 = "ab"	false	EC4: s1 is longer than s2, so it cannot be a prefix.
s1 = "hello", s2 = "world"	false	EC5: s1 is not a prefix of s2; the first characters differ.

Boundary Conditions:

1. BC1: Both s1 and s2 are empty strings.
2. BC2: s1 is an empty string, and s2 is an empty string.
3. BC3: s1 is one character long, and s2 is one character long.
4. BC4: s1 has one character, and s2 has multiple characters, with the first character of s2 matching s1.
5. BC5: s1 has one character, and s2 has multiple characters, with the first character of s2 not matching s1.

Input Data (s1, s2)	**Expected Outcome**	**Remark**

s1 = "", s2 = ""	true	BC1: An empty string is a prefix of another empty string.
s1 = "", s2 = ""	true	BC2: An empty string is a prefix of another empty string.
s1 = "a", s2 = "a"	true	BC3: Both strings are equal with one character.
s1 = "a", s2 = "abc"	true	BC4: s1 is a prefix of s2 as the first character matches.
s1 = "a", s2 = "b"	false	BC5: The first characters of s1 and s2 do not 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

Equivalence Classes:

1. EC1: Scalene Triangle (All sides are different, and triangle inequality holds).
2. EC2: Isosceles Triangle (Two sides are equal, and triangle inequality holds).
3. EC3: Equilateral Triangle (All three sides are equal).
4. EC4: Right-Angled Triangle (One side satisfies Pythagorean theorem).

5. EC5: Invalid Triangle (Triangle inequality does not hold).
6. EC6: Non-positive Input (Any side is less than or equal to zero).

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)

Input (A, B, C)	Expected Outcome	Equivalence Class	Remark
3.0, 4.0, 5.0	Right-Angled	EC4	Valid right-angled triangle.
5.0, 5.0, 8.0	Isosceles	EC2	Valid isosceles triangle.
3.0, 4.0, 6.0	Scalene	EC1	Valid scalene triangle.
2.0, 2.0, 2.0	Equilateral	EC3	Valid equilateral triangle.
1.0, 2.0, 3.0	Invalid	EC5	Does not satisfy triangle inequality.
0.0, 4.0, 5.0	Invalid	EC6	Non-positive input (0 side).
-1.0, 2.0, 2.0	Invalid	EC6	Non-positive input (-1 side).

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

Input (A, B, C)	Expected Outcome	Remark
3.0, 4.0, 5.0	Scalene	A + B > C; all sides different.
2.0, 3.0, 4.0	Scalene	A + B > C; valid scalene triangle.
2.0, 2.0, 3.9	Scalene	A + B > C; still valid but on boundary.

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

Input (A, B, C)	Expected Outcome	Remark
2.0, 3.0, 2.0	Isosceles	A = C; valid isosceles triangle.
3.0, 4.0, 3.0	Isosceles	A = C; valid isosceles triangle.
2.0, 1.0, 2.0	Isosceles	A = C; valid but on boundary.

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

Input (A, B, C)	Expected Outcome	Remark
2.0, 2.0, 2.0	Equilateral	All sides equal; valid equilateral triangle.
3.0, 3.0, 3.0	Equilateral	All sides equal; valid equilateral triangle.
1.0, 1.0, 1.0	Equilateral	All sides equal; valid equilateral triangle.

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

Input (A, B, C)	Expected Outcome	Remark
3.0, 4.0, 5.0	Right-Angled	Valid right-angled triangle (3-4-5).
5.0, 12.0, 13.0	Right-Angled	Valid right-angled triangle (5-12-13).
6.0, 8.0, 10.0	Right-Angled	Valid right-angled triangle (6-8-10).

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

Input (A, B, C)	Expected Outcome	Remark
1.0, 2.0, 3.0	Invalid	Does not satisfy triangle inequality.
4.0, 1.0, 2.0	Invalid	Does not satisfy triangle inequality.
5.0, 5.0, 11.0	Invalid	Does not satisfy triangle inequality.

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

Input (A, B, C)	Expected Outcome	Remark
0.0, 4.0, 5.0	Invalid	Non-positive input (0 side).
-1.0, 2.0, 2.0	Invalid	Non-positive input (-1 side).
2.0, -3.0, 2.0	Invalid	Non-positive input (-3 side).