Lab Session - Functional Testing (Black-Box)

Name: Praneel Vania
ID: 202201131

Q1.

Valid ranges for inputs(Day, Month, Year):

○ **1 ≤ Day ≤ 31**
○ **1 ≤ Month ≤ 12**
○ **1900 ≤ Year ≤ 2015**

Equivalence Partitioning (EP):

1. Valid Equivalence Classes:
   - Class 1: Valid day, valid month, valid year.
     - Example: (20, 8, 2011) → Expected: 19, 8, 2011
   - Class 2: First day of a month.
     - Example: (1, 5, 2005) → Expected: 30, 4, 2005
   - Class 3: End of the month for a 31-day month.
     - Example: (31, 12, 2010) → Expected: 30, 12, 2010
   - Class 4: End of the month for a 30-day month.
     - Example: (30, 6, 2012) → Expected: 29, 6, 2012
   - Class 5: Leap year: last day of February (February 29).
     - Example: (1, 3, 2016) → Expected: 29, 2, 2016
2. Invalid Equivalence Classes:
   - Class 1: Invalid day, valid month, valid year.
     - Example: (32, 7, 2010) → Expected: Error
   - Class 2: Valid day, invalid month, valid year.
     - Example: (15, 13, 2015) → Expected: Error
   - Class 3: Valid day, valid month, invalid year.
     - Example: (15, 5, 1890) → Expected: Error
     - ■

Boundary Value Analysis (BVA):

1. Valid Boundary Values:
   - Boundary 1: Day = 1, Month = 1, Year = 1900.
     - Example: (1, 1, 1900) → Expected: Error (no previous date)
   - Boundary 2: Day = 31, Month = 12, Year = 2015.
     - Example: (31, 12, 2015) → Expected: 30, 12, 2015
   - Boundary 3: Day = 1, Month = 2, Year = 2012 (Leap year).
     - Example: (1, 2, 2012) → Expected: 31, 1, 2012
2. Invalid Boundary Values:
   - Boundary 1: Day = 0, Month = 5, Year = 2010.
     - Example: (0, 5, 2010) → Expected: Error
   - Boundary 2: Day = 31, Month = 2, Year = 2011 (Non-leap year).
     - Example: (31, 2, 2011) → Expected: Error
   - Boundary 3: Day = 29, Month = 2, Year = 2014 (Non-leap year).
     - Example: (29, 2, 2014) → Expected: Error

Test Suite Table

1. Equivalence Partitioning

Category	Input	Expected Output
Valid input	06,05,2012	05,05,2012

Valid input (1st of month)	01,12,2012	30,11,2012
Valid input (end of 31-day month)	31,12,2012	30,12,2012
Invalid Day	32,12,2012	error
Invalid Month	31,13,2012	error
Invalid Year	31,12,2222	error

2. Boundary Value Analysis

Category	**Input**	**Expected Output**
Day before min boundary	00,05,2012	error
First valid boundary day (Leap)	01,03,2012	29,02,2012
End boundary (valid date)	31,12,2012	30,12,2012
Invalid leap year boundary	29,02,2011	error

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.

```c
int linearSearch(int v, int a[])

{

    int i = 0;

    while (i < a.length) // This line is incorrect as arrays in C/C++ do
not have a `.length` property.

    {

        if (a[i] == v)

            return(i); // Return index if value is found

        i++; // Move to the next element

    }

    return (-1); // Return -1 if value is not found

}
```

Test cases:

1. Equivalence Partitioning Test Cases:

Test Case	Array a[ ]	Value v	Size	Expected Output	Reasoning
Valid Partition 1:					
Case 1: Value present in array	{1 , 2 , 3 }	3	3	2	3 is at index 2.
Case 2: Value at first position	{1 , 2 , 3}	1	3	0	1 is at index 0.
Case 3: Value at last position	{1 , 2 , 3}	3	3	2	3 is at index 2

Boundary Value Analysis (BVA) Test Cases

Test Case	Array a[ ]	Value v	Size	Expected Output	Reasoning
Boundary 1: Array size					
Case 1: Empty array	{}	5	0	-1	Array is empty, so value cannot be found.
Case 2: Single element, present	{1}	1	1	0	Single element, value present at index 0.
Case 3: Single element, not present	{2}	1	1	-1	Single element, but value not present.
Boundary 2: First and last positions					
Case 4: Value at first position	{1 , 2 , 3}	1	3	0	Value 1 is at index 0.
Case 5: Value at last position	{1 , 2 , 3}	3	3	2	Value 3 is at index 2.

Boundary 3: Middle position

| Case 6: Value in the middle | {5, 15, 25} | 15 | 3 | 1 | Value 15 is at index 1 (middle). |

Boundary 4: Value extremes

| Case 7: Smallest possible value | {0, 1, 2, 3} | 0 | 4 | 0 | Value 0 at index 0. |
| Case 8: Largest possible value | {1, 2, 3, 2147483647} | 2147483647 | 4 | 3 | Largest possible integer at index 3. |

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

Equivalence Partitioning (EP) Test Cases

Test Case	Array a[]	Value v	Size	Expected Output	Reasoning

Valid Partition 1:

| Case 1: v appears multiple times | {1, 2, 3, 3, 3, 4, 5} | 3 | 7 | 3 | 3 appears 3 times in the array. |

Valid Partition 2:

| Case 2: v appears once | {10, 20, 30, 40} | 20 | 4 | 1 | 20 appears once at index 1. |

Valid Partition 3:

| Case 3: v does not appear | {5, 6, 7, 8} | 10 | 4 | 0 | 10 is not in the array. |

Valid Partition 4:

| Case 4: Empty array | {} | 5 | 0 | 0 | The array is empty, so count is 0. |

Invalid Partition 1:

| Case 5: Invalid array size | {1, 2, 3} | 2 | -1 | Error | Invalid size (-1) should cause an error. |

Invalid Partition 2:

| Case 6: Non-integer array elements | {'a', 'b', 'c'} | 'b' | 3 | Error | Array contains non-integer values. |

Boundary Value Analysis (BVA) Test Cases

Test Case	Array a[]	Value v	Size	Expected Output	Reasoning

Boundary 1: Array size

Case 1: Empty array	{}	5	0	0	Array is empty, so value cannot be found.
Case 2: Single element, value present	{10}	10	1	1	Single element 10 matches v.
Case 3: Single element, value not present	{20}	15	1	0	Single element 20 does not match v.

Boundary 2: Number of occurrences

| Case 4: v appears at start | {10, 20, 30} | 10 | 3 | 1 | Value 10 appears once at the start. |
| Case 5: v appears at the end | {10, 20, 30} | 30 | 3 | 1 | Value 30 appears once at the end. |

Test Case	Array a[ ]	Value v	Size	Expected Output	Reasoning
Case 6: v appears multiple times	{5, 15, 15, 25, 15}	15	5	3	Value 15 appears 3 times.

Boundary 3: Value extremes

Test Case	Array a[ ]	Value v	Size	Expected Output	Reasoning
Case 7: Smallest possible value	{0, 1, 2, 3}	0	4	1	Smallest integer value 0 appears once.
Case 8: Largest possible value	{1, 2, 3, 2147483647}	2147483647	4	1	Largest possible integer appears once.

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.
Equivalence Partitioning (EP) Test Cases

Test Case	Array a[ ]	Value v	Size	Expected Output	Reasoning
Valid Partition 1:					
Case 1: Value present in array	{1, 2, 3, 4, 5}	3	5	2	3 is at index 2.
Case 2: Value present at first pos	{10, 20, 30, 40, 50}	10	5	0	10 is at index 0.
Case 3: Value present at last pos	{10, 20, 30, 40, 50}	50	5	4	50 is at index 4.
Valid Partition 2:					

Test Case	Array	Value	Size	Expected Output	Reasoning
Case 4: Value not present	{5, 10, 15, 20, 25}	12	5	-1	12 is not in the array.

Valid Partition 3:

Case 5: Single-element array, present	{100}	100	1	0	100 is the only element and is found.

Valid Partition 4:

Case 6: Empty array	{}	20	0	-1	Array is empty, so value is not found.

Invalid Partition 1:

Case 7: Invalid array size	{1, 2, 3}	2	-1	Error	Invalid size (-1) should cause an error.

Boundary Value Analysis (BVA) Test Cases

Test Case	Array a[]	Value v	Size	Expected Output	Reasoning
Boundary 1: Array size					
Case 1: Empty array	{}	5	0	-1	Array is empty, so value cannot be found.
Case 2: Single-element, present	{10}	10	1	0	Single element 10 is present at index 0.
Case 3: Single-element, not present	{20}	15	1	-1	Single element 20, but 15 is not present.
Boundary 2: Value position					

Test Case	Array	v			Reasoning
Case 4: v at first position	{5, 10, 15, 20, 25}	5	5	0	Value 5 is at index 0.
Case 5: v in the middle	{5, 10, 15, 20, 25}	15	5	2	Value 15 is at index 2.
Case 6: v at last position	{5, 10, 15, 20, 25}	25	5	4	Value 25 is at index 4.

Boundary 3: Value extremes

Test Case	Array	v			Reasoning
Case 7: Smallest possible value	{0, 1, 2, 3, 4, 5}	0	6	0	Value 0 is at index 0.
Case 8: Largest possible value	{1, 2, 3, 4, 2147483647}	2147483647	5	4	Largest possible integer value at index 4.

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 Partitioning (EP) Test Cases

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Valid Partition 1:					
Case 1: Equilateral triangle	5	5	5	EQUILATERAL	All sides are equal.
Valid Partition 2:					

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Case 2: Isosceles triangle	5	5	8	ISOSCELES	Two sides are equal.
Case 3: Isosceles triangle	6	7	6	ISOSCELES	Two sides are equal.

Valid Partition 3:

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Case 4: Scalene triangle	5	6	7	SCALENE	No sides are equal.

Invalid Partition 1:

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Case 5: Invalid triangle	1	2	10	INVALID	Does not satisfy the triangle inequality.
Case 6: Invalid triangle	10	5	3	INVALID	One side is too long to form a valid triangle.

Invalid Partition 2:

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Case 7: Invalid (zero length)	0	5	5	INVALID	A side with length 0 does not form a valid triangle.
Case 8: Negative side length	-3	4	5	INVALID	Negative side length is not valid.

Boundary Value Analysis (BVA) Test Cases

Test Case	Side A	Side B	Side C	Expected Output	Reasoning

Boundary 1: Minimum valid lengths

Test Case	Side A	Side B	Side C	Expected Output	Reasoning
Case 1: Smallest valid triangle	1	1	1	EQUILATERAL	Smallest possible valid triangle (equilateral).
Case 2: Two sides at minimum	1	1	2	INVALID	Sum of two sides equals the third, so it's invalid.
Case 3: Scalene with minimum values	2	3	4	SCALENE	Small scalene triangle with distinct side lengths.

Case 4: Large equilateral	1000	1000	1000	EQUILATERAL	Large triangle with all sides equal.

Case 5: Large isosceles	1000	1000	1500	ISOSCELES	Two sides equal, one large side.

Boundary 3: Invalid cases

Case 6: Invalid due to one large side	1	2	1000	INVALID	One side is too long to form a valid triangle.

Case 7: Negative value	-1	5	5	INVALID	Negative side length makes it invalid.

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 Partitioning (EP) Test Cases

Test Case	String s1	String s2	Expected Output	Reasoning
Valid Partition 1:				
Case 1: Exact match	"apple"	"apple"	true	s1 is the exact same as s2, so it is a prefix.

| Case 2: s1 is a prefix | "app" | "appl e" | tru e | s1 is a valid prefix of s2. |

Valid Partition 3:

| Case 3: s1 is not a prefix | "dog" | "appl e" | fals e | s1 does not match the start of s2. |

Valid Partition 4:

| Case 4: s1 is longer than s2 | "applesau ce" | "appl e" | fals e | A longer s1 cannot be a prefix of a shorter s2. |

Invalid Partition:

| Case 5: Empty s1, non-empty s2 | "" | "appl e" | tru e | An empty string is a prefix of any string. |

| Case 6: Empty s1 and s2 | "" | "" | tru e | An empty string is a prefix of another empty string. |

For the `prefix` function, which checks whether s1 is a prefix of s2, we can design **Equivalence Partitioning (EP)** and **Boundary Value Analysis (BVA)** test cases. Here's a table showing the test cases for both testing techniques.

Equivalence Partitioning (EP) Test Cases

In **Equivalence Partitioning**, we divide the input space into valid and invalid partitions based on the function's requirements and expected outputs.

Test Case	String s1	String s2	Expected Output	Reasoning

Valid Partition 1:

Case 1: Exact match `"apple"` `"apple"` `true` s1 is the exact same as s2, so it is a prefix.

Valid Partition 2:

Case 2: s1 is a prefix `"app"` `"apple"` `true` s1 is a valid prefix of s2.

Valid Partition 3:

Case 3: s1 is not a prefix `"dog"` `"apple"` `false` s1 does not match the start of s2.

Valid Partition 4:

Case 4: s1 is longer than s2 `"applesauce"` `"apple"` `false` A longer s1 cannot be a prefix of a shorter s2.

Invalid Partition:

Case 5: Empty s1, non-empty s2 `""` `"apple"` `true` An empty string is a prefix of any string.

Case 6: Empty s1 and s2 `""` `""` `true` An empty string is a prefix of another empty string.

Boundary Value Analysis (BVA) Test Cases

Test Case	String s1	String s2	Expected Output	Reasoning
Boundary 1: Empty strings				
Case 1: s1 is empty	`""`	`"hello"`	`true`	An empty string is always a prefix.

Case 2: s2 is empty	"hello"	""	false	A non-empty s1 cannot be a prefix of an empty string.

Boundary 2: Single character

Case 3: Single character prefix	"h"	"hello"	true	A single character s1 matches the first character of s2.
Case 4: Single character not prefix	"x"	"hello"	false	A single character s1 does not match the first character.

Boundary 3: Length differences

Case 5: s1 length == s2 length	"hello"	"hello"	true	s1 and s2 are identical.
Case 6: s1 longer than s2	"hellothere"	"hello"	false	A longer s1 cannot be a prefix of a shorter s2.

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.

a) Identify the Equivalence Classes for the System

We identify equivalence classes based on the properties of triangles (scalene, isosceles, equilateral, right-angled) and invalid cases. Below are the relevant equivalence classes:

1. **Equilateral Triangle**: All three sides are equal ($A = B = C$).
2. **Isosceles Triangle**: Two sides are equal ($A = B \neq C$ or similar).
3. **Scalene Triangle**: All sides are different ($A \neq B \neq C$).
4. **Right-Angled Triangle**: The sides satisfy the Pythagorean theorem ($A^2 + B^2 = C^2$ or similar permutations).
5. **Invalid Triangle**: The sides do not satisfy the triangle inequality ($A + B \leq C$ or any similar case).
6. **Non-positive or Zero-Length Input**: Any side is non-positive ($A \leq 0, B \leq 0, C \leq 0$).

---

b) Identify Test Cases to Cover the Identified Equivalence Classes

Below are test cases to cover each equivalence class, and they are explicitly mapped to their respective classes.

Test Case	A	B	C	Expected Output	Equivalence Class
Case 1: Equilateral triangle	3.0	3.0	3.0	Equilateral	Equilateral Triangle
Case 2: Isosceles triangle (A = B)	5.0	5.0	7.0	Isosceles	Isosceles Triangle
Case 3: Scalene triangle	4.0	5.0	6.0	Scalene	Scalene Triangle
Case 4: Right-angled triangle	3.0	4.0	5.0	Right-angled	Right-angled Triangle

Test Case					
Case 5: Invalid triangle (A + B = C)	2.0	2.0	4.0	Invalid	Invalid Triangle (fails triangle inequality)
Case 6: Zero-length side	0.0	2.0	3.0	Invalid	Non-positive Input
Case 7: Negative side length	-1.0	2.0	2.0	Invalid	Non-positive Input

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

For a scalene triangle, the boundary condition is when the sum of two sides equals the third. We test values slightly below, equal to, and above the boundary:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: A + B = C (boundary)	3.0	4.0	7.0	Invalid	A + B = C (exact boundary, invalid)
Case 2: A + B > C	3.0	4.0	6.9	Scalene	A + B > C (just valid)
Case 3: A + B < C	3.0	4.0	7.1	Invalid	A + B < C (invalid)

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

For an isosceles triangle where two sides are equal, the boundary involves values slightly above, below, and equal to the side lengths:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: A = C	5.0	6.0	5.0	Isosceles	A = C (exact isosceles)
Case 2: A slightly greater than C	5.1	6.0	5.0	Scalene	A > C (scalene)

| Case 3: A slightly less than C | 4.9 | 6.0 | 5.0 | Scalene | A < C (scalene) |

---

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

For an equilateral triangle, all sides must be equal. Test cases include small differences around the boundary:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: A = B = C	5.0	5.0	5.0	Equilateral	A = B = C (exact equilateral)
Case 2: A slightly greater than B, C	5.1	5.0	5.0	Isosceles	A > B = C (isosceles)
Case 3: A slightly less than B, C	4.9	5.0	5.0	Isosceles	A < B = C (isosceles)

---

f) Test Cases for the Boundary Condition: A² + B² = C² (Right-Angle Triangle)

For right-angled triangles, we use the Pythagorean theorem. Test cases are set up around this condition:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: A² + B² = C² (exact)	3.0	4.0	5.0	Right-angled	A² + B² = C² (exact boundary, valid)
Case 2: A² + B² > C²	3.0	4.0	4.9	Scalene	A² + B² > C² (slightly smaller C, scalene)
Case 3: A² + B² < C²	3.0	4.0	5.1	Invalid	A² + B² < C² (slightly larger C, invalid)

---

g) Test Cases for the Non-Triangle Case (A + B ≤ C)

For non-triangles, we explore when the sum of two sides is less than or equal to the third side:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: A + B = C	3.0	4.0	7.0	Invalid	A + B = C (invalid)
Case 2: A + B < C	3.0	4.0	7.1	Invalid	A + B < C (invalid)

---

h) Test Cases for Non-Positive Input

For non-positive input, test cases cover values of zero and negative numbers:

Test Case	A	B	C	Expected Output	Boundary Tested
Case 1: Zero side	0.0	5.0	5.0	Invalid	A = 0 (invalid)
Case 2: Negative side	`-1.0				