LAB 8 : Black Box Testing

Vaishvik Patel
202201433

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?

Equivalence Class Partitioning (EP)

Here, we divide inputs into valid and invalid ranges to ensure that all possible conditions are tested.

Valid Input Classes (EP1):

1. **Valid days for months** (e.g., 30 days in April, 31 days in July, 28 or 29 days in February).
2. **Valid months**: (1 to 12).
3. **Valid years**: (1900 to 2015).

Invalid Input Classes (EP2):

1. **Invalid days**: e.g., day < 1 or day > 31.
2. **Invalid months**: month < 1 or month > 12.
3. **Invalid years**: year < 1900 or year > 2015.
4. **Invalid date combinations**: e.g., February 30, April 31.

Boundary Value Analysis (BVA)

Boundary conditions for the program are identified at the edge of valid ranges. Testing these conditions ensures robustness.

- **Day boundaries**: 1, 2, 30, 31 (or 28/29 for February).
- **Month boundaries**: 1, 2, 11, 12.
- **Year boundaries**: 1900, 1901, 2014, 2015.

Test ID	Input Data (Day, Month, Year)	Expected Outcome	Type
TC1	15, 8, 2010	Previous date: 14-08-2010	EP1
TC2	1, 1, 1900	Error: Previous date beyond range	BVA
TC3	29, 2, 2012	Previous date: 28-02-2012	EP1 (Leap Year Case)
TC4	30, 4, 2020	Error: Invalid year (beyond 2015)	EP4
TC5	31, 4, 2014	Error: Invalid day for April	EP2
TC6	1, 5, 2014	Previous date: 30-04-2014	BVA
TC7	1, 3, 2016	Previous date: 29-02-2016	EP1 (Leap Year Case)
TC8	0, 5, 2010	Error: Invalid day	EP2
TC9	31, 12, 2015	Previous date: 30-12-2015	BVA
TC10	1, 1, 2015	Previous date: 31-12-2014	BVA
TC11	32, 7, 2011	Error: Invalid day	EP2
TC12	15, 0, 2010	Error: Invalid month	EP3
TC13	15, 8, 1899	Error: Invalid year	EP4
TC14	15, 8, 2016	Error: Invalid year	EP4

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.

```
int linearSearch(int v, int a[])
{
        int i = 0;
        while (i < a.length)
        {
                if (a[i] == v)
                        return(i);
                i++;
        }
        return (-1);
}
```

Test Case ID	Input Array	Value vvv	Expected Output	Description
1	[1, 2, 3]	2	1	Match in the middle
2	[1, 2, 3]	4	-1	No match
3	[]	1	-1	Empty array
4	[5]	5	0	Match in single element
5	[5]	10	-1	No match in single element
6	[1, 2]	1	0	Match first element
7	[1, 2]	2	1	Match last element
8	[1, 2]	3	-1	No match in two-element array
9	[10, 20, 30, 40]	30	2	Match in larger array
10	[10, 20, 30, 40]	10	0	Match first element

11	[10, 20, 30, 40]	40	3	Match last element
12	[10, 20, 30, 40]	50	-1	No match in larger array
13	[1, 2, 3]	'2'	-	Your input datatype should int
14	[1.0, 2.0, 3.0]	2	-	Your input array datatype should int
15	[1, 2, 3]	2.0	-	Your input datatype should int
16	[null, 1, 2]	null	-	Both inputs are invalid
17	['a', 'b', 'c']	'b'	-	Your input array datatype should int
18	[1, 2, '2']	2	-	Your input array datatype should int

True Code :

```
long long int linearSearch(long long int v, long long int a[], int length) {
        for (int i = 0; i < length; i++) {
            if (a[i] == v) {
                return i;  // Return the index if the value is found
            }
        }
        return -1;  // Return -1 if the value is not found
}
```

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);
```

}

Test Case ID	Input Array	Value vvv	Expected Output	Description
1	[1, 2, 3]	2	1	Match occurs once
2	[1, 2, 3]	4	0	No match
3	[]	1	0	Empty array
4	[5]	5	1	Match in single element
5	[5]	10	0	No match in single element
6	[1, 2]	1	1	Match first element
7	[1, 2]	2	1	Match last element
8	[1, 2]	3	0	No match in two-element array
9	[10, 20, 30, 40]	30	1	Match occurs once in larger array
10	[10, 20, 30, 40]	10	1	Match first element
11	[10, 20, 30, 40]	40	1	Match last element
12	[10, 20, 30, 40]	50	0	No match in larger array
13	[1, 2, 3, 2, 2]	2	3	Match occurs three times
14	[1, 1, 1, 1]	1	4	All elements match
15	[1, 1, 2, 2, 3]	2	2	Match occurs twice
16	[1, 2, 3]	2.0	-	Input datatype should integer
17	[1.0, 2.0, 3.0]	2	-	Input datatype should integer
18	[1, 2, 3]	'2'	-	Input datatype should integer

19	**[null, 1, 2]**	**null**	**-**	**Input datatype should integer**
20	**['a', 'b', 'c']**	**'b'**	**-**	**Input datatype should integer**

True code :

```
long long int countItem(long long int v, long long int a[], int size) {
        long long int count = 0;
           for (int i = 0; i < size; i++) {
              if (a[i] == v) {
                 Count++;
              }
           }
     return count;
}
```

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.

```
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);
}
```

Equivalence Partitioning (EP) Table

Test Case	Input	Expected Output	Reason
TC1	v = 30, a = [10, 20, 30, 40, 50], size = 5	2	Return value is 2.
TC2	v = 35, a = [10, 20, 30, 40, 50], size = 5	-1	Return value is -1.
TC3	v = 10, a = [], size = 0	-1	Return value is -1.
TC4	v = 10, a = ["10", 20, 30], size = 3	**Error/Exception**	Non-integer element triggers error.
TC5	v = 2147483648, a = [10, 20, 30], size = 3	**Error/Exception**	Value out of 32-bit integer range.

Boundary Value Analysis (BVA) Table :

Test Case	Input	Expected Output	Reason
TC6	v = 10, a = [10, 20, 30], size = 3	0	Return value is 0.
TC7	v = 30, a = [10, 20, 30], size = 3	2	Return value is 2.
TC8	v = 9, a = [10, 20, 30], size = 3	-1	Return value is -1.
TC9	v = 31, a = [10, 20, 30], size = 3	-1	Return value is -1.
TC10	v = 10, a = [10], size = 1	0	Return value is 0.

TC11	v = 20, a = [10], size = 1	-1	Return value is -1.

True Code :

```
long long int binarySearch(long long int v, long long int a[], int size) {
	int lo = 0, mid, hi = size - 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;
}
```

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);
}
```

EP Test Cases

Test Case	Input	Expected Output	Reason
TC1	a = 3, b = 3, c = 3	0 (EQUILATERAL)	Return value is 0 (equilateral).
TC2	a = 3, b = 3, c = 2	1 (ISOSCELES)	Return value is 1 (isosceles).
TC3	a = 3, b = 4, c = 5	2 (SCALENE)	Return value is 2 (scalene).
TC4	a = 1, b = 2, c = 3	3 (INVALID)	Return value is 3 (triangle inequality fails).
TC5	a = 0, b = 4, c = 5	3 (INVALID)	Return value is 3 (non-positive side).
TC6	a = -1, b = 3, c = 4	3 (INVALID)	Return value is 3 (negative side).
TC7	a = 2147483648, b = 1, c = 1	3 (INVALID)	Return value is 3 (out of bound value).

BVA Test Cases

Test Case	Input	Expected Output	Reason
TC8	a = 1, b = 1, c = 1	0 (EQUILATERAL)	Return value is 0.
TC9	a = 2, b = 2, c = 1	1 (ISOSCELES)	Return value is 1.
TC10	a = 2, b = 2, c = 4	3 (INVALID)	Return value is 3 (triangle inequality).
TC11	a = 2147483648, b = 1, c = 1	3 (INVALID)	Return value is 3 (out of bound).
TC12	a = 1, b = 2147483648, c = 2	3 (INVALID)	Return value is 3 (out of bound).
TC13	a = 2, b = 2, c = 2147483648	3 (INVALID)	Return value is 3 (out of bound).

True Code :

```
#define EQUILATERAL 0
```

```
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3

int triangle(long long int a, long long int b, long long int c) {
	// Check for non-positive values and triangle inequality violations
	if (a <= 0 || b <= 0 || c <= 0 || a >= b + c || b >= a + c || c >= a + b)
		return INVALID;

	// Check for equilateral triangle
	if (a == b && b == c)
		return EQUILATERAL;

	// Check for isosceles triangle
	if (a == b || a == c || b == c)
		return ISOSCELES;

	// Otherwise, it's a scalene triangle
	return SCALENE;
}
```

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;
}
```

EP Test Cases

Test Case	Input	Expected Output	Reason
TC1	s1 = "pre", s2 = "prefix"	true	Return value is true (valid prefix).
TC2	s1 = "prefix", s2 = "prefix"	true	Return value is true (both strings equal).
TC3	s1 = "pre", s2 = "test"	false	Return value is false (not a prefix).
TC4	s1 = "long", s2 = "short"	false	Return value is false (not a prefix).
TC5	s1 = "hello", s2 = "hell"	false	Return value is false (s1 is longer).
TC6	s1 = "", s2 = "nonempty"	true	Return value is true (empty string is prefix).
TC7	s1 = "nonempty", s2 = ""	false	Return value is false (s1 is longer).

BVA Test Cases

Test Case	Input	Expected Output	Reason
TC11	s1 = "", s2 = ""	true	Return value is true (both empty).
TC12	s1 = "a", s2 = "a"	true	Return value is true (both strings equal).
TC13	s1 = "a", s2 = "ab"	true	Return value is true (s1 is prefix).
TC14	s1 = "ab", s2 = "a"	false	Return value is false (s1 is longer).
TC15	s1 = "abc", s2 = "ab"	false	Return value is false (s1 is longer).
TC16	s1 = "abc", s2 = "abcd"	true	Return value is true (s1 is prefix).

True Code :

```
public static boolean prefix(String s1, String s2) {
        // Check for null inputs
        if (s1 == null || s2 == null) {
                return false;
        }
```

```
        // Check if s1 is longer than s2
        if (s1.length() > s2.length()) {
                return false;
        }

        // Compare characters
        for (int i = 0; i < s1.length(); i++) {
                if (s1.charAt(i) != s2.charAt(i)) {
                        return false;  // Characters do not match
                }
        }
        return true;  // All characters match, s1 is a prefix of 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. Equivalence Classes Identification

1. **Valid Triangle Cases**
    - Equilateral Triangle: A=B=CA = B = CA=B=C
    - Isosceles Triangle**:** Two sides are equal (e.g., A=B≠CA = B \neq CA=B=C)
    - Scalene Triangle**:** All three sides are different (e.g., A≠B≠CA \neq B \neq CA=B=C)
    - Right-Angled Triangle**:** A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (or permutations of this relation)
2. **Invalid Triangle Cases (Non-triangle)**
    - Violates Triangle Inequality**:** A+B≤CA + B \leq CA+B≤C
    - Negative or Zero Inputs**:** Any side is non-positive (e.g., A≤0A \leq 0A≤0)

B. Test Cases for Equivalence Classes

Test Case	Input Values (A, B, C)	Expected Output	Equivalence Class Covered
TC1	3, 3, 3	Equilateral	Equilateral Triangle

TC2	5, 5, 8	Isosceles	Isosceles Triangle
TC3	4, 6, 7	Scalene	Scalene Triangle
TC4	3, 4, 5	Right-angled	Right-Angled Triangle
TC5	1, 2, 3	Not a Triangle	Violates Triangle Inequality
TC6	-3, 4, 5	Invalid Input	Non-positive Input
TC7	0, 5, 7	Invalid Input	Non-positive Input

C. Boundary Condition: A+B>C (Scalene Triangle)

Test Case	Input Values (A, B, C)	Expected Output	Boundary Condition
TC8	4.9, 5.0, 9.8	Scalene	Just satisfies A+B>CA + B > CA+B>C
TC9	5.0, 5.0, 10.0	Not a Triangle	Exactly A+B=CA + B = CA+B=C

D. Boundary Condition: A=C (Isosceles Triangle)

Test Case	Input Values (A, B, C)	Expected Output	Boundary Condition
TC10	7.0, 5.0, 7.0	Isosceles	Boundary for Isosceles
TC11	7.0, 7.0, 7.1	Scalene	Just beyond boundary

E. Boundary Condition: A=B=C (Equilateral Triangle)

Test Case	Input Values (A, B, C)	Expected Output	Boundary Condition
TC12	6.0, 6.0, 6.0	Equilateral	All sides equal
TC13	6.0, 6.0, 6.1	Isosceles	Just beyond boundary

F. Boundary Condition: A2+B2=C2 (Right-angled Triangle)

Test Case	Input Values (A, B, C)	Expected Output	Boundary Condition
TC14	3.0, 4.0, 5.0	Right-angled	Exact Pythagorean triplet
TC15	3.0, 4.0, 5.1	Scalene	Slightly beyond boundary

G. Boundary Condition: Non-triangle Case

Test Case	Input Values (A, B, C)	Expected Output	Boundary Condition
TC16	1.0, 2.0, 3.1	Not a Triangle	Just beyond triangle inequality
TC17	2.0, 2.0, 3.9	Not a Triangle	Just fails triangle inequality

---

H. Non-Positive Input Cases

Test Case	Input Values (A, B, C)	Expected Output	Non-positive Input Case
TC18	0, 4, 5	Invalid Input	Zero side
TC19	-1, 5, 7	Invalid Input	Negative side