
Dhirubhai Ambani
Institute of Information and Communication Technology

# Assignment 8

## Functional Testing - Black Box

## IT - 314 Software Engineering

***Name:Mavdiya Sujal Pravinbhai***

***ID: 202201040***

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 Partitioning**

| Test Case | Input/Equivalence Partitioning - (Month,Day,Year) | Expected Outcome |
|---|---|---|
| TC_001 - Valid Input | (3,13,2001) | 3/12/2001 |
| TC_002 - Month lower than 0 | (-1,1,2001) | Invalid Month |
| TC_003 - Month greater than 12 | (13,1,2001) | Invalid Month |
| TC_004 - Day less than 1 | (12,0,2000) | Invalid Day |
| TC_005 - Day greater than 31 | (5,41,1993) | Invalid Day |
| TC_006 - Year less than 1990 | (5,1,1193) | Invalid Year |

**Boundary Value Analysis**

| Test Case | Input/Boundary Value Analysis - (Month,Day,Year) | Expected Outcome |
|---|---|---|
| TC_007 - Leap year | (29,2,2014) | Invalid Day |
| TC_008 - Month with only 30 days | (31,4,2000) | Invalid Day |
| TC_009 - February | (30,2,2013) | Invalid Day |
| TC_0010 - Year = 2015 | (1,1,2015) | 12/31/2014 |
| TC_0011 - Year = 2016 | (2,3,2016) | Invalid Year |

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);
}
```

**Equivalence Partitioning**

| Test Case | Input/Equivalence Partitioning - (Search Value,Array) | Expected Outcome |
|---|---|---|
| TC_1 - Valid Input | (3,[1,2,3,4,5]) | 2 |
| TC_2 - Element at first position | (1,[1,2,3]) | 0 |
| TC_3 - Value at last position | (5,[3,8,5]) | 2 |

**Boundary Value Analysis**

| Test Case | Input/Boundary Value Analysis - (Search Value, Array) | Expected Outcome |
|---|---|---|
| TC_4 - Empty Array | (4,[]) | -1 |
| TC_5 - Value absent | (6,[4,7,8,2]) | -1 |
| TC_6 - Only 1 element | (1,[2]) | -1 |
| TC_7 - Only 1 element and | (2,[2]) | 0 |

| | | |
|---|---|---|
| value present | | |

***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);
}
```

**Equivalence Partitioning**

| Test Case | Input/Equivalence Partitioning - (Value,Array) | Expected Outcome |
|---|---|---|
| TC_01 - Valid Input | (3,[1,2,3,4,5]) | 1 |
| TC_02 - Element at multiple position | (2,[2,2,3,8,2]) | 3 |
| TC_03 - Value absent | (5,[3,8,2]) | 0 |

**Boundary Value Analysis**

| Test Case | Input/Boundary Value Analysis - (Value,Array) | Expected Outcome |
|---|---|---|
| TC_04 - Empty Array | (4,[]) | 0 |
| TC_05 - Value absent | (6,[4,7,8,2]) | |
| TC_06 - Only 1 element | (1,[2]) | 0 |
| TC_07 - Only 1 element and value present | (2,[2]) | 1 |

***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);
}
```

**Equivalence Partitioning**

| Test Case | Input/Equivalence Partitioning - (Search Value,Array) | Expected Outcome |
|---|---|---|
| TC01 - Valid Input | (3,[1,2,3,4,5]) | 2 |
| TC02 - Element at first position | (1,[1,2,3]) | 0 |
| TC03 - Value at last position | (5,[3,8,5]) | 2 |
| TC04 - Multiple Values | (6,[3,5,6,6,7,7,8]) | 3 |
| TC05 - Value absent | (6,[4,7,8,2]) | -1 |

**Boundary Value Analysis**

| Test Case | Input/Boundary Value Analysis - (Search Value, Array) | Expected Outcome |
|---|---|---|
| TC05 - Empty Array | (4,[]) | -1 |
| TC06 - Only 1 element | (1,[2]) | -1 |
| TC07 - Only 1 element and value present | (2,[2]) | 0 |

***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);
}
```

**Equivalence Partitioning**

| Test Case | Input/Equivalence Partitioning - (a,b,c) | Expected Outcome |
|---|---|---|
| TC_001 - Equilateral | (1,1,1) | Equilateral |
| TC_002 - Isosceles | (2,2,3) | Isosceles |
| TC_003 - Scalene | (4,5,6) | Scalene |
| TC_004 - Equilateral Invalid | (-1,-1,-1) | Invalid |
| TC_005 - Isoceles Invalid | (0,0,5) | Invalid |
| TC_006 - Scalene Invalid | (2,4,12) | Invalid |
| | Input/Boundary Value Analysis - (a,b,c) | |
| TC_007 - Isoceles Boundary | (1,1,2) | Invalid |
| TC_008 - Scalene Boundary | (1,2,3) | 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).***

```
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;
}
```

**Equivalence Partitioning**

| Test Case | Input/Equivalence Partitioning - (s1,s2) | Expected Outcome |
|---|---|---|
| TC1 - Valid Input | (abc,abcbbba) | True |
| TC2 - length of s1 > length of s2 | (xxxyzz,xyz) | False |
| TC3 - Valid Input but s1 ! pref(s2) | (pabc,pbbac) | False |

**Boundary Value Analysis**

| | Input/Boundary Value Analysis - (s1,s2) | |
|---|---|---|
| TC4 - Valid Input but first element not equal | (bbc,abcaa) | False |
| TC5 - Valid Input but last element not equal | (hello,hellWorld) | False |
| TC6 - length of s1 = length of s2 | (abc,abc) | True |
| TC7 - length of s1 = length of s2 + 1 | (abcd,abc) | False |

***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 Class | Description | Class Number |
|---|---|---|
| All three sides equal | All sides have the same length, forming an equilateral triangle. | C1 |
| Two sides equal | Exactly two sides have the same length, forming an isosceles triangle. | C2 |
| All sides different | Each side has a different length, forming a scalene triangle. | C3 |
| Right-angled triangle | The sides satisfy the Pythagorean theorem ($A^2 + B^2 = C^2$). | C4 |
| Invalid triangle lengths | The provided lengths do not satisfy the triangle inequality. | C5 |
| Non-positive side length | Any of the sides has a non-positive value, making it an invalid triangle. | C6 |

### *b. Identify Test Cases to Cover the Identified Equivalence Classes*

| Input Data (A, B, C) | Expected Outcome | Relevant Equivalence Classes |
|---|---|---|
| (3.0, 3.0, 3.0) | Equilateral | C1 |
| (3.0, 3.0, 5.0) | Isosceles | C2 |

| | | |
|---|---|---|
| (3.0, 4.0, 5.0) | Scalene | C3 |
| (3.0, 4.0, 5.0) | Right-angled | C4 |
| (1.0, 1.0, 3.0) | Invalid | C5 |
| (0.0, 1.0, 1.0) | Invalid | C6 |
| (-1.0, 2.0, 2.0) | Invalid | C6 |
| (0.0, 0.0, 0.0) | Invalid | C6 |

### *c. Boundary Condition A + B > C (Scalene Triangle)*

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (2.0, 3.0, 4.0) | Scalene | Valid triangle with all sides different lengths |
| (2.0, 2.0, 3.9) | Scalene | Valid triangle with two sides nearly equal |
| (2.0, 2.0, 4.0) | Invalid | Triangle is invalid due to violation of triangle inequality |

### *d. Boundary Condition A = C (Isosceles Triangle)*

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (3.0, 4.0, 3.0) | Isosceles | Valid isosceles triangle, with two equal sides. |
| (3.0, 4.0, 4.0) | Isosceles | Valid isosceles triangle, with two equal sides. |
| (3.0, 4.0, 2.9) | Invalid | Invalid triangle, fails triangle inequality. |

### *e. Boundary Condition A = B = C ( Equilateral Triangle)*

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (3.0, 3.0, 3.0) | Equilateral | equilateral triangle, all sides e |
| (2.0, 2.0, 2.0) | Equilateral | equilateral triangle, all sides e |
| (2.9, 2.9, 2.9) | Equilateral | equilateral triangle, all sides e |

***f. Boundary Condition*** $A^2 + B^2 = C^2$ ***or*** $A^2 + C^2 = B^2$ ***or*** $C^2 + B^2 = A^2$ ***( Right-Angle Triangle)***

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (3.0, 4.0, 5.0) | Right-angled | Valid right-angled triangle, satisfies pythagoras' theorem. |
| (5.0, 12.0, 13.0) | Right-angled | Valid right-angled triangle, satisfies pythagoras' theorem. |
| (3.0, 4.0, 4.9) | Invalid | Invalid triangle, does not meet right-angle condition. |

***g. Non-Triangle Case***

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (1.0, 2.0, 3.0) | Invalid | Invalid triangle, fails triangle inequality. |
| (5.0, 2.0, 2.0) | Invalid | Invalid triangle, fails triangle inequality. |

| | | |
|---|---|---|
| (10.0, 1.0, 1.0) | Invalid | Invalid triangle, fails triangle inequality. |

### *h. Non-Positve Input*

| Input Data (A, B, C) | Expected Outcome | Description |
|---|---|---|
| (0.0, 1.0, 1.0) | Invalid | Invalid input, side length cannot be zero. |
| (-1.0, 2.0, 2.0) | Invalid | Invalid input, side length cannot be negative. |
| (1.0, 0.0, 1.0) | Invalid | Invalid input, side length cannot be zero. |
| (1.0, 1.0, -1.0) | Invalid | Invalid input, side length cannot be negative. |