
Dhirubhai Ambani Institute of Information and Communication Technology

IT314-Software Engineering

Instructor - Prof. Saurabh Tiwari

# Lab08: Functional Testing (Black-Box)

Name: Tathya Prajapati

StudentId: 202201170

**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?
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.

## Derived Equivalence Class:

1. Day is between 1 and 31 **(valid)**
2. Day is less than 1 **(invalid)**
3. Day is greater than 31 **(invalid)**
4. Month is between 1 and 12 **(valid)**
5. Month is less than 1 **(invalid)**
6. Month is greater than 12 **(invalid)**
7. Year is between 1900 and 2015 **(valid)**
8. Year is less than 1900 **(invalid)**
9. Year is greater than 2015 **(invalid)**
10. Day is 29 in February in a leap year **(valid)**
11. Day is 29 in February in a non-leap year **(invalid)**
12. Day is 31 in months with 31 days **(valid)**
13. Day is 31 in months with less than 31 days (April, June, September, November) **(invalid)**

## TestCases:

| Test Case Number | Test Data(Day,Month,Year) | Expected Outcome | Classes Covered |
|---|---|---|---|
| TC1 | 12, 5, 2010 | T | 1, 4, 7 |
| TC2 | 0, 5, 2010 | F | 2, 4, 7 |
| TC3 | 32, 5, 2010 | F | 3, 4, 7 |
| TC4 | 12, 12, 2010 | T | 1, 4, 7 |
| TC5 | 12, 0, 2010 | F | 1, 5, 7 |
| TC6 | 12, 13, 2010 | F | 1, 6, 7 |
| TC7 | 12, 5, 1899 | F | 1, 4, 8 |
| TC8 | 12, 5, 2016 | F | 1, 4, 9 |
| TC9 | 29, 2, 2000 | T | 10, 4, 7 |
| TC10 | 29, 2, 2010 | F | 11, 4, 7 |
| TC11 | 31, 1, 2010 | T | 12, 4, 7 |
| TC12 | 31, 4, 2010 | F | 13, 4, 7 |
| TC13 | 1, 1, 1900 | T | 2, 5, 8 |
| TC14 | 31, 12, 2015 | T | 1, 4, 9 |
| TC15 | 1, 1, 2016 | F | 1, 4, 9 |
| TC16 | 31, 12, 1899 | F | 1, 4, 8 |
| TC17 | 28, 2, 2000 | T | 1, 10, 7 |
| TC18 | 29, 2, 2000 | T | 1, 10, 7 |
| TC19 | 30, 4, 2010 | T | 1, 13, 7 |
| TC20 | 1, 3, 2010 | T | 1, 4, 11 |

### Equivalence Partitioning (EP):

In **Equivalence Partitioning**, the input domain is divided into classes of equivalent data. Valid classes and invalid classes are identified.

**Valid Partitions:**

- **Month**: 1 ≤ month ≤ 12
- **Day**: 1 ≤ day ≤ valid number of days in a month (28 for February in non-leap years, 29 for leap years, 30 for months like April, June, September, November, and 31 for others).
- **Year**: 1900 ≤ year ≤ 2015

**Invalid Partitions:**

- **Month**: month < 1 or month > 12
- **Day**: day < 1 or day > 31 or more than the valid number of days in that specific month.
- **Year**: year < 1900 or year > 2015

## Q.2. Programs:

**P1.** The function linearSearch searches for a value v in an array of integers a. If v appears in the array, 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);
}
```

**Input - 1:**
v = 5
a[] = {1, 2, 3, 4, 5}
**Output - 1:**
True (4)
Explanation: 5 is found at index 4, so the function returns 4.

**Input - 2:**
v = 7
a[] = {10, 20, 30, 40, 50}
**Output - 2:**
False (-1)
Explanation: 7 is not found in the array, so the function returns -1.

**Input - 3:**
v = 3
a[] = {3, 3, 3, 3, 3}
**Output - 3:**
True (0)
Explanation: The first occurrence of 3 is at index 0, so the function returns 0.

**Input - 4:**
v = 2
a[] = {1, 2, 3, 4, 2}
**Output - 4:**
True (1)
Explanation: The first occurrence of 2 is at index 1, so the function returns 1.

**Input - 5:**
v = 0
a[] = {}
**Output - 5:**
False (-1)
Explanation: The array is empty, so the function directly returns -1.

**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);
}
```

**Input - 1:**
v = 3
a[] = {1, 2, 3, 3, 5}
**Output - 1:**
True (2)
Explanation: The value 3 appears twice in the array, so the function returns 2.

**Input - 2:**
v = 4
a[] = {10, 20, 30, 40, 50}
**Output - 2:**
False (0)
Explanation: The value 4 is not present in the array, so the function returns 0.

**Input - 3:**
v = 7
a[] = {7, 7, 7, 7, 7}
**Output - 3:**
True (5)
Explanation: The value 7 appears 5 times in the array, so the function returns 5.

**Input - 4:**
v = 2
a[] = {1, 2, 3, 4, 2}
**Output - 4:**
True (2)
Explanation: The value 2 appears twice in the array, so the function returns 2.

**Input - 5:**
v = 8
a[] = {}
**Output - 5:**
False (0)
Explanation: The array is empty, so the function directly returns 0, as there are no occurrences of the value 8.

**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);
        }
```

**Input - 1:**
v = 6
a[] = {1, 3, 5, 6, 8, 9}
**Output - 1:**
True (3)
Explanation: The value 6 is found at index 3, so the function returns 3.

**Input - 2:**
v = 10
a[] = {2, 4, 6, 8, 12, 14}
**Output - 2:**
False (-1)
Explanation: The value 10 is not present in the array, so the function returns -1.

**Input - 3:**
v = 7
a[] = {1, 3, 5, 7, 9, 11, 13}
**Output - 3:**
True (3)
Explanation: The value 7 is found at index 3, so the function returns 3.

**Input - 4:**
v = 1
a[] = {1, 2, 3, 4, 5}
**Output - 4:**
True (0)
Explanation: The value 1 is found at index 0, so the function returns 0.

**Input - 5:**
v = 15
a[] = {5, 10, 20, 30, 40}
**Output - 5:**
False (-1)
Explanation: The value 15 is not in the array, so the function returns -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);
}
```

**Input - 1:**
a = 3, b = 3, c = 3
**Output - 1:**
True (EQUILATERAL)
Explanation: All sides are equal, so the triangle is equilateral.

**Input - 2:**
a = 5, b = 5, c = 8
**Output - 2:**
True (ISOSCELES)
Explanation: Two sides are equal (5), making the triangle isosceles.

**Input - 3:**
a = 4, b = 5, c = 6
**Output - 3:**
True (SCALENE)
Explanation: All sides are different, so the triangle is scalene.

**Input - 4:**
a = 1, b = 2, c = 3
**Output - 4:**
False (INVALID)
Explanation: The sum of two sides equals the third (1 + 2 = 3), making it an invalid triangle.

**Input - 5:**
a = 2, b = 2, c = 3
**Output - 5:**
True (ISOSCELES)
Explanation: Two sides are equal (2), making the triangle isosceles.

**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;
}
```

**Input - 1:**
s1 = "pre"
s2 = "prefix"
**Output - 1:**
True
Explanation: The string "pre" is a prefix of "prefix."

**Input - 2:**
s1 = "hello"
s2 = "hello world"
**Output - 2:**
True
Explanation: The string "hello" is a prefix of "hello world."

**Input - 3:**
s1 = "test"
s2 = "testing"
**Output - 3:**
True
Explanation: The string "test" is a prefix of "testing."

**Input - 4:**
s1 = "world"
s2 = "hello"
**Output - 4:**
False
Explanation: The string "world" is not a prefix of "hello."

**Input - 5:**
s1 = "abc"
s2 = "ab"
**Output - 5:**
False
Explanation: The string "abc" is longer than "ab," so it cannot be a prefix.

**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 for this system can be categorized based on the possible properties of a triangle. We have classes for valid and invalid inputs, as well as for different types of triangles:

1. **Invalid Inputs:**
    - One or more sides are non-positive (invalid).
    - Sum of any two sides is not greater than the third side (non-triangle, invalid).
2. **Valid Triangles:**
    - Equilateral Triangle: All sides are equal.
    - Isosceles Triangle: Exactly two sides are equal.
    - Scalene Triangle: All three sides are different, and the triangle inequality holds.
    - Right-Angled Triangle: Satisfies the Pythagorean theorem, i.e., A2+B2=C2A^2 + B^2 = C^2A2+B2=C2.

**b) Identify Test Cases to Cover the Identified Equivalence Classes**

Here are test cases, each associated with an equivalence class:

1. **Test Case 1 (Invalid Input - Non-positive side):**
    - Input: A=−1,B=2,C=2
    - Covers: One side is non-positive (invalid).
    - Expected Output: "Invalid"

2. **Test Case 2 (Invalid Input - Non-triangle):**
    - Input: A=1,B=2,C=3
    - Covers: Sum of two sides is not greater than the third side.
    - Expected Output: "Not a triangle"

3. **Test Case 3 (Equilateral Triangle):**
    - Input: A=3,B=3,C=3
    - Covers: All sides are equal (equilateral).
    - Expected Output: "Equilateral"

4. **Test Case 4 (Isosceles Triangle):**
    - Input: A=3,B=3,C=2
    - Covers: Two sides are equal (isosceles).
    - Expected Output: "Isosceles"

5. **Test Case 5 (Scalene Triangle):**
    - Input: A=3,B=4,C=5
    - Covers: All sides are different (scalene).
    - Expected Output: "Scalene"

6. **Test Case 6 (Right-Angled Triangle):**
    - Input: A=3,B=4,C=5
    - Covers: Satisfies the Pythagorean theorem (right-angled).
    - Expected Output: "Right-angled"

**c) Boundary Condition for Scalene Triangle**

1. Test Case 7 (Boundary for Scalene Triangle):
    - Input: A=2,B=2,C=3
    - This input is on the boundary of forming a valid triangle.
    - Expected Output: "Scalene"

**d) Boundary Condition for Isosceles Triangle**

1. **Test Case 8 (Boundary for Isosceles Triangle):**
    - Input: A=3,B=2,C=3
    - Tests the boundary where exactly two sides are equal.
    - Expected Output: "Isosceles"

**e) Boundary Condition for Equilateral Triangle**

1. **Test Case 9 (Boundary for Equilateral Triangle):**
    - Input: A=3,B=3,C=3
    - Tests the condition where all sides are equal.
    - Expected Output: "Equilateral"

**f) Boundary Condition for Right-Angle Triangle**

1. **Test Case 10 (Boundary for Right-Angled Triangle):**
    - Input: A=5,B=12,C=13
    - Tests the Pythagorean theorem boundary.
    - Expected Output: "Right-angled"

**g) Boundary Condition for Non-Triangle**

1. **Test Case 11 (Non-triangle boundary case):**
    - Input: A=1,B=2,C=3
    - This case tests when the sum of two sides equals the third side, meaning it's not a valid triangle.
    - Expected Output: "Not a triangle"

**h) Non-positive Input Test Points**

1. **Test Case 12 (Non-positive input):**
    - Input: A=0,B=3,C=3
    - One side is zero, so this is invalid.
    - Expected Output: "Invalid"

2. **Test Case 13 (Negative side input):**
    - Input: A=−5,B=4,C=6
    - Negative value for one side.
    - Expected Output: "Invalid