



# Dhirubhai Ambani Institute of Information Communication Technology

Course : IT314 Software Engineering

---

## Lab 8

*Functional Testing (Black-Box)*

---

Student Name : Neel Patel

Student ID : 202201494

Group : G32

## Question 1

---

Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges  $1 \leq \text{month} \leq 12$ ,  $1 \leq \text{day} \leq 31$ ,  $1900 \leq \text{year} \leq 2015$ . 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 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.

The solution of each problem must be given in the format as follows:

***Tester Action and Input Data***  
***Equivalence Partitioning***

***Expected Outcome***

a, b, c

An Error message

a-1, b, c

Yes

***Boundary Value Analysis***

a, b, c-1

Yes

## Answer

### Equivalence Partitioning

Equivalence Partitioning is a technique used to divide a set of test cases into groups or partitions that are expected to exhibit similar behavior.

In this case, we have three input parameters: day, month, and year. Let's identify the equivalence classes for each parameter:

➤ Day

- Valid days: 1 to 31
- Invalid days: 0, negative numbers, and numbers greater than 31

➤ Month

- Valid months: 1 to 12
- Invalid months: 0, negative numbers, and numbers greater than 12

➤ Year

- Valid years: 1900 to 2015
- Invalid years: Years before 1900 and after 2015

## Equivalence Partitioning Test Cases

### Tester Action and Input Data

***Expected Outcome***

a. Valid day (15), valid month (6), valid year (2000)	Previous date or valid date
b. Invalid day (0), valid month (6), valid year (2000)	An Error message
c. Valid day (15), invalid month (0), valid year (2000)	An Error message
d. Valid day (15), valid month (13), valid year (2000)	An Error message
e. Valid day (15), valid month (6), invalid year (1899)	An Error message
f. Valid day (15), valid month (6), invalid year (2016)	An Error message
g. Invalid day (0), invalid month (0), valid year (2000)	An Error message
h. Invalid day (32), invalid month (13), valid year (2000)	An Error message
i. Valid day (1), valid month (1), invalid year (2016)	An Error message
j. Valid day (31), valid month (12), invalid year (1899)	An Error message
k. Invalid day (0), invalid month (0), invalid year (1899)	An Error message
l. Invalid day (32), invalid month (13), invalid year (2016)	An Error message

## Boundary Value Analysis

Boundary Value Analysis focuses on testing at the boundaries between equivalence partitions.

- Day
  - Test with values: 1, 31, 0, 32
  
- Month
  - Test with values: 1, 12, 0, 13
  
- Year
  - Test with values: 1900, 2015, 1899, 2016

## Boundary Value Analysis Test Cases

### ***Tester Action and Input Data***

- a. Valid day (1), valid month (1), valid year (1900)
- b. Valid day (31), valid month (12), valid year (2015)
- c. Invalid day (0), valid month (6), valid year (2000)
- d. Invalid day (32), valid month (6), valid year (2000)
- e. Valid day (15), invalid month (0), valid year (2000)
- f. Valid day (15), invalid month (13), valid year (2000)
- g. Valid day (15), valid month (6), invalid year (1899)
- h. Valid day (15), valid month (6), invalid year (2016)

### ***Expected Outcome***

- Previous date or valid date
- Previous date or valid date
- An Error message
- An Error message
- An Error message
- An Error message
- An Error message
- An Error message

## Question 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);
}
```

### Modified Code

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

**Test 1:** Search for 5  
Expected: 2, Got: 2

**Test 2:** Search for 7  
Expected: 3, Got: 3

<b>Test 3:</b> Search for 2 (not in array) Expected: -1, Got: -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);
}
```

#### Modified Code

<pre>int countItem(int v, int a[], int size) {     int count = 0;     for (int i = 0; i &lt; size; i++)     {         if (a[i] == v)             count++;     }     return count; }</pre>
<p><b>Test 1:</b> Count occurrences of 3 Expected: 3, Got: 3</p> <p><b>Test 2:</b> Count occurrences of 5 Expected: 1, Got: 1</p> <p><b>Test 3:</b> Count occurrences of 2 (not in array) Expected: 0, Got: 0</p>

**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);
}
```

#### Modified Code

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

**Test 1:** Search for 7

Expected: 3, Got: 3

**Test 2:** Search for 1

Expected: 0, Got: 0

**Test 3:** Search for 19

Expected: 9, Got: 9

**Test 4:** Search for 4 (not in array)

Expected: -1, Got: -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);
}
```

Modified Code

```
#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3
```



```

int triangle(int a, int b, int c)
{
    if (a <= 0 || b <= 0 || c <= 0 || 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;
}

```

**Test 1:** a=3, b=3, c=3 (Equilateral)

Expected: EQUILATERAL, Got: 0

**Test 2:** a=3, b=4, c=4 (Isosceles)

Expected: ISOSCELES, Got: 1

**Test 3:** a=3, b=4, c=5 (Scalene)

Expected: SCALENE, Got: 2

**Test 4:** a=1, b=10, c=12 (Invalid)

Expected: INVALID, Got: 3

**Test 5:** a=0, b=4, c=5 (Invalid - Zero Side Length)

Expected: INVALID, Got: 3

**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;
}

```

### Modified Code

```
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;  
}
```

**Test 1:** prefix("pre", "prefix")

Expected: true, Got: true

**Test 2:** prefix("fix", "prefix")

Expected: false, Got: false

**Test 3:** prefix("prefix", "prefix")

Expected: true, Got: true

**Test 4:** prefix("longer", "short")

Expected: false, Got: false

**Test 5:** prefix("", "empty")

Expected: true, Got: true

**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:

- Identify the equivalence classes for the system
- 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)
- 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  $A^2 + B^2 = C^2$  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.

## Answer

### a) Identify the equivalence classes for the system:

In this program, the equivalence classes can be identified as follows:

1. Equilateral Triangle (All sides are equal):  $A = B = C$
2. Isosceles Triangle (Two sides are equal):  $A = B, A = C, B = C$
3. Scalene Triangle (No sides are equal):  $A \neq B \neq C$
4. Right-Angled Triangle (Pythagorean Theorem holds):  $A^2 + B^2 = C^2$
5. Non-Triangle (Impossible lengths):  $A + B \leq C$  or  $B + C \leq A$  or  $A + C \leq B$
6. Boundary Condition  $A + B = C$  (Scalene Triangle):  $A + B = C$
7. Boundary Condition  $A = C$  (Isosceles Triangle):  $A = C$
8. Boundary Condition  $A = B = C$  (Equilateral Triangle):  $A = B = C$
9. Boundary Condition  $A^2 + B^2 = C^2$  (Right-Angled Triangle):  $A^2 + B^2 = C^2$
10. Non-Positive Inputs: A, B, or C is less than or equal to zero

### b) Identify test cases to cover the identified equivalence classes:

To ensure comprehensive test coverage, we can design test cases as follows:

#### 1. Equivalence Class: Equilateral Triangle

- Test case:  $A = 3, B = 3, C = 3$  (All sides are equal)

#### 2. Equivalence Class: Isosceles Triangle

- Test case 1:  $A = 3, B = 3, C = 4$  (Two sides are equal:  $A = B$ )
- Test case 2:  $A = 4, B = 3, C = 3$  (Two sides are equal:  $B = C$ )
- Test case 3:  $A = 3, B = 4, C = 3$  (Two sides are equal:  $A = C$ )

#### 3. Equivalence Class: Scalene Triangle

- Test case 1:  $A = 3, B = 4, C = 5$  (No sides are equal)
- Test case 2:  $A = 7, B = 24, C = 25$  (No sides are equal)

4. Equivalence Class: Right-Angled Triangle

- Test case 1:  $A = 3, B = 4, C = 5$  ( $A^2 + B^2 = C^2$ )

5. Equivalence Class: Non-Triangle

- Test case 1:  $A = 1, B = 2, C = 3$  ( $A + B \leq C$ )
- Test case 2:  $A = 3, B = 1, C = 2$  ( $B + C \leq A$ )

6. Boundary Condition:  $A + B = C$  (Scalene Triangle)

- Test case:  $A = 1, B = 2, C = 3$

7. Boundary Condition:  $A = C$  (Isosceles Triangle)

- Test case:  $A = 3, B = 4, C = 3$  ( $A = C$ )

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

- Test case:  $A = 5, B = 5, C = 5$  ( $A = B = C$ )

9. Boundary Condition:  $A^2 + B^2 = C^2$  (Right-Angled Triangle)

- Test case:  $A = 3, B = 4, C = 5$  ( $A^2 + B^2 = C^2$ )

**c) For the boundary condition  $A + B > C$  case (scalene triangle):**

- Test case:  $A = 2, B = 3, C = 5$

**d) For the boundary condition  $A = C$  case (isosceles triangle):**

- Test case:  $A = 3, B = 4, C = 3$  ( $A = C$ )

**e) For the boundary condition  $A = B = C$  case (equilateral triangle):**

- Test case:  $A = 4, B = 4, C = 4$  ( $A = B = C$ )

**f) For the boundary condition  $A^2 + B^2 = C^2$  case (right-angle triangle):**

- Test case:  $A = 3, B = 4, C = 5$  ( $A^2 + B^2 = C^2$ )

**g) For the non-triangle case:**

- Test case 1:  $A = 1, B = 2, C = 3$  ( $A + B \leq C$ )
- Test case 2:  $A = 3, B = 1, C = 2$  ( $B + C \leq A$ )
- Test case 3:  $A = 3, B = 2, C = 1$  ( $C + A \leq B$ )

**h) For non-positive input:**

- Test case 1:  $A = 0, B = 4, C = 5$  (A is non-positive)
- Test case 2:  $A = 4, B = -3, C = 2$  (B is non-positive)
- Test case 3:  $A = 3, B = 4, C = 0$  (C is non-positive)