

# SOFTWARE ENGINEERING



**Lab - 8**

**Name : Malhar Vaghasiya**

**ID : 202201183**

**Q.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 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.**

Equivalence Class Partitioning Test cases :

| Input        | Expected Outcome | Reasoning   |
|--------------|------------------|---|
| 15,5,2000    | Valid            | Typical Valid Date                                |
| 31,12,2015   | Valid            | Valid on upper year boundary                      |
| 29,2,2000    | Valid            | Valid leap year case for february                 |
| 28,1,2001    | Valid            | Valid date for February in a non-leap year.       |
| 32, 5, 2000  | Invalid          | Invalid day (May has 31 days)                     |
| 29, 2, 1900  | Invalid          | Invalid leap year case (1900 is not a leap year). |
| 15, 13, 2001 | Invalid          | Invalid month (greater than 12).                  |
| 0, 6, 2005   | Invalid          | Invalid day (less than 1).                        |

|           |         |                                  |
|-----------|---------|----------------------------------|
| 1,0,1999  | Invalid | Invalid month (less than 1).     |
| 29,2,2021 | Invalid | Invalid year (greater than 2015) |

Boundary Value Analysis Test Cases :

| Input      | Expected Outcome | Reasoning  |
|------------|------------------|--|
| 1,1,1990   | Valid            | Lower boundary of the year                         |
| 31,12,2015 | Valid            | Upper boundary of the year                         |
| 1,3,2000   | Valid            | Leap year boundary case                            |
| 0,1,2003   | Invalid          | Invalid day (less than 1)                          |
| 32,1,2013  | Invalid          | Invalid day (greater than 31)                      |
| 29,2,1900  | Invalid          | Invalid leap year boundary (1900 not a leap year). |
| 28,2,2001  | Invalid          | Valid boundary for february in a non leap year     |

Code :

```
#include <iostream>
using namespace std;

bool isLeapYear(int year) {
```

```

        if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))
            return true;
        return false;
    }

```

```

int daysInMonth(int month, int year)
{
    if (month == 2)
        return isLeapYear(year) ? 29 : 28;

    if (month == 4 || month == 6 || month == 9 || month == 11)
        return 30;

    return 31;
}

```

```

void previousDate(int day, int month, int year)
{
    if (year < 1900 || year > 2015 || month < 1 || month > 12 ||
        day < 1 || day > daysInMonth(month, year)) {
        cout << "Error: Invalid date" << endl;
        Return;
    }
    day--;

    if (day == 0) {
        month--;

        if (month == 0) {
            month = 12;
            Year--;
        }
        day = daysInMonth(month, year);
    }

    if (year < 1900) {
        cout << "Error: Invalid date" << endl;
        Return;
    }
}

```

```

        cout << "Previous date is: " << day << "/" << month << "/" <<
        year << endl;
    }

int main() {
    previousDate(1, 1, 2001);
    return 0;
}

```

## Q.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);
}

```

### Value Present:

- E1: The value `v` is present in the array and occurs once.
- E2: The value `v` is present in the array and occurs multiple times.
- E3: The value `v` is not present in the array.

### Array Edge Cases:

- E4: The array is empty.
- E5: The value `v` is at the first or last position in the array.

Equivalence Classes:

| Test case | Input              | Expected Outcome | Equivalence boundary |
|-----------|--------------------|------------------|----------------------|
| TC 1      | v = 5, [1,2,3,4,5] | 4                | E1                   |
| TC 2      | v = 3, [1,3,3,3,5] | 1                | E2                   |
| TC 3      | v = 6, [1,2,3,4,5] | -1               | E3                   |
| TC 4      | V = 2, []          | -1               | E4                   |
| TC 5      | v = 1, [1,2,3,4,5] | 0                | E5                   |

Boundary Points :

- BP1: Single-element array where v is present.
- BP2: Single-element array where v is not present.
- BP3: v is at the first position.
- BP4: v is at the last position.
- BP5: Array contains negative numbers and v is a negative number.

| Test case | Input             | Expected Outcome | Equivalence boundary |
|-----------|-------------------|------------------|----------------------|
| BP 1      | v=3, [3]          | 0                | BP 1                 |
| BP 2      | v=2, [1]          | 1                | BP 2                 |
| BP 3      | v=1, [1,2,3,4,5]  | -0               | BP 3                 |
| BP 4      | v=5, [1,2,3,4,5]  | 4                | BP 4                 |
| BP 5      | v=-3, [1,2,3,4,5] | 2                | BP 5                 |

**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);
}
```

Value Present:

- E1: The value v is present in the array and occurs once.
- E2: The value v is present in the array and occurs multiple times.
- E3: The value v is not present in the array.

Array Edge Cases:

- E4: The array is empty.
- E5: The value v is at the first or last position in the array.

Equivalence Classes :-

| Test case | Input              | Expected Outcome | Equivalence boundary |
|-----------|--------------------|------------------|----------------------|
| TC 1      | v = 5, [1,2,3,4,5] | 1                | E1                   |
| TC 2      | v = 3, [1,3,3,3,5] | 3                | E2                   |
| TC 3      | v = 6, [1,2,3,4,5] | 0                | E3                   |
| TC 4      | V = 2, []          | 0                | E4                   |
| TC 5      | v = 1, [1,2,3,4,5] | 1                | E5                   |

Boundary Points for countItem:

1. BP1: Single-element array where v is present.
2. BP2: Single-element array where v is not present.
3. BP3: v is at the first position.
4. BP4: v is at the last position.
5. BP5: Array contains negative numbers and v is a negative number.

| Test case | Input                      | Expected Outcome | Equivalence boundary |
|-----------|----------------------------|------------------|----------------------|
| BP 1      | v=3, [3]                   | 1                | BP 1                 |
| BP 2      | v=2, [1]                   | 0                | BP 2                 |
| BP 3      | v=1, [1,2,3,4,5]           | 1                | BP 3                 |
| BP 4      | v=5, [1,2,3,4,5]           | 1                | BP 4                 |
| BP 5      | v= -3,<br>[-5,-4,-3,-2,-1] | 1                | BP 5                 |

**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 Classes for binarySearch:

#### 1. Value Present:

- E1: The value v is present in the array and is located at the first position.
- E2: The value v is present in the array and is located at the last position.
- E3: The value v is present in the array and is located somewhere in the middle.

#### 2. Value Not Present:

- E4: The value v is less than the smallest element in the array.
- E5: The value v is greater than the largest element in the array.
- E6: The value v is not in the array but falls between two elements.

#### 3. Array Edge Cases:

- E7: The array is empty.
- E8: The array contains one element, which may or may not be equal to v.

### Equivalence Classes :-

| Test case | Input             | Expected Output | Equivalence Boundary |
|-----------|-------------------|-----------------|----------------------|
| TC1       | v=1,[1,2,3,4,5]   | 0               | E1                   |
| TC2       | v=5,[1,2,3,4,5]   | 4               | E2                   |
| TC3       | v=3,[1,2,3,4,5]   | 2               | E3                   |
| TC4       | v=0,[1,2,3,4,5]   | 1               | E4                   |
| TC5       | v=6,[1,2,3,4,5]   | -1              | E5                   |
| TC6       | v=2.5,[1,2,3,4,5] | -1              | E6                   |
| TC7       | v=3,[ ]           | -1              | E7                   |
| TC8       | v=1,[1]           | 0               | E8                   |
| TC9       | v=2,[1]           | -1              | E8                   |

Boundary Points for binarySearch:

1. BP1: Single-element array where v is equal to the element.
2. BP2: Single-element array where v is not equal to the element.
3. BP3: The value v is at the first position in a multi-element sorted array.
4. BP4: The value v is at the last position in a multi-element sorted array.
5. BP5: The array contains duplicate values of v.

| Test Case | Input                | Expected Output | Equivalence Boundary |
|-----------|----------------------|-----------------|----------------------|
| BP1       | v=3, [3]             | 0               | BP1                  |
| BP2       | v=2, [3]             | -1              | BP2                  |
| BP3       | v=1, [1,2,3,4,5]     | 0               | BP3                  |
| BP4       | v=5, [1,2,3,4,5]     | 4               | BP4                  |
| BP5       | v=3, [1,2,3,3,3,4,5] | 2               | BP5                  |

**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);
}
```

Done with P6.

**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;
}
```

#### Valid Prefix Cases:

- E1: s1 is a non-empty string and is a prefix of s2.
- E2: s1 is an empty string, which is considered a prefix of any string s2.
- E3: s1 is equal to s2.

#### Invalid Prefix Cases:

- E4: s1 is longer than s2.
- E5: s1 is not a prefix of s2 (they differ after some characters)

#### Equivalence Classes :

| Test Case | Input                            | Expected Output | Equivalence Boundary |
|-----------|----------------------------------|-----------------|----------------------|
| TC1       | s1 = "pre",<br>s2 = "prefix"     | true            | E1                   |
| TC2       | s1 = "",<br>s2 = "anything"      | true            | E2                   |
| TC3       | s1 = "same",<br>s2 = "same"      | true            | E3                   |
| TC4       | s1 = "longer",<br>s2 = "shorter" | false           | E4                   |
| TC5       | s1 = "prefix",<br>s2 = "pre"     | false           | E5                   |

#### Boundary Points for prefix Function:

1. BP1: s1 is a single character and is a prefix of s2.
2. BP2: s1 is a single character and is not a prefix of s2.

3. BP3: s1 is an empty string and s2 is a non-empty string.
4. BP4: s1 is equal to s2, which also has one character.
5. BP5: s1 is the same as the first few characters of s2, but does not cover the entire s2.

| Test Case | Input                         | Expected Output | Equivalence Boundary |
|-----------|-------------------------------|-----------------|----------------------|
| BP1       | s1 = "p",<br>s2 = "prefix"    | true            | BP1                  |
| BP2       | s1 = "x",<br>s2 = "prefix"    | false           | BP2                  |
| BP3       | s1 = "",<br>s2 = "hello"      | true            | BP3                  |
| BP4       | s1 = "a",<br>s2 = true        | true            | BP4                  |
| BP5       | s1 = "pref",<br>s2 = "prefix" | true            | BP5                  |

**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 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  $A^2 + B^2 = C^2$  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.

**Soln :-**

**a). Identify the equivalence classes for the system**

The equivalence classes for this system can be divided into valid and invalid triangles based on the properties of the triangle:

Valid Triangle:

- Equilateral Triangle (E1): All sides are equal:  $A=B=C$
- Isosceles Triangle (E2): Two sides are equal:  $A=B$ ,  $B=C$ , or  $C=A$
- Scalene Triangle (E3): All sides are different:  $A \neq B$ ,  $B \neq C$ , or  $C \neq A$

Invalid Triangle:

- Non-Triangle Case (I1): Sum of two sides is less than or equal to the third side:  $A+B \leq C$  or  $A+C \leq B$  or  $B+C \leq A$
- Non-Positive Inputs (I2): One or more sides have non-positive values:  $A \leq 0$ ,  $B \leq 0$ ,  $C \leq 0$ .

**b). Identify test cases to cover the identified equivalence classes.**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| TC 1      | 5,5,5         | Equilateral      | E1                   |
| TC 2      | 5,5,3         | Isosceles        | E2                   |
| TC 3      | 5,4,3         | Scalene          | E3                   |
| TC 4      | 3,4,5         | Right Angled     | E4                   |
| TC 5      | 1,2,3         | Invalid          | I1                   |
| TC 6      | 0,3,4         | Invalid          | I2                   |
| TC 7      | -1,2,3        | Invalid          | I2                   |

**c). Boundary condition  $A+B > C$  (Scalene triangle)**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 1      | 3,4,7         | Invalid          | $A + B = C$          |
| BC 2      | 3,4,6.99      | Scalene          | $A + B > C$          |
| BC 3      | 3,4,6         | Invalid          | $A + B > C$          |

**d). Boundary condition  $A+B > C$  (Scalene triangle)**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 5      | 5,5,8         | Isosceles        | $A = B$              |
| BC 6      | 5,8,5         | Isosceles        | $A = C$              |
| BC 7      | 8,5,5         | Isosceles        | $B = C$              |
| BC 8      | 5,5,10        | Invalid          | $A + B = C$          |

**e) Boundary condition  $A=B=C$  case (Equilateral triangle)**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 9      | 5,5,5         | Equilateral      | $A = B = C$          |
| BC 10     | 5.1,5.1,5.1   | Equilateral      | $A = B = C$          |

**f). Boundary condition  $A^2 + B^2 = C^2$  case (Right-angled triangle)**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 11     | 3,4,5         | Right Angled     | $A^2 + B^2 = C^2$    |
| BC 12     | 6,8,10        | Right Angled     | $A^2 + B^2 = C^2$    |
| BC 13     | 5,12,13       | Right Angled     | $A^2 + B^2 = C^2$    |



**g) Boundary condition for non-triangle case**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 14     | 1,2,3         | Invalid          | $A + B = C$          |
| BC 15     | 2,2,5         | Invalid          | $A + B < C$          |
| BC 16     | 1.5,2,3.5     | Invalid          | $A + B = C$          |

**h) Non-positive input test points**

| Test case | Input (A,B,C) | Expected Outcome | Equivalence boundary |
|-----------|---------------|------------------|----------------------|
| BC 17     | 0,3,4         | Invalid          | $A = 0$              |
| BC 18     | -1,4,5        | Invalid          | $A = -1$             |
| BC 19     | 5,0,5         | Invalid          | $B = 0$              |
| BC 20     | 3,4,-2        | Invalid          | $C = -2$             |