# Software Engineering

## IT-314


**Akshat Joshi - 202201185**

**Dhirubhai Ambani Institute Of Information And Communication Technology**

**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.**

**Answer :-**

## Equivalence Class Partitioning Test Cases :-

| Input | Expected Outcome | Reason |
|---|---|---|
| 14, 4, 2001 | Valid | Typical Valid Date |
| 31, 12, 2015 | Valid | Valid on upper year boundary |
| 29, 2, 2004 | Valid | Valid leap year case for February |
| 28, 2, 2001 | | Valid date for February |

| | | |
|---|---|---|
| | | in a non-leap year |
| 32, 5, 2000 | | Invalid day (May has only 31 days) |
| 29, 2, 1900 | | 1900 is not a leap year |
| 15, 13, 2002 | | Invalid month (greater than 12) |
| 0, 6, 2005 | | Invalid day (less than 1) |
| 1, 0, 1999 | | Invalid month (less than 1) |
| 29, 2, 2021 | | Invalid Year (greater than 2015) |

## Boundary Value Analysis Test Cases :-

| Input | Expected Outcome | Reason |
|---|---|---|
| 1, 1, 2020 | Invalid | Year is beyond the upper boundary of 2015 |
| 30, 2, 2004 | Invalid | Invalid leap year day (February cannot have 30 days) |
| 28, 2, 2001 | Valid | Valid date in February in non-leap year |
| 31, 4, 2010 | | Invalid day for April (April has only 30 days) |
| 0, 1, 1999 | | Invalid day (Day less than 1) |

| | | |
|---|---|---|
| 32, 5, 2000 | | Invalid day for May (May has only 31 days) |
| 1,3,2000 | | Leap year transition from february |
| 29, 2, 1900 | | 1900 is not a leap year |
| 31, 12, 2015 | | Upper boundary for the year |
| 1,1,1900 | | Lower boundary for the year |

**Code :-**

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

bool isLeapYear(int year) {
    return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
}

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;
}
```

```cpp
void nextDate(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 > daysInMonth(month, year)) {
        day = 1;
        month++;
        if (month > 12) {
            month = 1;
            year++;
        }
    }

    if (year > 2015) {
        cout << "Error: Invalid date" << endl;
        return;
    }

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

int main() {
    nextDate(31, 12, 2015);
    return 0;
}
```

## 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.**

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

**Answer :-**

**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.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | v = 8, a = [2, 4, 8, 10, 12] | 2 | E1: Value present once. |
| TC2 | v = 6, a = [1,6,6,7,8] | 1 | E2: Value present multiple times; return first index. |
| TC3 | v = 11, a = [1,2,3,4,5] | -1 | E3: Value not present. |

| | | | |
|---|---|---|---|
| TC4 | v = 3, a = [] | -1 | E4: Empty array |
| TC5 | v = 4, a = [4,5,6] | 0 | E5: Value present at the first position. |

**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 in a multi-element array.
- BP4: v is at the last position in a multi-element array.
- BP5: Array contains negative numbers, and v is a negative number.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BP1 | v = 10, a = [10] | 0 | BP1: Single-element array where v is present. |
| BP2 | v = 8, a = [10] | -1 | BP2: Single-element array where v is not present. |

| | | | |
|---|---|---|---|
| BP3 | v = 2, a = [2,5,9] | 0 | BP3: Value present at the first position. |
| BP4 | v = 30, a = [10, 20, 30] | 2 | BP4: Value present at the last position. |
| BP5 | v = -6, a = [-6,-5, -3, 0, 7] | 1 | BP5: Array contains negative numbers, v is negative. |

**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);
}
```

**Answer :-**

**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.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | v = 9, a = [2, 9, 10, 12, 15] | 1 | E1: Value present once. |
| TC2 | v = 4, a = [2, 4, 6, 4, 5] | 2 | E2: Value present multiple times. |
| TC3 | v = 11, a = [1, 2, 3, 4, 5] | 0 | E3: Value not present. |
| TC4 | v = 7 , a = [] | 0 | E4: Empty array, value cannot appear. |
| TC5 | v = 2, a = [2, 3, 4] | 1 | E5: Value present at the first position. |

**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 in a multi-element array.
- BP4: v is at the last position in a multi-element array.
- BP5: Array contains negative numbers, and v is a negative number.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BP1 | v = 7, a = [7] | 1 | BP1: Single-element array where v is present. |
| BP2 | v = 5, a = [3] | -1 | BP2: Single-element array where v is not present. |
| BP3 | v = 15, a = [15, 20, 30] | 1 | BP3: Value present at the first position. |

| | | | |
|---|---|---|---|
| BP4 | v = 30, a = [10, 20, 30] | 1 | BP4: Value present at the last position. |
| BP5 | v = -8, a = [-8, -6, 0, 5] | 0 | BP5: Array contains negative numbers, and v is |

**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;
}
```

## Answer:-

**Equivalence Classes:**

**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.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | v = 1, a = [1, 2, 3, 4, 5] | 0 | E1: Value present at the first position. |

| | | | |
|---|---|---|---|
| TC2 | v = 5, a = [1, 2, 3, 4, 5] | 4 | E2: Value present at the last position. |
| TC3 | v = 3, a = [1, 2, 3, 4, 5] | 2 | E3: Value present in the middle. |
| TC4 | v = 0, a = [1, 2, 3, 4, 5] | -1 | E4: Value is less than the smallest element. |
| TC5 | v = 6, a = [1, 2, 3, 4, 5] | -1 | E5: Value is greater than the largest element. |
| TC6 | v = 3, a = [1, 2, 4, 5] | -1 | E6: Value falls between two elements but is not present. |
| TC7 | v = 1, a = [] | -1 | E7: Array is empty. |
| TC8 | v = 10, a = [10] | 0 | E8: Array has one element, which is equal to the value. |

**Boundary Points:**

BP1: Single-element array where v is equal to the element.

BP2: Single-element array where v is not equal to the element.

BP3: The value v is at the first position in a multi-element sorted array.

BP4: The value v is at the last position in a multi-element sorted array.

BP5: The array contains duplicate values of v.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BP1 | v = 10, a = [10] | 0 | BP1: Single-element array where v is equal to the element. |
| BP2 | v = 5, a = [10] | -1 | BP2: Single-element array where v is not equal to the element. |
| BP3 | v = 1, a = [1, 2, 3, 4, 5] | 0 | BP3: Value is at the first position in a multi-element |

| | | | |
|---|---|---|---|
| BP4 | v = 5, a = [1, 2, 3, 4, 5] | 4 | BP4: Value is at the last position in a multi-element |
| BP5 | v = 3, a = [1, 3, 3, 3, 4] | 1 | BP5: Array contains duplicate values of v. |

**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);
}
```

## Answer:-

**Equivalence Classes:**

E1: All three sides are equal.

E2: Two sides are equal.

E3: All sides are different and valid.

E4: Sum of two sides not greater than the third side.

E5: One or more sides are zero or negative.

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | a = 3, b = 3, c = 3 | EQUILATERAL | E1: All three sides are equal. |
| TC2 | a = 5, b = 5, c = 3 | ISOSCELES | E2: Two sides are equal. |
| TC3 | a = 3, b = 4, c = 5 | SCALENE | E3: All sides are different and valid. |
| TC4 | a = 1, b = 2, c = 3 | INVALID | E4: Sum of two sides not greater than the third side. |

| | | | |
|---|---|---|---|
| TC5 | a = 0, b = 1, c = 1 | INVALID | E5: One or more sides are zero or negative. |

**Boundary Points:**

BP1: Smallest possible valid triangle (all sides equal).

BP2: Isosceles with third side slightly different.

BP3: Isosceles with one side on boundary of invalidity

BP4: Scalene with valid Pythagorean triplet.

BP5: Sum of two sides equals third side (invalid).

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BP1 | a = 1, b = 1, c = 1 | EQUILATERAL | BP1: Smallest possible valid triangle (all sides equal). |
| BP2 | a = 2, b = 2, c = 3 | ISOSCELES | BP2: Isosceles with third side slightly different. |

| | | | |
|---|---|---|---|
| BP3 | a = 1, b = 2, c = 2 | ISOSCELES | BP3: Isosceles with one side on boundary of invalidity |
| BP4 | a = 3, b = 4, c = 5 | SCALENE | BP4: Scalene with valid Pythagorean triplet. |
| BP5 | a = 1, b = 10, c = 12 | INVALID | BP5: Sum of two sides equals third side (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;
}
```

## Answer:-

**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).

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | s1 = "pre", s2 = "prefix" | true | E1: s1 is a non-empty string and is a prefix of s2. |
| TC2 | s1 = "", s2 = "hello" | true | E2: s1 is an empty string (prefix of any s2). |
| TC3 | s1 = "test", s2 = "test" | true | E3: s1 is equal to s2. |

| | | | |
|---|---|---|---|
| TC4 | s1 = "hello", s2 = "hi" | false | E4: s1 is longer than s2. |
| TC5 | s1 = "abc", s2 = "abz" | false | E5: s1 is not a prefix of s2 (they differ after some characters). |

**Boundary Points:**

BP1: s1 is a single character and is a prefix of s2.

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

BP3: s1 is an empty string and s2 is a non-empty string.

BP4: s1 is equal to s2, which also has one character.

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 Class** |
|---|---|---|---|
| BP1 | s1 = "a", s2 = "abc" | true | BP1: s1 is a single character and is a prefix of s2. |

| | | | |
|---|---|---|---|
| BP2 | s1 = "z", s2 = "abc" | false | BP2: s1 is a single character and is not a prefix of s2. |
| BP3 | s1 = "", s2 = "test" | true | BP3: s1 is an empty string and s2 is non-empty. |
| BP4 | s1 = "a", s2 = "a" | true | BP4: s1 is equal to s2, both having one character. |
| BP5 | s1 = "pre", s2 = "prefix" | true | BP5: s1 matches the initial part of s2 but not the entire string. |

**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 need to be 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 A2 + B2 = C2 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**

The equivalence classes for this system can be divided into valid and invalidtriangles 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≠B, B≠C, A≠C

- Right-angled Triangle (E4): Follows the Pythagorean theorem A^2 + B^2 = C^2 with A ≤ B ≤ C

**Invalid Triangle:**

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

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

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| TC1 | A = 5.0, B = 5.0, C = 5.0 | Equilateral Triangle | E1: All sides are equal. |
| TC2 | A = 4.0, B = 4.0, C = 3.0 | Isosceles Triangle | E2: Two sides are equal. |
| TC3 | A = 3.0, B = 4.0, C = 5.0 | Right-angled Triangle | E4: Satisfies Pythagorean theorem. |
| TC4 | A = 2.0, B = 3.0, C = 4.0 | Scalene Triangle | E3: All sides are different. |

| | | | |
|---|---|---|---|
| TC5 | A = 1.0, B = 2.0, C = 3.0 | Invalid Triangle | I1: Sum of two sides is not greater than the third side. |
| TC6 | A = -1.0, B = 2.0, C = 2.0 | Invalid Triangle | I2: One side is non-positive. |

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

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BC1 | A = 5.0, B = 5.0, C = 9.9 | Scalene Triangle | A+B > C |
| BC2 | A = 2.5, B = 2.6, C = 4.9 | Scalene Triangle | A+B > C |
| BC3 | A = 3.1, B = 3.2, C = 6.3 | Invalid Triangle | A+B = C |

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

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| BC1 | 5, 5, 8 | Isosceles | A = B |
| BC2 | 5, 8, 5 | Isosceles | A = C |
| BC3 | 8, 5, 5 | Isosceles | C = B |
| BC4 | 5, 5, 10 | Invalid | A + B = C |

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

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BC1 | 5, 5, 5 | Equilateral | A = B = C |
| BC2 | 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 | Expected Output | Equivalence Class |
|---|---|---|---|
| BC1 | 3, 4, 5 | Right Angled | A^2 + B^2 = C^2 |
| BC2 | 6, 8, 10 | Right Angled | A^2 + B^2 = C^2 |
| BC3 | 5, 12, 13 | Right Angled | A^2 + B^2 = C^2 |

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

| Test Case | Input | Expected Output | Equivalence Class |
|---|---|---|---|
| BC1 | 1, 2, 3 | Invalid | A + B = C |
| BC2 | 2, 2, 5 | Invalid | A + B < C |
| BC3 | 1.5, 2, 3.5 | Invalid | A + B = C |

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

| **Test Case** | **Input** | **Expected Output** | **Equivalence Class** |
|---|---|---|---|
| BC1 | 0, 3, 4 | Invalid | A = 0 |
| BC2 | -1, 4, 5 | Invalid | A = -1 |
| BC3 | 5, 0, 5 | Invalid | B = 0 |
| BC4 | 3, 4, -2 | Invalid | C = -2 |