



**IT314 - Software Engineering**

**Lab - 8 : Functional Testing (Black-Box)**

**Name : Priyank Ramani**

**ID : 202201497**

# Q1: Determining the Previous Date

## Problem :

We are tasked with designing a test suite for a program that calculates the previous date given a day, month, and year. Inputs should fall within the following ranges:

- **Day:** 1 to 31
- **Month:** 1 to 12
- **Year:** 1900 to 2015

The program will output either a valid previous date or an error indicating an invalid date.

## 1. Equivalence Class Partitioning:

We divide the input space into valid and invalid partitions to reduce the number of test cases.

### Equivalence Classes:

- **Valid Day:**  $1 \leq \text{day} \leq 31$
- **Invalid Day:**  $\text{day} < 1$  or  $\text{day} > 31$
- **Valid Month:**  $1 \leq \text{month} \leq 12$
- **Invalid Month:**  $\text{month} < 1$  or  $\text{month} > 12$
- **Valid Year:**  $1900 \leq \text{year} \leq 2015$
- **Invalid Year:**  $\text{year} < 1900$  or  $\text{year} > 2015$

### Test Cases for Equivalence Partitioning:

Test Case	Day	Month	Year	Expected Output	Equivalence Class
TC1	15	6	2010	Previous Date	All Valid Inputs
TC2	32	6	2010	Invalid Date	Invalid Day
TC3	0	6	2010	Invalid Date	Invalid Day
TC4	15	13	2010	Invalid Date	Invalid Month
TC5	15	0	2010	Invalid Date	Invalid Month
TC6	15	6	1899	Invalid Date	Invalid Year
TC7	15	6	2016	Invalid Date	Invalid Year

## 2. Boundary Value Analysis:

We test the boundaries of the valid input ranges to ensure correct behavior at the edges.

### Boundary Test Cases:

Test Case	Day	Month	Year	Expected Output	Equivalence Class
TC8	1	1	1900	Previous Date	Lower Bound of Day, Month, Year
TC9	31	12	2015	Previous Date	Upper Bound of Day, Month, Year
TC10	2	1	1900	Previous Date	Just Above Lower Bound of Day
TC11	30	12	2015	Previous Date	Just Below Upper Bound of Day

## Q2: Programs Testing

### P1: linearSearch ( int v, int a[ ] )

**Functionality:** Searches for the first occurrence of value **v** in array **a**. Returns the index if found, otherwise returns **-1**.

### Equivalence Classes:

1. **Value is present:** **v** is found in **a[ ]**.
2. **Value is not present:** **v** is not found in **a[ ]**.
3. **Empty array:** **a[ ]** has no elements.
4. **Single-element array:** **a[ ]** contains exactly one element.
5. **Duplicate values:** **a[ ]** contains multiple occurrences of **v**.

### Boundary Conditions:

1. **First element:** **v** is the first element of **a[ ]**.
2. **Last element:** **v** is the last element of **a[ ]**.
3. **Array length 1:** **a[ ]** contains only one element.

**Test Cases:**

Test Case	v	a	Expected Output	Equivalence Class	Boundary Condition
TC1	5	[1, 2, 3, 4, 5]	4	Value Present	Last element
TC2	7	[1, 2, 3, 4, 5]	-1	Value Not Present	-
TC3	3	[]	-1	Empty Array	-
TC4	1	[1]	0	Single Element Array, Value Present	First element and Single element
TC5	2	[1, 2, 2, 3, 4, 5]	1	Duplicate Values Present	First occurrence

**P2: countItem ( int v, int a[ ] )**

- **Functionality:** Counts how many times value **v** appears in array **a[ ]**.

**Equivalence Classes:**

1. **Multiple occurrences:** **v** appears multiple times in **a[ ]**.
2. **Single occurrence:** **v** appears once in **a[ ]**.
3. **No occurrences:** **v** does not appear in **a[ ]**.
4. **Empty array:** **a[ ]** has no elements.
5. **Single-element array:** **a[ ]** contains exactly one element.

**Boundary Conditions:**

1. **Array length 1:** **a[ ]** contains only one element.
2. **Array with duplicates:** **a[ ]** contains repeated values of **v**.

**Test Cases:**

Test Case	v	a	Expected Output	Equivalence Class	Boundary Condition
TC1	5	[1, 2, 5, 5, 5]	3	Multiple Occurrences	-
TC2	5	[1, 2, 3, 4, 5]	1	Single Occurrences	-
TC3	7	[1, 2, 3, 4, 5]	0	No Occurrences	-
TC4	5	[]	0	Empty Array	-
TC5	5	[5]	1	Single Occurrence in Single Element Array	Single element

**P3: binarySearch ( int v, int a[ ] )**

- **Functionality:** Searches for value **v** in a sorted array **a[ ]**. Returns the index if found, otherwise returns **-1**.

**Equivalence Classes:**

1. **Value is present:** **v** exists in **a[ ]**.
2. **Value is not present:** **v** does not exist in **a[ ]**.
3. **Empty array:** **a[ ]** has no elements.
4. **Single-element array:** **a[ ]** contains exactly one element.
5. **Value is smaller than all elements:** **v** is less than all elements in **a[ ]**.
6. **Value is larger than all elements:** **v** is greater than all elements in **a[ ]**.

**Boundary Conditions:**

1. **First element:** **v** is the first element of **a[ ]**.
2. **Middle element:** **v** is the middle element of **a[ ]**.
3. **Last element:** **v** is the last element of **a[ ]**.
4. **Single element array:** **a[ ]** contains one element.

**Test Cases:**

Test Case	v	a	Expected Output	Equivalence Class	Boundary Condition
TC1	1 0	[1, 5, 10, 15, 20]	2	Value Present	Middle element
TC2	1	[1, 5, 10, 15, 20]	0	Value Present	First element
TC3	2 0	[1, 5, 10, 15, 20]	4	Value Present	Last element
TC4	7	[1, 5, 10, 15, 20]	-1	Value Not Present	-
TC5	3	[]	-1	Empty Array	-
TC6	2	[1]	-1	Single Element Array, Value Not Present	Single element array

**P4: triangle ( int a, int b, int c )**

- **Functionality:** Classifies a triangle based on the side lengths **a**, **b**, and **c**. It returns equilateral, isosceles, scalene, or invalid.

**Equivalence Classes:**

1. **Equilateral Triangle:** All sides are equal (**a = b = c**).
2. **Isosceles Triangle:** Two sides are equal (**a = b, a ≠ c**).
3. **Scalene Triangle:** No sides are equal (**a ≠ b ≠ c**).
4. **Invalid Triangle:** Triangle inequality is violated (**a + b ≤ c** or similar).

**Boundary Conditions:**

1. **Invalid sides:** Check boundary conditions where sides sum exactly to or just greater than the third side.
2. **Edge cases for equilateral and isosceles:** **a = b = c**, **a = b**.

**Test Cases:**

Test Case	a	b	c	Expected Output	Equivalence Class	Boundary Condition
TC1	3	3	3	Equilateral	Equilateral Triangle	$a = b = c$
TC2	3	3	5	Isosceles	Isosceles Triangle	$a = b$
TC3	3	4	5	Scalene	Scalene Triangle	$a \neq b \neq c$
TC4	1	2	3	Invalid	Invalid Triangle ( $a + b = c$ )	$a + b = c$

**P5: prefix ( string s1, string s2 )**

- **Functionality:** Checks if **s1** is a prefix of **s2**. Returns **true** if **s1** is a prefix of **s2**, otherwise returns **false**.

**Equivalence Classes:**

1. **s1 is a prefix of s2:** **s1** appears at the start of **s2**.
2. **s1 is not a prefix of s2:** **s1** does not appear at the start of **s2**.
3. **s1 is longer than s2:** **s1.length() > s2.length()**.
4. **s1 and s2 are equal:** Both strings are exactly the same.

**Boundary Conditions:**

1. **Empty strings:** One or both strings are empty.
2. **Single-character strings:** Check with strings of length 1.

**Test Cases:**

Test Case	s1	s2	Expected Output	Equivalence Class	Boundary Condition
TC1	"abc"	"abcdef"	true	s1 is a prefix of s2	-
TC2	"xyz"	"abcdef"	false	s1 is not a prefix of s2	-
TC3	"abcdefg"	"abc"	false	s1 is longer than s2	-

TC4	""	"abcdef"	true	s1 is empty	Empty s1
TC5	"a"	"a"	true	s1 and s2 are equal	Single character strings equal

## P6: Floating-point Triangle Classification

- **Functionality:** This program classifies a triangle with floating-point side lengths and also determines if it is a **right-angled triangle** (based on the Pythagorean theorem).

### Equivalence Classes:

1. **Equilateral Triangle:**  $A = B = C$ .
2. **Isosceles Triangle:**  $A = B \neq C$ .
3. **Scalene Triangle:**  $A \neq B \neq C$ .
4. **Right-angled Triangle:**  $A^2 + B^2 = C^2$  (or similar).
5. **Invalid Triangle:** Triangle inequality is violated ( $A + B \leq C$ ).

### Boundary Conditions:

1. **Check boundaries for triangle inequality:**  $A + B = C$ .
2. **Right-angle triangle condition:** Test edge cases where  $A^2 + B^2 = C^2$ .

### Test Cases:

Test Case	A	B	C	Expected Output	Equivalence Class	Boundary Condition
TC1	3.0	3.0	3.0	Equilateral	Equilateral Triangle	-
TC2	3.0	3.0	4.0	Isosceles	Isosceles Triangle	-
TC3	3.0	4.0	5.0	Right-angled	Right-angled Triangle	$A^2 + B^2 = C^2$
TC4	2.0	3.0	4.0	Scalene	Scalene Triangle	-
TC5	1.0	2.0	3.0	Invalid	Invalid Triangle ( $A + B = C$ )	$A + B = C$