Lab-8_Software Testing
Lab Session - Functional Testing (Black-Box)
(ID-202201054)

Q.1: Equivalence Class Test Cases for Previous Date Program

1. Equivalence Partitioning (EP)

Equivalence Partitioning is a testing technique that divides input data into valid and invalid partitions to reduce the number of test cases. Here's how we can classify the inputs for the previous date program:

Equivalence Partitioning (EP) Test Cases

Input (day, month, year)	Expected Outcome	Type
(1, 1, 2000)	(31, 12, 1999)	Valid
(15, 5, 2015)	(14, 5, 2015)	Valid
(29, 2, 2004)	(28, 2, 2004)	Valid (Leap Year)
(1, 1, 2015)	(31, 12, 2014)	Valid
(0, 1, 2000)	Error message (Invalid date)	Invalid
(32, 1, 2000)	Error message (Invalid date)	Invalid

(1, 0, 2000)	Error message (Invalid date)	Invalid
(1, 1, 1899)	Error message (Invalid date)	Invalid
(31, 2, 2000)	Error message (Invalid date)	Invalid
(30, 2, 2001)	Error message (Invalid date)	Invalid

Boundary Value Analysis (BVA) Test Cases

Input (day, month, year)	Expected Outcome	Type
(1, 1, 1900)	(31, 12, 1899)	Boundary
(31, 12, 2015)	(30, 12, 2015)	Boundary
(29, 2, 2004)	(28, 2, 2004)	Boundary (Leap Year)
(29, 2, 2001)	Error message (Invalid date)	Boundary (Non-Leap Year)
(28, 2, 2001)	(1, 3, 2001)	Boundary

Test Suite Summary

Test Case No.	Input (day, month, year)	Expected Outcome	Type
1	(1, 1, 2000)	(31, 12, 1999)	Valid
2	(15, 5, 2015)	(14, 5, 2015)	Valid
3	(29, 2, 2004)	(28, 2, 2004)	Valid (Leap Year)

4	(1, 1, 2015)	(31, 12, 2014)	Valid
5	(0, 1, 2000)	Error message (Invalid date)	Invalid
6	(32, 1, 2000)	Error message (Invalid date)	Invalid
7	(1, 0, 2000)	Error message (Invalid date)	Invalid
8	(1, 1, 1899)	Error message (Invalid date)	Invalid
9	(31, 2, 2000)	Error message (Invalid date)	Invalid
10	(30, 2, 2001)	Error message (Invalid date)	Invalid
11	(1, 1, 1900)	(31, 12, 1899)	Boundary
12	(31, 12, 2015)	(30, 12, 2015)	Boundary
13	(29, 2, 2001)	Error message (Invalid date)	Boundary
14	(28, 2, 2001)	(1, 3, 2001)	Boundary

Program to Determine Previous Date

Here's the modified Python program implementing the logic to determine the previous date:

```
def is_leap_year(year):
    """Check if a year is a leap year."""
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

def get_previous_date(day, month, year):
    """Return the previous date given a day, month, and year."""
    # Validate year
    if year < 1900 or year > 2015:
        return "Error: Invalid date"

    # Validate month
    if month < 1 or month > 12:
        return "Error: Invalid date"
```

```
    # Days in each month (adjust for leap year in February)
    days_in_month = [31, 28 + is_leap_year(year), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    # Validate day
    if day < 1 or day > days_in_month[month - 1]:
        return "Error: Invalid date"

    # Calculate previous date
    if day > 1:
        return (day - 1, month, year)
    else:
        if month == 1:
            return (31, 12, year - 1)
        else:
            return (days_in_month[month - 2], month - 1, year)

# Test Cases Execution
test_cases = [
    (1, 1, 2000),
    (15, 5, 2015),
    (29, 2, 2004),
    (1, 1, 2015),
    (0, 1, 2000),
    (32, 1, 2000),
    (1, 0, 2000),
    (1, 1, 1899),
    (31, 2, 2000),
    (30, 2, 2001),
    (1, 1, 1900),
    (31, 12, 2015),
    (29, 2, 2001),
    (28, 2, 2001),
]

# Output Results
for day, month, year in test_cases:
    result = get_previous_date(day, month, year)
    print(f"Input: ({day}, {month}, {year}) => Previous Date: {result}")
```

Program Analysis

Programs Overview:The assignment contains six programs (P1-P6) implementing various algorithms:

1. Linear Search Implementation
2. Count Item Occurrence
3. Binary Search in Sorted Array
4. Triangle Classification
5. String Prefix Checker
6. Enhanced Triangle Classification with Test Cases

Let's analyze each program and develop test cases for P6 in detail. Detailed Analysis:

P1: Linear Search:

```
int l inear Search( int v , int a[])
```

Purpose: Finds first occurrence of value v in array a Return Value: Index of first occurrence or -1 if not found

Time Complexity: O(n) Key Characteristics: Sequential search, no preprocessing require

P2: Count Item:

```
int count I tem( int v , int a[])
```

Purpose:Counts occurrences of value v in array a Return Value: Total count of Occurrences.

Time Complexity: O(n)

P3: Binary Search:

```
int binary Search( int v, int a[])
```

Purpose: Efficiently finds value in sorted array

Prerequisite: Array must be sorted

Time Complexity: O(log n)

P4: Triangle Classification:

```
int triangle( int a, int b, int c)
```

Purpose: Classifies triangle type based on side lengths Return Values:

EQUILATERAL(0) ISOSCELES(1) SCALENE(2) INVALID(3)

P5: String Prefix Check:

```
boolean prefix( String s 1 , String s 2 )
```

Purpose: Checks if s1 is prefix of s2
Return Value: Boolean indicating prefix status
Key Characteristics: String comparison with length check

P6:

a) Equivalence Classes (EP)

Class	Test Case	Input Values	Expected Output	Explanation
Valid Equilateral	Equilateral	(3.0, 3.0, 3.0)	Equilateral	All three sides are equal.
Valid Isosceles	Isosceles	(5.0, 5.0, 8.0)	Isosceles	Two sides are equal, and A + B > C.
Valid Scalene	Scalene	(6.0, 7.0, 8.0)	Scalene	All sides are different, and the sum of any two sides is greater than the third.
Valid Right-Angled	Right-angled	(3.0, 4.0, 5.0)	Right-angled	Pythagorean triple ($A^2 + B^2 = C^2$).
Invalid - Sum Violation	Invalid Sum	(5.0, 3.0, 8.1)	Invalid	The sum of two sides is not greater than the third (5.0 + 3.0 ≤ 8.1).

Invalid - Non-Positive Values	Zero-length Side	(0, 5, 5)	Invalid	One side has zero length.
	Negative-length Side	(-3, 4, 5)	Invalid	One side is negative.

b) Test Cases for Identified Equivalence Classes

Equivalence Class	Test Case	Input Values	Expected Output	Reasoning
Equilateral	EP Equilateral	(7.0, 7.0, 7.0)	Equilateral	Ensures the function correctly identifies equal sides.
Isosceles	EP Isosceles	(10.0, 10.0, 15.0)	Isosceles	Verifies two equal sides with the third being different.
Scalene	EP Scalene	(5.0, 6.0, 7.0)	Scalene	All sides are distinct and the sum of any two sides is greater than the third.
Right-Angled	EP Right-angled	(5.0, 12.0, 13.0)	Right-angled	Checks for right-angle correctness using known Pythagorean triple.
Invalid - Non-Triangle	EP Invalid	(2.0, 2.0, 5.0)	Invalid	The sum of any two sides should be greater than the third, but it isn't.

c) Scalene Triangle Boundary Conditions

Boundary Condition	Test Case	Input Values	Expected Output	Reasoning
A + B > C	Boundary Scalene	(2.0, 3.0, 4.9)	Scalene	Just valid scalene triangle, checks boundary condition where A + B > C.
	Edge Scalene	(2.0, 3.0, 5.0)	Scalene	At the edge, A + B equals C.

d) Isosceles Triangle Boundary Conditions

Boundary Condition	Test Case	Input Values	Expected Output	Reasoning
A = C	Boundary Isosceles	(7.0, 7.0, 12.9)	Isosceles	Valid isosceles triangle close to the boundary.
	Edge Isosceles	(7.0, 7.0, 13.0)	Isosceles	A + C equals B at the boundary condition.

e) Equilateral Triangle Boundary Conditions

Boundary Condition	Test Case	Input Values	Expected Output	Reasoning
A = B = C	Boundary Equilateral	(5.0001, 5.0001, 5.0001)	Equilateral	Close values to check the precision for equilateral identification.
	Edge Equilateral	(5.0, 5.0, 5.0)	Equilateral	Exact equality at the boundary condition.

f) Right-Angle Triangle Boundary Conditions

Boundary Condition	Test Case	Input Values	Expected Output	Reasoning
$A^2 + B^2 = C^2$	Boundary Right-angled	(3.0, 4.0, 5.0)	Right-angled	Classic Pythagorean triple, verifying the boundary.
	Edge Right-angled	(6.0, 8.0, 10.0)	Right-angled	A larger triple at the boundary of right-angle detection.

g) Non-Triangle Boundary Conditions

Boundary Condition	Test Case	Input Values	Expected Output	Reasoning
A + B <= C	Boundary Invalid	(5.0, 5.0, 10.0)	Invalid	Sum of two sides is exactly equal to the third, making it invalid.
	Edge Invalid	(5.0, 5.0, 10.1)	Invalid	Sum of two sides is less than the third.

h) Non-Positive Inputs

Condition	Test Case	Input Values	Expected Output	Reasoning
Zero or Negative Length	Zero Side	(0, 4, 5)	Invalid	One side is zero, making it an invalid triangle.
	Negative Side	(-1, 5, 6)	Invalid	Negative side value, invalid triangle.

Modified Code:

```
#include <iostream>
#include <algorithm>
#include <cmath>

// Triangle type constants
const int EQUILATERAL = 0;
const int ISOSCELES = 1;
const int SCALENE = 2;
const int INVALID = 3;
const int RIGHT_ANGLED = 4;

int triangle(double a, double b, double c) {
    // Sort sides in non-decreasing order
    double sides[] = {a, b, c};
    std::sort(sides, sides + 3);
    a = sides[0];
    b = sides[1];
    c = sides[2];

    // Check if any side is non-positive or violates the triangle inequality
    if (a <= 0 || b <= 0 || c <= 0 || (a + b <= c)) {
        return INVALID;
    }

    // Check for an equilateral triangle
    if (a == b && b == c) {
        return EQUILATERAL;
    }

    // Check for an isosceles triangle
    if (a == b || b == c) {
        return ISOSCELES;
    }
```

```
    // Check if it's a right-angled triangle using Pythagorean theorem
    if (std::abs((a * a) + (b * b) - (c * c)) < 1e-6) {
        return RIGHT_ANGLED;
    }

    // If none of the above, it's a scalene triangle
    return SCALENE;
}

int main() {
    // Test the triangle classification function
    double a, b, c;

    // Example input
    std::cout << "Enter three sides of a triangle: ";
    std::cin >> a >> b >> c;

    int result = triangle(a, b, c);

    switch (result) {
        case EQUILATERAL:
            std::cout << "The triangle is Equilateral.\n";
            break;
        case ISOSCELES:
            std::cout << "The triangle is Isosceles.\n";
            break;
        case SCALENE:
            std::cout << "The triangle is Scalene.\n";
            break;
        case RIGHT_ANGLED:
            std::cout << "The triangle is Right-Angled.\n";
            break;
        case INVALID:
        default:
            std::cout << "The input does not form a valid triangle.\n";
            break;
    }

    return 0;
}
```