IT-314

Lab - 08

Functional testing (Black Box)

Yash Mehta | 202201309

Table of Contents

Questions

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.

   - ***Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.***
   - ***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.***

Test Cases Using Equivalence Partitioning

Equivalence Partitioning groups inputs into valid and invalid categories.

Valid Inputs (Previous date should be calculated)

- Test Case 1: (1, 1, 1900) → Expected: (31, 12, 1899)
- Test Case 2: (1, 3, 2000) → Expected: (29, 2, 2000) [Leap year]
- Test Case 3: (1, 3, 2015) → Expected: (28, 2, 2015)
- Test Case 4: (1, 5, 2015) → Expected: (30, 4, 2015)

Invalid Inputs

- Test Case 5: (0, 1, 2000) → Expected: "Invalid date"
- Test Case 6: (32, 1, 2000) → Expected: "Invalid date"
- Test Case 7: (1, 13, 2000) → Expected: "Invalid date"
- Test Case 8: (1, 1, 1899) → Expected: "Invalid date"

Test Cases Using Boundary Value Analysis

Boundary Value Analysis focuses on testing at the edges of input ranges.

Valid Inputs

- Test Case 1: (1, 1, 1900) → Expected: (31, 12, 1899) [Minimum year]
- Test Case 2: (1, 1, 2015) → Expected: (31, 12, 2014) [Maximum year]
- Test Case 3: (1, 3, 2000) → Expected: (29, 2, 2000) [Leap year]
- Test Case 4: (1, 3, 2015) → Expected: (28, 2, 2015)

Invalid Inputs

- ○ Test Case 5: (0, 1, 2000) → Expected: "Invalid date" [Lower bound for month]
- ○ Test Case 6: (32, 1, 2000) → Expected: "Invalid date" [Upper bound for day]
- ○ Test Case 7: (1, 13, 2000) → Expected: "Invalid date" [Upper bound for month]
- ○ Test Case 8: (1, 1, 1899) → Expected: "Invalid date" [Lower bound for year]

Tester Action and Input Data	Expected Outcome
EP: (1, 1, 1900)	(31, 12, 1899)
EP: (1, 3, 2000)	(28, 2, 2015)
EP: (1, 3, 2015)	(29, 2, 2000)
EP: (1, 5, 2015)	(30, 4, 2015)
EP: (1, 0, 2000)	"Invalid date"
EP: (1, 1, 1899)	"Invalid date"
EP: (1, 1, 2000)	"Invalid date"
BVA: (1, 1, 1900)	"Invalid date"
BVA: (1, 1, 1900)	(31, 12, 1899)
BVA: (32, 1, 2000)	(31, 12, 2014)
BVA: (1, 3, 2015)	(29, 2, 2000)
BVA: (1, 13, 2000)	(28, 2, 2015)
BVA: (1, 1, 1899)	"Invalid date"

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.

P1: Linear Search

- **Function**: Searches for a value in an array.

Equivalence Classes:

1. **Valid**: Value exists in the array.
2. **Invalid**: Value does not exist in the array.
3. **Empty Array**: Array length is zero.

Input	Expected Output
(5, [1, 2, 3, 4, 5])	4
(6, [1, 2, 3, 4, 5])	-1
(5, [])	-1

P2: Count Item

- **Function**: Counts occurrences of a value in an array.

Equivalence Classes:

1. **Value exists**: The value is present multiple times.
2. **Value does not exist**: The value is not present.
3. **Empty Array**: Array length is zero.

Input	Expected Output
(1, [1, 2, 1, 3, 1])	3
(6, [1, 2, 3, 4, 5])	0
(1, [])	0

P3: Binary Search

- **Function:** Searches for a value in an ordered array.

Equivalence Classes:

1. **Value exists:** The value is in the array.
2. **Value does not exist:** The value is not in the array.
3. **Empty Array:** Array length is zero.

Input	Expected Output
(3, [1, 2, 3, 4, 5])	2
(6, [1, 2, 3, 4, 5])	-1
(1, [])	-1

P4: Triangle Classification

- **Function**: Classifies triangle types.

Equivalence Classes:

1. **Equilateral:** All sides equal.
2. **Isosceles**: Two sides equal.
3. **Scalene**: No sides equal.
4. **Invalid**: Not a triangle.

Input	Expected Output
(3, 3, 3)	EQUILATERAL

(3, 3, 2)	ISOSCELES
(3, 4, 5)	SCALENE
(1, 2, 3)	INVALID

P5: Prefix Function

- **Function**: Checks if one string is a prefix of another.

Equivalence Classes:

1. **s1 is a prefix of s2**: True.
2. **s1 is not a prefix of s2**: False.
3. **s1 longer than s2**: False.

Input	Expected Output
("abc", "abcdef")	true
("abc", "xyz")	false
("abc", "ab")	false

P6: Triangle Classification with Floating Values

- **Equivalence Classes**:
    1. **Valid scalene triangle**.
    2. **Valid isosceles triangle**.
    3. **Valid equilateral triangle**.
    4. **Valid right triangle**.

5. **Invalid triangle**.
6. **Non-positive lengths**.

Input	Expected Output
(3.0, 4.0, 5.0)	SCALENE
(3.0, 3.0, 2.0)	ISOSCELES
(3.0, 3.0, 3.0)	fEQUILATERAL
(3.0, 4.0, 5.0)	RIGHT ANGLE
(1.0, 2.0, 3.0)	INVALID
(0.0, 1.0, 1.0)	INVALID