# Software Engineering (IT 314)

**Lab 8 - Submitted By :**Heet Thakkar(202201431)

## Question 1

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

- **Valid Inputs:**
  - (1, 1, 1900) to (31, 12, 2015) - All combinations of valid days, months, and years.
- **Invalid Inputs:**
  - Day out of range: (0, 1, 2000), (32, 1, 2000)
  - Month out of range: (1, 0, 2000), (1, 13, 2000)
  - Year out of range: (1, 1, 1899), (1, 1, 2016)
  - Invalid date combinations: (29, 2, 2015) - 2015 is not a leap year.

### B. Boundary Value Analysis (BVA)

- **Valid Boundaries:**
  - (1, 1, 1900) - Minimum valid date
  - (31, 12, 2015) - Maximum valid date
  - (1, 2, 1900) - Minimum valid date for February
  - (29, 2, 2016) - Leap year valid boundary
- **Invalid Boundaries:**
  - (0, 1, 2000) - Invalid day
  - (32, 1, 2000) - Invalid day
  - (1, 0, 2000) - Invalid month
  - (1, 13, 2000) - Invalid month
  - (1, 1, 1899) - Invalid year
  - (1, 1, 2016) - Invalid year

## 2. Test Suite

### Equivalence Partitioning Test Cases

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (0, 1, 2000) | An Error message |
| (32, 1, 2000) | An Error message |
| (1, 0, 2000) | An Error message |
| (1, 13, 2000) | An Error message |
| (1, 1, 1899) | An Error message |
| (1, 1, 2016) | An Error message |
| (29, 2, 2015) | An Error message |
| (1, 1, 1900) | Previous date: 31/12/1899 |
| (1, 1, 2000) | Previous date: 31/12/1999 |
| (1, 3, 2000) | Previous date: 29/2/2000 |
| (1, 5, 2000) | Previous date: 30/4/2000 |

**Boundary Value Analysis Test Cases**

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (1, 1, 1900) | Previous date: 31/12/1899 |
| (31, 12, 2015) | Previous date: 30/12/2015 |
| (1, 2, 1900) | Previous date: 31/1/1900 |
| (29, 2, 2016) | Previous date: 28/2/2016 |
| (0, 1, 2000) | An Error message |
| (32, 1, 2000) | An Error message |
| (1, 0, 2000) | An Error message |
| (1, 13, 2000) | An Error message |
| (1, 1, 1899) | An Error message |
| (1, 1, 2016) | An Error message |

**QUESTION 2 : For all the following programs**

1. **Test Cases Identification**
2. **Program Execution with test suites**

**Program 1**

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (5, {1, 2, 3, 4}, 4) | -1 (value not found) |
| (3, {1, 2, 3, 4}, 4) | 2 (value found at index 2) |
| (1, {1, 2, 3, 4}, 4) | 0 (value found at index 0) |
| (6, {1, 2, 3, 4}, 4) | -1 (value not found) |
| (2, {1, 2, 2, 3}, 4) | 1 (first occurrence at index 1) |

### B. Boundary Value Analysis (BVA)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (1, {1}, 1) | 0 (value found at index 0) |
| (2, {1}, 1) | -1 (value not found) |
| (5, {}, 0) | -1 (empty array, value not found) |
| (10, {10}, 1) | 0 (value found at index 0) |
| (0, {0, 1, 2}, 3) | 0 (value found at index 0) |

## 2. Program Execution with Test Suites

```c
#include <stdio.h>

int linearSearch(int v, int a[], int size) {
    for (int i = 0; i < size; i++) {
        if (a[i] == v) {
            return i; // Return the index of the first occurrence
        }
    }
    return -1; // Value not found
}

void runTests() {
    // Equivalence Partitioning Tests
    printf("EP Test 1: %d (Expected: -1)\n", linearSearch(5, (int[]){1, 2, 3, 
4}, 4));
    printf("EP Test 2: %d (Expected: 2)\n", linearSearch(3, (int[]){1, 2, 3, 
4}, 4));
    printf("EP Test 3: %d (Expected: 0)\n", linearSearch(1, (int[]){1, 2, 3, 
4}, 4));
    printf("EP Test 4: %d (Expected: -1)\n", linearSearch(6, (int[]){1, 2, 3, 
4}, 4));
    printf("EP Test 5: %d (Expected: 1)\n", linearSearch(2, (int[]){1, 2, 2, 
3}, 4));

    // Boundary Value Analysis Tests
    printf("BVA Test 1: %d (Expected: 0)\n", linearSearch(1, (int[]){1}, 1));
    printf("BVA Test 2: %d (Expected: -1)\n", linearSearch(2, (int[]){1}, 1));
    printf("BVA Test 3: %d (Expected: -1)\n", linearSearch(5, (int[]){}, 0));
    printf("BVA Test 4: %d (Expected: 0)\n", linearSearch(10, (int[]){10}, 
1));
    printf("BVA Test 5: %d (Expected: 0)\n", linearSearch(0, (int[]){0, 1, 2}, 
3));
}

int main() {
    runTests();
    return 0;
}
```

**Program 2**

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (5, {1, 2, 3, 4}, 4) | 0 (value not found) |
| (3, {1, 2, 3, 3, 4}, 5) | 2 (value found twice) |
| (1, {1, 1, 1, 1, 1}, 5) | 5 (value found five times) |
| (2, {1, 3, 4}, 3) | 0 (value not found) |
| (4, {4, 4, 4, 4}, 4) | 4 (value found four times) |

### B. Boundary Value Analysis (BVA)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (1, {1}, 1) | 1 (value found once) |
| (2, {1}, 1) | 0 (value not found) |
| (5, {}, 0) | 0 (empty array, value not found) |
| (0, {0}, 1) | 1 (value found once) |
| (10, {10, 10, 10, 10, 10}, 5) | 5 (value found five times) |

## 2. Program Execution with Test Suites

### Code Implementation

```c
#include <stdio.h>

int countItem(int v, int a[], int size) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (a[i] == v) {
            count++;
        }
    }
    return count;
}

void runCountItemTests() {
    // Equivalence Partitioning Tests
    printf("countItem EP Test 1: %d (Expected: 0)\n", countItem(5, (int[]){1, 2, 3,
4}, 4));
    printf("countItem EP Test 2: %d (Expected: 2)\n", countItem(3, (int[]){1, 2, 3, 3,
4}, 5));
    printf("countItem EP Test 3: %d (Expected: 5)\n", countItem(1, (int[]){1, 1, 1, 1,
1}, 5));
    printf("countItem EP Test 4: %d (Expected: 0)\n", countItem(2, (int[]){1, 3, 4},
3));
    printf("countItem EP Test 5: %d (Expected: 4)\n", countItem(4, (int[]){4, 4, 4,
4}, 4));

    // Boundary Value Analysis Tests
    printf("countItem BVA Test 1: %d (Expected: 1)\n", countItem(1, (int[]){1}, 1));
    printf("countItem BVA Test 2: %d (Expected: 0)\n", countItem(2, (int[]){1}, 1));
    printf("countItem BVA Test 3: %d (Expected: 0)\n", countItem(5, (int[]){}, 0));
    printf("countItem BVA Test 4: %d (Expected: 1)\n", countItem(0, (int[]){0}, 1));
    printf("countItem BVA Test 5: %d (Expected: 5)\n", countItem(10, (int[]){10, 10,
10, 10, 10}, 5));
}

int main() {
    runCountItemTests();
    return 0;
}
```

**Program 3**

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (5, {1, 2, 3, 4}, 4) | -1 (value not found) |
| (3, {1, 2, 3, 4, 5}, 5) | 2 (value found at index 2) |
| (1, {1, 2, 3, 4, 5}, 5) | 0 (value found at index 0) |
| (6, {1, 2, 3, 4, 5}, 5) | -1 (value not found) |
| (4, {1, 2, 3, 4, 5}, 5) | 3 (value found at index 3) |

### B. Boundary Value Analysis (BVA)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| (1, {1}, 1) | 0 (value found at index 0) |
| (2, {1}, 1) | -1 (value not found) |
| (5, {}, 0) | -1 (empty array, value not found) |
| (10, {10}, 1) | 0 (value found at index 0) |
| (0, {0, 1, 2, 3}, 4) | 0 (value found at index 0) |

## 2. Program Execution with Test Suites

### Code Implementation

```c
#include <stdio.h>
int binarySearch(int v, int a[], int size) {
    int lo = 0, hi = size - 1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (v == a[mid]) {
            return mid; // Found
        } else if (v < a[mid]) {
            hi = mid - 1;
        } else {
            lo = mid + 1;
        }
    }
    return -1; // Not found
}
void runBinarySearchTests() {
    // Equivalence Partitioning Tests
    printf("binarySearch EP Test 1: %d (Expected: -1)\n", binarySearch(5, (int[]){1,
2, 3, 4}, 4));
    printf("binarySearch EP Test 2: %d (Expected: 2)\n", binarySearch(3, (int[]){1, 2,
3, 4, 5}, 5));
    printf("binarySearch EP Test 3: %d (Expected: 0)\n", binarySearch(1, (int[]){1, 2,
3, 4, 5}, 5));
    printf("binarySearch EP Test 4: %d (Expected: -1)\n", binarySearch(6, (int[]){1,
2, 3, 4, 5}, 5));
    printf("binarySearch EP Test 5: %d (Expected: 3)\n", binarySearch(4, (int[]){1, 2,
3, 4, 5}, 5));

    // Boundary Value Analysis Tests
    printf("binarySearch BVA Test 1: %d (Expected: 0)\n", binarySearch(1, (int[]){1},
1));
    printf("binarySearch BVA Test 2: %d (Expected: -1)\n", binarySearch(2, (int[]){1},
1));
    printf("binarySearch BVA Test 3: %d (Expected: -1)\n", binarySearch(5, (int[]){},
0));
    printf("binarySearch BVA Test 4: %d (Expected: 0)\n", binarySearch(10,
(int[]){10}, 1));
    printf("binarySearch BVA Test 5: %d (Expected: 0)\n", binarySearch(0, (int[]){0,
1, 2, 3}, 4));
}
int main() {
    runBinarySearchTests();
    return 0;
}
```

Program 4

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| `triangle(3, 3, 3)` | EQUILATERAL (0) |
| `triangle(3, 4, 3)` | ISOSCELES (1) |
| `triangle(3, 4, 5)` | SCALENE (2) |
| `triangle(1, 1, 3)` | INVALID (3) |
| `triangle(0, 0, 0)` | INVALID (3) |
| `triangle(2, 2, 3)` | ISOSCELES (1) |
| `triangle(5, 5, 10)` | INVALID (3) |

### B. Boundary Value Analysis (BVA)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| `triangle(1, 1, 1)` | EQUILATERAL (0) |
| `triangle(1, 1, 2)` | ISOSCELES (1) |
| `triangle(1, 2, 3)` | INVALID (3) |
| `triangle(2, 2, 3)` | ISOSCELES (1) |
| `triangle(3, 4, 5)` | SCALENE (2) |
| `triangle(0, 1, 1)` | INVALID (3) |
| `triangle(1, 1, 0)` | INVALID (3) |
| `triangle(-1, -1, -1)` | INVALID (3) |

## 2. Program Execution with Test Suites

### Code Implementation

```c
#include <stdio.h>

#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3

int triangle(int a, int b, int c) {
    // Check for invalid triangle
    if (a >= b + c || b >= a + c || c >= a + b)
        return INVALID;
    // Check for equilateral triangle
    if (a == b && b == c)
        return EQUILATERAL;
    // Check for isosceles triangle
    if (a == b || a == c || b == c)
        return ISOSCELES;
    // Otherwise, it is scalene
    return SCALENE;
}

void runTriangleTests() {
    // Equivalence Partitioning Tests
    printf("Triangle EP Test 1: %d (Expected: %d)\n", triangle(3, 3, 3), EQUILATERAL);
    printf("Triangle EP Test 2: %d (Expected: %d)\n", triangle(3, 4, 3), ISOSCELES);
    printf("Triangle EP Test 3: %d (Expected: %d)\n", triangle(3, 4, 5), SCALENE);
    printf("Triangle EP Test 4: %d (Expected: %d)\n", triangle(1, 1, 3), INVALID);

    // Boundary Value Analysis Tests
    printf("Triangle BVA Test 1: %d (Expected: %d)\n", triangle(0, 0, 0), INVALID);
    printf("Triangle BVA Test 2: %d (Expected: %d)\n", triangle(1, 1, 1),
EQUILATERAL);
    printf("Triangle BVA Test 3: %d (Expected: %d)\n", triangle(1, 1, 2), ISOSCELES);
    printf("Triangle BVA Test 4: %d (Expected: %d)\n", triangle(1, 2, 3), INVALID);
    printf("Triangle BVA Test 5: %d (Expected: %d)\n", triangle(2, 2, 3), ISOSCELES);
}

int main() {
    runTriangleTests();
    return 0;
}
```

**Program 5**

## 1. Test Cases Identification

### A. Equivalence Partitioning (EP)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| `prefix("pre", "prefix")` | TRUE |
| `prefix("test", "testing")` | TRUE |
| `prefix("hello", "world")` | FALSE |
| `prefix("abc", "ab")` | FALSE |
| `prefix("test", "test123")` | TRUE |
| `prefix("test123", "test")` | FALSE |

### B. Boundary Value Analysis (BVA)

| Tester Action and Input Data | Expected Outcome |
|---|---|
| `prefix("", "any")` | TRUE |
| `prefix("any", "")` | FALSE |
| `prefix("", "")` | TRUE |
| `prefix("a", "a")` | TRUE |
| `prefix("abc", "abcd")` | TRUE |
| `prefix("abc", "ab")` | FALSE |
| `prefix("abcd", "abcde")` | TRUE |

## 2. Program Execution with Test Suites

### Code Implementation

```
public class PrefixChecker {

    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;
    }

    public static void runPrefixTests() {
        // Equivalence Partitioning Tests
        System.out.println("Prefix EP Test 1: " + prefix("pre", "prefix") + "
(Expected: true)");
        System.out.println("Prefix EP Test 2: " + prefix("test", "testing") + "
(Expected: true)");
        System.out.println("Prefix EP Test 3: " + prefix("hello", "world") + "
(Expected: false)");
        System.out.println("Prefix EP Test 4: " + prefix("abc", "ab") + " (Expected:
false)");

        // Boundary Value Analysis Tests
        System.out.println("Prefix BVA Test 1: " + prefix("", "any") + " (Expected:
true)"); // empty prefix
        System.out.println("Prefix BVA Test 2: " + prefix("any", "") + " (Expected:
false)"); // longer prefix
        System.out.println("Prefix BVA Test 3: " + prefix("", "") + " (Expected:
true)"); // both empty
        System.out.println("Prefix BVA Test 4: " + prefix("a", "a") + " (Expected:
true)"); // single char equal
        System.out.println("Prefix BVA Test 5: " + prefix("abc", "abcd") + "
(Expected: true)"); // exact prefix
    }

    public static void main(String[] args) {
        runPrefixTests();
    }
}
```

**Program 6**

## a) Identify the Equivalence Classes

1. **Equivalence Classes for Triangle Types**:
    - **Equilateral Triangle**: A = B = C
    - **Isosceles Triangle**: A = B ≠ C or A = C ≠ B or B = C ≠ A
    - **Scalene Triangle**: A ≠ B ≠ C
    - **Right-Angled Triangle**: A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (assuming C is the longest side)
    - **Invalid Triangle**: A + B ≤ C or A + C ≤ B or B + C ≤ A
    - **Non-Triangle (Non-positive lengths)**: A ≤ 0 or B ≤ 0 or C ≤ 0

## b) Identify Test Cases to Cover the Identified Equivalence Classes

| Test Case | Input Values (A, B, C) | Expected Outcome | Equivalence Class Covered |
|---|---|---|---|
| Test Case 1 | (3.0, 3.0, 3.0) | "Equilateral Triangle" | Equilateral Triangle |
| Test Case 2 | (5.0, 5.0, 3.0) | "Isosceles Triangle" | Isosceles Triangle |
| Test Case 3 | (4.0, 5.0, 6.0) | "Scalene Triangle" | Scalene Triangle |
| Test Case 4 | (3.0, 4.0, 5.0) | "Right-Angled Triangle" | Right-Angled Triangle |
| Test Case 5 | (1.0, 2.0, 3.0) | "Invalid Triangle" | Invalid Triangle |
| Test Case 6 | (1.0, 2.0, 0.0) | "Non-Triangle" | Non-Triangle (Non-positive lengths) |
| Test Case 7 | (0.0, 5.0, 5.0) | "Non-Triangle" | Non-Triangle (Non-positive lengths) |

### c) Boundary Condition for Scalene Triangle (A + B > C)

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (3.0, 4.0, 5.0) | "Scalene Triangle" |
| Test Case 2 | (2.0, 3.0, 4.0) | "Scalene Triangle" |
| Test Case 3 | (2.0, 2.0, 3.99999) | "Scalene Triangle" |
| Test Case 4 | (3.0, 4.0, 7.0) | "Invalid Triangle" |

### d) Boundary Condition for Isosceles Triangle (A = C)

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (5.0, 5.0, 3.0) | "Isosceles Triangle" |
| Test Case 2 | (5.0, 3.0, 5.0) | "Isosceles Triangle" |
| Test Case 3 | (5.0, 5.0, 5.0) | "Equilateral Triangle" |
| Test Case 4 | (0.0, 5.0, 0.0) | "Non-Triangle" |

### e) Boundary Condition for Equilateral Triangle (A = B = C)

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (3.0, 3.0, 3.0) | "Equilateral Triangle" |
| Test Case 2 | (0.0, 0.0, 0.0) | "Non-Triangle" |
| Test Case 3 | (5.0, 5.0, 5.0) | "Equilateral Triangle" |

### f) Boundary Condition for Right-Angled Triangle ($A^2 + B^2 = C^2$)

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (3.0, 4.0, 5.0) | "Right-Angled Triangle" |
| Test Case 2 | (5.0, 12.0, 13.0) | "Right-Angled Triangle" |
| Test Case 3 | (8.0, 15.0, 17.0) | "Right-Angled Triangle" |
| Test Case 4 | (5.0, 5.0, 7.0) | "Invalid Triangle" |

### g) Non-Triangle Case

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (1.0, 2.0, 3.0) | "Invalid Triangle" |
| Test Case 2 | (3.0, 1.0, 1.0) | "Invalid Triangle" |
| Test Case 3 | (5.0, 5.0, 10.0) | "Invalid Triangle" |
| Test Case 4 | (7.0, 3.0, 4.0) | "Invalid Triangle" |

### h) Non-Positive Input

| Test Case | Input Values (A, B, C) | Expected Outcome |
|---|---|---|
| Test Case 1 | (-1.0, 2.0, 3.0) | "Non-Triangle" |
| Test Case 2 | (0.0, 5.0, 5.0) | "Non-Triangle" |
| Test Case 3 | (5.0, 0.0, 5.0) | "Non-Triangle" |
| Test Case 4 | (5.0, 5.0, -1.0) | "Non-Triangle" |
| Test Case 5 | (0.0, 0.0, 0.0) | "Non-Triangle" |