IT314 Lab 8

Software Testing

Kenil sarang 202201194

Question 1:

1. **Equivalence Class Partitioning:**
   This method divides the input range into multiple classes, where each class is expected to behave in a similar way.
   - **Valid Equivalence Classes:**
     - **Day: 1≤day≤311 \leq \text{day} \leq 311≤day≤31**
     - **Month: 1≤month≤121 \leq \text{month} \leq 121≤month≤12**
     - **Year: 1900≤year≤20151900 \leq \text{year} \leq 20151900≤year≤2015**
   - **Invalid Equivalence Classes:**
     - **Day < 1 or Day > 31**
     - **Month < 1 or Month > 12**
     - **Year < 1900 or Year > 2015**
2. **Boundary Value Analysis (BVA):**
   This technique focuses on testing the values at the boundaries of input partitions.
   - **Day Boundaries:**
     - **Minimum: 1**
     - **Maximum: 31**
   - **Month Boundaries:**
     - **Minimum: 1**
     - **Maximum: 12**
   - **Year Boundaries:**
     - **Minimum: 1900**
     - **Maximum: 2015**

Input Data	Expected Outcome	Type
2, 5, 2000	Previous date: 01/05/2000	EP

1, 3, 2015	Previous date: 28/02/2015	EP
32, 5, 2000	error	EP
15, 13, 2000	error	EP
15, 10, 1899	error	EP
1, 1, 1900	error	BV
1, 2, 1900	Previous date: 31/01/1900	BV
1, 1, 2015	Previous date: 31/12/2014	BV
31, 12, 2015	Previous date: 30/12/2015	BV

Question 2:

P1:

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

```
    }
    return (-1);
}
```

1. **Equivalence Class Partitioning:**
   The input can be split into valid and invalid equivalence classes.

- **Valid Equivalence Classes:**
  - **The value v is found in the array a.**
  - **The value v is not found in the array a.**
  - **The array a has one or more elements.**
- **Invalid Equivalence Classes:**
  - **The array a is empty.**

2. **Boundary Value Analysis (BVA):**
   Test edge cases related to the size of the array a:

- **Array size boundaries:**
  - **Array with no elements (size 0).**
  - **Array with exactly one element.**
  - **Array with two elements (minimum size for non-trivial cases).**
  - **Array with many elements (large size).**
- **Boundaries for the element v:**
  - **v is located at the first position (index 0).**
  - **v is located at the last position (index a.length-1).**

- **Test Cases:**

Input Data	Expected Outcome	Type
[1, 2, 3, 4, 5], 3	Return index: 2	EP
[10, 20, 30, 40], 10	Return index: 0	EP

[9, 8, 7, 6], 5	error (not found)	EP
[], 5	error (array is empty)	EP
[3], 3	0	BV
[1, 2], 2	1	BV
[4, 5, 6], 6	2	BV

[10, 20, 30, ..., 1000], 1000	999	BV
[10, 20, 30, ..., 1000], 1	error (not found)	BV

P2:

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

Equivalence Class Partitioning:
The input can be divided into valid and invalid equivalence classes.

- **Valid Equivalence Classes**:
  - The value v occurs one or more times in the array a.
  - The value v does not appear in the array a.
  - The array a has one or more elements.
- **Invalid Equivalence Classes**:
  - The array a is empty.

Boundary Value Analysis (BVA):
Test boundary values for the size of the array a:

- **Array size boundaries**:
  - Array with no elements (size 0).
  - Array with exactly one element.
  - Array with two elements.
  - Array with a large number of elements.
- **Boundaries for v occurrences**:
  - v appears exactly once.
  - v appears multiple times.
  - v does not appear at all.

- **Test Cases:**

Input Data	Expected Outcome	Type
[1, 2, 3, 4, 5], 3	Return count: 1	EP
[10, 20, 30, 40], 10	Return count: 1	EP
[9, 8, 7, 6], 5	error (not found)	EP
[], 5	error (EMPTY)	EP

[3], 3	Return count: 1	BV
[1, 2], 2	Return count: 1	BV
[1, 2, 2, 2, 3], 2	Return count: 3	BV
[10, 20, 30, ..., 1000], 1000	Return count: 1	BV
[10, 20, 30, ..., 1000], 1	error (not found)	BV

P3:

```
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);
}
```

Equivalence Class Partitioning:
The input can be categorized into valid and invalid equivalence classes.

- **Valid Equivalence Classes:**
  - **The value v exists in the sorted array a.**
  - **The value v does not exist in the sorted array a.**
  - **The array a is non-empty and sorted.**
- **Invalid Equivalence Classes:**
  - **The array a is empty.**
  - **The array a is unsorted.**

Boundary Value Analysis (BVA):
Test boundary values for the size of the array a:

- **Array size boundaries:**
  - **An empty array (size 0).**
  - **An array with only one element.**
  - **An array with two elements.**
  - **An array with many elements.**
- **Boundaries for finding v:**
  - **v is located at the first index (index 0).**
  - **v is located at the last index (index a.length - 1).**
  - **v is not present in the array.**

- **Test Cases:**

Input Data	Expected Outcome	Type
[1, 2, 3, 4, 5], 3	Return index: 2	EP
[10, 20, 30, 40], 10	Return index: 0	EP
[9, 8, 7, 6], 5	error (array is unsorted)	EP
[5, 10, 15, 20], 25	error (not found)	EP
[], 5	error (empty)	EP
[3], 3	Return index: 0	BV
[1, 2], 2	Return index: 1	BV
[4, 5, 6], 6	Return count: 2	BV
[10, 20, 30, ..., 1000], 1000	Return index: 999	BV
[10, 20, 30, ..., 1000], 1	error (not found)	BV

P4:

```
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);

}
```

Equivalence Class Partitioning:
The input can be divided into valid and invalid equivalence classes.

- **Valid Equivalence Classes**:
    - **Equilateral triangle**: All three sides are equal.
    - **Isosceles triangle**: Two sides are equal.
    - **Scalene triangle**: All sides are of different lengths.
- **Invalid Equivalence Classes**:
    - The side lengths do not meet the triangle inequality conditions:
        - a≥b+ca b + ca≥b+c
        - b≥a+cb a + cb≥a+c
        - c≥a+bc a + bc≥a+b
    - One or more sides are non-positive (e.g., a≤0a \leq 0a≤0, b≤0b \leq 0b≤0, c≤0c \leq 0c≤0).

Boundary Value Analysis (BVA):
Test boundary values for the side lengths of the triangle:

- **Minimum positive side length**: 111.

- **Equal side lengths** for equilateral and isosceles triangles.
- **Slight differences in side lengths** for scalene and invalid triangles.
- Boundary cases for the **triangle inequality** (e.g., a+b=ca + b = ca+b=c).

- **Test Cases:**

Input Data	**Expected Outcome**	**Type**
(3, 3, 3)	Return: Equilateral(0)	EP
(5, 5, 8)	Return: Isosceles(1)	EP
(4, 5, 6)	Return: Scalene(2)	EP
(10, 5, 3)	error (triangle inequality)(3)	EP
(0, 5, 5)	error (non-positive side)(3)	EP
(1, 1, 1)	Return: Equilateral(0)	BV
(2, 2, 3)	Return: Isosceles(1)	BV
(3, 4, 5)	Return: Scalene(2)	BV
(1, 2, 3)	error (triangle inequality)(3)	BV

(-1, 2, 3)	error (invalid coordinates)(3)	BV

P5:

```
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;
}
```

1. **Equivalence Class Partitioning**:
   The inputs can be categorized into valid and invalid equivalence classes.
   - **Valid Equivalence Classes**:
     - s1 is a valid prefix of s2.
     - s1 is not a prefix of s2.
     - s1 is an empty string (an empty string is always a prefix of any string).
     - s1 is longer than s2.

2. **Boundary Value Analysis (BVA)**:
   Test boundary values for the lengths of s1 and s2:
   - Both s1 and s2 are empty strings.
   - s1 is empty and s2 is non-empty.
   - s1 consists of one character, and s2 starts with that same character.
   - s1 and s2 are of the same length and are identical.
   - s1 is longer than s2.

- **Test Cases:**

Input Data	**Expected Outcome**	**Type**
("pre", "prefix")	Return: true	EP

("fix", "prefix")	Return: false	EP
("longer", "short")	Return: false	EP
("prefix", "prefix")	Return: true	EP
("a", "abc")	Return: true	BV
("abc", "abc")	Return: true	BV
("abcdef", "abc")	Return: false	BV
("abc", "abx")	Return: false	BV

P6: Modified Triangle Classification Program

This program takes floating-point inputs representing the lengths of triangle sides and classifies the triangle as **scalene**, **isosceles**, **equilateral**, or **right-angled** based on the side lengths.

---

a) **Equivalence Class Partitioning**:

We can identify distinct equivalence classes based on triangle properties.

- **Valid Equivalence Classes**:
  - **Equilateral Triangle**: All sides are equal (A = B = C).
  - **Isosceles Triangle**: Two sides are equal (A = B or A = C or B = C).
  - **Scalene Triangle**: No sides are equal (A ≠ B ≠ C).
  - **Right-Angled Triangle**: The sides satisfy the Pythagorean theorem ($A^2 + B^2 = C^2$).
- **Invalid Equivalence Classes**:
  - The side lengths do not satisfy the triangle inequality (A + B ≤ C, A + C ≤ B, or B + C ≤ A).
  - One or more sides are non-positive (A ≤ 0, B ≤ 0, or C ≤ 0).

---

b) **Test Cases**:

Test cases should be created to cover each identified equivalence class:

1. **Equilateral Triangle**:
   - Input: A = 5, B = 5, C = 5
     Covers: Equilateral Triangle (A = B = C).
2. **Isosceles Triangle**:
   - Input: A = 5, B = 5, C = 8
     Covers: Isosceles Triangle (A = B).
3. **Scalene Triangle**:
   - Input: A = 3, B = 4, C = 5
     Covers: Scalene Triangle (A ≠ B ≠ C).
4. **Right-Angled Triangle**:

- Input: A = 3, B = 4, C = 5
  Covers: Right-Angled Triangle ($A^2 + B^2 = C^2$).

5. **Invalid Triangle - Triangle Inequality Not Satisfied**:
   - Input: A = 1, B = 2, C = 10
     Covers: Invalid Triangle ($A + B \leq C$).
6. **Invalid Triangle - Non-Positive Side**:
   - Input: A = 0, B = 3, C = 4
     Covers: Invalid Triangle ($A \leq 0$).

- **Test Cases:**

Input Data	**Expected Outcome**	**Equivalence Class**
(3.0, 3.0, 3.0)	Equilateral	Equilateral (A=B=C)
(5.0, 5.0, 8.0)	Isosceles	Isosceles (A=B, A≠C)
(3.0, 4.0, 5.0)	Right-Angled	Right-Angled ($A^2+B^2=C^2$)
(7.0, 8.0, 9.0)	Scalene	Scalene (A ≠ B ≠ C)
(1.0, 2.0, 3.0)	error	invalid
(0.0, 4.0, 5.0)	error	invalid
(-1.0, 2.0, 2.0)	error	invalid

(4.0, 4.0, 7.0)	error	invalid

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

This condition verifies that the sum of any two sides is greater than the third side.

- **Test Case 1:**
  - **Input: (3.0, 4.0, 7.0) – A + B = C (boundary value).**
  - **Expected Outcome: Invalid Triangle (fails the triangle inequality).**
- **Test Case 2:**
  - **Input: (4.0, 4.0, 7.0) – A + B = C (boundary value).**
  - **Expected Outcome: Isosceles Triangle.**

d) Boundary Condition: A = C for Isosceles Triangle

This checks whether two sides of the triangle are equal at the boundary.

- **Test Case:**
  - **Input: (5.0, 7.0, 5.0) – Two sides are equal (boundary value).**
  - **Expected Outcome: Isosceles Triangle.**

e) Boundary Condition: A = B = C for Equilateral Triangle

This verifies cases where all three sides are equal.

- **Test Case 1:**
  - **Input: (6.0, 6.0, 6.0) – All sides equal (boundary value).**
  - **Expected Outcome: Equilateral Triangle.**
- **Test Case 2:**
  - **Input: (7.0, 6.0, 6.0) – One side differs.**
  - **Expected Outcome: Not an Equilateral Triangle.**

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

This tests the Pythagorean theorem for a right-angled triangle.

- **Test Case:**
  - **Input: (3.0, 4.0, 5.0) – Classic Pythagorean triplet.**
  - **Expected Outcome: Right-Angled Triangle.**

g) Non-Triangle Case ($A + B \leq C$)

This tests the boundary where the sum of two sides is not greater than the third side.

- **Test Case:**
  - **Input: (1.0, 2.0, 3.0) – A + B = C (boundary value).**
  - **Expected Outcome: Invalid Triangle.**

h) Non-Positive Input

This tests cases where one or more sides have non-positive values.

- **Test Case 1:**
    - **Input: (0.0, 3.0, 4.0) – Zero-length side.**
    - **Expected Outcome: Invalid Triangle.**
- **Test Case 2:**
    - **Input: (-1.0, 3.0, 3.0) – Negative-length side.**
    - **Expected Outcome: Invalid Triangle.**