IT314: Software Engineering

Lab 7

Parth Agarwal(202001098)

Group: 2

12-04-2023

Black and White Box Testing:

Section: A

Input: *Day, Month, Year with valid ranges as-*

$1 \leq Month \leq 12$

$1 \leq Day \leq 31$

$1900 \leq Year \leq 2015$

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.

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

Design Boundary Value Test Cases.

Solution: Taking the **year as a Single Fault Assumption** i.e. year will be having values varying from 1900 to 2000 and others will have nominal values.

Test Cases	Month	Day	Year	Output
1	6	15	1990	14 June 1990
2	6	15	1901	14 June 1901
3	6	15	1960	14 June 1960
4	6	15	2014	14 June 2014
5	6	15	2015	14 June 2015

Taking **Day as Single Fault Assumption** i.e. Day will be having values varying from 1 to 31 and others will have nominal values.

Test Case	Month	Day	Year	Output
6	6	1	1960	31 May 1960
7	6	2	1960	1 June 1960
8	6	30	1960	29 June 1960
9	6	31	1960	Invalid day

Taking **Month as Single Fault Assumption** i.e. Month will be having values varying from 1 to 12 and others will have nominal values.

Test Case	Month	Day	Year	Output
10	1	15	1960	14 Jan 1960
11	2	15	1960	14 Feb 1960
12	11	15	1960	14 Nov 1960
13	12	15	1960	14 Dec 1960

For the n variable to be checked Maximum of 4n + 1 test case will be required. Therefore, for n = 3, the maximum test cases are-

4 × 3 + 1 =13

Code for running on the Ecclipse IDE:

```
import java.time.LocalDate;

public class PreviousDate {
   public static String getPreviousDate(int day, int month, int year) {
      LocalDate date = LocalDate.of(year, month, day);
      LocalDate previousDate = date.minusDays(1);

         if (previousDate.getYear() < 1900 || previousDate.getYear() >
         2015) {
                return "Invalid date";
         }
         else {
                 return previousDate.getDayOfMonth() + "/" +
         previousDate.getMonthValue() + "/" + previousDate.getYear();
               }

   }
}
```

In this example, we have created five test methods, each of which tests a different input scenario. The assertEquals method is used to compare the actual output of the **PreviousDate.getPreviousDate method** with the expected output. If the actual output matches the expected output, the test passes; otherwise, it fails.

Different Programs to be tested on the Ecllipse IDE:

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.

CODE:

```
public class LinearSearch {
    public static int linearSearch(int[] a, int v) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] == v) {
                return i;
            }
        }
        return -1;
    }
}
```

Test Cases:

Input: a = [2, 4, 6, 8, 10, 12], v = 8	Expected Output: 3
Input: a = [1, 3, 5, 7], v = 2	Expected Output: -1
Input: a = [], v = 1	Expected Output: -1

P2. The function countItem returns the number of times a value v appears in an array of integers a.

CODE:

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

Test Cases:

Input: a = [1, 2, 3, 4, 5], v = 3.	Expected output: 1.
Input: a = [1, 1, 1, 1, 1], v = 1.	Expected output: 5.
Input: a = [1, 2, 3, 4, 5], v = 6.	Expected output: 0.
Input: a = [], v = 3.	Expected output: 0.
Input: a = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4], v = 4.	Expected output: 4.
Input: a = [1, 1, 2, 2, 3, 3, 3], v = 3.	Expected output: 3.
Input: a = [1, 2, 3], v = 0.	Expected output: 0.
Input: a = [1, 1, 2, 2, 3, 3], v = 4.	Expected output: 0.
Input: a = [1, 1, 1, 1], v = 1.	Expected output: 4.
Input: a = [0, 0, 0, 0, 0], v = 1.	Expected output: 0.

P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

CODE:

```
public class BinarySearch {
    public static int binarySearch(int[] a, int v) {
        int left = 0;
        int right = a.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (a[mid] == v) {
                return mid;
            } else if (a[mid] < v) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1; }
}
```

Test Cases:

Input: a = [1, 3, 4, 7, 9], v = 4	Expected Output: 2
Input: a = [2, 3, 5, 8, 9], v = 2	Expected Output: 0
Input: a = [1, 3, 5, 7, 10], v = 10	Expected Output: 4
Input: a = [1, 2, 3, 5, 8], v = 6	Expected Output: -1
Input: a = [], v = 3	Expected Output: -1
Input: a = [7], v = 2	Expected Output: -1
Input: a = [5], v = 5	Expected Output: 0

P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

CODE:

```
public class Triangle {
    public static String triangle(int a, int b, int c) {
        if (a <= 0 || b <= 0 || c <= 0) {
            return "Invalid";
        } else if (a + b <= c || b + c <= a || a + c <= b) {
            return "Invalid";
        } else if (a == b && b == c) {
            return "Equilateral";
        } else if (a == b || b == c || a == c) {
            return "Isosceles";
        } else {
            return "Scalene";
        }
    }
}
```

Test Cases:

Input: triangle(3, 3, 3)	Expected Output: "Equilateral"
Input: triangle(3, 3, 4)	Expected Output: "Isosceles"
Input: triangle(3, 4, 5)	Expected Output: "Scalene"
Input: triangle(1, 2, 3)	Expected Output: "Invalid"
Input: triangle(0, 0, 0)	Expected Output: "Invalid"
Input: triangle(-1, -2, -3)	Expected Output: "Invalid"

P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

CODE:

```
public class Prefix {
    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;
    }
}
```

Test Cases:

1) Input: s1 = "hello"
   s2 = "hello world"
   Expected Output: true

2) Input: s1 = "world"
   s2 = "hello world"
   Expected Output: false

3) Input: s1 = "apple"
s2 = "app"
Expected Output: false

4) Input: s1 = "app"
s2 = "apple"
Expected Output: true

5) Input: s1 = ""
s2 = "hello"
Expected Output: true (empty string is always a prefix of any string)

6) Input: s1 = "hello"
s2 = "hello"
Expected Output: true (a string is always a prefix of itself)

P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled.
Determine the following for the above program:

CODE:

```
import java.util.Scanner;

public class Triangle {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the length of side A: ");
        float a = scanner.nextFloat();
        System.out.print("Enter the length of side B: ");
        float b = scanner.nextFloat();
        System.out.print("Enter the length of side C: ");
        float c = scanner.nextFloat();
        scanner.close();

        if (a <= 0 || b <= 0 || c <= 0) {
            System.out.println("Invalid triangle");
        } else if (a + b <= c || b + c <= a || a + c <= b) {
            System.out.println("Invalid triangle");
        } else if (a == b && b == c) {
```

```
            System.out.println("Equilateral triangle");
        } else if (a == b || b == c || a == c) {
            System.out.println("Isosceles triangle");
        } else if (a * a + b * b == c * c || b * b + c * c == a * a || a * a +
c * c == b * b) {
            System.out.println("Right angled triangle");
        } else {
            System.out.println("Scalene triangle");
        }
    }
}
```

Determine the following for the above program:

a) Identify the equivalence classes for the system

- **Invalid triangle:** A triangle is invalid if any of the side lengths is negative or zero.
- **Non-triangle:** A triangle is non-triangle if the sum of the lengths of any two sides is less than or equal to the length of the third side.
- **Scalene triangle:** A scalene triangle has three different side lengths.
- **Isosceles triangle:** An isosceles triangle has two sides of the same length and one different side.
- **Equilateral triangle**: An equilateral triangle has all three sides of the same length.
- **Right-angled triangle**: A right-angled triangle has one angle that measures 90 degrees.

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

Test cases:

- **Invalid triangle:** (-2, 3, 4), (0, 5, 7), (6, -1, 9)
- **Non-triangle:** (3, 5, 9), (6, 10, 20), (0.1, 0.2, 0.3)
- **Scalene triangle:** (3, 4, 5), (6.2, 7.3, 8.4), (2.5, 4.7, 6.8)
- **Isosceles triangle:** (4, 4, 6), (7.5, 7.5, 9.8), (12, 10, 10)
- **Equilateral triangle**: (3, 3, 3), (5.6, 5.6, 5.6), (9.2, 9.2, 9.2)
- **Right-angled triangle:** (3, 4, 5), (5, 12, 13), (7, 24, 25)

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

CASES: Scalene Triangle

1) A = 1, B = 2, C = 3 (minimum valid values for A, B, and C)
2) A = 100, B = 100, C = 199 (maximum valid values for A, B, and C)
3) A = 2, B = 2, C = 3 (boundary case where A + B = C)

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

CASES: Isosceles Triangle

1) A = 1, B = 1, C = 2 (minimum valid values for A, B, and C)
2) A = 100, B = 100, C = 199 (maximum valid values for A, B, and C)
3) A = 2, B = 2, C = 3 (boundary case where A = B < C)

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

CASES: Equilateral Triangle

1) A = 1, B = 1, C = 1 (minimum valid values for A, B, and C)
2) A = 100, B = 100, C = 100 (maximum valid values for A, B, and C)

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

CASES: Right-Angled Triangle

1) A = 3, B = 4, C = 5 (minimum valid values for A, B, and C)
2) A = 100, B = 1000, C = 1000 (maximum valid values for A, B, and C)
3) A = 5, B = 12, C = 13 (boundary case where A^2 + B^2 = C^2)

g) For the non-triangle case, identify test cases to explore the boundary.

CASES: Invalid Triangle

1) A = 0, B = 1, C = 1 (minimum invalid value for A)
2) A = 1, B = 0, C = 1 (minimum invalid value for B)
3) A = 1, B = 1, C = 0 (minimum invalid value for C)
4) A = -1, B = 1, C = 1 (negative value for A)
5) A = 1, B = -1, C = 1 (negative value for B)
6) A = 1, B = 1, C = -1 (negative value for C)
7) A = 1, B = 1, C = 3 (A + B < C, not a valid triangle)

h) For non-positive input, identify test points.

CASES:

1) A = -1, B = 3, C = 4
2) A = 3, B = -1, C = 4
3) A = 3, B = 4, C = -1
4) A = -1, B = -1, C = 4
5) A = 3, B = -1, C = -1
6) A = -1, B = 4, C = -1
7) A = -1, B = -1, C = -1

Test cases 1-3 represent cases where one of the input values is negative. Test cases 4-6 represent cases where two of the input values are negative. Test case 7 represents a case where all three input values are negative.

The expected outcome for all these test cases should be "invalid input".