IT314: Software Engineering

Kaushik Prajapati - 202201472

Lab-8: Functional Testing (Black-Box)

Q.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.

1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.
2. 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.

Let us denote month parameter as M, day parameter as D and Year parameter as Y.

Equivalence Class Partitioning

Equivalence Class	Description (Month, Day, Year)
E1	M<1, D<1, Y<1900
E2	M<1, D<1, 1900≤Y≤2015
E3	M<1, D<1, Y>2015
E4	M<1, 1≤D≤31, Y<1900
E5	M<1, 1≤D≤31, 1900≤Y≤2015
E6	M<1, 1≤D≤31, Y>2015
E7	M<1, D>31, Y<1900
E8	M<1, D>31, 1900≤Y≤2015
E9	M<1, D>31, Y>2015
E10	1≤M≤12, D<1, Y<1900
E11	1≤M≤12, D<1, 1900≤Y≤2015
E12	1≤M≤12, D<1, Y>2015
E13	1≤M≤12, 1≤D≤31, Y<1900
E14	1≤M≤12, 1≤D≤31, 1900≤Y≤2015
E15	1≤M≤12, 1≤D≤31, Y>2015
E16	1≤M≤12, D>31, Y<1900
E17	1≤M≤12, D>31, 1900≤Y≤2015
E18	1≤M≤12, D>31, Y>2015
E19	M>12, D<1, Y<1900
E20	M>12, D<1, 1900≤Y≤2015
E21	M>12, D<1, Y>2015
E22	M>12, 1≤D≤31, Y<1900
E23	M>12, 1≤D≤31, 1900≤Y≤2015
E24	M>12, 1≤D≤31, Y>2015
E25	M>12, D>31, Y<1900
E26	M>12, D>31, 1900≤Y≤2015
E27	M>12, D>31, Y>2015

Test Cases for Equivalent Class Partitioning

Sr. No	Test Case Value (Month, Day, Year)	Valid/Invalid	Equivalence Class Covered
1	(0, 0, 1899)	Invalid	E1
2	(0, 0, 2000)	Invalid	E2
3	(0, 0, 2020)	Invalid	E3
4	(0, 15, 1899)	Invalid	E4
5	(0, 15, 2000)	Invalid	E5
6	(0, 15, 2020)	Invalid	E6
7	(0, 32, 1899)	Invalid	E7
8	(0, 32, 2000)	Invalid	E8
9	(0, 32, 2020)	Invalid	E9
10	(5, 0, 1899)	Invalid	E10
11	(5, 0, 2000)	Invalid	E11
12	(5, 0, 2020)	Invalid	E12
13	(5, 15, 1899)	Invalid	E13
14	(5, 15, 2000)	Valid	E14
15	(5, 15, 2020)	Invalid	E15
16	(5, 32, 1899)	Invalid	E16
17	(5, 32, 2000)	Invalid	E17
18	(5, 32, 2020)	Invalid	E18
19	(13, 0, 1899)	Invalid	E19
20	(13, 0, 2000)	Invalid	E20
21	(13, 0, 2020)	Invalid	E21
22	(13, 15, 1899)	Invalid	E22
23	(13, 15, 2000)	Invalid	E23
24	(13, 15, 2020)	Invalid	E24
25	(13, 32, 1899)	Invalid	E25
26	(13, 32, 2000)	Invalid	E26
27	(13, 32, 2020)	Invalid	E27

Test Cases for Boundary Value Analysis

Sr. No	Test Case Value (Month, Day, Year)	Valid/Invalid	Description
1	(0, 1, 1900)	Invalid	Month below lower boundary
2	(1, 1, 1900)	Valid	First valid date
3	(1, 31, 1900)	Valid	Valid day for January
4	(1, 32, 1900)	Invalid	Day above upper boundary
5	(12, 31, 2015)	Valid	Last valid date
6	(12, 32, 2015)	Invalid	Day above upper boundary
7	(12, 1, 2016)	Invalid	Year above upper boundary
8	(12, 1, 1899)	Invalid	Year below lower boundary
9	(2, 29, 2000)	Valid	Leap year date
10	(2, 29, 2001)	Invalid	Non-leap year date
11	(2, 28, 2001)	Valid	Last valid date in non-leap year
12	(2, 28, 1900)	Valid	Last valid date in non-leap year
13	(2, 28, 2015)	Valid	Last valid date in non-leap year
14	(3, 1, 2015)	Valid	Valid date after February
15	(3, 0, 2015)	Invalid	Day below lower boundary

Program in Python

```python
def is_leap_year(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

def previous_date(day, month, year):

    if not (1 <= month <= 12) or not (1 <= day <= 31) or not (1900 <= year <= 2015):
        return "Invalid date"

    # Days in each month
    days_in_month = [31, 29 if is_leap_year(year) else 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    if day > 1:
        return (day - 1, month, year)
    else:
        month -= 1
        if month == 0:
            month, year = 12, year - 1
        return (days_in_month[month - 1], month, year)
```

Q.2 Do equivalence class partitioning and boundary value analysis for all the programs below.

Program 1: 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.

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

Equivalence Class Partitioning

Equivalence Class	Description
E1	Search value v is present in the array.
E2	Search value v is not present in the array.
E3	Array a is non-empty.
E4	Array a is empty.
E5	Array contains invalid data types (not integers).
E6	Invalid search value (e.g., null, if applicable).

Test Cases for Equivalent Class Partitioning

Test Case	Search Value (v)	Array (a)	Expected Result	Covers Equivalence Class
1	5	[1, 2, 3, 4, 5]	4	E1
2	10	[1, 2, 3, 4, 5]	-1	E2
3	1	[]	-1	E4
4	3	[3]	0	E1
5	-1	[1, 2, 3, -1]	3	E1
6	7	[]	-1	E4
7	5	[5, 5, 5]	0	E1
8	8	[1, 2, 3, 4]	-1	E2
9	null	[1, 2, 3, 4]	Error	E6
10	2	[1, 'a', 2]	Error	E5

Test Cases for Boundary Value Analysis

Test Case	Search Value (v)	Array (a)	Expected Result	Description
1	1	[1]	0	Single element array, present.
2	2	[1]	-1	Single element array, not present.
3	1	[1, 2]	0	Two elements, first is present.
4	2	[1, 2]	1	Two elements, last is present.
5	3	[1, 2]	-1	Two elements, neither is present.
6	1	[]	-1	Empty array.
7	2	[2]	0	Single element, present.
8	3	[2]	-1	Single element, not present.
9	5	[1, 2, 3, 4, 5]	4	Multiple elements, last is present.
10	0	[0, 1, 2]	0	Multiple elements, first is zero.

Program 2: The function countItem returns the number of times a value v appears in an array of integers a.

```
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

Equivalence Class	Description
E1	Search value v is present at least once in the array.
E2	Search value v is present multiple times in the array.
E3	Search value v is not present in the array.
E4	Array a is non-empty.
E5	Array a is empty.
E6	Array contains invalid data types (not integers).
E7	Invalid search value (e.g., null, if applicable).

Test Cases for Equivalent Class Partitioning

Test Case	Search Value (v)	Array (a)	Expected Result	Covers Equivalence Class
1	5	[1, 2, 3, 4, 5]	1	E1
2	5	[1, 2, 3, 4, 5, 5]	2	E2
3	10	[1, 2, 3, 4, 5]	0	E3
4	3	[]	0	E5
5	1	[1]	1	E1
6	3	[3, 3, 3]	3	E2
7	2	[]	0	E5
8	1	[1, 'a', 2]	Error	E6
9	null	[1, 2, 3, 4]	Error	E7
10	6	[1, 2, 3, 4]	0	E3

Test Cases for Boundary Value Analysis

Test Case	Search Value (v)	Array (a)	Expected Result	Description
1	1	[1]	1	Single element array, present.
2	2	[1]	0	Single element array, not present.
3	1	[1, 2]	1	Two elements, first is present.
4	2	[1, 2]	1	Two elements, last is present.
5	3	[1, 2]	0	Two elements, neither is present.
6	1	[]	0	Empty array.
7	2	[2]	1	Single element, present.
8	3	[2]	0	Single element, not present.
9	5	[1, 2, 3, 4, 5]	1	Multiple elements, last is present.
10	0	[0, 1, 2]	1	Multiple elements, zero present.

Program 3: The function countItem returns the number of times a value v appears in an array of integers a.

```
int binarySearch(int v, int a[]) {

    int lo = 0, hi = a.length-1;
    while (lo <= hi)
    {
        int 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

Equivalence Class	Description
E1	Search value v is present in the array.
E2	Search value v is the first element in the array.
E3	Search value v is the last element in the array.
E4	Search value v is present multiple times in the array.
E5	Search value v is less than the smallest element in the array.
E6	Search value v is greater than the largest element in the array.
E7	The array a is empty.
E8	The array contains invalid data types (not integers).

Test Cases for Equivalent Class Partitioning

Test Case	Search Value (v)	Array (a)	Expected Result	Covers Equivalence Class
1	5	[1, 2, 3, 4, 5]	4	E1
2	1	[1, 2, 3, 4, 5]	0	E2
3	5	[1, 2, 3, 4, 5]	4	E3
4	3	[1, 2, 3, 3, 4, 5]	2	E4
5	0	[1, 2, 3, 4, 5]	-1	E5
6	6	[1, 2, 3, 4, 5]	-1	E6
7	2	[]	-1	E7
8	a'	[1, 2, 3, 4]	Error	E8
9	4	[2, 3, 4, 4, 4, 5]	2	E4
10	7	[1, 2, 3, 4, 5]	-1	E6

Test Cases for Boundary Value Analysis

Test Case	Search Value (v)	Array (a)	Expected Result	Description
1	1	[1]	0	Single element array, present.
2	2	[1]	-1	Single element array, not present.
3	1	[1, 2]	0	Two elements, first is present.
4	2	[1, 2]	1	Two elements, last is present.
5	0	[1, 2]	-1	Two elements, neither is present.
6	1	[]	-1	Empty array.
7	2	[2]	0	Single element, present.
8	3	[2]	-1	Single element, not present.
9	5	[1, 2, 3, 4, 5]	4	Multiple elements, last is present.
10	0	[1, 2, 3]	-1	Multiple elements, zero not present.
11	4	[1, 2, 3, 4, 5]	3	Multiple elements, middle present.

Program 4: 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).

```
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

Equivalence Class	Description
E1	All three sides are equal (equilateral).
E2	Two sides are equal (isosceles).
E3	All sides are different (scalene).
E4	The lengths do not satisfy triangle inequality (invalid).
E5	One or more sides are non-positive (invalid).

Test Cases for Equivalent Class Partitioning

Test Case	Sides (a, b, c)	Expected Result	Covers Equivalence Class
1	(3, 3, 3)	0	E1
2	(3, 3, 4)	1	E2
3	(3, 4, 5)	2	E3
4	(1, 2, 3)	3	E4
5	(1, 1, 0)	3	E5
6	(5, 5, 5)	0	E1
7	(4, 4, 2)	1	E2
8	(5, 7, 9)	2	E3
9	(1, 1, 3)	3	E4
10	(0, 2, 2)	3	E5

Test Cases for Boundary Value Analysis

Test Case	Sides (a, b, c)	Expected Result	Description
1	(1, 1, 1)	0	Minimum valid equilateral triangle.
2	(2, 2, 3)	1	Minimum isosceles triangle.
3	(2, 3, 4)	2	Minimum scalene triangle.
4	(1, 2, 2)	1	Valid isosceles triangle.
5	(2, 2, 4)	3	Invalid triangle (fails inequality).
6	(0, 1, 1)	3	Invalid triangle (zero length).
7	(-1, 1, 1)	3	Invalid triangle (negative length).

8	(2, 2, 1)	1	Valid isosceles triangle.
9	(2, 1, 1)	1	Valid isosceles triangle.
10	(3, 4, 5)	2	Valid scalene triangle.
11	(3, 4, 8)	3	Invalid triangle (fails inequality).

Program 5: 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).

```java
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;
}
```

Equivalence Class Partitioning

Equivalence Class	Description
E1	s1 is equal to s2 (exact match).
E2	s1 is a prefix of s2.
E3	s1 is an empty string.
E4	s1 is longer than s2.
E5	s1 is shorter than s2 but not a prefix.

Test Cases for Equivalent Class Partitioning

Test Case	s1	s2	Expected Result	Covers Equivalence Class
1	"hello"	"hello"	TRUE	E1
2	"hello"	"hello world"	TRUE	E2
3	""	"anything"	TRUE	E3
4	"hello"	"hi"	FALSE	E4
5	"he"	"hello"	TRUE	E2
6	"world"	"hello world"	FALSE	E5
7	""	""	TRUE	E3
8	"longprefix"	"short"	FALSE	E4
9	"test"	"testing"	TRUE	E2
10	"abc"	"ab"	FALSE	E5

Test Cases for Boundary Value Analysis

Test Case	s1	s2	Expected Result	Description
1	""	"non-empty"	TRUE	Empty string as prefix.
2	"a"	"abc"	TRUE	Single character prefix.
3	"abc"	"abc"	TRUE	Exact match.
4	"abc"	"abcd"	TRUE	Valid prefix.
5	"abc"	"ab"	FALSE	s1 is longer than s2.
6	"abc"	"xyz"	FALSE	Prefix does not match.
7	"a"	"a"	TRUE	Single character match.
8	"ab"	"abc"	TRUE	s1 is a valid prefix.
9	"abcdef"	"abc"	FALSE	Longer s1 than s2.
10	""	""	TRUE	Both strings are empty.

Program 6: 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:

(a) Identify the Equivalence Classes

Valid Cases:

- E1: A valid scalene triangle (all sides are different).
- E2: A valid isosceles triangle (two sides are equal).
- E3: A valid equilateral triangle (all three sides are equal).
- E4: A valid right-angled triangle (satisfies Pythagorean theorem).
- E5: Valid triangle (sides form a triangle but not scalene, isosceles, equilateral, or right-angled).

Invalid Cases:

- E6: The sum of any two sides is not greater than the third side (not a valid triangle).
- E7: At least one side length is non-positive (zero or negative).

(b) Identify Test Cases to Cover the Equivalence Classes

Test Case	A	B	C	Expected Result	Covers Equivalence Class
1	3	4	5	Scalene	E1
2	5	5	3	Isosceles	E2
3	4	4	4	Equilateral	E3
4	3	4	5	Right-angled	E4
5	2	2	3	Isosceles	E2
6	5	5	6	Isosceles	E2
7	1	1	1	Equilateral	E3
8	3	3	3	Equilateral	E3
9	1	2	3	Invalid	E6
10	0	5	6	Invalid	E7
11	-1	2	2	Invalid	E7
12	3	4	10	Invalid	E6
13	5	12	13	Right-angled	E4

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

Test Case	A	B	C	Expected Result	Description
1	3	4	5	Scalene	Valid scalene triangle.
2	3	5	6	Scalene	Valid scalene triangle.
3	2.5	3.5	6	Invalid	A + B ≤ C (not a triangle).
4	2.9	3.1	6	Invalid	A + B = C (not a triangle).

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

Test Case	A	B	C	Expected Result	Description
1	4	4	3	Isosceles	Valid isosceles triangle.
2	5	5	7	Isosceles	Valid isosceles triangle.
3	5	5	5	Equilateral	All sides are equal.

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

Test Case	A	B	C	Expected Result	Description
1	3	3	3	Equilateral	Valid equilateral triangle.
2	2.5	2.5	2.5	Equilateral	Valid equilateral triangle.
3	1	1	1	Equilateral	Valid equilateral triangle.

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

Test Case	A	B	C	Expected Result	Description
1	3	4	5	Right-angled	Valid right-angled triangle.
2	6	8	10	Right-angled	Valid right-angled triangle.
3	5	12	13	Right-angled	Valid right-angled triangle.
4	5	5	5	Invalid	Not a right-angled triangle.

(g) Non-Triangle Case

Test Case	A	B	C	Expected Result	Description
1	1	1	3	Invalid	Not a valid triangle.
2	2	3	10	Invalid	Not a valid triangle.
3	0.5	0.5	1	Invalid	Not a valid triangle.

(h) Non-Positive Input

Test Case	A	B	C	Expected Result	Description
1	-1	2	2	Invalid	Non-positive input.
2	0	2	2	Invalid	Non-positive input.
3	1	-2	2	Invalid	Non-positive input.
4	1	2	0	Invalid	Non-positive input.