Software Engineering

Lab - 8

Name : Ragan Patel
Student ID: 202201152

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.

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

Q-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.

Equivalence Partition	Expected Outcome
12-7-2007	Previous Date
29-6-2024	Invalid(Year out of range)
19-11-2023	Invalid(Year out of range)
8-0-2001	Invalid(Month out of range)
28-3-2005	Previous Date
0-8-2014	Invalid(Date out of range)
32-2-2008	Invalid(Date out of range)
15-7-2004	Previous Date
1-2-1898	Invalid(Year out of range)
12-13-2006	Invalid(Month out of range)
15-7-2010	Previous Date

Boundary Value Analysis	**Expected Output**
24-1-2005	Previous Date
19-12-2005	Previous Date
10-22-1955	Invalid(Month out of range)
31-12-2015	Previous Date
1-1-1900	Previous Date
0-0-2009	Invalid(Month & Date Out of range)
12-4-0	Invalid(Year out of Range)
7-7-2007	Previous Date
2-2-2022	Invalid(Year out of Range)
8-9-1899	Invalid(Year out of Range)
15-7-2011	Previous Date

Q-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.

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

Equivalence Partitioning:

Input Data	**Description**	**Expected Outcome**
V = 0, a = {0,1,2,3,4,5}	V is present in array a[]	1(Valid)
V = 7, a = {0,1,2,3,4,5}	V is not present in array a[]	-1(valid)
V = 2, a = {}	Array a[] is empty	-1(valid)
V = 3, a = {0,1,3,3,4}	V is present multiple times in a[]	1 (valid)
V = 4, a = {1,'c',4,5}	Array a[] contains non-integer values	Error
V = 9, a = null	A[] is Null	Error
V = 'v', a = {1,2,3,3,4}	V is a non integer value	Error

Boundary Value Analysis:

Input Data	Description	Expected Outcome
V = 7, a = {5}	Array has only one element and v is not present	-1
V = 1, a = {}	Array is empty	-1
V = 9, a = {1,2,3,4,5}	V is not in the range of the array	-1
V = 8, a = {8}	V is present	0

P2. 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:

Input Data	Description	Expected Outcome
V = 0, a = {0,1,2,3,4,5}	V is present in array a[]	1(Valid)
V = 7, a = {0,1,2,3,4,5}	V is not present in array a[]	-1(valid)
V = 2, a = {}	Array a[] is empty	-1(valid)
V = 3, a = {0,1,3,3,4}	V is present multiple times in a[]	1 (valid)
V = 4, a = {1,'c',4,5}	Array a[] contains non-integer values	Error
V = 9, a = null	A[] is Null	Error
V = 'v', a = {1,2,3,3,4}	V is a non integer value	Error

Boundary Value Analysis:

Input Data	Description	Expected Outcome
V = 7, a = {5}	Array has only one element and v is not present	-1
V = 1, a = {}	Array is empty	-1
V = 9, a = {1,2,3,4,5}	V is not in the range of the array	-1
V = 8, a = {8}	V is present	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.
Assumption: the elements in the array a are sorted in non-decreasing order.

```
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);
}
```

Input Data	Description	Expected Outcome
V = 0, a = {0,1,2,3,4,5}	V is present in array a[]	1(Valid)
V = 7, a = {0,1,2,3,4,5}	V is not present in array a[]	-1(valid)
V = 2, a = {}	Array a[] is empty	-1(valid)

V = 3, a = {0,1,3,3,4}	V is present multiple times in a[]	1 (valid)
V = 4, a = {1,'c',4,5}	Array a[] contains non-integer values	Error
V = 9, a = null	A[] is Null	Error
V = 'v', a = {1,2,3,3,4}	V is a non integer value	Error

Boundary Value Analysis:

Input Data	**Description**	**Expected Outcome**
V = 7, a = {5}	Array has only one element and v is not present	-1
V = 1, a = {}	Array is empty	-1
V = 9, a = {1,2,3,4,5}	V is not in the range of the array	-1
V = 8, a = {8}	V is present	0
V = 8, a = {8,7}	Array has only two elements and v is present in first half	1
V = 7, a = {8,7}	Array has only two elements and v is present in 2nd half	1

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).

Equivalence Classes:

Input	Description	Expected outcome
a = 5,b =5 , c = 5	All sides are equal(equilateral triangle)	0
a = 5, b = 7, c = 7	Two sides are equal(isosceles triangle)	1
a = 3, b= 5 , c = 3	All sides are different	2
a = 4, b = 5, c = 10	Impossible lengths (invalid triangle)	3
a = 0, b = 0, c = 0	All sides are zero (invalid triangle)	3
a = 2, b = -3, c = 4	One side is negative (invalid triangle)	3
a = 7, b = 'z', c = 'w'	Side is not an integer (Error)	3

Boundary Value Analysis:

Input Data	Description	Expected output
a = 2, b = 2, c = 2	Minimum valid Equilateral Triangle	0
a = 5, b = 6, c = 7	Minimum valid Scalene Triangle	2
a = 3, b = 3, c = 4	Minimum valid Isosceles Triangle	1
a = 0, b = 4, c = 5	One Side is zero	3
a = 4, b = 2, c = 1	Impossible lengths (invalid triangle)	3
a = 0, b = 0, c = 0	All Side is zero	3

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).

```
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;
}
```

a) Identify the equivalence classes for the system:
Valid Equivalence Classes:
- **Equilateral Triangle: All sides are equal.**
- **Isosceles Triangle: Exactly two sides are equal.**
- **Scalene Triangle: All sides are different.**
- **Right-Angled Triangle: Satisfies the Pythagorean theorem (A2 + B2 = C2).**

Invalid Equivalence Classes:
● Non-Triangle: The sum of the lengths of any two sides is not greater than the third side.
● Negative Values: One or more sides have negative lengths.
● Zero Values: One or more sides are zero.

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence classes)

Input Data	Description	Covered Classes
2.0, 2.0, 2.0	All sides are equal	Equilateral Triangle
3.0, 3.0, 4.0	Two sides are equal	Isosceles Triangle
7.0, 9.0, 10.0	All sides are diffeíent	Scalene Triangle
3.0, 4.0, 5.0	Satis es the Pythagoíean theoíem	Right-Angled Triangle
1.0, 2.0, 3.0	The sum of the two shoíteí sides is equal to the longest	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Negative Values
0.0, 1.0, 1.0	One side is zero	zero Values

a) for the boundary condition A + B > C case (scalene Triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Classes
2.0, 3.0, 4.0	Valid scalene Triangle	Scalene Triangle
2.0, 2.5, 5.0	Just below boundary	Non Triangle
1.0, 1.0, 2.0	Exactly on the boundary	Non Triangle

b) for the boundary condition A = C case (isosceles Triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Classes
5.0, 5.0, 3.0	Two sides equal, valid isosceles	Isosceles Triangle
3.0, 3.0, 6.0	Just below boundary	Non Triangle
2.0, 2.0, 4.0	Exactly on the boundary	Non Triangle

c) for the boundary condition A = B = C case (Equilateral Triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Classes
3.0, 3.0, 3.0	All sides equal, valid Equilateral	Equilateral Triangle
2.0, 2.0, 2.0	Valid Equilateral	Equilateral Triangle
2.0, 2.0, 3.0	Just isosceles	Isosceles Triangle

d) for the boundary condition A2 + B2 = C2 case (right-angle Triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Classes
3.0, 4.0, 5.0	Valid right-angled Triangle	Right Angled

5.0, 12.0, 13.0	Valid right-angled Triangle	Right Angled
2.0, 2.0, 3.0	Not a Right-angled Triangle	Not Right-Angled

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

Input Data	**Description**	**Covered Classes**
1.0, 2.0, 3.0	Sum of two sides equals the thiíd	Non-Triangle
2.0, 5.0, 3.0	Valid Triangle	Scalene
10.0, 1.0, 1.0	Impossible lengths	Non-Triangle

f) for non-positive input, identify test points.

Input Data	**Description**	**Covered Classes**
0.0, 0.0, 0.0	All sides are zero	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Non-Triangle
2.0, 0.0, 2.0	One side is zero	Non-Triangle