Software Engineering (IT314)

Lab 08

Manan Patel
(202201310)

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) – speci c 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 identi ed 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 identi ed expected outcome (mentioned by you) is correct or not.

Equivalence Partition (Data Inputs)	Expected Outcome
15-6-2005	Previous date
10-0-2000	Invalid (Month out of range)
5-15-2010	Invalid (Month out of range)
31-12-2015	Previous date
30-4-2010	Previous date
28-2-2001	Previous date
0-10-1999	Invalid (Date out of range)
35-1-2010	Invalid (Date out of range)
1-1-1900	Previous date
10-10-1896	Invalid (Year out of range)
15-7-2019	Invalid (Year out of range)

Boundary Value Analysis (Input Data)	**Expected Outcome**
15-1-2000	Previous date
31-12-2015	Previous date
10-0-2000	Invalid (Month out of range)
5-13-2010	Invalid (Month out of range)
1-5-2010	Previous date
31-12-2015	Previous date
30-4-2010	Previous date
29-2-2004	Previous date
28-2-2001	Previous date
0-3-2000	Invalid (Date out of range)
32-1-2010	Invalid (Date out of range)
1-1-1900	Previous date
15-6-2015	Previous date
10-10-1899	Invalid (Year out of range)
15-7-2016	Invalid (Year out of range)

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 rst 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 = 3, a = {1, 2, 3, 4, 5}	v is present in array a[]	2 (valid)
v = 6, a = {1, 2, 3, 4, 5}	v is not present in array a[]	-1 (valid)
v = 1, a = {}	Array a[] is empty	-1 (valid)
v = 2, a = {1, 2, 2, 3, 4}	v is present multiple times in a[]	1 (valid)
v = 5, a = null	a[] is null	Error
v = 3, a = {1, 'a', 3, 4}	Array a[] contains non-integer values	Error
v = 'b', a = {1, 2, 3, 4, 5}	v is a non-integer value	Error

Boundary Value Analysis:

Input Data	Description	Expected Outcome
v = 5, a = {5}	Array has only one element and v is present	0
v = 3, a = {5}	Array has only one element and v is not present	-1
v = 10, a = {1, 2, 3, 4, 5}	v is not in the array but outside the range	-1
v = 5, a = {}	Empty array	-1

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 = 3, a = {1, 2, 3, 4, 5}	v is present in array a[]	1 (valid)
v = 6, a = {1, 2, 3, 4, 5}	v is not present in array a[]	0 (valid)

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

Boundary Value Analysis:

Input Data	**Description**	**Expected Outcome**
v = 5, a = {5}	Array has only one element and v is present	1
v = 3, a = {5}	Array has only one element and v is not present	0
v = 10, a = {1, 2, 3, 4, 5}	v is not in the array but outside the range	0
v = 5, a = {}	Empty array	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);
}
```

Equivalence class:

Input Data	**Description**	**Expected Outcome**
v = 3, a = {1, 2, 3, 4, 5}	v is present in array a[]	2 (valid)
v = 6, a = {1, 2, 3, 4, 5}	v is not present in array a[]	-1 (valid)
v = 1, a = {}	Array a[] is empty	-1 (valid)

v = 0, a = {1, 2, 3, 4, 5}	V is less than the smallest element	0 (valid)
v = 10, a = {1, 2, 3, 4, 5}	V is greater than the largest element	0 (valid)
v = 5, a = null	a[] is null	Error
v = 3, a = {1, 'a', 3, 4}	Array a[] contains non-integer values	Error
v = 'b', a = {1, 2, 3, 4, 5}	v is a non-integer value	Error

Boundary Value Analysis:

Input Data	**Description**	**Expected Outcome**
v = 2, a = {2, 3}	Array has only two elements and v is present in rst half	1
v = 4, a = {2, 4}	Array has only two elements and v is present in 2nd half	1
v = 5, a = {5}	Array has only one element and v is present	0
v = 3, a = {5}	Array has only one element and v is not present	-1
v = 10, a = {1, 2, 3, 4, 5}	v is not in the array but outside the range	-1
v = 5, a = {}	Empty array	-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).

```
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 Classes:

Input Data	**Description**	**Expected Outcome**
a = 3, b = 3, c = 3	All sides are equal (equilateral triangle)	0
a = 5, b = 5, c = 3	Two sides are equal (isosceles triangle)	1
a = 4, b = 5, c = 6	All sides are different (scalene triangle)	2
a = 1, b = 2, c = 3	Impossible lengths (invalid triangle)	3
a = 0, b = 0, c = 0	All sides are zero (invalid triangle)	3
a = 3, b = -1, c = 2	One side is negative (invalid triangle)	3
a = 9, b = ‘y’, c = ‘x’	Side is not an Integer	Error

Boundary Value Analysis:

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

P5. The function pre x (String s1, String s2) returns whether or not the string s1 is a pre x 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;
}
```

Input Data	Description	Expected Outcome
s1 = "pre", s2 = "pre x"	s1 is a valid pre x of s2	TRUE
s1 = "pre", s2 = "pretext"	s1 is a valid pre x of s2	TRUE
s1 = "text", s2 = "pre x"	s1 is not a pre x of s2	FALSE
s1 = "", s2 = "pre x"	s1 is an empty string (always a pre x)	TRUE
s1 = "prE", s2 = "pre x"	s1 is not a pre x (case-sensitive comparison)	FALSE

Boundary Value Analysis:

Input Data	Description	Expected Outcome
s1 = "", s2 = ""	Both strings are empty (empty is a pre x of empty)	TRUE
s1 = "abc", s2 = "abc"	s1 is equal to s2 (is a pre x of itself)	TRUE
s1 = "abcd", s2 = "abc"	s1 is longer than s2 (invalid case)	FALSE
s1 = "ab", s2 = "abc"	s1 is a valid pre x of s2	TRUE

P6: Consider again the triangle classi cation program (P4) with a slightly different speci cation: The program reads oating 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 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: Satis es the Pythagorean theorem ($A^2 + B^2 = C^2$).

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 identi ed equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must need to be ensure that the identi ed set of test cases cover all identi ed 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 different	Scalene Triangle
3.0, 4.0, 5.0	Satis es the Pythagorean theorem	Right-Angled Triangle
1.0, 2.0, 3.0	The sum of the two shorter 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

c) 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

d) 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

e) 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

f) 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

g) 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 third	Non-Triangle
2.0, 5.0, 3.0	Valid triangle	Scalene
10.0, 1.0, 1.0	Impossible lengths	Non-Triangle

h) 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