Software Testing Lab Session - Functional Testing (Black-Box)

Nandini Mandaviya (202201487)

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.

The solution of each problem must be given in the format as follows:

Tester Action and Input Data	*Expected Outcome*
Equivalence Partitioning	
a, b, c	An Error message
a-1, b, c	Yes Yes
Boundary Value Analysis	
a, b, c-1	

Without Boundary Value Analysis:

Input	Expected Output
(31, 12, 2015)	Yes
(1, 1, 1900)	Yes
(15, 6, 2010)	Yes
(30, 2, 2012)	Error: Invalid Date
(30, 2, 2013)	Error: Invalid Date
(31, 4, 2010)	Error: Invalid Date
(32, 7, 2010)	Error: Invalid Date
(0, 5, 2020)	Error: Invalid Date
(1, 13, 2000)	Error: Invalid Month
(1, 0, 2000)	Error: Invalid Month
(15, 5, 1899)	Error: Invalid Year
(15, 5, 2016)	Error: Invalid Year

With Boundary Value Analysis:

Input	Expected Output	Comments
(1, 1, 1900)	Yes	Valid: Boundary for the minimum year.
(31, 12, 2015)	Yes	Valid: Boundary for the maximum year.
(1, 1, 1899)	Error: Invalid Year	Invalid: Year less than 1900.
(1, 1, 2016)	Error: invalid Year	Invalid: Year greater than 2015.
(1, 2, 1900)	Yes	Valid: Non-leap year, Jan to Feb transition.
(29, 2, 1900)	Error; Invalid Date	Invalid: 29th Feb in a non-leap year.
(28, 2, 1900)	Yes	Valid: Boundary for non-leap year February
(30, 2, 1900)	Error: Invalid Date	Invalid: 30th Feb in non-leap year.
(1, 3, 1900)	Yes	Valid: Non-leap year February to March.
(29, 2, 2000)	Yes	Valid: Leap year boundary, February.
(30, 2, 2000)	Error: Invalid Date	Invalid: 30th Feb, leap year.

Input	Expected Outcome	Comments
(1, 3, 2000)	Yes	Valid: Leap year transition from Feb to Mar.
(32, 1, 2000)	Error: Invalid Day	Invalid: Day greater than 31.
(1, 4, 2000)	Yes	Valid: Boundary, month transition Mar to Apr.
(30, 4, 2000)	Yes	Valid: Boundary, 30-day month (April).
(31, 4, 2000)	Error: invalid Date	Invalid: 31st day in April (April has 30).
(1, 5, 2000)	Yes	Valid: Month transition from Apr to May.
(31, 12, 2000)	Yes	Valid: Last day of the year.
(31, 6, 2000)	Error: Invalid Date	Invalid: 31st day in June (June has 30 days).
(1, 0, 2000)	Error: Invalid Month	Invalid: Month less than 1.
(1, 13, 2000)	Error: Invalid Month	Invalid: Month greater than 12.
(31, 12, 1999)	Yes	Valid: Boundary for the 20th century.
(1, 1, 2001)	Yes	Valid: Year transition from Dec to Jan.

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);
}
```

Test Case	Input	Expected Outcome	Class
EP1	v=3, a=[1, 2, 3, 4]	2	Valid v in array
EP2	v=5, a=[1, 2, 3, 4]	-1	Valid: v not in array
EP3	v=3, a=[]	-1	Invalid: Empty array
BVA4	v=3, a=[3]	0	Single element in array
BVA5	v=4, a=[1, 2, 3, 4]	3	Last element in array
BVA6	v=1, a=[1, 2, 3, 4]	0	First element in array

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);

}
```

Test Case	Input	Expected Outcome	Class
EP1	v=2, a=[1, 2, 3, 2, 2]	3	Valid: multiple occurrences
EP2	v=4, a=[1, 2, 3, 5]	0	Valid: no occurrence
EP3	v=1, a=[]	0	Invalid: Empty array
BVA4	v=3, a=[3]	1	Single element in array
BVA5	v=5, a=[1, 5, 5, 5, 5]	4	Last element repeated

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);
}
```

Test Case	Input	Expected Outcome	Class
EP1	v=6, a=[1, 3, 6, 7, 9]	2	Valid: v present
EP2	v=4, a=[1, 3, 6, 7, 9]	-1	Valid: v absent
EP3	v=3, a=[]	-1	Invalid: empty array
BVA4	v=1, a=[1]	0	Single element in array
BVA5	v=9, a=[1, 3, 6, 7, 9]	4	Last element in array
BVA6	v=1, a=[1, 3, 6, 7, 9]	0	First element in array

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);
}
```

Test Case	Input	Expected Outcome	Class
EP1	3, 3, 3	Equilateral	Valid: all sides equal
EP2	4, 4, 5	Isosceles	Valid: two sides equal
EP3	4, 5, 6	Scalene	Valid: all sides different
EP4	1, 2, 3	Invalid	Invalid: cannot form a triangle
BVA5	1, 1, 1	Equilateral	Minimal valid triangle
BVA6	1, 2, 2	Isosceles	Two sides equal (minimal)
BVA7	1, 1, 2	Invalid	Triangle inequality violated

Potential issues:
Non-positive side lengths: The program doesn't check for non-positive side lengths (e.g., a, b, or c being 0 or negative), which could result in logically incorrect classification.
Fix: Add a check at the start to ensure all sides are positive

```
if (a <= 0 || b <= 0 || c <= 0) {
   return "Invalid";
}
```

All the codes are correct and gives expected output.

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;
  }
```

Test Case	Input	Expected Outcome	Class
EP1	s1="pre", s2="prefix"	True	Valid: s1 is a prefix
EP2	s1="post", s2="prefix"	False	Valid: s1 not a prefix
EP3	s1="prefix", s2="pre"	False	Invalid: s1 longer
BVA54	s1="", s2="prefix"	True	Empty string is prefix
BVA5	s1="p", s2="p"	True	Single character strings
BVA6	s1="prefix", s2="prefix"	True	Exact match for prefix

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:

a) Identify the equivalence classes for the system 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)

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

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

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

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

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

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

a) Equivalence Classes for Triangle Program

- Valid triangles:
  - Equilateral (A = B = C).
  - Isosceles (A = B ≠ C).
  - Scalene (A ≠ B ≠ C).
  - Right-angled triangle ($A^2 + B^2 = C^2$).
- Invalid triangles: Any set of sides violating the triangle inequality.
- Non-positive inputs: One or more sides ≤ 0.

b-f) Test Cases to Cover the Identified Equivalence Classes

Test Case	Input	Expected Outcome	Class
EP1	A=3.0, B=3.0, C=3.0	Equilateral	Equilateral triangle
EP2	A=4.0, B=4.0, C=5.0	Isosceles	Isosceles triangle
EP3	A=4.0, B=5.0, C=6.0	Scalene	Scalene triangle
EP4	A=3.0, B=4.0, C=5.0	Right-angled	Right-angled triangle
EP5	A=1.0, B=2.0, C=3.0	Invalid	Invalid triangle
EP6	A=0.0, B=1.0, C=2.0	Invalid	Non-positive input
BVA1	A=1.0, B=1.0, C=1.0	Equilateral	A = B = C case
BVA2	A=3.0, B=4.0, C=5.0	Right-angled	$A^2 + B^2 = C^2$ case
BVA3	A=2.0, B=2.0, C=1.0	Isosceles	A = B, A + B > C case
BVA4	A=1.0, B=2.0, C=3.0	Invalid	Non-triangle boundary