IT313-Software Engineering
Lab-8
Abhinav-202201112

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.

-**Equivalence** class partitioning:

Equivalence Class	**Equivalence Class Description**
E1	month < 1, day < 1, year < 1900
E2	month < 1, day < 1, 1900 ≤ year ≤ 2015
E3	month < 1, day < 1, year > 2015
E4	month < 1, 1 ≤ day ≤ 31, year < 1900
E5	month < 1, 1 ≤ day ≤ 31, 1900 ≤ year ≤ 2015
E6	month < 1, 1 ≤ day ≤ 31, year > 2015
E7	month < 1, day > 31, year < 1900
E8	month < 1, day > 31, 1900 ≤ year ≤ 2015

E9	month < 1, day > 31, year > 2015
E10	1 ≤ month ≤ 12, day < 1, year < 1900
E11	1 ≤ month ≤ 12, day < 1, 1900 ≤ year ≤ 2015
E12	1 ≤ month ≤ 12, day < 1, year > 2015
E13	1 ≤ month ≤ 12, 1 ≤ day ≤ 31, year < 1900
E14	1 ≤ month ≤ 12, 1 ≤ day ≤ 31, 1900 ≤ year ≤ 2015
E15	1 ≤ month ≤ 12, 1 ≤ day ≤ 31, year > 2015
E16	1 ≤ month ≤ 12, day > 31, year < 1900
E17	1 ≤ month ≤ 12, day > 31, 1900 ≤ year ≤ 2015
E18	1 ≤ month ≤ 12, day > 31, year > 2015
E19	month > 12, day < 1, year < 1900

E20	month > 12, day < 1, 1900 ≤ year ≤ 2015
E21	month > 12, day < 1, year > 2015
E22	month > 12, 1 ≤ day ≤ 31, year < 1900
E23	month > 12, 1 ≤ day ≤ 31, 1900 ≤ year ≤ 2015
E24	month > 12, 1 ≤ day ≤ 31, year > 2015
E25	month > 12, day > 31, year < 1900
E26	month > 12, day > 31, 1900 ≤ year ≤ 2015
E27	month > 12, day > 31, year > 2015

Equivalence test cases analysis:

Covered Equivalence Class	Valid / Invalid	Test Case
1	Invalid	month=0, day=0, year=1899
2	Invalid	month=0, day=0, year=2000
3	Invalid	month=0, day=0, year=2016
4	Invalid	month=0, day=1, year=1899
5	Invalid	month=0, day=15, year=2000
6	Invalid	month=0, day=31, year=2016
7	Invalid	month=0, day=32, year=1899
8	Invalid	month=0, day=32, year=2000
9	Invalid	month=0, day=32, year=2016
10	Invalid	month=1, day=0, year=1899
11	Invalid	month=1, day=0, year=2000
12	Invalid	month=1, day=0,

		year=2016
13	Invalid	month=1, day=1, year=1899
14	Valid	month=1, day=15, year=2000
15	Invalid	month=1, day=15, year=2016
16	Invalid	month=1, day=32, year=1899
17	Invalid	month=1, day=32, year=2000
18	Invalid	month=1, day=32, year=2016
19	Invalid	month=13, day=0, year=1899

20	Invalid	month=13, day=0, year=2000
21	Invalid	month=13, day=0, year=2016
22	Invalid	month=13, day=1, year=1899
23	Invalid	month=13,day=15, year=2000
24	Invalid	month=13,day=15, year=2016
25	Invalid	month=13,day=32,

		year=1899
26	Invalid	month=13,day=32, year=2000
27	Invalid	month=13,day=32, year=2016

Boundary Test-Cases analysis:

TEST CASE	VALID/INVALID
Month = 0	Invalid
Month = 13	Invalid
Day = 0	Invalid
Day = 32	Invalid
Year = 1899	Invalid
Year = 2016	Invalid
Month = 1, Day = 1, Year = 1900	Valid
Month = 12, Day = 1, Year = 1900	Valid
Month = 1, Day = 31, Year = 1900	Valid
Month = 12, Day = 31, Year = 1900	Valid
Month = 1, Day = 1, Year = 2015	Valid
Month = 12, Day = 1, Year = 2015	Valid
Month = 1, Day = 31, Year = 2015	Valid
Month = 12, Day = 31, Year = 2015	Valid

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.

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 Cases for Linear Search:

Tester action & Input Data	**Expected Outcome**
(10, ``)	0
(10, ``)	-1
(10, [20,10])	1
(10, [20,-10])	-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);

}
```

Test cases for Count item:

Tester action & Input Data	**Expected Outcome**
(10, ``)	1
(10, ``)	0
(10, [10,-10,-10])	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);
}
```

Test cases for Binary Search:

Tester action & Input Data	**Expected Outcome**
(10, ``)	0
(10, ``)	-1
(10, [20,-10])	-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);
}
```

Test cases for classification of Triangle:

Tester action & Input Data	**Expected Outcome**
(3,3,3)	EQUILATERAL
(3,3,4)	ISOSCELES
(3,4,5)	SCALENE

(5,5,5)	EQUILATERAL
(1,1,3)	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).

```
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 for String:

Tester action & Input Data	Expected Outcome
("abc", "abcde")	True
("hello", "hello world")	True
("test", "testing")	True
("prefix", "suffixed")	False
("long", "longer")	True
("abc", "ab")	False
("java", "javascript")	True
("", "anything")	True
("nonempty", "")	False

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. Identify Equivalence Classes

- *Equilateral: All sides equal.*
- *Isosceles: Two sides equal.*
- *Scalene: All sides different.*
- *Invalid: Sides do not form a triangle.*

b. Identify Test Cases Covering Equivalence Classes

Test Cases Summary

- *(3 ,3 ,3) → EQUILATERAL*
- *(3 ,3 ,4) → ISOSCELES*
- *(3 ,4 ,5) → SCALENE*
- *(1 ,2 ,3) → INVALID*

c. Boundary Condition A + B > C Case

Test case to verify boundary:

- *(3 +4 >7) should yield INVALID.*

d. Boundary Condition A = C Case

Test case to verify boundary:

- *(3 = C) should yield ISOSCELES.*

e. Boundary Condition A = B = C Case

Test case to verify boundary:
- *(4 =4=4) should yield EQUILATERAL.*

f. Boundary Condition $A^2 + B^2 = C^2$ Case
Test case to verify boundary:
- *($3^2 + 4^2 = 5^2$) should yield SCALENE.*

g. Non-Triangle Case Identification
Test case to explore the boundary:
- *(10 +2 <3) should yield INVALID.*

h. Non-positive Input Points
Test cases for non-positive input:
- *(-3 ,-4 ,-5) should yield INVALID*