SOFTWARE ENGINEERING

(IT-314)

Lab - 8

Khushi Vora - 202201332

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?

Ans 1. Valid Equivalence Classes:
E1 - Valid month (1 to 12) , month>=1 && month<=12
E2- Valid day (1 to 31, considering month constraints), day>=1 && day<=31
E3 - Valid year (1900 to 2015),year>=1900 && year>=2015

Invalid Equivalence Classes:

E1- Invalid month (less than 1 or greater than 12), month<1 || month>12

E2: day<1 || day>31,day<1 || day>31

E3: Invalid year (less than 1900 or greater than 2015),year<1900 || year>2015

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 Partitioning:

Tester Action and Input Data	Expected Outcome
03/01/2000	29/02/2000
07/05/2010	14/07/2010
10/31/2015	30/10/2015
0/ 05/2015	An Error message
05/32/2015	An Error message
04/05/1899	An Error message
0103/2000	29/02/2000

***Test Case with leap year**

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
01/01/1901	31/12/1900
01/01/2015	31/12/2014
03/29/2004	30/12/2004
02/29/2016	28/02/2016
04/30/2015	29/04/2015
12/31/2015	An Error message
12/0/2015	An Error message
08/31/2015	30/08/2015

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.

Q2.
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);
```

}

Ans.
Ex: [5, 7,1, 7, 2,4]
Equivalence Class:
E1: v→present in the array
E2: v→not present
E3: Array is empty

Input	**Output**
1	2
9	-1

Boundary Value Class:
BV1: v → present on 0th Idx
BV2: v→ present on last Idx
BV3: v→ not present
BV4: v→present in given range

Input	**Output**
5	0
4	5
9	-1
6	1

P2. The function count Item 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:
E1: v→not present in array ⇒ count(v)=0
E2: v→ present in an array
E3: Array is empty

Input(v, array)	**Output**
(2,[34,1,3,2,3])	1
(7,[1,2,3,4])	0
(5,[])	Invalid or 0

Boundary Value Class:
BV1: count(v)==0
BV2: count(v)==arr.length()

Input	**Output**
(3,[3,3,3])	3
(5,[1,2,3,4])	0
(5,[])	Invalid or 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 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:
E1: v→not present in array
E2: v→present
E3: array is empty

Input(v,[array])	**Output**
(5,[1,2,3,4])	-1
(4,[1,2,3,4,5,6])	3

Boundary Value Class:
BV1: v→present on 0th Idx
BV2: v→present on last Idx

BV3: v→Array is empty

Input	Output
(2,[2,2,2])	0
(3,[1,2,3,4,5])	2
(7,[])	Invalid (-1)
(4,[1,2,3,5,6])	Invalid (-1)
(0,[1,2,3,4,5])	Invalid (-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);
}
```

Ans:
Equivalence Class:
E1: a==b==c
E2: a==b || a==c || c==b (Any two sides equal)
E3: b>=c+a

E4:c>=b+a
E5:a>=a+b

Input(a,b,c)

Input(a,b,c)	**Output**
(5,5,5)	Equilateral
(17,17,12)	Isosceles
(2,3,6)	Invalid
(3,4,5)	Scalene

Boundary Value Class:
BV1: a==b+c
BV2: b==c+a
BV3: c==a+b

Input(a,b,c)	**Output**
(2,2,4)	Isosceles
(5,0,0)	Invalid
(5,1,4)	Invalid
(4,5,8)	Scalene
(17,17,17)	Equilateral

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;
        }
}
```

Equivalence Class:

Valid Cases:

E1: s1 is a prefix of s2.

E2: s2 is equal to s2.

E3:s1is an empty string, and s2 is not.

Invalid Cases:

E4:s1 is longer than s2.

E5: s1 is not a prefix of s2.

E6: Both s1and s2 are empty (not applicable here as per the assumption).

Input (s1, s2)	Expected Outcome
("pre", "prefix")	TRUE
("prefix", "prefix")	TRUE
("", "notEmpty")	TRUE
("longprefix", "short")	FALSE
("not", "prefix")	FALSE

("pre", "noPrefixHere")	FALSE
("", "")	TRUE

Boundary Value Analysis:

Input (s1, s2)	Expected Outcome
("", "a")	TRUE
("a", "a")	TRUE
("a", "ab")	TRUE
("a", "b")	FALSE
("abc", "abcd")	TRUE
("abcd", "abc")	FALSE
("", "")	TRUE

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

Valid Triangle Classes:

- **E1: Equilateral Triangle**: All sides are equal, i.e., A=B=CA = B = CA=B=C.
- **E2: Isosceles Triangle**: Two sides are equal while the third is different, i.e., A=B≠CA = B \neq CA=B=C, A=C≠BA = C \neq BA=C=B, or B=C≠AB = C \neq AB=C=A.
- **E3: Scalene Triangle**: All sides are different, i.e., A≠B≠CA \neq B \neq CA=B=C and A≠CA \neq CA=C.
- **E4: Right-Angled Triangle**: The square of the longest side equals the sum of the squares of the other two sides, i.e., A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (with CCC being the longest side).

Invalid Triangle Classes:

- **E5: Not a Triangle**: The sum of any two sides is less than or equal to the third side, i.e., A+B≤CA + B \leq CA+B≤C, A+C≤BA + C \leq BA+C≤B, or B+C≤AB + C \leq AB+C≤A.
- **E6: Non-positive Inputs**: At least one side is less than or equal to zero, i.e., A≤0A \leq 0A≤0, B≤0B \leq 0B≤0, or C≤0C \leq 0C≤0.

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)

Test Case	Input (A, B, C)	Expected Output	Equivalence Class
TC1	(12, 12, 12)	"Equilateral"	Equilateral Triangle
TC2	(17, 17, 5)	"Isosceles"	Isosceles Triangle
TC3	(4, 5, 6)	"Scalene"	Scalene Triangle
TC4	(6, 12, 17)	"Scalene"	Scalene Triangle
TC5	(2, 2, 5)	"Not a Triangle"	Not a Triangle
TC6	(0, 5, 17)	"Not a Triangle"	Non-positive Input
TC7	(5, 6, 11)	"Not a Triangle"	Not a Triangle
TC8	(0, 0, 0)	"Not a Triangle"	Non-positive Input
TC9	(5, 12, 17)	"Right-Angled"	Right-Angled Triangle
TC10	$(4, 4, \sqrt{32})$	"Right-Angled"	Right-Angled Triangle

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

Test Case	Input (A, B, C)	Expected Output	Equivalence Class
TC11	(3, 4, 5)	"Scalene"	Equilateral Triangle
TC12	(2, 3, 5)	"Not a Triangle"	Isosceles Triangle
TC13	(2, 2.5, 5)	"Not a Triangle"	Scalene Triangle

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

Test Case	Input (A, B, C)	Expected Output
TC14	(3, 3, 2)	"Isosceles"
TC15	(3, 3, 6)	"Not a Triangle"
TC16	(2, 2, 4)	"Not a Triangle"

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

Test Case	Input (A, B, C)	Expected Output
TC17	(5, 5, 5)	"Equilateral"
TC18	(0, 0, 0)	"Not a Triangle"

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

Test Case	Input (A, B, C)	Expected Output
TC19	(3, 4, 5)	"Right-Angled"
TC20	(1, 1,$\sqrt{2}$)	"Right-Angled"
TC21	(1, 2, $\sqrt{5}$)	"Right-Angled"
TC22	(2, 2, $\sqrt{8}$)	"Not a Triangle"

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

Test Case	Input (A, B, C)	Expected Output
TC23	(1, 2, 3)	"Not a Triangle"
TC24	(2, 3, 1)	"Not a Triangle"
TC25	(4, 4,7)	"Not a Triangle"

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

Test Case	Input (A, B, C)	Expected Output
TC26	(1, 0, 1)	"Not a Triangle"
TC27	(-1, -3, 1)	"Not a Triangle"
TC28	(0, 0, 1)	"Not a Triangle"
TC29	(-2, 1, -1)	"Not a Triangle"