IT314 Lab-08
Functional Testing (Black-Box)
202201022 - Gunesha Shahu

Question 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?

Equivalence Classes:

E1 : Input of month is <1
E2 : Input of month is >12
E3 : Input of month is <=12 and >=1
E4: Input of date is <1
E5: Input of date is >31
E6: Input of date is >=1 and <=31
E7: Input of year is <1900 202201412
E8: Input of year is >2015
E9: Input of year is >=1900 and <=2015

Boundary Cases:

B1: Input of month=1
B2: Input of month=2
B3: Input of month=11
B4: Input of month=12
B5: Input of day=1
B6: Input of day=2
B7: Input of day=30

B8: Input of day=31
B9: Input of year=1900
B10: Input of year=1901
B11: Input of year=2014
B12: Input of year=2015
B13: Input of year=1899
B14: Input of year=2016
B15: Input of day=0
B16: Input of day=32
B17: Input of month=0
B18: Input of month=13
B19: Input of month=6
B20: Input of day=26
B21: Input of year=2000

Tester Actions: (Equivalence Partitioning)

#	Test Input (Month,Day,Year)	Expected Outcome	Classes Covered
1	7,29,2004	True	3,6,9
2	0,26,2007	False	1
3	15,17,1998	False	2
4	4,0,1907	False	4
5	11,40,1932	False	5
6	6,15,1875	False	7
7	4,4,2020	False	8

Boundary Value Analysis:

#	Test Input (Month,Day,Year)	Expected Outcome	Boundary Cases
1	1,1,1900	Yes	1,5,9
2	2,2,1901	Yes	2,6,10
3	11,30,2014	Yes	3,7,11
4	12,31,2015	Yes	4,8,12
5	0,1,1900	No	17
6	13,31,2015	No	18
7	1,0,1900	No	15
8	1,32,2015	No	16
9	1,1,1899	No	13
10	12,31,2016	No	14
11	6,26,2000	Yes	19,20,21

Question 2) Programs

a) 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 Classes:

E1: The value v is null

E2: The value v is not null and is not an integer

E3: The value v is not null and is an integer

E4: The array a[] is empty

E5: The array a[] is not empty but does not contain integer as elements

E6: The array a[] is not empty, contains integers as elements, and the value v is present in the array single time

E7: The array a[] is not empty, contains integers as elements, and the value v is present in the array multiple times

E8: The array a[] is not empty, contains integer as element, and the value v is not present in the array

Tester Actions:

#	**Test Input** **(Value v,Array a[])**	**Expected Outcome**	**Classes Covered**
1	1,[1,2,3,4]	0 (Valid Input)	3,6
2	2,[4,5,-1,2,5,2]	3 (Valid Input)	3,7
3	7,[2,-7,0,12]	-1 (Valid Input)	3,8

4	NULL,[10,9,3]	Invalid Input	1
5	a,[0,32,54]	Invalid Input	2
6	-1,[]	Invalid Input	4
7	10,[a,&,-,bfh]	Invalid Input	5

Boundary Cases:

Test Case	Input (v, a[])	Expected Outcome	Description
1	(1, [])	-1	Empty array (no occurrences).
2	(1, [1])	0	Array with one element equal to the search value.
3	(1, [2])	-1	Array with one element not equal to the search value.
4	(1, [1, 1, 1])	0	Array with multiple occurrences of the search value.
5	(1, [2, 3, 4])	-1	Array with no occurrences of the search value.
6	(1, [0, -1, 2, 3])	-1	Array with negative numbers and zero, no occurrences.
7	(1, [1, 2, 3, 1])	0	Array with multiple occurrences, returns the first index.
8	(1, [-1, 2, 3])	-1	Array with negative numbers and no occurrences.
9	(1, [2, 1, 1, 3])	1	Array with the search value present at a non-first index.

10	(0, [0, 0, 0, 0])	0	Array where all elements are zero, counting occurrences of zero.

b) 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 Classes:

E1: The value v is null
E2: The value v is not null and is not an integer
E3: The value v is not null and is an integer
E4: The array a[] is empty
E5: The array a[] is not empty but does not contain integers as elements
E6: The array a[] is not empty, contains integers as elements

Tester Actions:

#	**Test Input (Value v,Array a[])**	**Expected Outcome**	**Classes Covered**
1	5,[45,7,5,0,-1,5]	2(Valid Input)	3,6

2	31,[-5,53,12,7]	0(Valid Input)	3,6
3	NULL,[1,2,3,4]	Invalid Input	1
4	t,[2,76,0,1]	Invalid Input	2
5	-7,[]	Invalid Input	4
6	12,[abg,%,f]	Invalid Input	5

Boundary Cases:

Test Case	Input (v, a[])	Expected Outcome	Description
1	(5, [])	0	Empty array (no occurrences).
2	(5, [5])	1	Array with one element equal to the search value.
3	(5, [5])	0	Array with one element not equal to the search value.
4	(5, [5, 5, 5])	3	Array with multiple occurrences of the search value.
5	(5, [1, 2, 3, 4])	0	Array with no occurrences of the search value.
6	(5, [0, -1, 2, 3, 4])	0	Array with negative numbers and zero, no occurrences.
7	(5, [5, 1, 2, 5, 3, 4])	2	Array with multiple occurrences of the search value.
8	(5, [-5, -1, 2, 3, 4])	0	Array with negative numbers and no occurrences of the

			search value.
9	(5, [5, -5, 1, -1, 2, 3])	1	Array with a mix of values, including one occurrence of the search value.
10	(1, [1, 1, 1, 1])	4	Array where all elements are equal to the search value.
11	(0, [0, 0, 0, 0])	4	Array where all elements are zero, counting occurrences of zero.

a) 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 Classes:

E1: The value v is null

E2: The value v is not null and is not an integer

E3: The value v is not null and is an integer

E4: The array a[] is empty

E5: The array a[] is not empty but does not contain integer as elements

E6: The array a[] is not empty, not sorted in increasing order

E7: The array a[] is not empty, is sorted in increasing order, contains integers as elements, and the value v is present in the array

E8: The array a[] is not empty, is sorted in increasing order, contains integer as element, and the value v is not present in the array. Value v is greater than all elements of the array.

E9: The array a[] is not empty, is sorted in increasing order, contains integer as element, and the value v is not present in the array. Value v is lesser than all elements of the array.

E10: The array a[] is not empty, is sorted in increasing order, contains integer as element, and the value v is not present in the array. Exact value v is not present between the elements of the array.

Tester Actions:

#	**Test Input** **(Value v,Array a[])**	**Expected Outcome**	**Classes Covered**
1	3,[0,2,3,9,11]	2(Valid Input)	3,7
2	134,[23,67,98,120]	-1(Valid Input)	3,8

3	-3,[0,12,54]	-1(Valid Input)	3,9
4	42,[15,40,41,45,69]	-1(Valid Input)	3,10
5	NULL,[1,2,3,4]	Invalid Input	1
6	a,[0,12,54,332]	Invalid Input	2
7	43,[]	Invalid Input	4
8	10,[sd,8,%,~]	Invalid Input	5
9	32,[545,21,89,0]	Invalid Input	6

Boundary Cases:

Test Case	Input (v, a[])	Expected Outcome	Description
1	(0, [])	-1	Empty array (invalid search)
2	(0, [])	-1	Search for a value greater than the only element.
3	(0, [])	0	Search for the only element in the array.
4	(0, [])	-1	Search for a value less than the only element.
5	(5, [1, 2, 3, 4])	-1	Search for a value greater than the maximum element in the array.
6	(1, [1, 2, 3, 4])	0	Search for the minimum element in a sorted array.

7	(4, [1, 2, 3, 4])	3	Search for the maximum element in a sorted array.
8	(2, [1, 2, 3, 4])	1	Search for a middle value in a sorted array.
9	(3, [3, 3, 3, 3])	0	Search for a value that is repeated multiple times in the array.
10	(2, [1, 2, 3, 4, 5])	1	Search for a value that is exactly between elements in a sorted array.
11	(10, [1, 2, 3, 4, 5])	-1	Search for a value that is greater than the last element in the array.
12	(0, [0, 1, 2])	0	Search for zero in a sorted array that includes zero.
13	(1, [1, 1, 1, 1])	0	Search for a repeated value in an array of identical elements.
14	(1, [1, 1, 1, 1])	2	Search for an even number in a sorted array of even integers.

a) 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:

E1: The value a is NULL

E2: The value a is not NULL but not an integer or an integer less than zero

E3: The value a is not NULL and an integer>0

E4: The value b is NULL

E5: The value b is not NULL but not an integer or an integer less than zero

E6: The value b is not NULL and an integer>0

E7: The value c is NULL

E8: The value c is not NULL but not an integer or an integer less than zero

E9: The value c is not NULL and an integer>0

E10: The values a>=b+c or b>=a+c or c>=a+b

E11: The values a==b==c

E12: The values a==b!=c or b==c!=a or c==a!=b

E13: The values do not satisfy the above three equivalence cases (a!= b, a!= c, b!=c, and a+b>c, a+c>b, b+c>a)

Tester Actions:

#	Test Input (Value a,b,c)	Expected Outcome	Classes Covered
1	(1,2,3)	INVALID (Valid Input)	3,6,9,10
2	(5,5,5)	EQUILATERAL (Valid Input)	3,6,9,11
3	(7,9,7)	ISOSCELES (Valid Input)	3,6,9,12
4	(3,4,5)	SCALENE (Valid Input)	3,6,9,13
5	(NULL,3,1)	Invalid Input	1
6	(-3,1,4)	Invalid Input	2
7	(7,NULL,13)	Invalid Input	4
8	(9,a,1)	Invalid Input	5
9	(54,12,NULL)	Invalid Input	7
10	(2,78,0)	Invalid Input	8

Boundary Cases:

Test Case	Input (a, b, c)	Expected Outcome	Description
1	(0, 0, 0)	INVALID	All sides are zero (invalid triangle).
2	(1, 1, 1)	EQUILATERAL	All sides are equal (equilateral triangle).

3	(1, 1, 2)	INVALID	Two sides equal, but do not satisfy triangle inequality.
4	(1, 2, 2)	ISOSCELES	Two sides equal (isosceles triangle).
5	(3, 4, 5)	SCALENE	All sides are different (scalene triangle).
6	(2, 2, 5)	INVALID	Two sides equal but do not satisfy triangle inequality.
7	(1, -1, 1)	INVALID	Negative length (invalid triangle).
8	(1, 1, -1)	INVALID	Negative length (invalid triangle).
9	(5, 10, 5)	ISOSCELES	Two sides equal (isosceles triangle).
10	(3, 5, 2)	SCALENE	All sides are different (scalene triangle).
11	(3, 6, 9)	INVALID	Fails triangle inequality condition (invalid triangle).
12	(0, 1, 1)	INVALID	One side is zero (invalid triangle).

13	(1, 1, 0)	INVALID	One side is zero (invalid triangle).
14	(1, Integer.MAX_VALUE, 1)	INVALID	Large side value, invalid due to triangle inequality.
15	(Integer.MAX_VALUE, Integer.MAX_VALUE, Integer.MAX_VALUE)	EQUILATERAL	All sides equal to the maximum integer value (valid equilateral triangle).

e) 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 Classes

E1: s1 is an empty string and s2 is a non empty string

E2: s1 is a non empty string and s2 is an empty string
E3: s1,s2 are not empty strings and s1 is an exact match of s2
E4: s1,s2 are not empty strings and s1 is a prefix of s2
E5: s1,s2 are not empty strings and s1 is longer than s2
E6: s1,s2 are not empty strings and s1 is not a prefix of s2 E7: Both s1 and s2 are empty strings.

Tester Actions:

#	Test Input (String s1,s2)	Expected Outcome	Classes Covered
1	“”, “SE”	true (Valid Input)	1
2	“SE”, “”	false (Valid Input)	2
3	“DA”, “DA”	true (Valid Input)	3
4	“IT”, “IT304”	true (Valid Input)	4
5	“abcd”, “i8”	false (Valid Input)	5
6	“DA”, “IT304”	false (Valid Input)	6
7	“”, “”	true (Valid Input)	7

Boundary Cases:

Test Case	s1	s2	Expected Result	Description
1	""	""	true	Both strings are empty.

2	""	"hello"	true	s1 is an empty string (valid prefix).
3	"a"	""	false	s1 is non-empty, s2 is empty (invalid prefix).
4	"hello"	"hello"	true	s1 equals s2.
5	"he"	"hello"	true	s1 is a valid prefix of s2.
6	"hello"	"he"	false	s1 is longer than s2 (invalid prefix).
7	"hello"	"world"	false	s1 is not a prefix of s2.
8	"he"	"hi"	false	s1 is a valid prefix but s2 starts with different letters.
9	"hell"	"hello"	true	s1 is a valid prefix of s2.
10	"hello"	"helloo"	true	s1 is a prefix of a longer s2.

f) Consider again the triangle classification program (d) 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**
Equivalence Classes:

E1: The value a is NULL.

E2: The value a is not NULL but is not a number (NaN) or is less than or equal to zero.

E3: The value a is a positive floating-point number.

E4: The value b is NULL.

E5: The value b is not NULL but is not a number (NaN) or is less than or equal to zero.

E6: The value b is a positive floating-point number.

E7: The value c is NULL.

E8: The value c is not NULL but is not a number (NaN) or is less than or equal to zero.

E9: The value c is a positive floating-point number.

E10: The values a,b, and c satisfy the triangle inequality (i.e., a<b+ca < b + ca<b+c, b<a+cb < a + cb<a+c, and c<a+bc
< a + bc<a+b).

E11: The values a, b, and c are all equal (equilateral triangle).

E12: The values a, b, and c are two equal and one different (isosceles triangle).

E13: The values a, b, and c are all different (scalene triangle).

E14: The values form a right-angled triangle (using the Pythagorean theorem).

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

#	Test Input (Value a,b,c)	Expected Outcome	Classes Covered
1	(2.0, 3.0, 4.0)	SCALENE	E3, E6, E9, E10, E13
2	(5.0, 5.0, 5.0)	EQUILATERAL	E3, E6, E9, E10, E11
3	(5.0, 5.0, 3.0)	ISOSCELES	E3, E6, E9, E10, E12
4	(3.0, 4.0, 5.0)	SCALENE	E3, E6, E9, E10, E13
5	(1.0, 2.0, 3.0)	INVALID	E3, E6, E9, E10
6	(0.0, 3.0, 4.0)	INVALID	E3, E6, E2, E10
7	(-1.0, 2.0, 3.0)	INVALID	E2, E3, E6
8	(2.0, 2.0, 0.0)	INVALID	E2, E3, E6, E10
9	(NaN, 3.0, 4.0)	INVALID	E1, E2
10	(3.0, 4.0, NaN)	INVALID	E1, E2
11	(4.0, NaN, 5.0)	INVALID	E1, E2
12	(0.0, 0.0, 0.0)	INVALID	E2
13	(1.0, 1.0, 1.0)	EQUILATERAL	E3, E6, E9, E10, E11
14	(3.0, 4.0, 5.0)	SCALENE	E3, E6, E9, E10, E13
15	(3.0, 4.0, 6.0)	INVALID	E3, E6, E9, E10, E11

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

Here are the boundary test cases for the condition A+B>CA + B > CA+B>C for a scalene triangle:

- (2.0, 3.0, 4.0) - Expected Outcome: SCALENE (Valid triangle where A+B>CA + B > CA+B>C).
- (2.0, 3.0, 5.0) - Expected Outcome: INVALID (Edge case where A+B=CA + B = CA+B=C).
- (3.0, 4.0, 8.0) - Expected Outcome: INVALID (Invalid triangle where A+B<CA + B < CA+B<C).

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

Here are boundary test cases for the condition A=CA = CA=C for an isosceles triangle:

- (3.0, 4.0, 3.0) - Expected Outcome: ISOSCELES (Valid triangle where A=CA = CA=C).
- (5.0, 7.0, 5.0) - Expected Outcome: ISOSCELES (Valid triangle with equal sides).
- (0.0, 1.0, 0.0) - Expected Outcome: INVALID (Invalid triangle with zero-length sides).

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

Here are boundary test cases for the condition A=B=CA = B = CA=B=C for an equilateral triangle:

- (4.0, 4.0, 4.0) - Expected Outcome: EQUILATERAL (Valid triangle with all sides equal).
- (0.0, 0.0, 0.0) - Expected Outcome: INVALID (Invalid triangle with zero-length sides).
- (2.0, 2.0, 2.0) - Expected Outcome: EQUILATERAL (Valid triangle with equal sides).

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

Here are boundary test cases for the condition A^2+B^2=C^2 for a right-angled triangle:

- (3.0, 4.0, 5.0) - Expected Outcome: RIGHT-ANGLED (Valid triangle).
- (5.0, 12.0, 13.0) - Expected Outcome: RIGHT-ANGLED (Valid triangle).
- (6.0, 8.0, 10.0) - Expected Outcome: RIGHT-ANGLED (Valid triangle).
- (1.0, 2.0, 3.0) - Expected Outcome: INVALID (Not a valid triangle).

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

Here are boundary test cases for the non-triangle case:

- (1.0, 2.0, 3.0) - Expected Outcome: INVALID (The sum of the two smaller sides equals the length of the largest side, which does not form a triangle).
- (2.0, 3.0, 6.0) - Expected Outcome: INVALID (The sum of the two smaller sides is less than the length of the largest side).

- (0.0, 2.0, 2.0) - Expected Outcome: INVALID (A side cannot have zero length).
- (-1.0, 1.0, 1.0) - Expected Outcome: INVALID (A side cannot have a negative length).
- (4.0, 1.0, 2.0) - Expected Outcome: INVALID (The sum of the two smaller sides is less than the length of the largest side).

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

Here are test cases for non-positive input:

- (0.0, 0.0, 0.0) - Expected Outcome: INVALID (All sides are zero, which cannot form a triangle).
- (-3.0, 4.0, 5.0) - Expected Outcome: INVALID (One side is negative).
- (5.0, -2.0, 3.0) - Expected Outcome: INVALID (One side is negative).
- (2.0, 0.0, 3.0) - Expected Outcome: INVALID (One side is zero).
- (1.0, -1.0, 1.0) - Expected Outcome: INVALID (One side is negative).