202201492
Shravan N. Makwana
IT-314
Lab 8

Functional Testing (Black-Box)

Question 1:

Equivalence Classes for the problem are:

E1: Month is less than 1 (invalid)
E2: Month is between 1 and 12 (valid)
E3: Month is greater than 12 (invalid)
E4: Day is less than 1 (invalid)
E5: Day is between 1 and the maximum valid day for the given month and year (valid)
E6: Day is greater than the maximum valid day for the given month and year (invalid)
E7: Year is less than 1900 (invalid)
E8: Year is between 1900 and 2015 (valid)
E9: Year is greater than 2015 (invalid)
E10: Month is a non-numeric value (alphabetic or special characters, invalid)
E11: Day is a non-numeric value (alphabetic or special characters, invalid)
E12: Year is a non-numeric value (alphabetic or special characters, invalid)
E13: Month is empty (invalid)
E14: Day is empty (invalid)
E15: Year is empty (invalid)
E16: Month is a decimal value (invalid)
E17: Day is a decimal value (invalid)
E18: Year is a decimal value (invalid)

Equivalence Partitioning:

Valid Inputs:

Input (Day, Month, Year)	Expected Outcome	Classes Covered
7, 9, 2004	**6, 9, 2004**	**E5, E2, E8**
1, 1, 2015	**31, 12, 2014**	**E5, E2, E8**
31, 12, 2015	**30, 12, 2015**	**E5, E2, E8**
2, 3, 2000	**1, 3, 2000**	**E5, E2, E8**
1, 3, 2000	**29, 2, 2000 (Leap Year)**	**E5, E2, E8**
29, 2, 2016 (Leap Year)	**28, 2, 2016**	**E5, E2, E8**
1, 5, 2010	**30, 4, 2010**	**E5, E2, E8**
1, 6, 2007	**31, 5, 2007**	**E5, E2, E8**
1, 12, 2015	**30, 11, 2015**	**E5, E2, E8**
1, 2, 2001	**31, 1, 2001**	**E5, E2, E8**
1, 7, 2012	**30, 6, 2012**	**E5, E2, E8**
30, 12, 2015	**29, 12, 2015**	**E5, E2, E8**
15, 8, 1947	**14, 8, 1947**	**E5, E2, E8**
15, 1, 1900	**14, 1, 1900**	**E5, E2, E8**

1, 11, 1999	**31, 10, 1999**	**E5, E2, E8**

Invalid Inputs:

Input (Day, Month, Year)	Expected Outcome	Classes Covered
'two', 2, 1945	**Invalid Date**	**E11, E2, E8**
2.3, 4, 2003	**Invalid Date**	**E17, E2, E8**
%, 24, 2003	**Invalid Date**	**E20, E2, E8**
'', 12, 1967	**Invalid Date**	**E14, E2, E8**
0, 12, 1999	**Invalid Date**	**E4, E2, E8**
14, 25, 2006	**Invalid Date**	**E5, E3, E8**
11, 'three', 1988	**Invalid Date**	**E5, E11, E8**
10, 14.6, 1932	**Invalid Date**	**E5, E16, E8**
2, &, 1922	**Invalid Date**	**E5, E20, E8**
4, '', 2007	**Invalid Date**	**E5, E13, E8**

Boundary Value Analysis:

Valid Inputs:

Input (Day, Month, Year)	Expected Outcome	Classes Covered
1, 1, 1900	**31, 12, 1899**	**E5, E2, E8**
31, 12, 2015	**30, 12, 2015**	**E5, E2, E8**
29, 2, 2016	**28, 2, 2016**	**E5, E2, E8**
1, 3, 2000	**29, 2, 2000 (Leap Year)**	**E5, E2, E8**
2, 1, 1900	**1, 1, 1900**	**E5, E2, E8**

Invalid Inputs:

Input (Day, Month, Year)	Expected Outcome	Classes Covered
0, 1, 2000	**Invalid Date**	**E4, E2, E8**
32, 12, 1999	**Invalid Date**	**E6, E2, E8**
31, 4, 2015	**Invalid Date**	**E6, E2, E8**
29, 2, 1900	**Invalid Date**	**E6, E2, E8**
1, 0, 2000	**Invalid Date**	**E5, E3, E8**
1, 13, 2000	**Invalid Date**	**E5, E3, E8**
1, 1, 1899	**Invalid Date**	**E5, E2, E7**

1, 1, 2016	**Invalid Date**	**E5, E2, E9**
", 12, 1999	**Invalid Date**	**E14, E2, E8**
1, ", 1999	**Invalid Date**	**E5, E13, E8**

Question 2:

P1: Linear Search Function

Equivalence Partitioning (EP):

- **EP1**: Value exists in the array → Valid case.
- **EP2**: Value does not exist in the array → Invalid case.
- **EP3**: The array is empty → Invalid case.
- **EP4**: Negative values or zero → Special case.

Boundary Value Analysis (BVA):

- **BVA1**: Value at the first position.
- **BVA2**: Value at the last position.
- **BVA3**: Single element in the array.

Test Cases:

Input (v, a[], size)	Expected Outcome	Equivalence Class / Boundary Condition
v = 3, a = [1, 2, 3, 4], 4	2	EP1
v = 5, a = [1, 2, 3, 4], 4	-1	EP2
v = 3, a = [], 0	-1	EP3
v = -1, a = [1, -1, 2, 3], 4	1	EP4

v = 1, a = [1, 2, 3, 4], 4	0	BVA1
v = 4, a = [1, 2, 3, 4], 4	3	BVA2
v = 2, a = [2], 1	0	BVA3

Original Code:

```
int linearSearch(int v, int a[])
{
    int i = 0;
    while (i < a.length)
    {
        if (a[i] == v)
            return(i);
        i++;
    }
    return (-1);
}
```

Corrected Code:

```
int linearSearch(int v, int a[], int size)
{
    for (int i = 0; i < size; i++)
    {
        if (a[i] == v)
            return i;
    }
    return -1;
}
```

P2: Count Item Function

Equivalence Partitioning (EP):

- EP1: Value appears multiple times → Valid case.
- EP2: Value does not appear → Invalid case.
- EP3: Empty array → Invalid case.

- **EP4**: Negative values → Special case.

Boundary Value Analysis (BVA):

- **BVA1**: Single occurrence at the start.
- **BVA2**: Single occurrence at the end.
- **BVA3**: Single occurrence in the middle.

Test Cases:

Input (v, a[], size)	Expected Outcome	Equivalence Class / Boundary Condition
v = 3, a = [1, 2, 3, 3, 4], 5	2	EP1
v = 5, a = [1, 2, 3, 4], 4	0	EP2
v = 2, a = [], 0	0	EP3
v = -1, a = [-1, -1, 2, 3], 4	2	EP4
v = 1, a = [1, 2, 3, 4], 4	1	BVA1
v = 4, a = [1, 2, 3, 4], 4	1	BVA2
v = 3, a = [1, 3, 3], 3	2	BVA3

Original Code:

```
int countItem(int v, int a[])
{
    int count = 0;
    for (int i = 0; i < a.length; i++)
    {
        if (a[i] == v)
            count++;
```

```
    }
    return (count);
}
```

Corrected Code:

```
int countItem(int v, int a[], int size)
{
    int count = 0;
    for (int i = 0; i < size; i++)
    {
        if (a[i] == v)
            count++;
    }
    return count;
}
```

P3: Binary Search Function

Equivalence Partitioning (EP):

- EP1: Value exists in the array → Valid case.
- EP2: Value does not exist → Invalid case.
- EP3: Array is sorted and non-empty → Valid case.
- EP4: Array is empty → Invalid case.

Boundary Value Analysis (BVA):

- BVA1: Value at the start.
- BVA2: Value at the end.
- BVA3: Single element in the array.

Test Cases:

Input (v, a[], size)	Expected Outcome	Equivalence Class / Boundary Condition
v = 3, a = [1, 2, 3, 4, 5], 5	2	EP1

v = 6, a = [1, 2, 3, 4, 5], 5	-1	EP2
v = 2, a = [], 0	-1	EP4
v = 1, a = [1], 1	0	BVA3
v = 1, a = [1, 2, 3, 4, 5], 5	0	BVA1
v = 5, a = [1, 2, 3, 4, 5], 5	4	BVA2

Original Code:

```
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);
}
```

Corrected Code:

```
int binarySearch(int v, int a[], int size)
{
   int lo = 0, hi = size - 1;
   while (lo <= hi)
```

```
    {
        int mid = (lo + hi) / 2;
        if (a[mid] == v)
            return mid;
        else if (v < a[mid])
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    return -1;
}
```

P4: Triangle Function

Equivalence Partitioning (EP):

- **EP1**: Equilateral triangle (all sides equal).
- **EP2**: Isosceles triangle (two sides equal).
- **EP3**: Scalene triangle (all sides different).
- **EP4**: Invalid triangle (side lengths violate the triangle inequality).

Boundary Value Analysis (BVA):

- **BVA1**: Near the triangle inequality boundary.
- **BVA2**: Equal sides boundary for isosceles.
- **BVA3**: Equal sides boundary for equilateral.

Test Cases:

Input (a, b, c)	Expected Outcome	Equivalence Class / Boundary Condition
3, 3, 3	Equilateral	EP1, BVA3
3, 3, 4	Isosceles	EP2, BVA2
3, 4, 5	Scalene	EP3, BVA1

1, 2, 3	Invalid	EP4
1, 1, 2	Invalid	BVA1

Original Code:

```
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 || b == c || a == c)
        return ISOSCELES;
    return SCALENE;
}
```

Corrected Code:

```
public class TriangleType {
    final int EQUILATERAL = 0;
    final int ISOSCELES = 1;
    final int SCALENE = 2;
    final int INVALID = 3;

    public int triangle(int a, int b, int c) {
        if (a <= 0 || b <= 0 || c <= 0 || 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;
    }
}
```

P5: Prefix Function

Equivalence Partitioning (EP):

- **EP1:** s1 is a prefix of s2.
- **EP2:** s1 is not a prefix of s2 (either s1 is longer or the characters do not match).
- **EP3:** s1 is an empty string.
- **EP4:** s2 is an empty string.

Boundary Value Analysis (BVA):

- **BVA1:** Length of s1 equal to length of s2.
- **BVA2:** Length of s1 one less than length of s2.
- **BVA3:** Length of s1 greater than length of s2.

Test Cases:

Input (s1, s2)	Expected Outcome	Equivalence Class / Boundary Condition
"hello", "hello world"	true	EP1
"hell", "hello world"	true	EP1, BVA2
"hello world", "hello"	false	EP2, BVA3
"hi", "hello"	false	EP2
"", "hello"	true	EP3
"hello", ""	false	EP4

Original Code:

```
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;
}
```

Corrected Code:

```
public class StringPrefix {
    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;
    }
}
```

P6: Triangle Classification Program

a) Identify the equivalence classes for the system

1. **EP1:** Equilateral triangle (all sides equal).
2. **EP2:** Isosceles triangle (two sides equal).
3. **EP3:** Scalene triangle (all sides different).
4. **EP4:** Right-angled triangle (satisfies the Pythagorean theorem).
5. **EP5:** Invalid triangle (side lengths violate the triangle inequality).
6. **EP6:** Non-positive input (at least one side is less than or equal to zero).

b) Identify test cases to cover the identified equivalence classes

Input (A, B, C)	Expected Outcome	Equivalence Class
3.0, 3.0, 3.0	Equilateral	EP1
3.0, 3.0, 4.0	Isosceles	EP2
3.0, 4.0, 5.0	Scalene	EP3
5.0, 12.0, 13.0	Right-angled	EP4
1.0, 2.0, 3.0	Invalid	EP5
0.0, 4.0, 5.0	Invalid	EP6
-1.0, 1.0, 1.0	Invalid	EP6

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

Input (A, B, C)	Expected Outcome	Description
3.0, 4.0, 5.0	Scalene	Valid boundary, scalene
2.0, 2.0, 4.0	Invalid	On boundary, invalid
2.0, 3.0, 4.0	Scalene	Valid boundary, scalene

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

Input (A, B, C)	Expected Outcome	Description
2.0, 2.0, 3.0	Isosceles	Valid boundary, isosceles

2.0, 3.0, 2.0	Isosceles	Valid boundary, isosceles
1.0, 1.0, 2.0	Invalid	On boundary, invalid

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

Input (A, B, C)	Expected Outcome	Description
3.0, 3.0, 3.0	Equilateral	Valid boundary, equilateral
2.0, 2.0, 2.0	Equilateral	Valid boundary, equilateral
0.0, 0.0, 0.0	Invalid	On boundary, invalid

f) For the boundary condition A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 case (right-angle triangle), identify test cases to verify the boundary.

Input (A, B, C)	Expected Outcome	Description
3.0, 4.0, 5.0	Right-angled	Valid boundary, right-angled
5.0, 12.0, 13.0	Right-angled	Valid boundary, right-angled
1.0, 1.0, Math.sqrt(2)	Right-angled	Valid boundary, right-angled

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

Input (A, B, C)	Expected Outcome	Description
1.0, 2.0, 3.0	Invalid	Non-triangle (equal sides)
3.0, 3.0, 6.0	Invalid	Non-triangle (exceeds)

2.0, 2.0, 4.0	Invalid	Non-triangle (equals)

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

Input (A, B, C)	Expected Outcome	Description
0.0, 1.0, 1.0	Invalid	Non-positive input
-1.0, 2.0, 2.0	Invalid	Non-positive input
1.0, 0.0, 1.0	Invalid	Non-positive input
1.0, 1.0, -1.0	Invalid	Non-positive input