Lab-8
Functional Testing (Black-Box)

Kevin Shingala
202201465

<u>Q1)</u>

Equivalence Classes for Input Data

Month Value Classes

- E1: Month value is alphabetic (Invalid)
- E2: Month value is numeric (Valid)
- E3: Month value is decimal (Invalid)
- E4: Month value is non-alphabetic (Invalid)
- E5: Month value is empty (Invalid)
- E6: Month value is less than 1 (Invalid)
- E7: Month value is in the range 1 to 12 (Valid)
- E8: Month value is more than 12 (Invalid)

Day Value Classes

- E9: Day value is alphabetic (Invalid)
- E10: Day value is numeric (Valid)
- E11: Day value is decimal (Invalid)
- E12: Day value is non-alphabetic (Invalid)
- E13: Day value is empty (Invalid)
- E14: Day value is less than 1 (Invalid)
- E15: Day value is in the range 1 to 31 (Valid)
- E16: Day value is more than 31 (Invalid)

Year Value Classes

- E17: Year value is less than 1900 (Invalid)
- E18: Year value is in the range 1900 to 2015 (Valid)
- E19: Year value is more than 2015 (Invalid)
- E20: Year value is alphabetic (Invalid)
- E21: Year value is numeric (Valid)
- E22: Year value is decimal (Invalid)
- E23: Year value is non-alphabetic (Invalid)
- E24: Year value is empty (Invalid)

Test Case No.	Input Values	Expected Outcome	Classes Covered
1	(7, 19, 2004)	Previous Date	E2, E7, E10, E15, E18, E21
2	(two, 2, 1945)	Invalid Date	E1
3	(2.3, 4, 2003)	Invalid Date	E3
4	(%, 24, 2003)	Invalid Date	E4
5	(, 12, 1967)	Invalid Date	E5
6	(0, 12, 1999)	Invalid Date	E6
7	(14, 25, 2006)	Invalid Date	E8
8	(11, three, 1988)	Invalid Date	E9
9	(10, 14.6, 1932)	Invalid Date	E11
10	(2, &, 1922)	Invalid Date	E12
11	(4, , 2007)	Invalid Date	E13
12	(5, 0, 2000)	Invalid Date	E14
13	(11, 35, 1992)	Invalid Date	E16
14	(8, 22, twenty)	Invalid Date	E20

15	(5, 19, 1987.6)	Invalid Date	E22
16	(12, 13, $)	Invalid Date	E23
17	(4, 20,)	Invalid Date	E24
18	(9, 17, 1899)	Invalid Date	E17
19	(1, 23, 2016)	Invalid Date	E19
20	(1, 1, 1900)	Previous Date	E7, E15, E18, E21
21	(2, 29, 2000)	Previous Date	E7, E15, E18, E21
22	(3, 1, 2015)	Previous Date	E7, E15, E18, E21
23	(12, 31, 2015)	Previous Date	E7, E15, E18, E21
24	(4, 30, 2010)	Previous Date	E7, E15, E18, E21
25	(11, 30, 2011)	Previous Date	E7, E15, E18, E21

Q2)

P1:

Equivalence Classes:

1. Valid input where v is present in a[]:

2. Valid input where v is not present in a[]:
3. Empty array (a[] is empty):
4. Single-element array (a[] has exactly one element):
5. Array with duplicate elements:

Test Case #	Input Array a[]	Search Value v	Expected Output	Equivalence Classes Covered
1	{1, 2, 3, 4, 5}	3	2	v present in a[]
2	{1, 2, 3, 4, 5}	6	-1	v not present in a[]
3	{}	3	-1	Empty array
4	{3}	3	0	Single-element array, v present
5	{3}	5	-1	Single-element array, v not present
6	{2, 4, 2, 4, 2}	4	1	Array with duplicates

Boundaries for Array Length:

- Empty array (a.length = 0).
- Single-element array (a.length = 1).
- Multiple-element array (a.length > 1).

Boundaries for Search Value (v):

- Search for the smallest element in the array.
- Search for the largest element in the array.
- Search for a value just outside the range (less than the smallest or greater than the largest).

Test Case #	Input Array a[]	Search Value v	Expected Output	Boundary Conditions Covered
1	{}	3	-1	Empty array
2	{5}	5	0	Single-element array, v present
3	{5}	4	-1	Single-element array, v not present
4	{1, 2, 3, 4, 5}	1	0	Search smallest element
5	{1, 2, 3, 4, 5}	5	4	Search largest element
6	{1, 2, 3, 4, 5}	0	-1	Search less than smallest
7	{1, 2, 3, 4, 5}	6	-1	Search greater than largest

Modified Code –

```
            public class SearchFunctions {
              // Modified linearSearch function to handle null arrays
              public static int linearSearch(int v, int[] a) {
                if (a == null || a.length == 0) {
                  return -1; // Return -1 if the array is null or empty
                }

                for (int i = 0; i < a.length; i++) {
                  if (a[i] == v) {
                    return i; // Return the index if the value is found
                  }
                }
                return -1; // Return -1 if the value is not found
              }
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

P2

Equivalence Classes:

1. Valid input where v is present in a[] multiple times:
2. Valid input where v is present in a[] exactly once:
3. Valid input where v is not present in a[]:
4. Empty array (a[] is empty):
5. Single-element array (a[] has exactly one element):

Test Case #	Input Array a[]	Search Value v	Expected Output	Equivalence Classes Covered
1	{1, 2, 3, 4, 5}	3	1	v present once in a[]
2	{1, 2, 3, 3, 3, 4}	3	3	v present multiple times in a[]
3	{1, 2, 3, 4, 5}	6	0	v not present in a[]
4	{}	3	0	Empty array
5	{3}	3	1	Single-element array, v present
6	{3}	5	0	Single-element array, v not present

Boundary Conditions:

1. Array size boundaries:
   - Empty array (a.length = 0).
   - Single-element array (a.length = 1).

    - Multi-element array.
2. Value boundaries (v):
    - Searching for the smallest element in the array.
    - Searching for the largest element in the array.
    - Searching for a value just outside the range of the elements in the array.

Test Cases for Boundary Value Analysis

Test Case #	Input Array a[]	Search Value v	Expected Output	Boundary Conditions Covered
1	{}	3	0	Empty array
2	{5}	5	1	Single-element array, v present
3	{5}	4	0	Single-element array, v not present
4	{1, 2, 3, 4, 5}	1	1	Search smallest element
5	{1, 2, 3, 4, 5}	5	1	Search largest element
6	{1, 2, 3, 4, 5}	0	0	Search less than smallest element
7	{1, 2, 3, 4, 5}	6	0	Search greater than largest element

Modified Code –

```
#include <iostream>
using namespace std;

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

```
            count++;
    }
    return count;
}
```

P3

Equivalence Classes:

1. Valid input where v is present in a[]:.
2. Valid input where v is not present in a[]:
3. Empty array (a[] is empty):
4. Single-element array (a[] has exactly one element):
5. Array where v is at the first, last, or middle position:

Test Cases for Equivalence Class Partitioning

Test Case #	Input Array a[]	Search Value v	Expected Output	Equivalence Classes Covered
1	{1, 2, 3, 4, 5}	3	2	v present in a[]
2	{1, 2, 3, 4, 5}	6	-1	v not present in a[]
3	{}	3	-1	Empty array
4	{3}	3	0	Single-element array, v present
5	{3}	5	-1	Single-element array, v not present
6	{1, 2, 3, 4, 5}	1	0	v at first position
7	{1, 2, 3, 4, 5}	5	4	v at last position

Boundary Conditions:

1. Array size boundaries:
   - Empty array (a.length = 0).
   - Single-element array (a.length = 1).
   - Multi-element array (a.length > 1).
2. Value boundaries (v):
   - Searching for the smallest element in the array.
   - Searching for the largest element in the array.
   - Searching for a value just outside the range of the elements in the array (less than the smallest or greater than the largest).

Test Cases for Boundary Value Analysis

Test Case #	Input Array a[]	Search Value v	Expected Output	Boundary Conditions Covered
1	{}	3	-1	Empty array
2	{5}	5	0	Single-element array, v present
3	{5}	4	-1	Single-element array, v not present
4	{1, 2, 3, 4, 5}	1	0	Search smallest element
5	{1, 2, 3, 4, 5}	5	4	Search largest element
6	{1, 2, 3, 4, 5}	0	-1	Search less than smallest element
7	{1, 2, 3, 4, 5}	6	-1	Search greater than largest element

Modified Code –

```
#include <iostream>
using namespace std;
```

```
// Modified binarySearch function to accept array size
int binarySearch(int v, int a[], int length) {
    int lo = 0, hi = length - 1, mid;
    while (lo <= hi) {
        mid = lo + (hi - lo) / 2;
        if (v == a[mid])
            return mid;
        else if (v < a[mid])
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    return -1;
}
```

P4

Equivalence Classes:

1. Valid equilateral triangle:.
2. Valid isosceles triangle:
3. Valid scalene triangle:
4. Invalid triangle:

Test Cases for Equivalence Class Partitioning

Test Case #	Sides of Triangle (a, b, c)	Expected Output	Equivalence Classes Covered
1	3,3,33, 3, 33,3,3	EQUILATERAL	Valid equilateral triangle
2	3,3,23, 3, 23,3,2	ISOSCELES	Valid isosceles triangle
3	3,4,53, 4, 53,4,5	SCALENE	Valid scalene triangle
4	1,2,31, 2, 31,2,3	INVALID	Invalid triangle
5	0,0,00, 0, 00,0,0	INVALID	Invalid triangle (zero lengths)
6	−1,2,2-1, 2, 2−1,2,2	INVALID	Invalid triangle (negative side)

Boundary Conditions:

1. Smallest non-zero lengths:
2. Equal sides, but close to violating triangle inequality:
3. Zero or negative lengths:

Test Case #	Sides of Triangle (a, b, c)	Expected Output	Boundary Conditions Covered
1	1,1,11, 1, 11,1,1	EQUILATERAL	Smallest valid triangle
2	1,2,21, 2, 21,2,2	ISOSCELES	Valid isosceles, close to boundary
3	1,1,21, 1, 21,1,2	INVALID	Boundary where sum of two sides equals third
4	1,2,31, 2, 31,2,3	INVALID	Triangle inequality violated
5	0,1,10, 1, 10,1,1	INVALID	Zero length
6	−1,1,1-1, 1, 1−1,1,1	INVALID	Negative length

Modified 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) {
        // Check if the triangle is invalid
        if (a <= 0 || b <= 0 || c <= 0 || a >= b + c || b >= a + c || c >= a + b) {
            return INVALID;
        }
        // Check if the triangle is equilateral
        if (a == b && b == c) {
            return EQUILATERAL;
        }
        // Check if the triangle is isosceles
        if (a == b || a == c || b == c) {
            return ISOSCELES;
        }
        // Otherwise, it must be scalene
        return SCALENE;
    }
}
```

P5

Equivalence Classes:

1. Valid prefix (s1 is a prefix of s2):
2. Invalid prefix (s1 is not a prefix of s2):
3. s1 is longer than s2:

Test Cases for Equivalence Class Partitioning

Test Case #	String s1	String s2	Expected Output	Equivalence Classes Covered
1	"test"	"testing"	true	Valid prefix
2	"hello"	"hell"	false	s1 longer than s2
3	"cat"	"dog"	false	Invalid prefix
4	"prefix"	"prefixation"	true	Valid prefix
5	"sample"	"samples"	true	Valid prefix
6	"wrong"	"wrang"	false	Invalid prefix (mismatch in middle)
7	""	"empty"	true	Empty string as prefix

Boundary Conditions:

1. Empty strings:
2. Strings with equal lengths:
3. Single-character strings:
4. s1 is longer than s2:

Test Cases for Boundary Value Analysis

Test Case #	String s1	String s2	Expected Output	Boundary Conditions Covered
1	""	"empty"	true	Empty string as prefix
2	"a"	"apple"	true	Single-character prefix
3	"a"	"b"	false	Single-character mismatch
4	"same"	"same"	true	Strings of equal length, match
5	"diff"	"differ"	false	Invalid prefix, mismatch after partial match
6	"testinglong"	"test"	false	s1 longer than s2
7	"long"	"longer"	true	Valid prefix, exact match for part of the string

Modified 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

a) Identify the equivalence classes for the system

- ➔ The Equivalence Classes are –
  E1 : All sides are positive (Valid)
  E2 : One or more sides are negative (Invalid)
  E3 : Valid triangle inequality i.e. sum of two sides is greater than the third (Valid)
  E4 : Invalid triangle inequality (Invalid)
  E5 : All the sides are equal to form an Equilateral triangle (Valid)
  E6 : Two sides are equal to form an Isosceles triangle (Valid)
  E7 : All the sides are unequal to form an Scalene triangle (Valid)
  E8 : Sides are entered to form a Right-angled triangle (Valid)
  E9 : One of the sides has length 0 (Invalid)

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)

- ➔ The Test Cases are –

Test Case No.	**Input Values**	**Expected Outcome**	**Covered Equivalence Class**
1	3, 4, 5	Right-angled Triangle	E1, E3, E8
2	3, 3, 3	Equilateral Triangle	E1, E3, E5
3	4, 5, 4	Isosceles Triangle	E1, E3, E6
4	2, 3, 4	Scalene Triangle	E1, E3, E7
5	1, 2, 3	Invalid Triangle	E1, E4
6	0, 2, 3	Invalid Input	E9
7	-1, 2, 3	Invalid Input	E2

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

- ➔ The Test Case are –

Test Case No.	**Input Values**	**Expected Outcome**

1	2.9999, 4, 7	Scalene Triangle
2	3, 4, 7.0001	Scalene Triangle

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

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	5, 7.12, 5	Isosceles Triangle
2	7, 7, 13,2	Isosceles Triangle

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

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	8, 8, 8	Equilateral Triangle
2	2.0, 2.0, 2.0	Equilateral Triangle

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

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	5, 12, 13	Right-angled Triangle
2	6, 8, 10	Right-angled Triangle

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

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	1, 2, 3	Invalid Triangle
2	4, 4, 8	Invalid Triangle

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

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	0, 5, 3	Invalid Input
2	-1, -5, 3	Invalid Input