Name: Sahil Chaudhari

StudentID: 202201171

Step 1: Equivalence Partitioning

Equivalence Partitioning is a technique used to divide a set of test cases into groups or partitions that are expected to exhibit similar behavior.In this case, we have three input parameters: day, month, and year. Let's identify the equivalence classes for each parameter:

Day

- Valid days: 1 to 31
- Invalid days: 0, negative numbers, and numbers greater than 31

Month

- Valid months: 1 to 12
- Invalid months: 0, negative numbers, and numbers greater than 12

Year

- Valid years: 1900 to 2015
- Invalid years: Years before 1900 and after 2015

Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome
a. Valid day (15), valid month (6), valid year (2000)	Previous date or valid date
b. Invalid day (0), valid month (6), valid year (2000)	An Error message
c. Valid day (15), invalid month (0), valid year (2000)	An Error message
d. Valid day (15), invalid month (13), valid year (2000)	An Error message
e. Valid day (15), valid month (6), invalid year (1899)	An Error message
f. Valid day (15), valid month (6), invalid year (2016)	An Error message
g. Invalid day (0), invalid month (0), valid year (2000)	An Error message
h. Invalid day (32), invalid month (13), valid year (2000)	An Error message
i. Valid day (1), valid month (1), invalid year (2016)	An Error message
j. Valid day (31), valid month (12), invalid year (1899)	An Error message
k. Invalid day (0), invalid month (0), invalid year (1899)	An Error message
l. Invalid day (32), invalid month (13), invalid year (2016)	An Error message

Step 2: Boundary Value Analysis
Boundary Value Analysis focuses on testing at the boundaries between equivalence partitions.
Day
● Test with values: 1, 31, 0, 32
Month
● Test with values: 1, 12, 0, 13
Year
● Test with values: 1900, 2015, 1899, 2016

Boundary Value Analysis Test Cases:

Tester Action and Input Data	Expected Outcome
a. Valid day (1), valid month (1), valid year (1900)	Previous date or valid date
b. Valid day (31), valid month (12), valid year (2015)	Previous date or valid date
c. Invalid day (0), valid month (6), valid year (2000)	An Error message
d. Invalid day (32), valid month (6), valid year (2000)	An Error message
e. Valid day (15), invalid month (0), valid year (2000)	An Error message
f. Valid day (15), invalid month (13), valid year (2000)	An Error message
g. Valid day (15), valid month (6), invalid year (1899)	An Error message
h. Valid day (15), valid month (6), invalid year (2016)	An Error message

ANS-2

P1: linearSearch Function

Searches for value `v` in array `a`. Returns the first index `i` where `a[i] == v`, otherwise returns `-1`.

Equivalence Partitioning Test Cases:

- Input: (5, [1, 2, 3, 4, 5]), Expected Output: 4

- Input: (10, [1, 2, 3, 4, 5]), Expected Output: -1

Boundary Value Analysis Test Cases:

- Input: (1, [1]), Expected Output: 0

- Input: (5, [1, 2, 3, 4, 5]), Expected Output: 4

- Input: 2, [1], Expected Output: -1

```c
#include <stdio.h>
#include <stdlib.h>
int linearSearch(int v, int a[], int size) {
if (a == NULL) {
printf("Error: Array is null.\n");
return -1; // Handle null array
}
for (int i = 0; i < size; i++) {
if (a[i] == v) {
return i; // Found
}
}
return -1; // Not found
}
int main() {
int testCase1[] = {1, 2, 3, 4, 5};
int testCase2[] = {1, 2, 3, 4, 5};
int testCase3[] = {};
int testCase4[] = {1, 2, 3};
int testCase5[] = {1, 0, 2}; // Mixed types would require a differentimplementation.

printf("TC1: %d\n", linearSearch(5, testCase1, 5)); // Expected: 4
printf("TC2: %d\n", linearSearch(10, testCase2, 5)); // Expected: -1
printf("TC3: %d\n", linearSearch(0, testCase3, 0)); // Expected: -1
printf("TC4: %d\n", linearSearch(5, NULL, 0)); // Expected: Errormessage
printf("TC7: %d\n", linearSearch(1, (int[]){1}, 1)); // Expected: 0
printf("TC8: %d\n", linearSearch(2, (int[]){1}, 1)); // Expected: -1
printf("TC9: %d\n", linearSearch(1, testCase1, 5)); // Expected: 0
printf("TC10: %d\n", linearSearch(5, testCase1, 5)); // Expected: 4
return 0;
}
```

P2: `COUNTITEM` Function

Returns the number of times value v appears in array a.

Equivalence Partitioning Test Cases:

- Input: (3, [1, 3, 3, 4, 3, 5]), Expected Output: 3
- Input: (10, [1, 2, 3, 4, 5]), Expected Output: 0
- Input: (5, [1, 2, 3, 4, 5]), Expected Output: 1
- Input: (1, []), Expected Output: 0
- Input: ("a", [1, 2, 3]), Expected Output: ERROR

Boundary Value Analysis Test Cases:

- Input: (1, [1]), Expected Output: 1
- Input: (5, [1, 2, 3, 4, 5]), Expected Output: 1
- Input: (2, [1]), Expected Output: 0
- Input: (1, [1, 2, 3, 4, 5]), Expected Output: 1

```c
#include <stdio.h>
#include <stdlib.h>

int countItem(int v, int a[], int size) {
if (a == NULL) {
printf("Error: Array is null.\n");
return -1; // Handle null array
}
int count = 0;
for (int i = 0; i < size; i++) {
if (a[i] == v) {
count++; // Increment count if value is found
}
}
return count; // Return the count of occurrences
}
int main() {
int testCase1[] = {1, 3, 3, 4, 3, 5}; // 3 appears 3 times
int testCase2[] = {1, 2, 3, 4, 5}; // 5 appears 1 time
int testCase3[] = {}; // Empty array
int testCase4[] = {1, 2, 3}; // Non-integer input would
require different handling
int testCase5[] = {1, 0, 2}; // Mixed types simulated here
printf("TC1: %d\n", countItem(3, testCase1, 6)); // Expected: 3
printf("TC2: %d\n", countItem(5, testCase2, 5)); // Expected: 1
printf("TC3: %d\n", countItem(10, testCase2, 5)); // Expected: 0
printf("TC4: %d\n", countItem(1, testCase3, 0)); // Expected: 0
printf("TC5: %d\n", countItem(3, NULL, 0)); // Expected: Error message
printf("TC8: %d\n", countItem(1, (int[]){1}, 1)); // Expected: 1
printf("TC9: %d\n", countItem(2, (int[]){1}, 1)); // Expected: 0
printf("TC10: %d\n", countItem(1, testCase2, 5)); // Expected: 1
printf("TC11: %d\n", countItem(5, testCase2, 5)); // Expected: 1
return 0;
}
```

P3: binarySearch Function

Searches for value `v` in a sorted array `a`. Returns index `i` if `a[i] == v`, otherwise returns `-1`.

Equivalence Partitioning Test Cases:

- Input: (3, [1, 2, 3, 4, 5]), Expected Output: 2
- Input: (10, [1, 2, 3, 4, 5]), Expected Output: -1
- Input: (1, [1, 2, 3, 4, 5]]), Expected Output: 0
- Input: (5, [1, 2, 3, 4, 5]), Expected Output:4
- Input: (3, []), Expected Output: -1

Boundary Value Analysis Test Cases:

- Input: (1, [1]), Expected Output: 0
- Input: (2, [1]), Expected Output: -1
- Input: (1, [1, 2, 3, 4, 5]), Expected Output: 0
- Input: (3, [1, 2, 3, 4, 5] Expected Output: 0
- Input: (5, [1, 2, 3, 4, 5]), Expected Output: 4

```c
#include <stdio.h>
int binarySearch(int v, int a[], int size) {
    if (a == NULL) {
        printf("Error: Array is null.\n");
        return -1; // Handle null array
    }
    int lo = 0;
    int hi = size - 1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (v == a[mid]) {
            return mid; // Found
        } else if (v < a[mid]) {
            hi = mid - 1; // Search in the left half

        } else {
            lo = mid + 1; // Search in the right half
        }
    }
    return -1; // Not found
}
int main() {
    int testCase1[] = {1, 2, 3, 4, 5}; // Sorted array
    int testCase2[] = {}; // Empty array
    int testCase3[] = {1}; // Single element array
    int testCase4[] = {1, 1, 1, 1, 1}; // All elements are the same
    printf("TC1: %d\n", binarySearch(3, testCase1, 5)); // Expected: 2
    printf("TC2: %d\n", binarySearch(10, testCase1, 5)); // Expected: -1
    printf("TC3: %d\n", binarySearch(1, testCase1, 5)); // Expected: 0
    printf("TC4: %d\n", binarySearch(5, testCase1, 5)); // Expected: 4
    printf("TC5: %d\n", binarySearch(3, testCase2, 0)); // Expected: -1
    printf("TC6: %d\n", binarySearch(3, NULL, 0)); // Expected: Error
    printf("TC7: %d\n", binarySearch(1, testCase3, 1)); // Expected: 0
    printf("TC8: %d\n", binarySearch(2, testCase3, 1)); // Expected: -1
    printf("TC9: %d\n", binarySearch(1, testCase1, 5)); // Expected: 0
    printf("TC10: %d\n", binarySearch(5, testCase1, 5)); // Expected: 4
```

P4: `TRIANGLE` Function Takes three integer parameters as the lengths of a triangle's sides. Returns the type of triangle (Equilateral, Isosceles, Scalene, or Invalid).

i) Test Suite

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
3, 3, 3	0
3, 3, 5	1
3, 4, 5	2
1, 2, 3	3
0, 0, 0	3
-1, 2, 3	3
Boundary Value Analysis	
1, 1, 1	0
2, 2, 3	1
2, 2, 5	3

1, 2, 2	1
0, 1, 1	3

```c
#include <stdio.h>
#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3
int triangle(int a, int b, int c) {
    if (a <= 0 || b <= 0 || c <= 0) {
        return INVALID; // Handle invalid lengths
    }
    if (a >= b + c || b >= a + c || c >= a + b) {
        return INVALID; // Check for triangle inequality
    }
    if (a == b && b == c) {
        return EQUILATERAL; // All sides equal
    }
    if (a == b || a == c || b == c) {
        return ISOSCELES; // Two sides equal
    }
    return SCALENE; // No sides equal
}
int main() {
// Test Cases
    printf("TC1: %d\n", triangle(3, 3, 3)); // Expected: 0 (Equilateral)
    printf("TC2: %d\n", triangle(3, 3, 5)); // Expected: 1 (Isosceles)
    printf("TC3: %d\n", triangle(3, 4, 5)); // Expected: 2 (Scalene)
    printf("TC4: %d\n", triangle(1, 2, 3)); // Expected: 3 (Invalid)
    printf("TC5: %d\n", triangle(0, 0, 0)); // Expected: 3 (Invalid)
    printf("TC6: %d\n", triangle(-1, 2, 3)); // Expected: 3 (Invalid)
    printf("TC7: %d\n", triangle(1, 1, 1)); // Expected: 0 (Equilateral)
    printf("TC8: %d\n", triangle(2, 2, 3)); // Expected: 1 (Isosceles)
    printf("TC9: %d\n", triangle(2, 2, 5)); // Expected: 3 (Invalid)
    printf("TC10: %d\n", triangle(1, 2, 2)); // Expected: 1 (Isosceles)
    printf("TC11: %d\n", triangle(0, 1, 1)); // Expected: 3 (Invalid)
    return 0;
}
```

P5: prefix Function

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).

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
"abc", "abcdef"	true
"abc", "ab"	false
"abc", "def"	false
"abc", "abc"	true
"", "abcdef"	true
"abcdef", ""	false

Boundary Value Analysis	
"", ""	true
"a", "a"	true
"a", "b"	false
"abc", "abcd"	true

"abc", "ab"	false

```java
public class StringPrefix {
    public static boolean prefix(String s1, String s2) {
// Check if s1 is longer than s2
        if (s1.length() > s2.length()) {
            return false;
        }
// Check each character for matching
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) != s2.charAt(i)) {
                return false; // Mismatch found
            }
        }
        return true; // All characters matched
    }
    public static void main(String[] args) {
        System.out.println("TC1: " + prefix("abc", "abcdef")); //Expected:  true

        System.out.println("TC2: " + prefix("abc", "ab")); // Expected: false
        System.out.println("TC3: " + prefix("abc", "def")); // Expected:    false
        System.out.println("TC4: " + prefix("abc", "abc")); // Expected:    true
        System.out.println("TC5: " + prefix("", "abcdef")); // Expected:    true
        System.out.println("TC6: " + prefix("abcdef", "")); // Expected:    false
        System.out.println("TC7: " + prefix("", "")); // Expected: true
        System.out.println("TC8: " + prefix("a", "a")); // Expected: true
        System.out.println("TC9: " + prefix("a", "b")); // Expected: false
        System.out.println("TC10: " + prefix("abc", "abcd")); // Expected:  true
        System.out.println("TC11: " + prefix("abc", "ab")); // Expected:    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

1. Equivalence Class for Valid Triangles:

○ Equilateral: All sides equal (A = B = C).
○ Isosceles: Two sides equal (A = B, A = C, or B = C).

○ Scalene: All sides different (A ≠ B, A ≠ C, B ≠ C).
○ Right-angled: Satisfies Pythagorean theorem (A2 + B2 = C2,considering A, B, C as sides).

2. Equivalence Class for Invalid Triangles:

○ Non-Triangle: Sides do not satisfy triangle inequality (A + B ≤C, A + C ≤ B, B + C ≤ A).
○ Non-positive Values: Any of A, B, or C is less than or equal to zero.

b) Identify Test Cases

Test Case ID	Description	Input (A, B, C)	Expected Outcome	Equivalence Class
TC1	Equilateral Triangle	(3.0, 3.0, 3.0)	"Equilatera l"	Equilateral
TC2	Isosceles Triangle	(3.0, 3.0, 5.0)	"Isosceles"	Isosceles
TC3	Scalene Triangle	(3.0, 4.0, 5.0)	"Scalene"	Scalene
TC4	Right-Angled Triangle	(3.0, 4.0, 5.0)	"Right-angled"	Right-angled
TC5	Non-Triangle	(1.0, 2.0, 3.0)	"Not a triangle"	Non-Triangle
TC6	Non-Triangle	(5.0, 2.0, 3.0)	"Not a triangle"	Non-Triangle
TC7	Non-positive Input	(0.0, 2.0, 3.0)	"Invalid"	Non-positive Values
TC8	Non-positive Input	(-1.0, 2.0, 3.0)	"Invalid"	Non-positive Values

a) Boundary Test Cases for Scalene Triangle (A + B > C)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC9	Boundary scalene case	(2.0, 3.0, 4.0)	"Scalene"
TC10	Just not forming scalene case	(2.0, 2.0, 4.0)	"Not a triangle"

b) Boundary Test Cases for Isosceles Triangle (A = C)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC11	Boundary isosceles case	(3.0, 3.0, 5.0)	"Isosceles"
TC12	Just not forming isosceles case	(3.0, 2.0, 5.0)	"Scalene"

c) Boundary Test Cases for Equilateral Triangle (A = B = C)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC13	Boundary equilateral case	(3.0, 3.0, 3.0)	"Equilateral"
TC14	Just not forming equilateral case	(2.0, 2.0, 3.0)	"Isosceles"

d) Boundary Test Cases for Right-Angled Triangle ($A^2 + B^2 = C^2$)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC15	Boundary right-angled case	(3.0, 4.0, 5.0)	"Right- angled"

TC16	Just not forming right-angled	(3.0, 4.0, 6.0)	"Scalene"

e) Boundary Test Cases for Non-Triangle

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC17	Not satisfying triangle inequality	(1.0, 1.0, 3.0)	"Not a triangle"
TC18	Not satisfying triangle inequality	(1.0, 2.0, 2.0)	"Not a triangle"

f) Test Cases for Non-Positive Input

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC19	Zero input	(0.0, 2.0, 3.0)	"Invalid"
TC20	Negative input	(-1.0, 2.0, 3.0)	"Invalid"