IT314
SOFTWARE ENGINEERING

LAB 8

Name: Heer Shah
SID: 202201525

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

<u>Equivalence Class:</u>

- E1: 1 <= month <= 12 (Valid)
- E2: month < 1 (Invalid)
- E3: month > 12 (Invalid)
- E4: Valid Day (Day valid for the given month and year)
- E5: Day < 1 (Invalid)
- E6: Day > 31 (Invalid)
- E7: Valid Year (1900 <= year <= 2015)
- E8: Year < 1900 (Invalid)
- E9: Year > 2015 (Invalid)

<u>Test Cases Table</u>

Test Case	Classes Covered	Expected Output
(1, 1, 1900)	**E1, E4, E7**	**Previous Date**
(0, 1, 2010)	**E2**	**Invalid Date**
(13, 1, 2010)	**E3**	**Invalid Date**
(1, 0, 2010)	**E5**	**Invalid Date**
(1, 32, 2010)	**E6**	**Invalid Date**
(1, 1, 1899)	**E8**	**Invalid Date**
(1, 1, 2016)	**E9**	**Invalid Date**

Q2. 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.
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.

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.

i) Test Suite

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
5, [1, 2, 3, 4, 5]	4
10, [1, 2, 3, 4, 5]	-1
0, []	-1
5, null	Error message
"a", [1, 2, 3]	Error message
5, [1, "2", 3, 4]	Error message

Boundary Value Analysis	
1, [1]	0
2, [1]	-1
1, [1, 2, 3, 4, 5]	0
5, [1, 2, 3, 4, 5]	4

ii) Modified Program

```
#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 different
implementation.
```

```
    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: Error
message

    // Test Case 5: Non-integer value
    // Note: In C, we cannot pass a non-integer directly. This case would
be handled in a different context (e.g., strings).

    // Test Case 6: Mixed data types (simulated; in practice, use arrays
of a common type)
    // Here we just simulate how it could be handled; in C, you would need
unions or struct.
    // This requires a more complex implementation to actually check
types.

    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. The function countItem returns the number of times a value v appears in an array of integers a.

i) Test Suite

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
3, [1, 3, 3, 4, 3, 5]	3
5, [1, 2, 3, 4, 5]	1
10, [1, 2, 3, 4, 5]	0
1, []	0
3, null	Error message
"a", [1, 2, 3]	Error message
3, [1, "3", 3, 4]	Error message
Boundary Value Analysis	
1, [1]	1
2, [1]	0
1, [1, 2, 3, 4, 5]	1
5, [1, 2, 3, 4, 5]	1

ii) Modified Program

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

```c
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

    // Test Case 6: Non-integer value
    // Note: Handling for non-integer input is not directly applicable in
C.
```

```c
    // Test Case 7: Mixed data types (simulated; requires a more complex implementation)
    // This would require a different design; not implemented here.

    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. 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 a are sorted in non-decreasing order.
i) Test Suite

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
3, [1, 2, 3, 4, 5]	2
10, [1, 2, 3, 4, 5]	-1
1, [1, 2, 3, 4, 5]	0
5, [1, 2, 3, 4, 5]	4
3, []	-1

3, null	Error message
Boundary Value Analysis	
1, [1]	0
2, [1]	-1
1, [1, 2, 3, 4, 5]	0
5, [1, 2, 3, 4, 5]	4
3, [1, 1, 1, 1, 1]	0

ii) Modified Program

```
#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
```

```c
        } 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
message


    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
    printf("TC11: %d\n", binarySearch(3, testCase4, 5)); // Expected: 0
(first occurrence)

    return 0;
}
```

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).

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

ii) Modified Program

```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. 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).

i) Test Suite

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

ii) Modified Program

```
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 ($A^2 + B^2 = C^2$, considering A, B, C as sides).
2. **Equivalence Class for Invalid Triangles**:
    - **Non-Triangle**: Sides do not satisfy triangle inequality ($A + B \leq C$, $A + C \leq B$, $B + C \leq A$).
    - **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)	"Equilateral"	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

c) 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"

d) 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"

e) 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"

f) 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"

g) 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"

h) 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"