Software Engineering (IT314)

Lab :- 8

Student Name :- Fajil Chauhan.
Student ID :- 202201221.

1. Equivalence Partitioning (EP):

Month:1 <= month <= 12
Day:1 <= day <= 31(depending on the month)
Year:1900 <= year <= 2015

For equivalence classes, we consider:

1. Valid month: 1–12
2.Invalid month: Less than 1 or greater than 12
3.Valid day: Within the correct range for the given month
4. Invalid day: Out of range for the specific month
5. Valid year: 1900–2015
6. Invalid year: Less than 1900 or greater than 2015

2. Boundary Value Analysis (BVA)

1. The lowest and highest possible values of months: 1, 12

2. The lowest and highest possible values of day: 1, 31

3. The lowest and highest possible values of years: 1900, 2015

Equivalence Partitioning Test Cases:

Input Data	**Expected Outcome**	
15, 7, 2010	14, 7, 2010	
1, 1, 1900	Invalid	
31, 12, 2015	30, 12, 2015	
29, 2, 2012	28, 2, 2012	
0, 10, 2005	Invalid Day	
15, 13, 2010	Invalid Month	

Boundary Value Analysis Test Cases:

Input Data	**Expected Outcome**
1, 1, 1900	Invalid date
31, 12, 2015	30, 12, 2015
28, 2, 2012	27, 2, 2012
1, 3, 2012	29, 2, 2012
31, 1, 2010	30, 1, 2010
1, 3, 2013	**28, 2, 2013**

b) 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.

C++ Program for Determining the Previous Date:-

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

bool isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400
== 0);
}

string getPreviousDate(int day, int month, int year) {
    // Validate inputs
    if (year < 1900 || year > 2015 || month < 1 || month > 12 ||
day < 1 || day > 31) {
        return "Invalid date";
    }

    // Days in each month
    int daysInMonth[] = {31, isLeapYear(year) ? 29 : 28, 31,
30, 31, 30, 31, 31, 30, 31, 30, 31};

    // Check for the valid day in the given month
    if (day > daysInMonth[month - 1]) {
        return "Invalid date";
    }
```

```
    // Calculate previous date
    if (day > 1) {
        return to_string(day - 1) + "/" + to_string(month) + "/"
+ to_string(year);
    } else {
        if (month == 1) {
            // January goes to December of the previous year
            return to_string(31) + "/12/" + to_string(year - 1);
        } else {
            // Go to the last day of the previous month
            return to_string(daysInMonth[month - 2]) + "/" +
to_string(month - 1) + "/" + to_string(year);
        }
    }
}

int main() {
    int day, month, year;

    // Input: day, month, year
    cout << "Enter day: ";
    cin >> day;
    cout << "Enter month: ";
    cin >> month;
    cout << "Enter year: ";
    cin >> year;
    // Calculate and print the previous date
    string previousDate = getPreviousDate(day, month,
year);
    cout << "Previous date: " << previousDate << endl;

    return 0;
}
```

Testing the Program

Test Case Input	Expected Output
1, 1, 2015	31/12/2014
1, 3, 2015	28/2/2015
29, 2, 2012	28/2/2012
1, 5, 2015	30/4/2015
31, 1, 2015	30/1/2015
1, 13, 2015	Invalid date
32, 1, 2015	Invalid date
1, 1, 1899	Invalid date

Checking Outcomes:-

For each input, check if the output matches the expected outcome:

1. Run the program.
2. Input the day, month, and year as specified in the test cases.
3. Compare the output to the expected result.

Question 2: Programs

(P1)

Equivalence Classes:

Class 1: Empty array → Output: -1

Class 2: Value exists once in the array → Output: Index of v

Class 3: Value exists multiple times in the array → Output: First index of v

Class 4: Value does not exist in the array → Output: -1

Class 5: Value exists as the first element in the array → Output: 0

Class 6: Value exists as the last element in the array → Output: Last index of v

Test Cases

Input (v, a)	Expected Output	Covers E-Class
(5, [])	-1	1
(3, [1, 2, 3, 4, 5])	2	2
(2, [1, 2, 3, 2, 5])	1	3
(9, [1, 2, 3, 4, 5])	-1	4
(1, [1, 2, 3, 4, 5])	0	5

(5, [1, 2, 3, 4, 5])	4	6

(P2)

Equivalence Classes:

Class 1: Empty array → Output: 0
Class 2: Value exists once in the array → Output: 1
Class 3: Value exists multiple times in the array → Output: Count of occurrences of v
Class 4: Value does not exist in the array → Output: 0
Class 5: All elements are equal to v → Output: Length of the array

Test Cases

Input (v, a)	Expected Output	Covers E-Class
(5, [])	0	1
(3, [1, 2, 3, 4, 5])	1	2
(2, [1, 2, 3, 2, 5])	2	3
(9, [1, 2, 3, 4, 5])	0	4
(1, [1, 1, 1, 1, 1])	5	5
(2, [2, 2, 3, 2, 4, 2])	4	3

(P3)

Equivalence Classes:

Class 1: Empty array → Output: -1
Class 2: Value exists in the array → Output: Index of v
Class 3: Value does not exist in the array → Output: -1
Class 4: Value exists as the first element in the array → Output: 0
Class 5: Value exists as the last element in the array → Output: Last index of v
Class 6: Array has only one element, and it is v → Output: 0
Class 7: Array has only one element, and it is not v → Output: -1

Test Cases

Input (v, a)	Expected Output	Covers E-Class
(5, [])	-1	1
(3, [1, 2, 3, 4, 5])	2	2
(6, [1, 2, 3, 4, 5])	-1	3
(1, [1, 2, 3, 4, 5])	0	4
(2, [2])	0	6

(3, [2])	-1	7

(P4)

Equivalence Classes:

Class 1: Invalid triangle (sum of two sides is less than or equal to the third) → Output: INVALID
Class 2: Equilateral triangle (all sides are equal) → Output: EQUILATERAL
Class 3: Isosceles triangle (two sides are equal) → Output: ISOSCELES
Class 4: Scalene triangle (no sides are equal) → Output: SCALENE

Test Cases

Input (v, a)	Expected Output	Covers E-Class
(1, 2, 3)	3	1
(5, 5, 5)	0	2
(5, 5, 3)	1	3
(4, 5, 6)	2	4
(1, 10, 12)	3	1
(3, 3, 5)	1	3

(P5)

Equivalence Classes:

Class 1: s1 is a valid prefix of s2
Class 2: s1 is not a prefix of s2
Class 3: s1 is equal to s2
Class 4: s1 is longer than s2
Class 5: s1 is an empty string (prefix of any string)
Class 6: s2 is empty but s1 is non-empty

Test Cases

Input (s1, s2)	**Expected Output**	**Covers E-Class**
(abc, abcdef)	true	1
(xyz, abcdef)	false	2
(abc, abc)	true	3
(abcd, abc)	false	4
(, abc)	true	5
(abc,)	false	6
(prefix, prefixTest)	true	1

(prefixTest, prefix)	false	4

(P6)

(A) Equivalence Classes:

Class 1: Invalid triangle (A+B<=C, A+C<=B, B+ C<=A)
Class 2: Equilateral triangle (A = B = C)
Class 3: Isosceles triangle (A=B or A=C or B=C,A+B >C)
Class 4: Scalene triangle (A,B,and C are all different,A+ B >C)
Class 5: Right-angled triangle ($A^2 + B^2 = C^2$)
Class 6: Non-positive input (A, B, C <= 0)

(B) Test Cases for Equivalence Classes

Input (A, B, C)	Expected Output	Covers E-Class
(1.0, 2.0, 3.0)	Invalid triangle	1
(3.0, 3.0, 3.0)	Equilateral triangle	2
(4.0, 4.0, 5.0)	Isosceles triangle	3
(3.0, 4.0, 5.0)	Scalene triangle	4
(3.0, 4.0, 5.0)	Right-angled triangle	5
(-1.0, 1.0, 1.0)	Non-positive input	6

(C) Boundary Condition for Scalene Triangle (A+B>C)

Input (A, B, C)	Expected Output	Triangle Type
(4.0, 3.0, 5.0)	Valid	Scalene triangle
(3.0, 4.0, 5.0)	Valid	Scalene triangle

(D) Boundary Condition for Isosceles Triangle (A=C)

Input (A, B, C)	Expected Output	Triangle Type
(5.0, 5.0, 8.0)	Valid	Isosceles triangle
(5.0, 8.0, 5.0)	Valid	Isosceles triangle

(E) Boundary Condition for Equilateral Triangle (A = B = C)

Input (A, B, C)	Expected Output	Triangle Type
(5.0, 5.0, 5.0)	Valid	Equilateral triangle
(3.0, 3.0, 3.0)	Valid	Equilateral triangle

(F) Boundary Condition for Right-Angled Triangle ($A^2 + B^2 = C^2$)

Input (A, B, C)	Expected Output	Triangle Type
(3.0, 4.0, 5.0)	Valid	Right-angled triangle
(5.0, 12.0, 13.0)	Valid	Right-angled triangle

(G) Non-Triangle Test Cases

Input (A, B, C)	Expected Output	Triangle Type
(1.0, 1.0, 2.0)	Invalid	Non-triangle
(1.0, 2.0, 3.0)	Invalid	Non-triangle

(H) Non-Positive Input Test Cases

Input (A, B, C)	Expected Output	Triangle Type
(0.0, 1.0, 1.0)	Non-positive input	Non-positive input
(-1.0, 1.0, 1.0)	Non-positive input	Non-positive input
(1.0, 0.0, 1.0)	Non-positive input	Non-positive input
(1.0, 1.0, -1.0)	Non-positive input	Non-positive input