Software Engineering

Lab-8

202201490

Maunil Modi

1)

Equivalence Partitioning Test Cases

Valid Date Equivalence Classes:

- **Valid Date (Regular Day):**
  - Input: (10, 6, 2020)
  - Expected Outcome: (9, 6, 2020)
- **Valid Date (End of February, Leap Year):**
  - Input: (29, 2, 2024)
  - Expected Outcome: (28, 2, 2024)
- **Valid Date (End of February, Non-Leap Year):**
  - Input: (28, 2, 2021)
  - Expected Outcome: (27, 2, 2021)
- **Valid Date (End of Month):**
  - Input: (31, 5, 2018)
  - Expected Outcome: (30, 5, 2018)
- **Valid Date (Month with 30 Days):**
  - Input: (30, 9, 2022)
  - Expected Outcome: (29, 9, 2022)

Invalid Date Equivalence Classes:

- **Invalid Day (Zero Day):**
  - Input: (0, 8, 2020)
  - Expected Outcome: An Error message
- **Invalid Day (Negative Day):**

    - Input: (-3, 11, 2021)
    - Expected Outcome: An Error message
- **Invalid Month (Zero Month):**
    - Input: (12, 0, 2019)
    - Expected Outcome: An Error message
- **Invalid Month (Negative Month):**
    - Input: (5, -2, 2023)
    - Expected Outcome: An Error message
- **Invalid Year (Future Year):**
    - Input: (20, 12, 2030)
    - Expected Outcome: An Error message

Boundary Value Analysis Test Cases

Boundary Values for Valid Inputs:

- **Day Before First of the Month:**
    - Input: (1, 7, 2019)
    - Expected Outcome: (30, 6, 2019)
- **Last Day of February, Non-Leap Year:**
    - Input: (28, 2, 2021)
    - Expected Outcome: (27, 2, 2021)
- **Last Day of February, Leap Year:**
    - Input: (29, 2, 2024)
    - Expected Outcome: (28, 2, 2024)
- **Day Boundary (31st Day):**
    - Input: (31, 8, 2020)
    - Expected Outcome: (30, 8, 2020)
- **Year Lower Boundary:**
    - Input: (1, 1, 1905)
    - Expected Outcome: (31, 12, 1904)
- **Year Upper Boundary:**
    - Input: (1, 1, 2020)
    - Expected Outcome: (31, 12, 2019)
- **Day Maximum for Months with 30 Days:**
    - Input: (30, 6, 2021)
    - Expected Outcome: (29, 6, 2021)

- **Last Valid Input for Valid Year:**
  - Input: (31, 12, 2020)
  - Expected Outcome: (30, 12, 2020)

C++ Program

```cpp
#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 > 2025 || 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_str
        }
    }
}

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

Here are the updated test cases:

Test Case Input

- **(2, 1, 2020)** → Expected Output: "1/1/2020"
- **(1, 4, 2020)** → Expected Output: "31/3/2020"
- **(29, 2, 2024)** → Expected Output: "28/2/2024"
- **(1, 10, 2020)** → Expected Output: "30/9/2020"
- **(31, 5, 2020)** → Expected Output: "30/5/2020"
- **(1, 14, 2020)** → Expected Output: "Invalid date"
- **(32, 8, 2020)** → Expected Output: "Invalid date"
- **(1, 1, 1899)** → Expected Output: "Invalid date"

2)

P1)

Equivalence Classes

- **Class 1**: Empty list → Output: `-1`
- **Class 2**: Value exists (first occurrence at index 0) → Output: `0`
- **Class 3**: Value exists (first occurrence at index n, n > 0) → Output: `n`
- **Class 4**: Value is absent from the list → Output: `-1`
- **Class 5**: Value exists with multiple occurrences → Output: Index of the first occurrence.

Test Cases

Input (v, a)	Expected Output	Covers Equivalence Class
(7, [])	-1	1
(2, [2, 8, 9])	0	2
(6, [4, 6, 3])	1	3
(9, [2, 4, 6, 9])	3	4
(15, [3, 5, 7])	-1	5
(3, [3, 5, 3, 9])	0	6
(4, [4, 4, 4, 4])	0	6
(7, [2, 3, 5, 7, 7])	3	6
(10, [1, 3, 10, 4, 10])	2	6
(8, [8, 9, 10, 11])	0	2

P2)

Equivalence Classes

- **Class 1**: Empty list → Output: `0`
- **Class 2**: Value exists once → Output: `1`
- **Class 3**: Value exists more than once → Output: Count of occurrences
- **Class 4**: Value is absent → Output: `0`
- **Class 5**: All elements in the list are equal to `v` → Output: Length of the list

Test Cases

Input (v, a)	Expected Output	Covers Equivalence Class
(5, [])	0	1
(8, [3, 8, 5])	1	2
(12, [3, 6, 9])	0	4

Input (v, a)	Expected Output	Covers Equivalence Class
(4, [4, 4, 4, 4])	4	5
(6, [6, 8, 6, 9])	2	3
(11, [3, 5, 7, 8])	0	4
(5, [5, 5, 5, 5, 5])	5	5
(0, [0, 0, 2])	2	3
(14, [5, 6, 7, 8])	0	4
(9, [1, 3, 5, 9, 9, 9])	3	3

P3)

Equivalence Classes

- **Class 1**: Empty list → Output: `-1`
- **Class 2**: Value exists at the first index → Output: Index `0`
- **Class 3**: Value exists at a middle index → Output: Index of `v`
- **Class 4**: Value exists at the last index → Output: Index of last occurrence
- **Class 5**: Value is smaller than any element in the list → Output: `-1`
- **Class 6**: Value is larger than any element in the list → Output: `-1`
- **Class 7**: Value does not exist and lies between two elements → Output: `-1`
- **Class 8**: Value exists with multiple occurrences → Output: Index of any occurrence of `v`

Test Cases

Input (v, a)	Expected Output	Covers Equivalence Class
(8, [])	-1	1
(4, [2, 4, 6, 8])	1	2
(2, [2, 4, 6, 8])	0	2
(8, [2, 4, 6, 8])	3	4
(0, [2, 4, 6, 8])	-1	5
(10, [2, 4, 6, 8])	-1	6

Input (v, a)	Expected Output	Covers Equivalence Class
(3, [2, 3, 3, 5, 8])	1	8
(9, [2, 4, 6, 7, 8])	-1	6
(4, [1, 2, 4, 4, 7])	2	8
(5, [1, 1, 1, 1])	-1	5

P4)

Equivalence Classes

- **Class 1**: Invalid triangle (non-positive sides) → Output: `INVALID`
- **Class 2**: Invalid triangle (triangle inequality not met) → Output: `INVALID`
- **Class 3**: Equilateral triangle (all sides equal) → Output: `EQUILATERAL`
- **Class 4**: Isosceles triangle (two sides equal) → Output: `ISOSCELES`
- **Class 5**: Scalene triangle (all sides different) → Output: `SCALENE`

Test Cases

Input (a, b, c)	Expected Outcome	Covers Equivalence Class
(0, 0, 0)	INVALID	1
(-3, 4, 5)	INVALID	1
(5, 5, 5)	EQUILATERAL	3
(4, 4, 6)	ISOSCELES	4
(3, 4, 5)	SCALENE	5
(9, 9, 12)	ISOSCELES	4
(2, 3, 5)	INVALID	2
(3, 3, 7)	INVALID	2
(6, 7, 8)	SCALENE	5
(7, 4, 11)	INVALID	2

P5)

Equivalence Classes

- **Class 1**: s1 is longer than s2 → Output: `false`
- **Class 2**: s1 is an exact prefix of s2 → Output: `true`
- **Class 3**: s1 is a partial prefix of s2 → Output: `false`
- **Class 4**: s1 is empty → Output: `true`
- **Class 5**: s2 is empty and s1 is not → Output: `false`
- **Class 6**: s1 is identical to s2 → Output: `true`

Test Cases

Input (s1, s2)	Expected Outcome	Covers Equivalence Class
("def", "defghi")	true	2
("xyz", "xy")	false	3
("xyz", "abcxyz")	false	3
("", "defghi")	true	4
("a", "")	false	5
("xyz", "xyz")	true	6
("long", "shorter")	false	1
("cat", "catch")	true	2
("word", "wor")	false	3
("prefix", "prefixsuffix")	true	2

P6)

a) Identify the Equivalence Classes

Valid Triangle Types:

- **Equilateral Triangle**: A = B = C
- **Isosceles Triangle**: A = B, or A = C, or B = C

- **Scalene Triangle**: $A \neq B \neq C$
- **Right-Angled Triangle**: $A^2 + B^2 = C^2$ (or permutations)

Invalid Triangle Types:

- **Not a Triangle**: $A + B \leq C$, $A + C \leq B$, or $B + C \leq A$
- **Non-positive Input**: $A \leq 0$, $B \leq 0$, or $C \leq 0$

b) Test Cases to Cover the Identified Equivalence Classes

Input (A, B, C)	Expected Outcome	Equivalence Classes Covered
(3, 3, 3)	Equilateral Triangle	Equilateral Triangle
(4, 4, 5)	Isosceles Triangle	Isosceles Triangle
(2, 3, 4)	Scalene Triangle	Scalene Triangle
(3, 4, 5)	Right-Angled Triangle	Right-Angled Triangle
(2, 3, 5)	Not a Triangle	Not a Triangle
(0, 4, 5)	Invalid	Non-positive Input

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

Input (A, B, C)	Expected Outcome
(3, 4, 5)	Scalene Triangle
(3, 5, 7)	Scalene Triangle
(4, 5, 9)	Not a Triangle
(4, 5, 8)	Scalene Triangle

d) Boundary Condition A = C (Isosceles Triangle)

Input (A, B, C)	Expected Outcome
(3, 4, 3)	Isosceles Triangle
(3, 5, 5)	Isosceles Triangle
(4, 5, 9)	Not a Triangle
(5, 5, 5)	Equilateral Triangle

e) Boundary Condition A = B = C (Equilateral Triangle)

Input (A, B, C)	Expected Outcome
(3, 3, 3)	Equilateral Triangle
(5, 5, 5)	Equilateral Triangle
(4, 5, 9)	Not a Triangle
(4, 5, 8)	Scalene Triangle

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

Input (A, B, C)	Expected Outcome
(3, 4, 5)	Right-Angled Triangle
(5, 12, 13)	Right-Angled Triangle
(4, 5, 9)	Not a Triangle
(4, 5, 8)	Scalene Triangle

g) Non-Triangle Case

Input (A, B, C)	Expected Outcome
(3, 4, 5)	Scalene Triangle
(5, 10, 16)	Not a Triangle
(4, 5, 9)	Not a Triangle
(4, 5, 8)	Scalene Triangle

h) Non-Positive Input

Input (A, B, C)	Expected Outcome
(3, 4, 0)	Invalid
(3, 5, -1)	Invalid
(4, 0, 9)	Invalid
(0, 5, 8)	Invalid