Software Engineering (IT314)
Lab :- 8

Student Name :- Prajapati Harshit
Student ID :- 202201500

Q-1):-

❖ Equivalence Partitioning Test Cases

Valid Date Equivalence Classes:-

1. **Valid Date (Regular Day)**:

   Input: (15, 8, 2010)

   Expected Outcome: (14, 8, 2010)

2. **Valid Date (End of February, Leap Year)**:

   Input: (29, 2, 2000)

   Expected Outcome: (28, 2, 2000)

3. **Valid Date (End of February, Non-Leap Year)**:

   Input: (28, 2, 2013)

   Expected Outcome: (27, 2, 2013)

4. **Valid Date (End of Month)**:

   Input: (30, 4, 2015)

   Expected Outcome: (29, 4, 2015)

5. **Valid Date (Month with 30 Days)**:

Input: (30, 6, 2015)

Expected Outcome: (29, 6, 2015)

Invalid Date Equivalence Classes:-

1. **Invalid Day (Zero Day)**:

   Input: (0, 10, 2015)

   Expected Outcome: An Error message

2. **Invalid Day (Negative Day)**:

   Input: (-1, 10, 2015)

   Expected Outcome: An Error message

3. **Invalid Month (Zero Month)**:

   Input: (1, 0, 2015)

   Expected Outcome: An Error message

4. **Invalid Month (Negative Month)**:

   Input: (1, -1, 2015)

   Expected Outcome: An Error message

5. **Invalid Year (Future Year)**:

   Input: (1, 1, 2025)

   Expected Outcome: An Error message

Boundary Value Analysis Test Cases

Boundary Values for Valid Inputs:-

1. **Day Before First of the Month**:

   Input: (1, 1, 2015)

   Expected Outcome: (31, 12, 2014)

2. **Last Day of February, Non-Leap Year**:

   Input: (28, 2, 2013)

   Expected Outcome: (27, 2, 2013)

3. **Last Day of February, Leap Year**:

   Input: (29, 2, 2016)

   Expected Outcome: (28, 2, 2016)

4. **Day Boundary (31st Day)**:

   Input: (31, 12, 2015)

   Expected Outcome: (30, 12, 2015)

5. **Year Lower Boundary**:

   Input: (1, 1, 1900)

   Expected Outcome: (31, 12, 1899)

6. **Year Upper Boundary**:

   Input: (1, 1, 2015)

   Expected Outcome: (31, 12, 2014)

7. **Day Maximum for Months with 30 Days**:

   Input: (30, 4, 2015)

   Expected Outcome: (29, 4, 2015)

8. **Last Valid Input for Valid Year**:

   Input: (31, 12, 2015)

   Expected Outcome: (30, 12, 2015)

Equivalence Partitioning Test Cases:-

Tester Action and Input Data	Expected Outcome
(15, 8, 2010)	(14, 8, 2010)
(29, 2, 2000)	(28, 2, 2000)
(28, 2, 2013)	(27, 2, 2013)
(30, 4, 2015)	(29, 4, 2015)
(30, 6, 2015)	(29, 6, 2015)
(0, 10, 2015)	An Error message
(-1, 10, 2015)	An Error message
(1, 0, 2015)	An Error message
(1, -1, 2015)	An Error message
(1, 1, 2025)	An Error message

Boundary Value Analysis Test Cases:-

Tester Action and Input Data	Expected Outcome
(1, 1, 2015)	(31, 12, 2014)

(28, 2, 2013)	(27, 2, 2013)
(29, 2, 2016)	(28, 2, 2016)
(31, 12, 2015)	(30, 12, 2015)
(1, 1, 1900)	(31, 12, 1899)
(1, 1, 2015)	(31, 12, 2014)
(30, 4, 2015)	(29, 4, 2015)
(31, 12, 2015)	(30, 12, 2015)

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

Using the previously defined test cases, you can input the values manually. Here are some test cases you can use:

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.

If they match, the test case passes; if not, it fails.

Q-2) :-

a) Identify the equivalence classes for the system

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)

P1)

Equivalence Classes:-

- **Class 1**: Empty array → 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 does not exist in the array → Output: -1
- **Class 5**: Value exists with duplicates → Output: Index of the first occurrence.

Test Cases:-

Here's a table summarizing the test cases for the linearSearch function, including the input, expected outcome, and the equivalence class each case covers:

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

This table provides a clear overview of the test cases, the inputs provided, the expected outputs, and the corresponding equivalence classes each test case covers.

P2)

Equivalence Classes:-

- **Class 1**: Empty array → Output: 0
- **Class 2**: Value exists once → Output: 1
- **Class 3**: Value exists multiple times → Output: Count of occurrences (e.g., n if v appears n times)
- **Class 4**: Value does not exist → Output: 0
- **Class 5**: All elements are equal to v → Output: Length of the array

Test Cases:-

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

P3)

Equivalence Classes:-

- **Class 1**: Empty array → 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 does not exist (less than smallest element) → Output: -1
- **Class 6**: Value does not exist (greater than largest element) → Output: -1
- **Class 7**: Value does not exist (between two elements) → Output: -1
- **Class 8**: Value exists with duplicates → Output: Index of any occurrence of v

Test Cases:-

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

P4)

Equivalence Classes:-

- **Class 1**: Invalid triangle (non-positive sides) → Output: INVALID

- **Class 2**: Invalid triangle (triangle inequality not satisfied) → 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
(-1, 2, 3)	INVALID	1
(1, 1, 1)	EQUILATERAL	3
(2, 2, 3)	ISOSCELES	4
(2, 3, 4)	SCALENE	5
(5, 2, 2)	ISOSCELES	4
(1, 2, 3)	INVALID	2
(3, 3, 6)	INVALID	2
(2, 5, 3)	SCALENE	5
(7, 3, 10)	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 equal to s2 → Output: true

Test Cases:-

Input (s1, s2)	Expected Outcome	Covers Equivalence Class

("abc", "abcdef")	true	2
("abc", "ab")	false	3
Input (s1, s2)	**Expected Outcome**	**Covers Equivalence Class**
("abc", "xyzabc")	false	3
("", "abcdef")	true	4
("a", "")	false	5
("abc", "abc")	true	6
("longerPrefix", "short")	false	1
("abc", "abcde")	true	2
("prefix", "pre")	false	3
("xyz", "xyzxyz")	true	2

P6)

a) Identifying the Equivalence Classes:-

Valid Triangle Types:

- **Equilateral Triangle**: Side A = Side B = Side C
- **Isosceles Triangle**: Side A = Side B, or Side A = Side C, or Side B = Side C
- **Scalene Triangle**: All sides unequal ($A \neq B \neq C$)
- **Right-Angled Triangle**: $A^2 + B^2 = C^2$ (Pythagorean theorem) or its permutations

Invalid Triangle Cases:

- **Not a Triangle**: $A + B \leq C$, $A + C \leq B$, or $B + C \leq A$
- **Non-positive Input**: Any side A, B, or C is less than or equal to zero

b) Test Cases Covering the Identified Equivalence Classes:-

Input (A, B, C)	Expected Output	Equivalence Classes Covered
(6, 6, 6)	Equilateral Triangle	Equilateral Triangle

Input (A, B, C)	Expected Output	Equivalence Classes Covered
(7, 7, 8)	Isosceles Triangle	Isosceles Triangle
(5, 7, 9)	Scalene Triangle	Scalene Triangle
(6, 8, 10)	Right-Angled Triangle	Right-Angled Triangle
(4, 5, 10)	Not a Triangle	Not a Triangle
(0, 5, 8)	Invalid	Non-positive Input

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

Input (A, B, C)	Expected Output
(4, 5, 6)	Scalene Triangle
(6, 7, 12)	Scalene Triangle
(6, 7, 13)	Not a Triangle
(5, 7, 11)	Scalene Triangle

d) Boundary Condition A = C (Isosceles Triangle):-

Input (A, B, C)	Expected Output
(6, 7, 6)	Isosceles Triangle
(7, 10, 10)	Isosceles Triangle
(5, 9, 14)	Not a Triangle
(9, 9, 9)	Equilateral Triangle

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

Input (A, B, C)	Expected Output
(6, 6, 6)	Equilateral Triangle
(8, 8, 8)	Equilateral Triangle
(7, 8, 14)	Not a Triangle
(7, 8, 13)	Scalene Triangle

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

Input (A, B, C)	Expected Output
(6, 8, 10)	Right-Angled Triangle
(9, 12, 15)	Right-Angled Triangle
(6, 9, 14)	Not a Triangle
(7, 10, 12)	Scalene Triangle

g) Non-Triangle Case:-

Input (A, B, C)	Expected Output
(5, 6, 7)	Scalene Triangle
(7, 12, 20)	Not a Triangle
(5, 9, 14)	Not a Triangle
(6, 8, 14)	Scalene Triangle

h) Non-Positive Input Case:-

Input (A, B, C)	Expected Output
(4, 6, 0)	Invalid
(5, 7, -3)	Invalid
(0, 8, 10)	Invalid
(-4, 6, 9)	Invalid