Software Engineering

LAB 8

Functional Testing (Black-Box)

Name : Kathan Kadiya
Student ID : 202201175

Program Specification and Test Cases

1. Program Specification :

- **Input**: Triple of day, month, and year
- **Input ranges**:
  Valid **month** from **1 to 12**
  Valid **days** are from **1 to 31**
  Valid **year** from **1900 to 2015**
- **Output**: Previous date or "Invalid date"

2. Test Suite :

2.1 Equivalence Classes :-

- **Valid cases** :
  - Normal days (excluding month end or year end)
  - Month end (excluding year end)
  - Year end (December 31)
  - Leap year (February 29)
- **Invalid cases** :
  - Invalid month **(< 1 or > 12)**
  - Invalid day (< 1 or > max days in month)
  - Invalid year **(<1900 or >2015)**
  - Invalid day for specific month (e.g., February 30)

2.2 Boundary Value Analysis :-

- First day of year: January 1 of any valid year
- Last day of year: December 31 of any valid year

- First day of month: DD 1 of any valid year
- Last day of month: 30/31, MM YYYY (in February its 28/29)
- Minimum and maximum valid year: 1900 and 2015

2.3 Test Cases

Tester Action and Input Data	Expected Outcome	Remarks
Date, month, year	An Error message	Invalid input(should be numbers)
15, 7, 1990	14, 7, 1990	Normal day
1, 7, 1995	30, 7, 1995	Month end
1, 1, 1998	31, 12, 1997	Year end
1, 10, 1990	29, 2, 1990	Leap year
1, 10, 1999	28, 2, 1999	Non-leap year
0, 7, 1990	Invalid date	Invalid day (too low)
32, 7, 1990	Invalid date	Invalid day (too high)
15, 0, 1990	Invalid date	Invalid month (too low)
15, 110, 1989	Invalid date	Invalid month (too high)
15, 7, 1899	Invalid date	Invalid year (too low)
15, 7, 2016	Invalid date	Invalid year (too high)
31, 4, 1989	Invalid date	Invalid day for April
29, 2, 1999	Invalid date	Invalid day for Februaryin non-leap year
1, 1, 1900	31, 12, 1899	Minimumvalid year - 1

31, 12, 2015	30, 12, 2015	Maximumvalid year
1, 1, 1989	31, 12, 1999	First day ofyear
31, 12, 1989	30, 12, 1989	Last day ofyear
1, 5, 1989	30, 4, 1989	First day ofmonth
31, 5, 1989	30, 5, 1989	Last day of31-day month
30, 4, 1989	29, 4, 1989	Last day of30-day month
29, 2, 1989	28, 2, 1989	Last day ofFebruary in leap year
28, 2, 1999	27, 2, 1999	Last day ofFebruary in non-leap year

Cpp Code for the same :

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

// Function to check if a year is a leap year
bool isLeapYear(int year) {
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}

// Function to get the number of days in a given month of a given year
int daysInMonth(int month, int year) {
    vector<int> days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (month == 2 && isLeapYear(year)) {
        return 29;
    }
    return days[month - 1];
}

// Function to calculate the previous date
string previousDate(int day, int month, int year) {
    if (!(1 <= month && month <= 12 && 1900 <= year && year <= 2015)) {
        return "Invalid date";
    }

    int maxDays = daysInMonth(month, year);
    if (!(1 <= day && day <= maxDays)) {
        return "Invalid date";
    }

    if (day > 1) {
        return to_string(day - 1) + ", " + to_string(month) + ", " + to_string(year);
    } else if (month > 1) {
        int prevMonth = month - 1;
        return to_string(daysInMonth(prevMonth, year)) + ", " + to_string(prevMonth) + ", " + to_string(year);
    } else {
        return "31, 12, " + to_string(year - 1);
    }
}
```

```cpp
// Function to run the test cases
void runTests() {
    vector<pair<vector<int>, string>> testCases = {
        {{15, 6, 2000}, "14, 6, 2000"},
        {{1, 7, 2010}, "30, 6, 2010"},
        {{1, 1, 2005}, "31, 12, 2004"},
        {{1, 3, 2000}, "29, 2, 2000"},
        {{1, 3, 2001}, "28, 2, 2001"},
        {{0, 6, 2000}, "Invalid date"},
        {{32, 6, 2000}, "Invalid date"},
        {{15, 0, 2000}, "Invalid date"},
        {{15, 13, 2000}, "Invalid date"},
        {{15, 6, 1899}, "Invalid date"},
        {{15, 6, 2016}, "Invalid date"},
        {{31, 4, 2000}, "Invalid date"},
        {{29, 2, 2001}, "Invalid date"},
        {{1, 1, 1900}, "31, 12, 1899"},
        {{31, 12, 2015}, "30, 12, 2015"},
        {{1, 1, 2000}, "31, 12, 1999"},
        {{31, 12, 2000}, "30, 12, 2000"},
        {{1, 5, 2000}, "30, 4, 2000"},
        {{31, 5, 2000}, "30, 5, 2000"},
        {{30, 4, 2000}, "29, 4, 2000"},
        {{29, 2, 2000}, "28, 2, 2000"},
        {{28, 2, 2001}, "27, 2, 2001"}
    };

    for (int i = 0; i < testCases.size(); i++) {
        vector<int> input = testCases[i].first;
        string expected = testCases[i].second;
        string result = previousDate(input[0], input[1], input[2]);
        cout << "Test " << i + 1 << ": " << (result == expected ? "PASS" : "FAIL") << endl;
        cout << "  Input: " << input[0] << ", " << input[1] << ", " << input[2] << endl;
        cout << "  Expected: " << expected << endl;
        cout << "  Actual: " << result << endl;
        cout << endl;
    }
}

int main() {
    runTests();
    return 0;
}
```

Problem 1:

Equivalence Partitioning

Input Data	Expected Outcome
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
-1, {-1, 0, 1}	0
1, {}	-1
4, {4}	0
1, {1, 2, 3}	0
3, {1, 2, 3}	2
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Boundary Value Analysis:

Input Data	Expected Outcome
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem 2 :

Equivalence Partitioning:

Input Data	Expected Outcome
5, {1, 2, 3}	0
2, {1, 2, 3}	1
-1, {-1, 0, 1}	1
1, {}	0
4, {4, 4, 4}	3
1, {1, 2, 3, 1, 1}	3
3, {1, 2, 3, 3, 3, 3}	4
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Boundary Value Analysis:

Input Data	Expected Outcome
5, {}	0
-2147483648, {-2147483648, 0, 2147483647}	1
2147483647, {-2147483648, 0, 2147483647}	1
1, {1, 2}	1
2, {1, 2, 2}	2
4, {1, 2, 3}	0
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem 3 :

Equivalence Partitioning:

Input Data	Expected Outcome
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
1, {1, 2, 3}	0
3, {1, 2, 3}	2
4, {1, 4, 7, 8}	1
0, {0, 1, 2, 3}	0
100, {10, 20, 30, 100}	3
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Boundary Value Analysis:

Input Data	Expected Outcome
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem 4 :

Equivalence Partitioning:

Input Data	Expected Outcome
3, 3, 3	EQUILATERAL (0)
3, 3, 2	ISOSCELES (1)
3, 4, 5	SCALENE (2)
1, 2, 3	INVALID (3)
1, 1, 2	INVALID (3)
5, 1, 1	INVALID (3)
2, 2, 3	ISOSCELES (1)
0, 1, 1	An Error message
1, 0, 1	An Error message

Boundary Value Analysis:

Input Data	Expected Outcome
1, 1, 1	EQUILATERAL (0)
1, 1, 2	INVALID (3)
2, 2, 4	INVALID (3)
2, 3, 5	INVALID (3)
3, 4, 7	INVALID (3)
1, 2, 2	ISOSCELES (1)
1, 2, 3	INVALID (3)
0, 1, 1	An Error message
1, 1, 0	An Error message

Problem 5:

Equivalence Partitioning:

Input Data	Expected Outcome
"pre", "prefix"	true
"pre", "postfix"	false
"prefix", "pre"	false
"test", "test"	true
"", "anything"	true
"anything", ""	false
"pre", "preparation"	true
null, "prefix"	An Error message
"prefix", null	An Error message

Boundary Value Analysis:

Input Data	Expected Outcome
"test", ""	false
"a", "a"	true
"a", "b"	false
"", ""	true
"start", "startmiddle"	true
"longprefix", "short"	false
"short", "longprefix"	true
null, "anything"	An Error message
"anything", null	An Error message

Problem 6:

a) Identify the Equivalence Classes

Equilateral Triangle: All three sides are equal.

Isosceles Triangle: Exactly two sides are equal.

Scalene Triangle: No sides are equal.

Right-Angled Triangle: Satisfies a2+b2=c2.

Invalid Triangle: Does not satisfy the triangle inequality a+b>c.

Non-positive Input: One or more sides are non-positive.

b) Identify Test Cases to Cover the Equivalence Classes

Equivalence Partitioning:

Input Data	Expected Outcome	Equivalence Class
3.0, 3.0, 3.0	Equilateral	Equilateral Triangle
3.0, 3.0, 2.0	Isosceles	Isosceles Triangle
3.0, 4.0, 5.0	Scalene	Scalene Triangle
3.0, 4.0, 0.0	Invalid	Invalid Triangle
0.0, 0.0, 0.0	Invalid	Non-positive Input
5.0, 1.0, 1.0	Invalid	Invalid Triangle
3.0, 4.0, 6.0	Scalene	Scalene Triangle

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

Boundary Value Analysis:

Input Data	Expected Outcome
2.0, 2.0, 3.99	Scalene
2.0, 2.0, 4.0	Invalid
2.0, 2.0, 4.01	Invalid

d) Boundary Condition A = C (Isosceles Triangle)

Boundary Value Analysis:

Input Data	Expected Outcome
3.0, 4.0, 3.0	Isosceles
3.0, 3.0, 3.0	Equilateral
3.0, 3.0, 4.0	Isosceles

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

Boundary Value Analysis:

Input Data	Expected Outcome
3.0, 3.0, 3.0	Equilateral
1.0, 1.0, 1.0	Equilateral
2.5, 2.5, 2.5	Equilateral

f) Boundary Condition A2+B2=C2 (Right-Angle Triangle)

Boundary Value Analysis:

Input Data	Expected Outcome
3.0, 4.0, 5.0	Right Angled
6.0, 8.0, 10.0	Right Angled
5.0, 12.0, 13.0	Right Angled

g) Non-Triangle Case

Boundary Value Analysis:

Input Data	Expected Outcome
1.0, 2.0, 3.0	Invalid
1.0, 2.0, 4.0	Invalid
1.0, 1.0, 2.0	Invalid

h) Non-Positive Input

Boundary Value Analysis:

Input Data	Expected Outcome
0.0, 1.0, 1.0	Invalid
-1.0, 1.0, 1.0	Invalid
1.0, 0.0, 1.0	Invalid