LAB 8: Lab Session - Functional Testing (Black-Box)

Name: Harshit Kumar
Id: 202201034

Equivalence Classes

1. Valid Equivalence Classes:

Valid Dates:

Class1: Valid dates in a non-leap year.

Example: (1,1,2000) - January 1, 2000

Example: (31,12,2015) - December 31, 2015

Class 2: Valid dates in a leap year.

Example: (29,2,2020) - February 29,2020

Class 3: Boundary dates.

Example: (1,1,1900) - January 1,1900

Example: (1,1,2015) - January 1,2015

Example: (1, 3, 2015) - March 1, 2015 (previous date should be Feb 28)

2. Invalid Equivalence Classes:

Invalid Dates:

Class 4: Day out of range.

Example: (32,1,2000) - Invalid day

Example: (0,1,2000) - Invalid day

Class 5: Month out of range.

Example: (15,0,2000) - Invalid month

Example: (15, 13, 2000) - Invalid month

Class 6: Year out of range.

Example: (15,1,1899) - Invalid year

Example: (15,1,2016) - Invalid year

Class 7: Invalid date combinations.

Example: (30, 2, 2000) - February 30 is invalid

Example: (31,4,2000) - April 31is invalid

TESTING TABLE:

TestCase	Day	Month	Year	Expected Output
TC1	1	1	2000	Previous date: 31/12/1999
TC2	31	12	2015	Previous date: 30/12/2015
TC3	29	2	2020	Previous date: 28/2/2020
TC4	32	1	2000	Invalid date
TC5	15	0	2000	Invalid date
TC6	15	13	2000	Invalid date
TC7	15	1	1899	Invalid date
TC8	15	1	2016	Invalid date
TC9	30	2	2000	Invalid date
TC10	31	4	2000	Invalid date

1. Test Cases Identified Using Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
(1, 1, 2000)	Previous date: 31/12/1999
(31, 12, 2015)	Previous date: 30/12/2015
(29, 2, 2012)	Previous date: 28/2/2012
(32, 1, 2000)	Invalid date
(15, 0, 2000)	Invalid date
(15, 13, 2000)	Invalid date
(15, 1, 1899)	Invalid date
(15, 1, 2016)	Invalid date
(30, 2, 2000)	Invalid date
(31, 4, 2000)	Invalid date

2. Test Cases Identified Using Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
(1,1,1900)	Invalid date (year out of range)
(1,1,2015)	Previous date: 31/12/2014
(31,12,1900)	Previous date: 30/12/1900
(31,12,2015)	Previous date: 30/12/2015
(29,2,1900)	Invalid date (not a leap year)
(29,2,2000)	Previous date: 28/2/2000
(1,1,2000)	Previous date: 31/12/1999
(31,12,2000)	Previous date: 30/12/2000
(1,13,2000)	Invalid date (month out of range)
(32,1,2000)	Invalid date (day out of range)

C++ implementation of the program:

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <sstream>
#include <vector>
#include <tuple>

std::string get_previous_date(int day, int month, int year) {
    // Struct to hold date information
    struct tm date = {0};
    date.tm_mday = day;
    date.tm_mon = month - 1; // Months are 0-11 in tm struct
    date.tm_year = year - 1900; // Years since 1900 in tm struct

    // Normalize and check the validity of the date
    if (mktime(&date) == -1) {
        return "Invalid date";
    }

    // Subtract one day (86400 seconds)
    const time_t one_day = 24 * 60 * 60;
    time_t date_time = mktime(&date) - one_day;

    // Convert the resulting time_t back to struct tm
    struct tm* previous_date = localtime(&date_time);

    // Format the output as dd/mm/yyyy
    std::ostringstream oss;
    oss << std::setfill('0') << std::setw(2) << previous_date->tm_mday << "/"
        << std::setfill('0') << std::setw(2) << previous_date->tm_mon + 1 << "/"
        << previous_date->tm_year + 1900;

    return oss.str();
}
```

```
// Test Suite
int main() {
 std::vector<std::tuple<int, int, int>> test_cases = {
 {1, 1, 2000}, {31, 12, 2015}, {29, 2, 2012}, {32, 1, 2000},
 {15, 0, 2000}, {15, 13, 2000}, {15, 1, 1899}, {15, 1, 2016},
 {30, 2, 2000}, {31, 4, 2000}, {1, 1, 1900}, {1, 1, 2015},
 {31, 12, 1900}, {31, 12, 2015}, {29, 2, 1900}, {29, 2, 2000},
 {1, 1, 2000}, {31, 12, 2000}, {1, 13, 2000}, {1, 1, 2000},
 {31, 1, 2000}, {32, 1, 2000}
 };
 for (const auto& test_case : test_cases) {
 int day, month, year;
 std::tie(day, month, year) = test_case;
 std::string expected_output = get_previous_date(day, month, year);
 std::cout << "Input: (" << day << ", " << month << ", " << year
 << ") - Expected Output: " << expected_output << std::endl;
 }
 return 0;
}
```

Execution and Verification

- Run the Program: Execute the modified program to see the outputs for each test case.
- Verify Outcomes: Compare the printed outputs against the expected outcomes listed in the tables above to check for correctness.

Program Analysis

Programs Overview

The assignment contains six programs (P1-P6) implementing various

algorithms:

1. *Linear Search Implementation*
2. *Count Item Occurrence*
3. *Binary Search in Sorted Array*
4. *Triangle Classification*
5. *String Prefix Checker*
6. *Enhanced Triangle Classification with Test Cases*

Let's analyze each program and develop test cases for P6 in detail.

Detailed Analysis

P1: Linear Search

```
int linearSearch(int v, int a[])
```

Purpose: Finds first occurrence of value v in array a
Return Value: Index of first occurrence or -1 if not found
Time Complexity: O(n)
Key Characteristics: Sequential search, no preprocessing required

P2: Count Item

```
int countItem(int v, int a[])
```

Purpose: Counts occurrences of value v in array a
Return Value: Total count of occurrences
Time Complexity: O(n)
Key Characteristics: Complete array traversal required

P3: Binary Search

```
int binarySearch(int v, int a[])
```

Purpose: Efficiently finds value in sorted array
Prerequisite: Array must be sorted
Time Complexity: O(log n)
Key Characteristics: Divide and conquer approach

P4: Triangle Classification

```
int triangle(int a, int b, int c)
```

Purpose: Classifies triangle type based on side lengths
Return Values:
EQUILATERAL(0)
ISOSCELES(1)
SCALENE(2)
INVALID(3)

P5: String Prefix Check

```
boolean prefix(String s1, String s2)
```

Purpose: Checks if s1 is prefix of s2
Return Value: Boolean indicating prefix status
Key Characteristics: String comparison with length check

Detailed Analysis of P6

A) Equivalence Classes Identification

1. Valid Triangle Classes:
   EC1:Equilateraltriangles(allsidesequal)
   EC2: Isosceles triangles (two sides equal)
   EC3: Scalene triangles (no sides equal)
   EC4: Right-angled triangles
2. Invalid Triangle Classes:
   EC5: Non-triangle cases (sum of any two sides ≤ third side)
   EC6: Zero or negative length sides
   EC7: Non-numeric inputs

B) Test Cases for Equivalence Classes

Test ID	Test Case	Equivalence Class	Expected Output
TC1	(5, 5, 5)	EC1	Equilateral
TC2	(5, 5, 3)	EC2	Isosceles
TC3	(3, 4, 5)	EC3, EC4	Right-angled Scalene
TC4	(2, 3, 6)	EC5	Invalid
TC5	(0, 4, 5)	EC6	Invalid
TC6	(7, 8, 9)	EC3	Scalene

C) Boundary Testing for A + B > C

TestID	TestCase	Description	Expected Output
TC7	(4, 5, 8.9)	Just within boundary	Valid Scalene
TC8	(4, 5, 9)	Onboundary	Invalid
TC9	(4, 5, 9.1)	Just outside boundary	Invalid

D) Boundary Testing for Isosceles (A = C)

Test ID	Test Case	Description	Expected Output
TC10	(5, 3, 5)	Perfect isosceles	Isosceles
TC11	(5, 3, 5.0001)	Just outside equality	Scalene
TC12	(5, 3, 4.9999)	Just inside equality	Scalene

E) Boundary Testing for Equilateral (A = B = C)

Test ID	Test Case	Description	Expected Output
TC13	(5, 5, 5)	Perfect equilateral	Equilateral
TC14	(5, 5, 5.0001)	Just outside equality	Isosceles
TC15	(5.0001, 5, 5)	Just outside equality	Isosceles

F) Right-Angle Triangle Testing ($A^2 + B^2 = C^2$)

TestID	TestCase	Description	Expected Output
TC16	(3, 4, 5)	Perfect right angle	Right-angled
TC17	(5, 12, 13)	Perfect right angle	Right-angled
TC18	(8, 15, 17)	Perfect right angle	Right-angled

G) Non-Triangle Case Testing

TestID	TestCase	Description	Expected Output
TC19	(2, 3, 5)	Sum equals	Invalid
TC20	(2, 3, 4.9)	Just valid	Valid
TC21	(2, 3, 5.1)	Just invalid	Invalid

H) Non-Positive Input Testing

Test ID	Test Case	Description	Expected Output
TC22	(0, 4, 5)	Zero length	Invalid
TC23	(-1, 3, 5)	Negative length	Invalid
TC24	(3, -2, 5)	Negative length	Invalid

Test Implementation Guidelines

1. Setup Requirements:
   Floating-point comparison tolerance should be defined
   Input validation mechanisms should be in place
   Error handling should be implemented
2. Test Execution:
   Tests should be automated where possible
   Each test case should be independent
   Results should be logged for analysis
3. Validation:
   Compare actual vs expected results
   Document any deviations
   Track test coverage

Conclusion

The test cases designed cover all major equivalence classes and boundary conditions for the triangle classification program. Special attention has been paid to:

- Boundaryconditions
- Invalid inputs
- Special cases (right angles, equilateral, isosceles)
- Error conditions