Dhirubhai Ambani Institute of Information and Communication Technology

IT314-Software Engineering

Lab -08

Ayush Chaudhari-202201517

Q.1. Consider a program for determining the previous date. Its input is triple of day, month and year
with the following ranges 1 <= month <= 12, 1 <= day <= 31, 1900 <= year <= 2015.The possible output
dates would be previous date or invalid date. Design the equivalence class test cases?

Write a set of test cases (i.e., test suite) – specific set of data – to properly test the programs. Your
test suite should include both correct and incorrect inputs.
1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

Equivalence Partitioning and Boundary Value Analysis Test Cases

Equivalence Partitioning

Test Input	Tester Action & Input Data	Expected Outcome
Year: (2, 11, 1887)	Enter invalid year	Error message: Year is less than 1900
Year: (5, 7, 2016)	Enter invalid year	Error message: Year is greater than 2015
Year: (19, 8, 2020)	Enter invalid year	Error message: Year is greater than 2015
Month: (17, 0, 2013)	Enter invalid month	Error message: Month is less than 1
Month: (28, 11, 2003)	Enter valid month	Previous date: 27 November, 2003
Month: (3, 23, 2001)	Enter invalid month	Error message: Month is greater than 12
Day: (0, 3, 2005)	Enter invalid day	Error message: Day is less than 1
Day: (2, 6, 2007)	Enter valid day	Previous date: 1 June, 2007
Day: (35, 9, 1999)	Enter invalid day	Error message: Day is greater than 31

Leap Year: (29, 2, 2000)	Enter valid leap year	Previous date: 28 February, 2000
Leap Year: (29, 2, 2004)	Enter valid leap year	Previous date: 28 February, 2004
Not a Leap Year: (29, 2, 1900)	Enter invalid leap year	Error message: Year is not a leap year
Invalid Day: (29, 2, 2001)	Enter invalid day for February	Error message: Day is greater than valid days for February

Boundary Value Analysis

Test Input	Tester Action & Input Data	Expected Outcome
Year: (1, 1, 1900)	Enter minimum valid year	Error message: Month, Day, and Year at minimum values
Year: (31, 12, 2015)	Enter maximum valid year	Previous date: 30 December, 2015
Year: (2, 1, 1900)	Enter day after 1 January 1900	Previous date: 31 December, 1899
Month: (1, 1, 1900)	Enter minimum valid month	Error message: Month, Day, and Year at minimum values
Month: (31, 12, 2015)	Enter maximum valid month	Previous date: 30 December, 2015
Month: (1, 12, 2016)	Enter invalid year	Error message: Year is greater than maximum value
Day: (1, 1, 1900)	Enter minimum valid day	Error message: Month, Day, and Year at minimum values

Day: (31, 12, 2015)	Enter maximum valid day	Previous date: 30 December, 2015
Day: (31, 12, 2016)	Enter invalid year	Error message: Year is greater than its maximum value
Day: (1, 1, 1901)	Enter day after 31 December 1900	Previous date: 31 December, 1900
Day (Leap Year): (1, 2, 2000)	Enter valid leap year day	Previous date: 31 January, 2000

2. 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.

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

using namespace std;

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

// Function to validate the date input
bool isValidDate(int day, int month, int year) {
    // Check year range
    if (year < 1900 || year > 2015) {
        return false; // Year is out of valid range
    }
    // Check month range
    if (month < 1 || month > 12) {
        return false; // Month is out of valid range
    }
    // Check day range
    if (day < 1 || day > 31) {
        return false; // Day is out of valid range
    }
```

```
    // Validate the number of days in the month
    // Months with only 30 days
    if ((month == 4 || month == 6 || month == 9 || month == 11) && day
> 30) {
        return false; // Invalid day for this month
    }
    // Check February days
    if (month == 2) {
        // If it's a leap year, February can have 29 days
        if (isLeapYear(year)) {
            return day <= 29;
        } else {
            return day <= 28; // Non-leap year February has 28 days
        }
    }

    return true; // All checks passed, date is valid
}

// Function to find the previous date
tuple<int, int, int> getPreviousDate(int day, int month, int year) {
    // Check if the date is valid before processing
    if (!isValidDate(day, month, year)) {
        throw invalid_argument("Invalid date provided.");
    }

    // Logic to calculate the previous date
    if (day > 1) {
        // Just decrease the day by one
        return make_tuple(day - 1, month, year);
    } else {
        // If day is 1, we need to adjust the month and possibly the
year
        if (month == 1) { // If the month is January
            return make_tuple(31, 12, year - 1); // Go to December of
the previous year
        } else if (month == 3) { // If the month is March
            // Return the last day of February based on leap year
            if (isLeapYear(year)) {
                return make_tuple(29, 2, year); // Leap year
            } else {
                return make_tuple(28, 2, year); // Non-leap year
            }
        } else {
```

```
            // For other months, return the last day of the previous
month
            return make_tuple(31, month - 1, year);
        }
    }
}

// Main function to run the program
int main() {
    // List of test cases with various scenarios
    vector<tuple<int, int, int>> testCases = {
        {2, 11, 1887},  // Invalid year
        {5, 7, 2016},   // Invalid year
        {19, 8, 2020},  // Invalid year
        {17, 0, 2013},  // Invalid month
        {28, 11, 2003}, // Valid date
        {3, 23, 2001},  // Invalid month
        {0, 3, 2005},   // Invalid day
        {2, 6, 2007},   // Valid date
        {35, 9, 1999},  // Invalid day
        {1, 1, 1900},   // Valid minimum date
        {31, 12, 2015}, // Valid maximum date
        {2, 1, 1900}    // Valid date
    };

    // Iterate through test cases and execute
    for (const auto& testCase : testCases) {
        int day = get<0>(testCase);
        int month = get<1>(testCase);
        int year = get<2>(testCase);

        try {
            auto previousDate = getPreviousDate(day, month, year);
            cout << "Input: (" << day << ", " << month << ", " << year
<< ") -> "
                 << "Previous Date: (" << get<0>(previousDate) << ", "
                 << get<1>(previousDate) << ", " <<
get<2>(previousDate) << ")\n";
        } catch (const invalid_argument& e) {
            cout << "Input: (" << day << ", " << month << ", " << year
<< ") -> "
                 << e.what() << endl; // Display error message for
invalid date
        }
    }
```

```
    return 0; // End of program
}
```

Output

```
Input: (2, 11, 1887) -> Invalid date provided.
Input: (5, 7, 2016) -> Invalid date provided.
Input: (19, 8, 2020) -> Invalid date provided.
Input: (17, 0, 2013) -> Invalid date provided.
Input: (28, 11, 2003) -> Previous Date: (27, 11, 2003)
Input: (3, 23, 2001) -> Invalid date provided.
Input: (0, 3, 2005) -> Invalid date provided.
Input: (2, 6, 2007) -> Previous Date: (1, 6, 2007)
Input: (35, 9, 1999) -> Invalid date provided.
Input: (1, 1, 1900) -> Previous Date: (31, 12, 1899)
Input: (31, 12, 2015) -> Previous Date: (30, 12, 2015)
Input: (2, 1, 1900) -> Previous Date: (1, 1, 1900)
```

Q.2. Programs:

P1: Linear Search

Derived Equivalence Classes

Class Number	Description
1	Value v is present in the array (valid).
2	Value v is not present in the array (invalid).
3	Array is empty (invalid).
4	Array contains one element (valid and invalid depending on whether it matches v).
5	Array contains multiple elements (valid).

6	All elements in the array are positive integers (valid).
7	All elements in the array are negative integers (valid).
8	All elements in the array include zero (valid).
9	Mixed integers (positive, negative, zero) in the array (valid).

Black Box Test Cases

Test Case	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
T1	v=3,a=[1,2,3]v = 3, a = [1, 2, 3]v=3,a=[1,2,3]	Returns 2	1, 4, 5	Valid	v is present at index 2
T2	v=4,a=[1,2,3]v = 4, a = [1, 2, 3]v=4,a=[1,2,3]	Returns -1	2, 3, 5	Invalid	v is not present
T3	v=1,a=[]v = 1, a = []v=1,a=[]	Returns -1	3	Invalid	Array is empty
T4	v=5,a=[5]v = 5, a = [5]v=5,a=[5]	Returns 0	1, 4	Valid	v is present at index 0
T5	v=2,a=[1,2,1]v = 2, a = [1, 2, 1]v=2,a=[1,2,1]	Returns 1	1, 5	Valid	First occurrence of v at index 1
T6	v=0,a=[0,−1,−2]v = 0, a = [0, -1, -2]v=0,a=[0,−1,−2]	Returns 0	1, 5	Valid	v is present at index 0
T7	v=−1,a=[−1,0]v = -1, a = [-1, 0]v=−1,a=[−1,0]	Returns 0	1, 5	Valid	v is present at index 0
T8	v=3,a=[−1,0]v = 3, a = [-1, 0]v=3,a=[−1,0]	Returns -1	2, 5	Invalid	v is not present

T9	v=1,a=[1,2,3]v = 1, a = [1, 2, 3]v=1,a=[1,2,3]	Returns 0	1, 5	Valid	v is present at index 0
T10	v=1,a=[2,3,4]v = 1, a = [2, 3, 4]v=1,a=[2,3,4]	Returns -1	2, 5	Invalid	v is not present

Boundary Value Analysis

Boundary Condition	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
Minimum size of array	v=1,a=[]v = 1, a = []v=1,a=[]	Returns -1	3	Invalid	Array is empty
One element present	v=1,a=[1]v = 1, a = [1]v=1,a=[1]	Returns 1	1, 4	Valid	v is present
One element not present	v=2,a=[1]v = 2, a = [1]v=2,a=[1]	Returns -1	2, 4	Invalid	v is not present
Multiple elements, all match	v=2,a=[2,2,2]v = 2, a = [2, 2, 2]v=2,a=[2,2,2]	Returns 0	1, 6	Valid	All elements match v
Multiple elements, none match	v=3,a=[1,2,4]v = 3, a = [1, 2, 4]v=3,a=[1,2,4]	Returns -1	2, 5	Invalid	No match for v
Mixed integers	v=0,a=[−1,0,1]v = 0, a = [-1, 0, 1]v=0,a=[−1,0,1]	Returns 1	1, 10	Valid	v is present

Code Implementation with Test Cases

```
#include <iostream>
#include <vector>

int linearSearch(int v, const std::vector<int>& a) {
    for (int i = 0; i < a.size(); i++) {
```

```
        if (a[i] == v)
            return i;  // Found at index i
    }
    return -1; // Not found
}

int main() {
    // Test cases
    std::cout << "T1: " << linearSearch(3, {1, 2, 3}) << " (Expected: 2)\n"; // Found at index 2
    std::cout << "T2: " << linearSearch(4, {1, 2, 3}) << " (Expected: -1)\n"; // Not found
    std::cout << "T3: " << linearSearch(1, {}) << " (Expected: -1)\n"; // Empty array
    std::cout << "T4: " << linearSearch(5, {5}) << " (Expected: 0)\n"; // Found at index 0
    std::cout << "T5: " << linearSearch(2, {1, 2, 1}) << " (Expected: 1)\n"; // First occurrence
    std::cout << "T6: " << linearSearch(0, {0, -1, -2}) << " (Expected: 0)\n"; // Found at index 0
    std::cout << "T7: " << linearSearch(-1, {-1, 0}) << " (Expected: 0)\n"; // Found at index 0
    std::cout << "T8: " << linearSearch(3, {-1, 0}) << " (Expected: -1)\n"; // Not found
    std::cout << "T9: " << linearSearch(1, {1, 2, 3}) << " (Expected: 0)\n"; // Found at index 0
    std::cout << "T10: " << linearSearch(1, {2, 3, 4}) << " (Expected: -1)\n"; // Not found

    return 0;
}
```

Output
T1: 2 (Expected: 2)
T2: -1 (Expected: -1)
T3: -1 (Expected: -1)
T4: 0 (Expected: 0)
T5: 1 (Expected: 1)
T6: 0 (Expected: 0)
T7: 0 (Expected: 0)
T8: -1 (Expected: -1)
T9: 0 (Expected: 0)
T10: -1 (Expected: -1)

P2: Count Item

Derived Equivalence Classes

Class Number	Description

1	Value v is present in the array (valid).
2	Value v is not present in the array (invalid).
3	Array is empty (invalid).
4	Array contains one element (valid and invalid depending on whether it matches v).
5	Array contains multiple elements (valid).
6	All elements in the array are the same as v (valid).
7	All elements in the array are different from v (invalid).
8	Array contains positive integers (valid).
9	Array contains negative integers (valid).
10	Array contains a mix of positive, negative, and zero (valid).

Black Box Test Cases

Test Case	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
T1	v=2,a=[1,2,3,2,2]v = 2, a = [1, 2, 3, 2, 2]v=2,a=[1,2,3,2,2]	Returns 3	1, 4, 5, 6	Valid	v is present multiple times
T2	v=4,a=[1,2,3]v = 4, a = [1, 2, 3]v=4,a=[1,2,3]	Returns 0	2, 3, 5	Invalid	v is not present
T3	v=1,a=[]v = 1, a = []v=1,a=[]	Returns 0	3	Invalid	Array is empty

T4	v=1,a=[1]v = 1, a = [1]v=1,a=[1]	Returns 1	1, 4	Valid	v is present
T5	v=2,a=[2,2,2,2,3]v = 2, a = [2, 2, 2, 2, 3]v=2,a=[2,2,2,2,3]	Returns 4	1, 6	Valid	All elements match v
T6	v=3,a=[1,1,1,1,1]v = 3, a = [1, 1, 1, 1, 1]v=3,a=[1,1,1,1,1]	Returns 0	2, 7	Invalid	No match for v
T7	v=−1,a=[−1,−1,0,1,1]v = -1, a = [-1, -1, 0, 1, 1]v=−1,a=[−1,−1,0,1,1]	Returns 2	1, 9	Valid	vvv is present
T8	v=0,a=[1,2,3]v = 0, a = [1, 2, 3]v=0,a=[1,2,3]	Returns 0	2, 5	Invalid	v is not present
T9	v=2,a=[2]v = 2, a = [2]v=2,a=[2]	Returns 1	1, 4	Valid	v is present
T10	v=5,a=[1,2,3]v = 5, a = [1, 2, 3]v=5,a=[1,2,3]	Returns 0	2, 5	Invalid	v is not present

Boundary Value Analysis

Boundary Condition	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
Minimum size of array	v=1,a=[]v = 1, a = []v=1,a=[]	Returns 0	3	Invalid	Array is empty
One element present	v=1,a=[1]v = 1, a = [1]v=1,a=[1]	Returns 1	1, 4	Valid	v is present
One element not present	v=2,a=[1]v = 2, a = [1]v=2,a=[1]	Returns 0	2, 4	Invalid	v is not present

Multiple elements, all match	v=2,a=[2,2,2]v = 2, a = [2, 2, 2]v=2,a=[2,2,2]	Returns 3	1, 6	Valid	All elements match v
Multiple elements, none match	v=3,a=[1,2,4]v = 3, a = [1, 2, 4]v=3,a=[1,2,4]	Returns 0	2, 5	Invalid	No match for v
Mixed integers	v=0,a=[−1,0,1]v = 0, a = [-1, 0, 1]v=0,a=[−1,0,1]	Returns 1	1, 10	Valid	v is present

Code Implementation with Test Cases

```
#include <iostream>
#include <vector>

int countItem(int v, const std::vector<int>& a) {
    int count = 0;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == v)
            count++;
    }
    return count; // Return count of occurrences
}

int main() {
    // Test cases
    std::cout << "T1: " << countItem(2, {1, 2, 3, 2, 2}) << "
(Expected: 3)\n"; // 3 occurrences
    std::cout << "T2: " << countItem(4, {1, 2, 3}) << " (Expected:
0)\n"; // Not found
    std::cout << "T3: " << countItem(1, {}) << " (Expected: 0)\n"; //
Empty array
    std::cout << "T4: " << countItem(1, {1}) << " (Expected: 1)\n"; //
Found at index 0
    std::cout << "T5: " << countItem(2, {2, 2, 2, 2, 3}) << "
(Expected: 4)\n"; // All match
```

```
    std::cout << "T6: " << countItem(3, {1, 1, 1, 1, 1}) << "
(Expected: 0)\n"; // No match
    std::cout << "T7: " << countItem(-1, {-1, -1, 0, 1, 1}) << "
(Expected: 2)\n"; // 2 occurrences
    std::cout << "T8: " << countItem(0, {1, 2, 3}) << " (Expected:
0)\n"; // Not found
    std::cout << "T9: " << countItem(2, {2}) << " (Expected: 1)\n"; //
Found at index 0
    std::cout << "T10: " << countItem(5, {1, 2, 3}) << " (Expected:
0)\n"; // Not found

    return 0;
}
```

Output:
T1: 3 (Expected: 3)
T2: 0 (Expected: 0)
T3: 0 (Expected: 0)
T4: 1 (Expected: 1)
T5: 4 (Expected: 4)
T6: 0 (Expected: 0)
T7: 2 (Expected: 2)
T8: 0 (Expected: 0)
T9: 1 (Expected: 1)
T10: 0 (Expected: 0)

P3: Binary Search

Derived Equivalence Classes

Class Number	Description
1	Value v is present in the array (valid).
2	Value v is not present in the array (invalid).
3	Array is empty (invalid).

4	Array contains one element (valid and invalid depending on whether it matches v).
5	Array is sorted in non-decreasing order (valid).
6	Array contains negative integers (valid).
7	Array contains positive integers (valid).
8	Mixed integers (positive, negative, zero) in the array (valid).

Black Box Test Cases

Test Case	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
T1	v=3,a=[1,2,3]v = 3, a = [1, 2, 3]v=3,a=[1,2,3]	Returns 2	1, 4, 5	Valid	v is present at index 2
T2	v=4,a=[1,2,3]v = 4, a = [1, 2, 3]v=4,a=[1,2,3]	Returns -1	2, 3, 5	Invalid	v is not present
T3	v=1,a=[]v = 1, a = []v=1,a=[]	Returns -1	3	Invalid	Array is empty
T4	v=5,a=[5]v = 5, a = [5]v=5,a=[5]	Returns 0	1, 4	Valid	v is present at index 0
T5	v=2,a=[1,2,2,3]v = 2, a = [1, 2, 2, 3]v=2,a=[1,2,2,3]	Returns 1	1, 5	Valid	First occurrence of v at index 1
T6	v=−1,a=[−1,0,1]v = -1, a = [-1, 0, 1]v=−1,a=[−1,0,1]	Returns 0	1, 5	Valid	Found at index 0
T7	v=2,a=[−1,0,1]v = 2, a = [-1, 0, 1]v=2,a=[−1,0,1]	Returns -1	2, 5	Invalid	v is not present

T8	v=0,a=[−1,0,1]v = 0, a = [-1, 0, 1]v=0,a=[−1,0,1]	Returns 1	1, 5	Valid	v is present at index 1
T9	v=5,a=[−2,0,1]v = 5, a = [-2, 0, 1]v=5,a=[−2,0,1]	Returns -1	2, 5	Invalid	v is not present
T10	v=2,a=[1,2,3]v = 2, a = [1, 2, 3]v=2,a=[1,2,3]	Returns 1	1, 5	Valid	v is present at index 1

Boundary Value Analysis

Boundary Condition	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
Minimum size of array	v=1,a=[]v = 1, a = []v=1,a=[]	Returns -1	3	Invalid	Array is empty
One element present	v=1,a=[1]v = 1, a = [1]v=1,a=[1]	Returns 0	1, 4	Valid	v is present
One element not present	v=2,a=[1]v = 2, a = [1]v=2,a=[1]	Returns -1	2, 4	Invalid	v is not present
Multiple elements, all match	v=2,a=[2,2,2,2]v = 2, a = [2, 2, 2, 2]v=2,a=[2,2,2,2]	Returns 0	1, 6	Valid	All elements match v
Multiple elements, none match	v=3,a=[1,2,4]v = 3, a = [1, 2, 4]v=3,a=[1,2,4]	Returns -1	2, 5	Invalid	No match for v

Mixed integers	v=0,a=[−1,0,1]v = 0, a = [-1, 0, 1]v=0,a=[−1,0,1]	Returns 1	1, 10	Valid	v is present

Code Implementation with Test Cases

```
#include <iostream>
#include <vector>

int binarySearch(int v, const std::vector<int>& a) {
    int left = 0, right = a.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (a[mid] == v)
            return mid; // Found at index mid
        else if (a[mid] < v)
            left = mid + 1; // Search in the right half
        else
            right = mid - 1; // Search in the left half
    }
    return -1; // Not found
}

int main() {
    // Test cases
    std::cout << "T1: " << binarySearch(3, {1, 2, 3}) << " (Expected:
2)\n"; // Found at index 2
    std::cout << "T2: " << binarySearch(4, {1, 2, 3}) << " (Expected:
-1)\n"; // Not found
    std::cout << "T3: " << binarySearch(1, {}) << " (Expected: -1)\n";
// Empty array
    std::cout << "T4: " << binarySearch(5, {5}) << " (Expected: 0)\n";
// Found at index 0
    std::cout << "T5: " << binarySearch(2, {1, 2, 2, 3}) << "
(Expected: 1)\n"; // First occurrence
```

```
    std::cout << "T6: " << binarySearch(-1, {-1, 0, 1}) << "
(Expected: 0)\n"; // Found at index 0
    std::cout << "T7: " << binarySearch(2, {-1, 0, 1}) << " (Expected:
-1)\n"; // Not found
    std::cout << "T8: " << binarySearch(0, {-1, 0, 1}) << " (Expected:
1)\n"; // Found at index 1
    std::cout << "T9: " << binarySearch(5, {-2, 0, 1}) << " (Expected:
-1)\n"; // Not found
    std::cout << "T10: " << binarySearch(2, {1, 2, 3}) << " (Expected:
1)\n"; // Found at index 1

    return 0;
}
```

Output:
T1: 2 (Expected: 2)
T2: -1 (Expected: -1)
T3: -1 (Expected: -1)
T4: 0 (Expected: 0)
T5: 1 (Expected: 1)
T6: 0 (Expected: 0)
T7: -1 (Expected: -1)
T8: 1 (Expected: 1)
T9: -1 (Expected: -1)
T10: 1 (Expected: 1)

P4: Prefix Function

Derived Equivalence Classes

Class Number	Description
1	s1 is a prefix of s2 (valid).
2	s1 is not a prefix of s2 (invalid).
3	s1 is longer than s2 (invalid).

4	s1 is empty (valid).
5	s2 is empty (invalid).
6	s1 and s2 are identical (valid).
7	s1 is one character long (valid/invalid depending on s2).

Black Box Test Cases

Test Case	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
T1	s1="pre",s2="prefix"s1 = "pre", s2 = "prefix"s1="pre",s2="prefix"	Returns true	1, 4	Valid	s1 is a prefix
T2	s1="fix",s2="prefix"s1 = "fix", s2 = "prefix"s1="fix",s2="prefix"	Returns false	2	Invalid	s1 is not a prefix
T3	s1="prefix",s2="pre"s1 = "prefix", s2 = "pre"s1="prefix",s2="pre"	Returns false	3	Invalid	s1 is longer than s2
T4	s1="",s2="hello"s1 = "", s2 = "hello"s1="",s2="hello"	Returns true	4	Valid	Empty prefix is a prefix of any string
T5	s1="hello",s2=""s1 = "hello", s2 = ""s1="hello",s2=""	Returns false	5	Invalid	Non-empty string cannot be a prefix of an empty string
T6	s1="test",s2="test"s1 = "test", s2 = "test"s1="test",s2="test"	Returns true	6	Valid	Identical strings
T7	s1="t",s2="test"s1 = "t", s2 = "test"s1="t",s2="test"	Returns true	1, 7	Valid	s1 is a prefix

T8	s1="test",s2="te"s1 = "test", s2 = "te"s1="test",s2="te"	Returns false	2, 3	Invalid	s1 is longer than s2

Boundary Value Analysis

Boundary Condition	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
Minimum size of s1s1s1	s1="",s2="a"s1 = "", s2 = "a"s1="",s2="a"	Returns true	4	Valid	Empty prefix
s1s1s1 longer than s2s2s2	s1="abcd",s2="abc"s1 = "abcd", s2 = "abc"s1="abcd",s2="abc"	Returns false	3	Invalid	Length mismatch
Identical strings	s1="abc",s2="abc"s1 = "abc", s2 = "abc"s1="abc",s2="abc"	Returns true	6	Valid	Both strings are identical
One character prefix	s1="a",s2="abc"s1 = "a", s2 = "abc"s1="a",s2="abc"	Returns true	1, 7	Valid	s1 is a prefix
Empty s2s2s2	s1="abc",s2=""s1 = "abc", s2 = ""s1="abc",s2=""	Returns false	5	Invalid	Non-empty cannot be a prefix of empty

Code Implementation with Test Cases

```
#include <iostream>
#include <string>

bool prefix(const std::string& s1, const std::string& s2) {
    if (s1.length() > s2.length()) {
        return false; // s1 cannot be a prefix if it is longer than s2
```

```
    }
    for (int i = 0; i < s1.length(); i++) {
        if (s1[i] != s2[i]) {
            return false; // Characters don't match
        }
    }
    return true; // All characters matched
}

int main() {
    // Test cases
    std::cout << "T1: " << std::boolalpha << prefix("pre", "prefix")
<< " (Expected: true)\n"; // True
    std::cout << "T2: " << std::boolalpha << prefix("fix", "prefix")
<< " (Expected: false)\n"; // False
    std::cout << "T3: " << std::boolalpha << prefix("prefix", "pre")
<< " (Expected: false)\n"; // False
    std::cout << "T4: " << std::boolalpha << prefix("", "hello") << "
(Expected: true)\n"; // True
    std::cout << "T5: " << std::boolalpha << prefix("hello", "") << "
(Expected: false)\n"; // False
    std::cout << "T6: " << std::boolalpha << prefix("test", "test") <<
" (Expected: true)\n"; // True
    std::cout << "T7: " << std::boolalpha << prefix("t", "test") << "
(Expected: true)\n"; // True
    std::cout << "T8: " << std::boolalpha << prefix("test", "te") << "
(Expected: false)\n"; // False

    return 0;
}
```

Output:
T1: true (Expected: true)
T2: false (Expected: false)
T3: false (Expected: false)
T4: true (Expected: true)
T5: false (Expected: false)
T6: true (Expected: true)
T7: true (Expected: true)
T8: false (Expected: false)

P5: Character Counting

Derived Equivalence Classes

Class Number	Description
1	Character c is present in the string (valid).
2	Character c is not present in the string (invalid).
3	String is empty (invalid).
4	c is a single character (valid).
5	String consists of repeated characters (valid).
6	String consists of special characters (valid).

Black Box Test Cases

Test Case	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
T1	c=′a′,s="banana"c = 'a', s = "banana"c=′a′,s="banana"	Returns 3	1, 4	Valid	c is present
T2	c=′x′,s="banana"c = 'x', s = "banana"c=′x′,s="banana"	Returns 0	2	Invalid	c is not present
T3	c=′a′,s=""c = 'a', s = ""c=′a′,s=""	Returns 0	3	Invalid	Empty string
T4	c=′b′,s="bbbb"c = 'b', s = "bbbb"c=′b′,s="bbbb"	Returns 4	5	Valid	All characters are c
T5	c=′1′,s="1a1b1"c = '1', s = "1a1b1"c=′1′,s="1a1b1"	Returns 3	1, 6	Valid	c is present

T6	c='!',s="hello!"c = '!', s = "hello!"c='!',s="hello!"	Returns 1	1, 6	Valid	c is present
T7	c = ' ', s = " "	Returns 3	1, 6	Valid	Spaces are counted
T8	c='a',s="apple"c = 'a', s = "apple"c='a',s="apple"	Returns 1	1, 4	Valid	c is present
T9	c='a',s="apple"c = 'a', s = "apple"c='a',s="apple"	Returns 1	1, 4	Valid	c is present
T10	c='z',s="apple"c = 'z', s = "apple"c='z',s="apple"	Returns 0	2	Invalid	c is not present

Boundary Value Analysis

Boundary Condition	Test Data	Expected Outcome	Classes Covered	Valid/Invalid	Reason
Minimum size of string	c='a',s=""c = 'a', s = ""c='a',s=""	Returns 0	3	Invalid	Empty string
Character present in string	c='a',s="aaa"c = 'a', s = "aaa"c='a',s="aaa"	Returns 3	1, 4	Valid	c is present
Character not present in string	c='b',s="aaa"c = 'b', s = "aaa"c='b',s="aaa"	Returns 0	2	Invalid	c is not present
All characters in string are ccc	c='b',s="bb"c = 'b', s = "bb"c='b',s="bb"	Returns 2	5	Valid	All characters are c
Mixed string with special characters	c='@',s="abc@!@"c = '@', s = "abc@!@"c='@',s="abc@!@"	Returns 2	1, 6	Valid	c is present

Code Implementation with Test Cases

```
#include <iostream>
#include <string>

int countCharacter(char c, const std::string& s) {
    int count = 0;
    for (char ch : s) {
        if (ch == c)
            count++;
    }
    return count; // Return count of occurrences
}

int main() {
    // Test cases
    std::cout << "T1: " << countCharacter('a', "banana") << "
(Expected: 3)\n"; // 3 occurrences
    std::cout << "T2: " << countCharacter('x', "banana") << "
(Expected: 0)\n"; // Not found
    std::cout << "T3: " << countCharacter('a', "") << " (Expected:
0)\n"; // Empty string
    std::cout << "T4: " << countCharacter('b', "bbbb") << " (Expected:
4)\n"; // All match
    std::cout << "T5: " << countCharacter('1', "1a1b1") << "
(Expected: 3)\n"; // 3 occurrences
    std::cout << "T6: " << countCharacter('!', "hello!") << "
(Expected: 1)\n"; // 1 occurrence
    std::cout << "T7: " << countCharacter(' ', "   ") << " (Expected:
3)\n"; // Count spaces
    std::cout << "T8: " << countCharacter('a', "apple") << "
(Expected: 1)\n"; // 1 occurrence
    std::cout << "T9: " << countCharacter('a', "apple") << "
(Expected: 1)\n"; // 1 occurrence
    std::cout << "T10: " << countCharacter('z', "apple") << "
(Expected: 0)\n"; // Not found

    return 0;
```

```
}
```
Output:
T1: 3 (Expected: 3)
T2: 0 (Expected: 0)
T3: 0 (Expected: 0)
T4: 4 (Expected: 4)
T5: 3 (Expected: 3)
T6: 1 (Expected: 1)
T7: 3 (Expected: 3)
T8: 1 (Expected: 1)
T9: 1 (Expected: 1)
T10: 0 (Expected: 0)

P6: Triangle Classification Program

a) Equivalence Classes

1. **Valid Triangle Cases**
   - **Equilateral Triangle**: All sides equal (e.g., A=B=C).
   - **Isosceles Triangle**: Two sides equal (e.g., A=B≠C).
   - **Scalene Triangle**: All sides different (e.g., A≠B≠C).
   - **Right-Angled Triangle**: Fulfills the Pythagorean theorem (e.g., A^2 + B^2 = C^2).
2. **Invalid Triangle Cases**
   - **Non-Triangle**: The sum of any two sides is less than or equal to the third side (e.g., A+B≤C).
   - **Non-positive Values**: One or more sides are non-positive (e.g., A≤0 B≤0 or C≤0).

b) Test Cases for Equivalence Classes

Test Case	Input Values	Expected Outcome	Equivalence Classes Covered	Reason
T1	A=3,B=3,C=3A = 3, B = 3, C = 3A=3,B=3,C=3	Equilateral	1	All sides are equal
T2	A=5,B=5,C=3A = 5, B = 5, C = 3A=5,B=5,C=3	Isosceles	2	Two sides are equal

T3	A=4,B=5,C=6A = 4, B = 5, C = 6A=4,B=5,C=6	Scalene	3	All sides are different
T4	A=3,B=4,C=5A = 3, B = 4, C = 5A=3,B=4,C=5	Right-Angled	4	Satisfies Pythagorean theorem
T5	A=1,B=2,C=3A = 1, B = 2, C = 3A=1,B=2,C=3	Invalid	5	Sum of two sides equal to the third
T6	A=0,B=5,C=5A = 0, B = 5, C = 5A=0,B=5,C=5	Invalid	6	One side is non-positive
T7	A=−1,B=2,C=2A = -1, B = 2, C = 2A=−1,B=2,C=2	Invalid	6	One side is negative
T8	A=2,B=2,C=3A = 2, B = 2, C = 3A=2,B=2,C=3	Isosceles	2	Two sides are equal
T9	A=2,B=2,C=2A = 2, B = 2, C = 2A=2,B=2,C=2	Equilateral	1	All sides are equal

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

Test Case	Input Values	Expected Outcome	Reason
T1	A=3,B=4,C=6A = 3, B = 4, C = 6A=3,B=4,C=6	Scalene	3+4>6 is valid
T2	A=2,B=5,C=6A = 2, B = 5, C = 6A=2,B=5,C=6	Scalene	2+5>6 is valid
T3	A=5,B=8,C=10A = 5, B = 8, C = 10A=5,B=8,C=10	Scalene	5+8>10 is valid
T4	A=7,B=5,C=11A = 7, B = 5, C = 11A=7,B=5,C=11	Invalid	7+5≤11

d) Boundary Condition A=CA = CA=C (Isosceles Triangle)

Test Case	Input Values	Expected Outcome	Reason
T1	A=5,B=3,C=5A = 5, B = 3, C = 5A=5,B=3,C=5	Isosceles	Two sides are equal
T2	A=6,B=4,C=6A = 6, B = 4, C = 6A=6,B=4,C=6	Isosceles	Two sides are equal
T3	A=10,B=8,C=10A = 10, B = 8, C = 10A=10,B=8,C=10	Isosceles	Two sides are equal
T4	A=4,B=4,C=8A = 4, B = 4, C = 8A=4,B=4,C=8	Invalid	4+4≤8

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

Test Case	Input Values	Expected Outcome	Reason
T1	A=5,B=5,C=5A = 5, B = 5, C = 5A=5,B=5,C=5	Equilateral	All sides are equal
T2	A=10,B=10,C=10A = 10, B = 10, C = 10A=10,B=10,C=10	Equilateral	All sides are equal
T3	A=1,B=1,C=1A = 1, B = 1, C = 1A=1,B=1,C=1	Equilateral	All sides are equal
T4	A=2.5,B=2.5,C=2.5A = 2.5, B = 2.5, C = 2.5A=2.5,B=2.5,C=2.5	Equilateral	All sides are equal

f) Boundary Condition A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (Right-Angled Triangle)

Test Case	Input Values	Expected Outcome	Reason

T1	A=3,B=4,C=5A = 3, B = 4, C = 5A=3,B=4,C=5	Right-Angled	3^2 + 4^2 = 5^2
T2	A=6,B=8,C=10A = 6, B = 8, C = 10A=6,B=8,C=10	Right-Angled	6^2 + 8^2 = 10^2
T3	A=5,B=12,C=13A = 5, B = 12, C = 13A=5,B=12,C=13	Right-Angled	5^2 + 12^2 = 13^2
T4	A=7,B=24,C=25A = 7, B = 24, C = 25A=7,B=24,C=25	Right-Angled	7^2 + 24^2 = 25^2

g) Non-Triangle Case

Test Case	Input Values	Expected Outcome	Reason
T1	A=1,B=2,C=3A = 1, B = 2, C = 3A=1,B=2,C=3	Invalid	A+B=C
T2	A=2,B=2,C=5A = 2, B = 2, C = 5A=2,B=2,C=5	Invalid	A+B<C
T3	A=1,B=4,C=2A = 1, B = 4, C = 2A=1,B=4,C=2	Invalid	A+C<B
T4	A=0,B=1,C=1A = 0, B = 1, C = 1A=0,B=1,C=1	Invalid	Non-positive side

h) Non-Positive Input

Test Case	Input Values	Expected Outcome	Reason
T1	A=0,B=2,C=2A = 0, B = 2, C = 2A=0,B=2,C=2	Invalid	A is non-positive
T2	A=3,B=−1,C=3A = 3, B = -1, C = 3A=3,B=−1,C=3	Invalid	B is negative

T3	A=−2,B=5,C=5A = -2, B = 5, C = 5A=−2,B=5,C=5	Invalid	A is negative
T4	A=3,B=3,C=0A = 3, B = 3, C = 0A=3,B=3,C=0	Invalid	C is non-positive