IT314: Software Engineering

Software Testing
Lab Session - Functional Testing (Black-Box)

Pratham Patel : 202201485

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?

Equivalence Partitions:

- **Valid Date:** A valid combination of day, month, and year, such as (15, 8, 2014).
- **Invalid Day:** The day is invalid (e.g., day < 1 or day > 31), like (0, 5, 2010) or (32, 1, 2011).
- **Invalid Month:** The month is invalid (e.g., month < 1 or month > 12), like (15, 0, 2010) or (12, 13, 2010).
- **Invalid Year:** The year is invalid (e.g., year < 1900 or year > 2015), like (10, 5, 1899) or (15, 7, 2016).

Equivalence Partitioning Test Cases:

Tester Action	Input Data	Expected Outcome
Valid Date	15, 8, 2014	14, 8, 2014
Invalid Day (<1)	0, 5, 2010	"Invalid Date"
Invalid Day (>31)	32, 1, 2011	"Invalid Date"
Invalid Month (<1)	15, 0, 2010	"Invalid Date"
Invalid Month (>12)	12, 13, 2010	"Invalid Date"
Invalid Year (<1900)	10, 5, 1899	"Invalid Date"
Invalid Year (>2015)	15, 7, 2016	"Invalid Date"

Boundary Value Analysis (BVA):

In BVA, we focus on the boundary values, as errors are more likely to occur at the edges of input ranges.

Boundary Values:

- **Day Boundary:**
  - Lower boundary: 1 (minimum day).
  - Upper boundary: 31 (maximum day for months with 31 days).
- **Month Boundary:**
  - Lower boundary: 1 (minimum month).
  - Upper boundary: 12 (maximum month).
- **Year Boundary:**
  - Lower boundary: 1900 (minimum year).
  - Upper boundary: 2015 (maximum year).

Boundary Value Analysis Test Cases:

Tester Action	Input Data	Expected Outcome
Lower boundary for day	1, 5, 2012	30, 4, 2012
Upper boundary for day	31, 12, 2014	30, 12, 2014
Lower boundary for month	15, 1, 2010	14, 1, 2010
Upper boundary for month	12, 12, 2015	11, 12, 2015
Lower boundary for year	1, 1, 1900	31, 12, 1899
Upper boundary for year	1, 1, 2015	31, 12, 2014

Updated Program:

```
#include <iostream>
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 check if the given date is valid
bool isValidDate(int day, int month, int year) {
    if (year < 1900 || year > 2015) return false;
    if (month < 1 || month > 12) return false;
    if (day < 1 || day > 31) return false;

    // Handle months with 30 days
    if ((month == 4 || month == 6 || month == 9 || month == 11) && day >
30) return false;

    // Handle February
    if (month == 2) {
        if (isLeapYear(year)) {
            if (day > 29) return false;
        } else {
            if (day > 28) return false;
        }
    }
    return true;
}

// Function to determine the previous date
void previousDate(int day, int month, int year) {
    if (!isValidDate(day, month, year)) {
        cout << "Invalid Date" << endl;
        return;
    }

    // Adjust for the first day of the month
    if (day == 1) {
```

```
        if (month == 1) {  // If it's January, go to the previous year
            day = 31;
            month = 12;
            year -= 1;
        } else if (month == 3) {  // March, handle February (leap year
case)
            month = 2;
            day = (isLeapYear(year)) ? 29 : 28;
        } else if (month == 5 || month == 7 || month == 10 || month == 12)
{  // Months that follow 30-day months
            month -= 1;
            day = 30;
        } else {  // All other cases, previous month has 31 days
            month -= 1;
            day = 31;
        }
    } else {
        day -= 1;
    }

    cout << "Previous Date: " << day << "/" << month << "/" << year <<
endl;
}
```

Test Suite:

Test Case ID	Tester Action	Input Data	Expected Output	Equivalence Partitioning / BVA	Remarks
TC-01	Valid Date	15, 8, 2014	14, 8, 2014	EP - Valid Date	Valid case
TC-02	Invalid Day (<1)	0, 5, 2010	"Invalid Date"	EP - Invalid Day (<1)	Error

TC-03	Invalid Day (>31)	32, 1, 2011	"Invalid Date"	EP - Invalid Day (>31)	Error
TC-04	Invalid Month (<1)	15, 0, 2010	"Invalid Date"	EP - Invalid Month (<1)	Error
TC-05	Invalid Month (>12)	12, 13, 2010	"Invalid Date"	EP - Invalid Month (>12)	Error
TC-06	Invalid Year (<1900)	10, 5, 1899	"Invalid Date"	EP - Invalid Year (<1900)	Error
TC-07	Invalid Year (>2015)	15, 7, 2016	"Invalid Date"	EP - Invalid Year (>2015)	Error
TC-08	First day of the year	1, 1, 1900	31, 12, 1899	BVA - Lower Year Boundary	Valid case
TC-09	Last day of the year	31, 12, 2015	30, 12, 2015	BVA - Upper Year Boundary	Valid case
TC-10	Lower Day Boundary	1, 5, 2012	30, 4, 2012	BVA - Lower Day Boundary	Valid case
TC-11	Upper Day Boundary (31)	31, 12, 2014	30, 12, 2014	BVA - Upper Day Boundary	Valid case
TC-12	Lower Month Boundary	15, 1, 2010	14, 1, 2010	BVA - Lower Month Boundary	Valid case
TC-13	Upper Month Boundary	12, 12, 2015	11, 12, 2015	BVA - Upper Month Boundary	Valid case
TC-14	Leap Year (29 Feb)	1, 3, 2012	29, 2, 2012	EP - Leap Year, Valid Date	Valid case
TC-15	Non-Leap Year (28 Feb)	1, 3, 2011	28, 2, 2011	EP - Non-Leap Year, Valid Date	Valid case

Q.2. Programs:

P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

1. Equivalence Partitioning (EP) Test Cases

Test Case	Description	Array `a`	Value `v`	Expected Output
TC1	`v` exists in the middle of the array	`[1, 2, 3, 4, 5]`	4	3
TC2	`v` exists at the first index	`[1, 2, 3, 4, 5]`	1	0
TC3	`v` does not exist in the array	`[1, 2, 3, 4, 5]`	6	-1
TC4	Empty array	`[]`	1	-1
TC5	Single element, `v` is present	`[5]`	5	0
TC6	Single element, `v` is not present	`[5]`	2	-1

---

2. Boundary Value Analysis (BVA) Test Cases

Test Case	Description	Array `a`	Value `v`	Expected Output
TC1	`v` exists at the boundary (first element)	`[10, 20, 30, 40]`	10	0

TC2	`v` exists at the boundary (last element)	`[10, 20, 30, 40]`	`40`	`3`
TC3	`v` does not exist, near first boundary	`[10, 20, 30, 40]`	`5`	`-1`
TC4	`v` does not exist, near last boundary	`[10, 20, 30, 40]`	`45`	`-1`
TC5	Minimum input size (empty array)	`[]`	`10`	`-1`
TC6	Maximum input size with `v` at last index	`[1, 2, ..., 1000]`	`1000`	`999`

---

3. Updated Code

```cpp
#include <iostream>
using namespace std;

int linearSearch(int v, int a[], int size) {
    for (int i = 0; i < size; i++) {
        if (a[i] == v)
            return i;  // Return the index if value is found
    }
    return -1;  // Return -1 if value is not found
}
```

4. Test Suite

Test Case	Array `a`	Value `v`	Expected Output	Actual Output	Result (Pass/Fail)
TC1	`[1, 2, 3, 4, 5]`	`4`	`3`	`3`	Pass

TC2	`[1, 2, 3, 4, 5]`	`1`	`0`	`0`	**Pass**
TC3	`[1, 2, 3, 4, 5]`	`6`	`-1`	`-1`	**Pass**
TC4	`[]`	`1`	`-1`	`-1`	**Pass**
TC5	`[5]`	`5`	`0`	`0`	**Pass**
TC6	`[5]`	`2`	`-1`	`-1`	**Pass**
TC7 (BVA)	`[10, 20, 30, 40]`	`10`	`0`	`0`	**Pass**
TC8 (BVA)	`[10, 20, 30, 40]`	`40`	`3`	`3`	**Pass**
TC9 (BVA)	`[10, 20, 30, 40]`	`5`	`-1`	`-1`	**Pass**
TC10 (BVA)	`[10, 20, 30, 40]`	`45`	`-1`	`-1`	**Pass**
TC11 (BVA)	`[]`	`10`	`-1`	`-1`	**Pass**
TC12 (BVA)	`[1, 2, ..., 1000]`	`1000`	`999`	`999`	**Pass**

P2. The function countItem returns the number of times a value v appears in an array of integers a.

1. Equivalence Partitioning (EP) Test Cases

Test Case	Description	Array `a`	Value `v`	Expected Output
TC1	`v` appears multiple times in the array	`[1, 2, 2, 3, 4, 2, 5]`	`2`	`3`
TC2	`v` appears once in the array	`[1, 2, 3, 4, 5]`	`5`	`1`
TC3	`v` does not appear in the array	`[1, 2, 3, 4, 5]`	`6`	`0`
TC4	Empty array	`[]`	`1`	`0`
TC5	Single element, `v` is present	`[5]`	`5`	`1`
TC6	Single element, `v` is not present	`[5]`	`2`	`0`

---

2. Boundary Value Analysis (BVA) Test Cases

Test Case	Description	Array `a`	Value `v`	Expected Output
TC1	`v` is at the boundary (first element)	`[10, 20, 30, 40]`	`10`	`1`
TC2	`v` is at the boundary (last element)	`[10, 20, 30, 40]`	`40`	`1`

TC3	`v` is not in the array, near first boundary	`[10, 20, 30, 40]`	`5`	`0`
TC4	`v` is not in the array, near last boundary	`[10, 20, 30, 40]`	`45`	`0`
TC5	Empty array (minimum size)	`[]`	`10`	`0`
TC6	Maximum input size with multiple occurrences	`[1, 2, ..., 1000]`	`500`	`1`

---

3. Updated Code

```cpp
#include <iostream>
using namespace std;

int countItem(int v, int a[], int size) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (a[i] == v)
            count++;  // Increment count if value matches
    }
    return count;  // Return the count of occurrences
}
```

4. Test Suite

Test Case	Array `a`	Value `v`	Expected Output	Actual Output	Result (Pass/Fail)
TC1 (EP)	`[1, 2, 2, 3, 4, 2, 5]`	`2`	`3`	`3`	Pass
TC2 (EP)	`[1, 2, 3, 4, 5]`	`5`	`1`	`1`	Pass

TC3 (EP)	`[1, 2, 3, 4, 5]`	`6`	`0`	`0`	**Pass**
TC4 (EP)	`[]`	`1`	`0`	`0`	**Pass**
TC5 (EP)	`[5]`	`5`	`1`	`1`	**Pass**
TC6 (EP)	`[5]`	`2`	`0`	`0`	**Pass**
TC7 (BVA)	`[10, 20, 30, 40]`	`10`	`1`	`1`	**Pass**
TC8 (BVA)	`[10, 20, 30, 40]`	`40`	`1`	`1`	**Pass**
TC9 (BVA)	`[10, 20, 30, 40]`	`5`	`0`	`0`	**Pass**
TC10 (BVA)	`[10, 20, 30, 40]`	`45`	`0`	`0`	**Pass**
TC11 (BVA)	`[]`	`10`	`0`	`0`	**Pass**
TC12 (BVA)	`[1, 2, ..., 1000]`	`500`	`1`	`1`	**Pass**

P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.

Assumption: the elements in the array a are sorted in non-decreasing order.

1. Equivalence Partitioning (EP) Test Cases

Test Case	Description	Array `a`	Value `v`	Expected Output
TC1 (EP)	`v` is present in the middle of the array	`[1, 2, 3, 4, 5]`	3	2
TC2 (EP)	`v` is present at the start of the array	`[1, 2, 3, 4, 5]`	1	0
TC3 (EP)	`v` is present at the end of the array	`[1, 2, 3, 4, 5]`	5	4
TC4 (EP)	`v` is not present in the array	`[1, 2, 3, 4, 5]`	6	-1
TC5 (EP)	Empty array	`[]`	1	-1
TC6 (EP)	Single element, `v` is present	`[5]`	5	0
TC7 (EP)	Single element, `v` is not present	`[5]`	2	-1
TC8 (Invalid)	Unsorted array (invalid input)	`[3, 1, 2, 5, 4]`	3	Invalid Input

---

2. Boundary Value Analysis (BVA) Test Cases

Test Case	Description	Array a	Value v	Expected Output
TC1 (BVA)	v is at the boundary (first element)	[10, 20, 30, 40]	10	0
TC2 (BVA)	v is at the boundary (last element)	[10, 20, 30, 40]	40	3
TC3 (BVA)	v is not in the array, near first boundary	[10, 20, 30, 40]	5	-1
TC4 (BVA)	v is not in the array, near last boundary	[10, 20, 30, 40]	45	-1
TC5 (BVA)	Empty array (minimum size)	[]	10	-1
TC6 (BVA)	Maximum input size	[1, 2, ..., 1000]	500	499

---

3. Updated Code

```
#include <iostream>
using namespace std;

bool isSorted(int a[], int size) {
    for (int i = 1; i < size; i++) {
        if (a[i] < a[i-1])
            return false;
    }
    return true;
}

int binarySearch(int v, int a[], int size) {
    if (!isSorted(a, size)) {
        cout << "Invalid Input: Array is not sorted" << endl;
        return -2;  // Indicate invalid input
    }
```

```
    int low = 0, high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (a[mid] == v)
            return mid;  // Return index if found
        else if (a[mid] < v)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1;  // Return -1 if value not found
}
```

4. Test Suite

Test Case	Array a	Value v	Expected Output	Actual Output	Result (Pass/Fail)
TC1 (EP)	`[1, 2, 3, 4, 5]`	3	2	2	Pass
TC2 (EP)	`[1, 2, 3, 4, 5]`	1	0	0	Pass
TC3 (EP)	`[1, 2, 3, 4, 5]`	5	4	4	Pass
TC4 (EP)	`[1, 2, 3, 4, 5]`	6	-1	-1	Pass
TC5 (EP)	`[]`	1	-1	-1	Pass
TC6 (EP)	`[5]`	5	0	0	Pass

TC7 (EP)	`[5]`	2	-1	-1	**Pass**
TC8 (Invalid)	`[3, 1, 2, 5, 4]`	3	Invalid Input	Invalid Input	**Pass**
TC9 (BVA)	`[10, 20, 30, 40]`	10	0	**`0**	

P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

1. Equivalence Partitioning (EP) Test Cases

Test Case	Description	Input (a, b, c)	Expected Output
TC1 (EP)	All sides equal (equilateral triangle)	`3, 3, 3`	`0`
TC2 (EP)	Two sides equal (isosceles triangle)	`3, 3, 2`	`1`
TC3 (EP)	All sides different (scalene triangle)	`3, 4, 5`	`2`
TC4 (EP)	Invalid triangle (sum of two sides less than third)	`1, 2, 3`	`3`
TC5 (EP)	Valid scalene triangle	`5, 6, 7`	`2`
TC6 (EP)	All sides zero (invalid triangle)	`0, 0, 0`	`3`
TC7 (EP)	Negative side length (invalid triangle)	`-3, 4, 5`	`3`

---

2. Boundary Value Analysis (BVA) Test Cases

Test Case	Description	Input (a, b, c)	Expected Output

TC1 (BVA)	**Minimum valid triangle (1, 1, 1)**	`1, 1, 1`	0
TC2 (BVA)	**Maximum valid triangle (biggest integer values)**	`2147483647, 2147483647, 2147483647`	0
TC3 (BVA)	**Invalid triangle (sum of two sides equal to the third)**	`1, 2, 3`	3
TC4 (BVA)	**Two sides equal (edge case for isosceles)**	`2, 2, 1`	1
TC5 (BVA)	**Zero length sides (invalid triangle)**	`0, 1, 1`	3

---

3. Updated Code

```
#include <iostream>
using namespace std;

final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;

int triangle(int a, int b, int c) {
    // Check for non-positive lengths
    if (a <= 0 || b <= 0 || c <= 0) {
        return INVALID; // Invalid triangle
    }
    // Check for triangle inequality
    if (a >= b + c || b >= a + c || c >= a + b) {
        return INVALID; // Invalid triangle
    }
    // Check for equilateral triangle
    if (a == b && b == c) {
        return EQUILATERAL;
    }
```

```
    // Check for isosceles triangle
    if (a == b || a == c || b == c) {
        return ISOSCELES;
    }
    return SCALENE; // Scalene triangle
}
```

4. Test Suite

Test Case	Input (a, b, c)	Expected Output	Actual Output	Result (Pass/Fail)
TC1 (EP)	`3, 3, 3`	0	0	**Pass**
TC2 (EP)	`3, 3, 2`	1	1	**Pass**
TC3 (EP)	`3, 4, 5`	2	2	**Pass**
TC4 (EP)	`1, 2, 3`	3	3	**Pass**
TC5 (EP)	`5, 6, 7`	2	2	**Pass**
TC6 (EP)	`0, 0, 0`	3	3	**Pass**
TC7 (EP)	`-3, 4, 5`	3	3	**Pass**
TC8 (BVA)	`1, 1, 1`	0	0	**Pass**
TC9 (BVA)	`2147483647, 2147483647, 2147483647`	0	0	**Pass**

P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

1. Equivalence Partitioning (EP) Test Cases

Test Case	Description	Input (s1, s2)	Expected Output
TC1 (EP)	`s1` is an exact match (prefix) of `s2`	`"test", "testcase"`	`true`
TC2 (EP)	`s1` is a partial match (prefix) of `s2`	`"te", "testcase"`	`true`
TC3 (EP)	`s1` is longer than `s2`	`"testing", "test"`	`false`
TC4 (EP)	`s1` is not a prefix of `s2`	`"case", "testcase"`	`false`
TC5 (EP)	`s1` is an empty string	`"", "testcase"`	`true`
TC6 (EP)	`s2` is an empty string	`"test", ""`	`false`
TC7 (EP)	`s1` and `s2` are equal	`"same", "same"`	`true`
TC8 (EP)	`s1` is a prefix but has different case	`"Test", "testcase"`	`false`

2. Boundary Value Analysis (BVA) Test Cases

Test Case	Description	Input (s1, s2)	Expected Output
TC1 (BVA)	Both strings are empty (valid prefix case)	`""`, `""`	`true`
TC2 (BVA)	`s1` is empty and `s2` is non-empty	`""`, `"nonempty"`	`true`
TC3 (BVA)	`s1` is non-empty and `s2` is empty	`"nonempty"`, `""`	`false`
TC4 (BVA)	Maximum length string as `s1` and `s2` (exact match)	``"A"*1000, "A"*1000	`true`
TC5 (BVA)	`s1` is maximum length and `s2` is longer	``"A"*1000, "A"*1001	`true`

---

3. Updated Code

```java
public class PrefixChecker {
    public static boolean prefix(String s1, String s2) {
        // Check if s1 is longer than s2
        if (s1.length() > s2.length()) {
            return false;
        }
        // Compare each character in s1 with s2
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) != s2.charAt(i)) {
                return false; // Mismatch found
            }
        }
        return true; // All characters matched
    }
}
```

4. Test Suite Execution

Test Case	Input (s1, s2)	Expected Output	Actual Output	Result (Pass/Fail)
TC1 (EP)	"test", "testcase"	true	true	Pass
TC2 (EP)	"te", "testcase"	true	true	Pass
TC3 (EP)	"testing", "test"	false	false	Pass
TC4 (EP)	"case", "testcase"	false	false	Pass
TC5 (EP)	"", "testcase"	true	true	Pass
TC6 (EP)	"test", ""	false	false	Pass
TC7 (EP)	"same", "same"	true	true	Pass
TC8 (EP)	"Test", "testcase"	false	false	Pass
TC9 (BVA)	"", ""	true	true	Pass
TC10 (BVA)	"", "nonempty"	true	true	Pass
TC11 (BVA)	"nonempty", ""	false	false	Pass
TC12 (BVA)	`"A"*1000, "A"*1000	true	true	Pass

P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled.

Determine the following for the above program:

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)
c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.
d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.
e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.
f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify
the boundary.
g) For the non-triangle case, identify test cases to explore the boundary.
h) For non-positive input, identify test points.

A. Equivalence Classes Identification

1. **Valid Equivalence Classes:**
   - **Equilateral Triangle:** All sides are equal (A = B = C).
   - **Isosceles Triangle:** Exactly two sides are equal (A = B ≠ C, A = C ≠ B, B = C ≠ A).
   - **Scalene Triangle:** All sides are different (A ≠ B ≠ C).
   - **Right-Angled Triangle:** Follows the Pythagorean theorem ($A^2 + B^2 = C^2$ or similar permutations).

2. **Invalid Equivalence Classes:**
   - **Non-Triangle:** The lengths do not satisfy the triangle inequality (A + B ≤ C, A + C ≤ B, B + C ≤ A).
   - **Non-Positive Inputs:** At least one of the sides is less than or equal to zero.

B. Test Cases for Equivalence Classes

Test Case	Description	Input (A, B, C)	Equivalence Class Covered
TC1	Equilateral Triangle	`3.0, 3.0, 3.0`	Equilateral Triangle
TC2	Isosceles Triangle	`4.0, 4.0, 6.0`	Isosceles Triangle
TC3	Scalene Triangle	`3.0, 4.0, 5.0`	Scalene Triangle
TC4	Right-Angled Triangle	`3.0, 4.0, 5.0`	Right-Angled Triangle
TC5	Non-Triangle	`1.0, 2.0, 3.0`	Non-Triangle
TC6	Non-Positive Inputs	`0.0, 2.0, 3.0`	Non-Positive Inputs
TC7	Non-Positive Inputs	`-1.0, 2.0, 3.0`	Non-Positive Inputs

C. Boundary Test Cases for A + B > C (Scalene Triangle)

Test Case	Description	Input (A, B, C)	Expected Outcome
TC1	Just above the boundary	`2.0, 3.0, 4.0`	Scalene Triangle
TC2	Exactly at the boundary	`2.0, 2.0, 4.0`	Non-Triangle

TC3	Just below the boundary	`1.0, 1.0, 2.0`	Non-Triangle

D. Boundary Test Cases for A = C (Isosceles Triangle)

Test Case	Description	Input (A, B, C)	Expected Outcome
TC1	Just above the boundary	`3.0, 4.0, 3.0`	Isosceles Triangle
TC2	Exactly at the boundary	`3.0, 4.0, 3.0`	Isosceles Triangle
TC3	Just below the boundary	`2.0, 4.0, 2.0`	Isosceles Triangle

E. Boundary Test Cases for A = B = C (Equilateral Triangle)

Test Case	Description	Input (A, B, C)	Expected Outcome
TC1	Just above the boundary	`3.0, 3.0, 3.0`	Equilateral Triangle
TC2	Exactly at the boundary	`2.0, 2.0, 2.0`	Equilateral Triangle
TC3	Just below the boundary	`1.0, 1.0, 1.0`	Equilateral Triangle

F. Boundary Test Cases for $A^2 + B^2 = C^2$ (Right-Angle Triangle)

Test Case	Description	Input (A, B, C)	Expected Outcome

TC1	**Just above the boundary**	`3.0, 4.0, 5.0`	**Right-Angled Triangle**
TC2	**Exactly at the boundary**	`3.0, 4.0, 5.0`	**Right-Angled Triangle**

G. Test Cases for Non-Triangle

Test Case	Descriptio n	Input (A, B, C)	Expected Outcome
TC1	**A + B ≤ C**	`2.0, 2.0, 5.0`	**Non-Triangle**
TC2	**A + C ≤ B**	`1.0, 2.0, 2.0`	**Non-Triangle**
TC3	**B + C ≤ A**	`5.0, 1.0, 2.0`	**Non-Triangle**

H. Test Cases for Non-Positive Input

Test Case	Description	Input (A, B, C)	Expected Outcome
TC1	**A = 0 (invalid side length)**	`0.0, 1.0, 1.0`	**Non-Positive Input**
TC2	**B = 0 (invalid side length)**	`1.0, 0.0, 1.0`	**Non-Positive Input**
TC3	**C = 0 (invalid side length)**	`1.0, 1.0, 0.0`	**Non-Positive Input**
TC4	**All sides non-positive**	`-1.0, -2.0, -3.0`	**Non-Positive Input**