IT-314 LAB – 8

Functional Testing (Black-Box)

Software Engineering

Lab Group – 1

Name : Dholariya Parth Narendra

Student ID: 202201085

Question-1

1. Equivalence Class Partitioning

Equivalence partitioning splits the input domain into multiple groups, where each group is expected to exhibit similar behaviour.

Valid Equivalence Classes

- Year :(1900 ≤ Year ≤ 2015)
- Month :(1 ≤ Month ≤ 12)
- Day :(1 ≤ Day ≤ 31)

Invalid Equivalence Classes

- Year >2015 or Year < 1900
- Month >12 or Month<1
- Day >31 or Day <1

2. Boundary Value Analysis

Boundary Value Tests at the Boundaries between partitions.

- Boundary For Year
  - Minimum value: 1900
  - Maximum value: 2015
- Boundary For Month
  - Minimum value: 1
  - Maximum value: 12
- Boundary For Day
  - Minimum value: 1

- Maximum value: 31

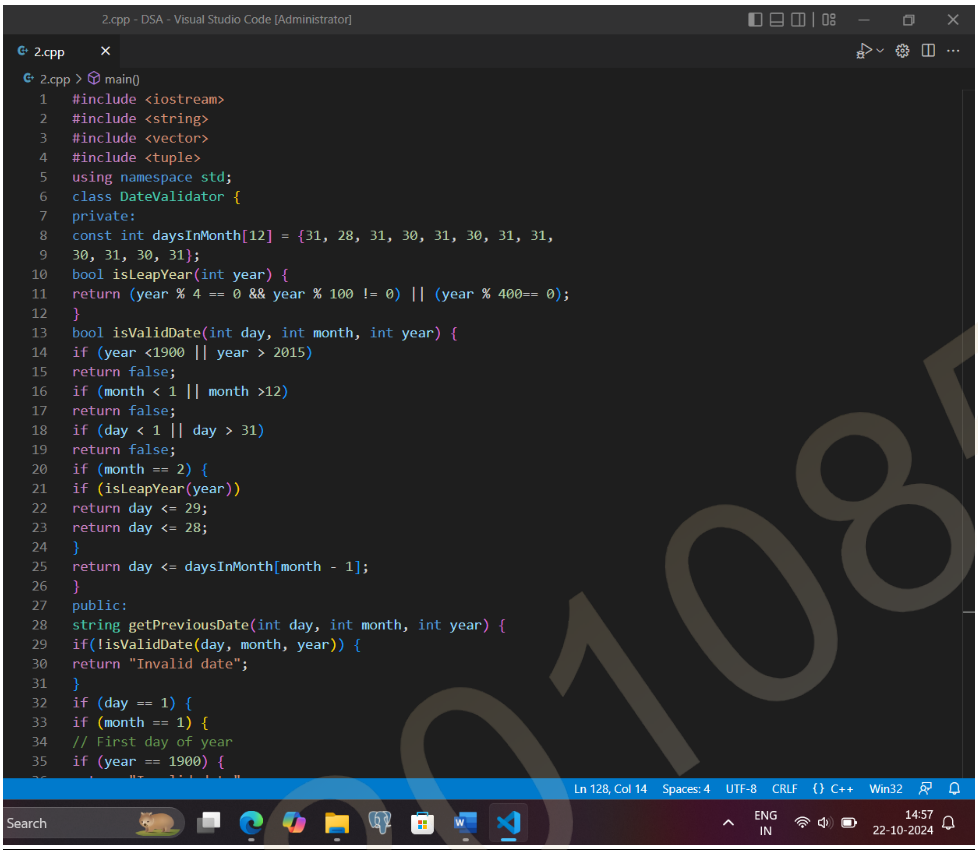
❖ Test Cases

EP = Equivalence Partitioning

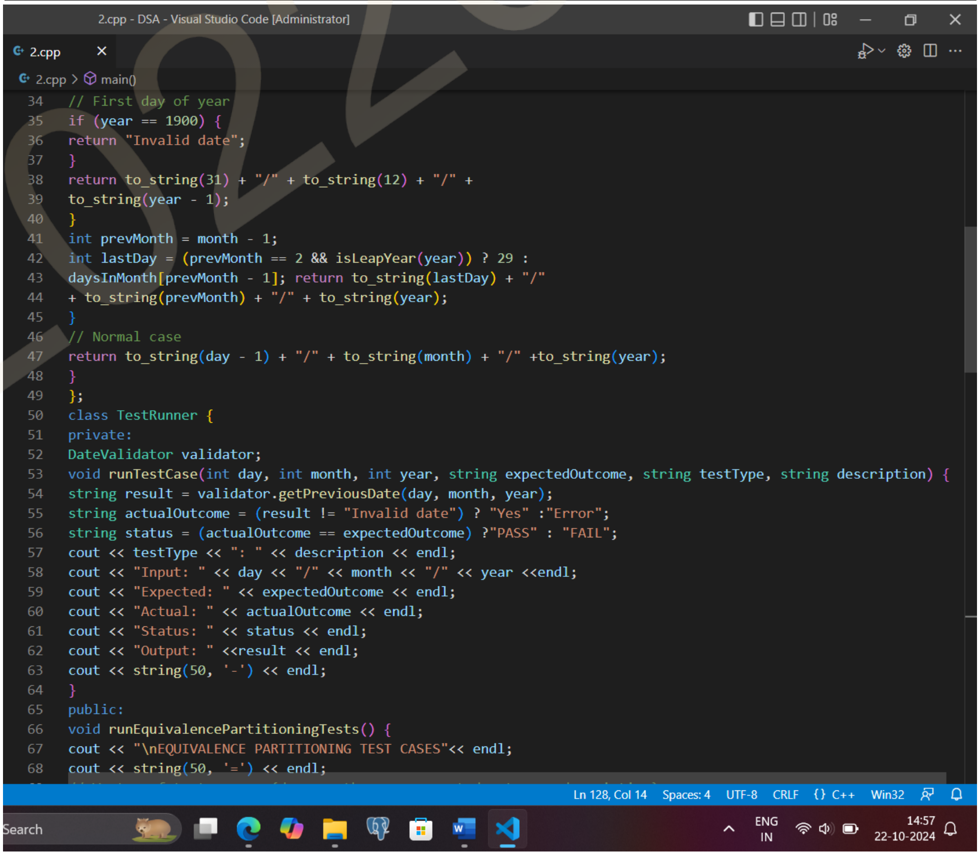
BV= Boundary Value

Input data	Expected Output	Type
15,6,2000	14/06/2000	EP
1,7,2000	30/06/2000	EP
31,12,2000	30/12/2000	EP
0,6,2000	Error	EP
32,6,2000	Error	EP
15,0,2000	Error	EP
29,2,2000	28/02/2000	EP
31,3,2000	30/03/2000	EP
30,4,2000	29/04/2000	EP
1,6,2000	31/05/2000	BV
31,7,2000	30/07/2000	BV
0,6,2000	Error	BV
15,12,2000	14/12/2000	BV
15,0,2000	Error	BV
15,13,2000	Error	BV
15,6,1900	14/06/1900	BV
15,6,2016	Error	BV
15,6,1899	Error	BV

Modified Program




```cpp
#include <iostream>
#include <string>
#include <vector>
#include <tuple>
using namespace std;
class DateValidator {
private:
const int daysInMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31};
bool isLeapYear(int year) {
return (year % 4 == 0 && year % 100 != 0) || (year % 400== 0);
}
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;
if (month == 2) {
if (isLeapYear(year))
return day <= 29;
return day <= 28;
}
return day <= daysInMonth[month - 1];
}
public:
string getPreviousDate(int day, int month, int year) {
if(!isValidDate(day, month, year)) {
return "Invalid date";
}
if (day == 1) {
if (month == 1) {
// First day of year
if (year == 1900) {
```




```cpp
// First day of year
if (year == 1900) {
return "Invalid date";
}
return to_string(31) + "/" + to_string(12) + "/" +
to_string(year - 1);
}
int prevMonth = month - 1;
int lastDay = (prevMonth == 2 && isLeapYear(year)) ? 29 :
daysInMonth[prevMonth - 1]; return to_string(lastDay) + "/"
+ to_string(prevMonth) + "/" + to_string(year);
}
// Normal case
return to_string(day - 1) + "/" + to_string(month) + "/" +to_string(year);
}
};
class TestRunner {
private:
DateValidator validator;
void runTestCase(int day, int month, int year, string expectedOutcome, string testType, string description) {
string result = validator.getPreviousDate(day, month, year);
string actualOutcome = (result != "Invalid date") ? "Yes" :"Error";
string status = (actualOutcome == expectedOutcome) ?"PASS" : "FAIL";
cout << testType << ": " << description << endl;
cout << "Input: " << day << "/" << month << "/" << year <<endl;
cout << "Expected: " << expectedOutcome << endl;
cout << "Actual: " << actualOutcome << endl;
cout << "Status: " << status << endl;
cout << "Output: " <<result << endl;
cout << string(50, '-') << endl;
}
public:
void runEquivalencePartitioningTests() {
cout << "\nEQUIVALENCE PARTITIONING TEST CASES"<< endl;
cout << string(50, '=') << endl;
```

```cpp
cout << string(50, '=') << endl;
// Vector of test cases: {day, month,year, expected outcome, description}
vector<tuple<int, int, int, string,
string>> testCases = {
// Valid Dates
{15, 6, 2000, "Yes", "Valid middle date"},
{1, 7, 2000, "Yes", "First day of month"},
{31, 12, 2000, "Yes", "Last day of year"},
// Invalid Dates
{0, 6, 2000, "Error", "Invalid day - below range"},
{32, 6, 2000, "Error", "Invalid day - above range"},
{15, 0, 2000, "Error", "Invalid month - below range"},
{15, 13, 2000, "Error", "Invalid month -above range"},
{15, 6, 1899, "Error", "Invalid year - below range"},
{15, 6, 2016, "Error", "Invalid year - above range"},
{31, 4, 2000, "Error", "Invalid day for month"},
{29, 2, 2001, "Error", "Invalid day for February non-leap year"}
};
for (const auto& test : testCases) {
runTestCase(get<0>(test),
get<1>(test), get<2>(test),
get<3>(test), "EP", get<4>(test));
}
}
void runBoundaryValueTests() {
cout << "\nBOUNDARY VALUE ANALYSIS TEST CASES" << endl;
cout << string(50,'=') << endl;
vector<tuple<int, int, int, string, string>>
testCases = {
// Day boundaries
{1, 6, 2000,
"Yes", "Minimum valid day"},
{31, 12, 2000,
"Yes", "Maximum valid day"},
{0, 6, 2000,
```

```cpp
{31, 12, 2000,
"Yes", "Maximum valid day"},
{0, 6, 2000,
"Error", "Day below minimum"},
{32, 6, 2000,
"Error", "Day above maximum"},
// Month boundaries
{15, 1, 2000, "Yes", "Minimum valid month"},
{15, 12, 2000, "Yes", "Maximum valid month"},
{15, 0, 2000, "Error", "Month below minimum"},
{15, 13, 2000, "Error", "Month above maximum"},
// Year boundaries
{15, 6, 1900, "Yes", "Minimum valid year"},
{15, 6, 2015, "Yes", "Maximum valid year"},
{15, 6, 1899, "Error", "Year below minimum"},
{15, 6, 2016, "Error", "Year above maximum"}
};
for (const auto& test :
testCases) {
runTestCase(
get<0>(test),
get<1>(test),
get<2>(test),
get<3>(test), "BVA",
get<4>(test));
}
}
};
int main() {
TestRunner runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests();
return 0;
}
```

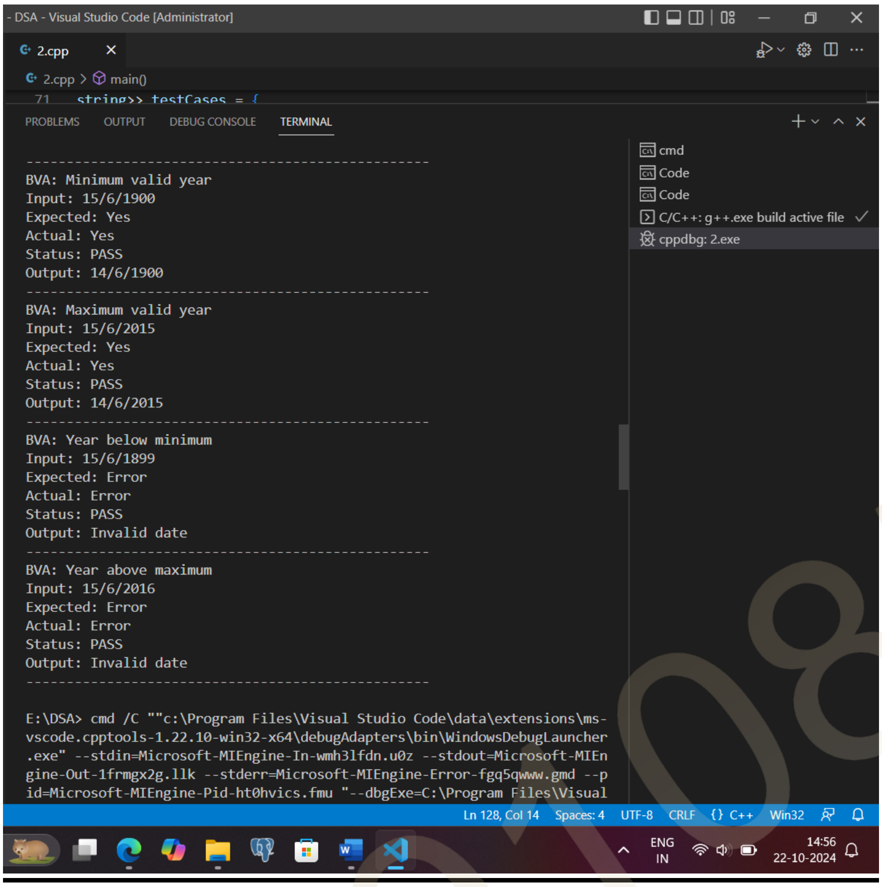
```
EQUIVALENCE PARTITIONING TEST CASES
==================================================
EP: Valid middle date
Input: 15/6/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 14/6/2000
--------------------------------------------------
EP: First day of month
Input: 1/7/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 30/6/2000
--------------------------------------------------
EP: Last day of year
Input: 31/12/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 30/12/2000
--------------------------------------------------
EP: Invalid day - below range
Input: 0/6/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
--------------------------------------------------
EP: Invalid day - above range
Input: 32/6/2000
Expected: Error
Actual: Error
Status: PASS
```

```
--------------------------------------------------
EP: Invalid month - below range
Input: 15/0/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
--------------------------------------------------
EP: Invalid month -above range
Input: 15/13/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
--------------------------------------------------
EP: Invalid year - below range
Input: 15/6/1899
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
--------------------------------------------------
EP: Invalid year - above range
Input: 15/6/2016
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
--------------------------------------------------
EP: Invalid day for month
Input: 31/4/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
```

```
-------------------------------------------------
EP: Invalid day for February non-leap year
Input: 29/2/2001
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
-------------------------------------------------

BOUNDARY VALUE ANALYSIS TEST CASES
=================================================
BVA: Minimum valid day
Input: 1/6/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 31/5/2000
-------------------------------------------------
BVA: Maximum valid day
Input: 31/12/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 30/12/2000
-------------------------------------------------
BVA: Day below minimum
Input: 0/6/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
-------------------------------------------------
BVA: Day above maximum
Input: 32/6/2000
Expected: Error
```

```
-------------------------------------------------
BVA: Day above maximum
Input: 32/6/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
-------------------------------------------------
BVA: Minimum valid month
Input: 15/1/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 14/1/2000
-------------------------------------------------
BVA: Maximum valid month
Input: 15/12/2000
Expected: Yes
Actual: Yes
Status: PASS
Output: 14/12/2000
-------------------------------------------------
BVA: Month below minimum
Input: 15/0/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
-------------------------------------------------
BVA: Month above maximum
Input: 15/13/2000
Expected: Error
Actual: Error
Status: PASS
Output: Invalid date
```



Question -2

P1

```
int linearSearch(int v, int a[]) {
 int i = 0;
 while (i < a.length) {
 if (a[i] == v)
return(i);
i++;
}
return (-1);
 }
```

1. Equivalence Class Partitioning

We can Divide the inputs into valid and invalid equivalence classes.

Valid Equivalence Classes

- v is present in the array a.
- v is not present in the array a.
- The array a contains one or more than one elements.

Invalid Equivalence Classes

- Array is empty.

2. Boundary Value Analysis

Test Boundary values for array a

- Array with one element.
- Array with two elements (minimum non-trivial size).
- Empty Array.
- Array with a large number of elements.

Boundary for the element v to be found.

- v is at the last index (index a.length-1).
- v is at the first index (index 0).

Test Cases

Input Data	Expected Outcome	Type
[1,2,3,4,5] , 4	Return index: 3	EP
[100,200,300,400],100	Return index: 0	EP
[19,18, 17, 16], 5	Error (not found)	EP
[], 7	Error (array is empty)	EP
[9], 9	0	BV
[10, 20], 20	1	BV
[24, 25, 26], 26	2	BV
[101, 201, 301, ..., 1001], 1	Error (not found)	BV
[10, 20, 30, ..., 1000], 1000	99	BV

P2

```
int countItem(int v, int a[]){
int count = 0;
for (int i = 0; i < a.length; i++)
{
if (a[i] == v)
count++;
}
return (count);
}
```

1. Equivalence Class Partitioning

We can divide the inputs into valid and invalid Equivalence partitioning.

Valid Equivalence classes

- v appears one or more times in the array a.
- v does not appear in the array a.
- The array a contains one or more elements.

Invalid Equivalence Classes

- The array a is empty.

2. Boundary Value Analysis

Test boundary values for the size of the array a

- Empty array (a with size 0).
- Array with one element.
- Array with two elements.
- Array with a large number of elements.

Boundary for the element v occurrence:

- v appears once.
- v appears multiple times.
- v does not appear at all.

Test Cases

Input Data	Expected Output	Type
[1, 2, 3, 4, 5], 3	Return count: 1	EP
[10, 20, 30, 40], 10	Return count: 1	EP
[9, 8, 7, 6], 5	error (not found)	EP
[], 5	error (EMPTY)	EP
[3], 3	Return count: 1	BV
[1, 2], 2	Return count: 1	BV
[1, 2, 2, 2, 3], 2	Return count: 3	BV
[10, 20, 30, ..., 1000], 1000	Return count: 1	BV
[10, 20, 30, ..., 1000], 1	error (not found)	BV

P3

```
int binarySearch(int v, int a[]){
int lo,mid,hi;
lo = 0;
hi = a.length-1;
while (lo <= hi){
mid = (lo+hi)/2;
if (v == a[mid])
return (mid);
else if (v < a[mid])
hi = mid-1;
else
lo = mid+1;}
return(-1);}
```

1. **Equivalence Class Partitioning**

We can divide the inputs into valid and invalid equivalence classes.

Valid Equivalence Classes

- v is present in the array a.
- v is not present in the array a.
- The array a contains one or more elements, and is sorted.

Invalid Equivalence Classes

- The array a is empty.
- The array a is unsorted.

2. **Boundary Value Analysis**

Test boundary values for the size of the array a:

- Empty array (a with size 0).
- Array with one element.
- Array with two elements.
- Array with a large number of elements.

Boundary for the element v to be found:

- v is at the first index (index 0).
- v is at the last index (index a.length-1).
- v is not in the array at all.

Test Cases

Input Data	Expected Output	Type
[1, 2, 3, 4, 5], 3	Return index: 2	EP
[10, 20, 30, 40], 10	Return index: 0	EP
[9, 8, 7, 6], 5	error (array is unsorted)	EP
[], 5	error (empty)	EP
[3], 3	Return index: 0	EP
[1, 2], 2	Return index: 1	BV
[4, 5, 6], 6	Return count: 2	BV
[10, 20, 30, ..., 1000], 1000	Return index: 999	BV
[10, 20, 30, ..., 1000], 1	error (not found)	BV
[5, 10, 15, 20], 25	error (not found)	EP

P4

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c){
if (a >= b+c || b >= a+c || c >= a+b)
return(INVALID);
if (a == b && b == c)
return(EQUILATERAL);
if (a == b || a == c || b == c)
return(ISOSCELES);
return(SCALENE);
}
```

1. Equivalence Class Partitioning

We can divide the inputs into valid and invalid equivalence classes.

Valid Equivalence Classes

- Equilateral triangle (all three sides are equal).
- Isosceles triangle (two sides are equal).
- Scalene triangle (all sides are different).

Invalid Equivalence Classes

- The side lengths do not satisfy the triangle inequality:
  - ○ a >= b + c
  - ○ b >= a + c
  - ○ c >= a + b
- One or more sides are non-positive (i.e., a <= 0, b <= 0, c <= 0).

2. **Boundary Value Analysis**

Test boundary values for the lengths of the sides of the triangle:

- ● Minimal positive length (1).
- ● Equal side lengths for equilateral and isosceles.
- ● Slight variations in side lengths for scalene and invalid cases.
- ● Triangle inequality boundary conditions.

Test Cases

Input Data	Expected Outcome	Type
(3, 3, 3)	Return: Equilateral(0)	EP
(5, 5, 8)	Return: Isosceles(1)	EP
(4, 5, 6)	Return: Scalene(2)	EP
(10, 5, 3)	error (triangle inequality)(3)	EP
(0, 5, 5)	error (non-positive side)(3)	EP
(1, 1, 1)	Return: Equilateral(0)	BV
(2, 2, 3)	Return: Isosceles(1)	BV
(3, 4, 5)	Return: Scalene(2)	BV
(1, 2, 3)	error (triangle inequality)(3)	BV
(-1, 2, 3)	error (invalid coordinates)(3)	BV

P5

```
public static boolean prefix(String s1, String s2){
if (s1.length() > s2.length()){
return false;  }
for (int i = 0; i < s1.length(); i++){
if (s1.charAt(i) != s2.charAt(i)){
return false;  }
}
return true;   }
```

1. **Equivalence Class Partitioning**

We can divide the inputs into valid and invalid equivalence classes.

Valid Equivalence Classes

- s1 is a valid prefix of s2.
- s1 is not a prefix of s2.
- s1 is an empty string (an empty string is a prefix of any string).
- s1 is longer than s2.

2. Boundary Value Analysis

Test boundary values for the lengths of s1 and s2:

- s1 and s2 are both empty strings.
- s1 is an empty string and s2 is non-empty.
- s1 has one character and s2 has the same character at the start.
- s1 and s2 have the same length and are equal.
- s1 is longer than s2.

Test Cases

Input Data	Expected Outcome	Type
("pre", "prefix")	Return: true	EP
("fix", "prefix")	Return: false	EP
("longer", "short")	Return: false	EP
("prefix", "prefix")	Return: true	EP
("a", "abc")	Return: true	BV
("abc", "abc")	Return: true	BV
("abcdef", "abc")	Return: false	BV
("abc", "abx")	Return: false	BV

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.

a) <u>Identify the equivalence classes for the system</u>

Equivalence Classes

We can identify different equivalence classes based on the properties of a triangle.

<u>Valid Equivalence Classes</u>

- Equilateral Triangle: All sides are equal ($A = B = C$).
- Isosceles Triangle: Two sides are equal ($A = B$ or $A = C$ or $B=C$).
- Scalene Triangle: No sides are equal ($A \neq B \neq C$).
- Right-Angled Triangle: The sides satisfy the Pythagorean theorem (A^2 + B^2 = C^2).

<u>Invalid Equivalence Classes</u>

- The sides do not satisfy the triangle inequality ($A + B \leq C$ or $A+ C \leq B$ or $B + C \leq A$).
- One or more sides are non-positive ($A \leq 0$ or $B \leq 0$ or $C \leq 0$).

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

Test Cases

Input Data	Expected Outcome	Equivalence Class
(3.0, 3.0, 3.0)	Equilateral	Equilateral (A=B=C)
(5.0, 5.0, 8.0)	Isosceles	Isosceles (A=B, A≠C)
(3.0, 4.0, 5.0)	Right-Angled	Right-Angled (A2+B2=C2)
(7.0, 8.0, 9.0)	Scalene	Scalene (A ≠ B ≠ C)
(1.0, 2.0, 3.0)	error	invalid
(0.0, 7.0, 8.0)	error	invalid
(-4.0, 6.0, 6.0)	error	invalid
(4.0, 4.0, 7.0)	error	invalid

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

This condition ensures that the sum of two sides is greater than the third.

Test Case 1

- Input: (3.0, 4.0, 7.0) (Boundary value where A + B = C)
- Expected Outcome: Invalid Triangle (fails triangle inequality).

Test Case 2

- Input: (7.0, 8.0, 9.0) (Boundary value where A + B >C)
- Expected Outcome: Scalene Triangle.

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

This condition checks whether two sides of the triangle are equal.

Test Case

- Input: (10.0, 8.0, 10.0) (Two sides are equal at the boundary).
- Expected Outcome: Isosceles Triangle.

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

This checks for cases where all three sides are equal.

Test Case 1

- Input: (8.0, 8.0, 8.0) (All sides are equal at the boundary).
- Expected Outcome: Equilateral Triangle.

Test Case 2

- Input: (13.0, 8.0, 8.0).
- Expected Outcome: Not an Equilateral Triangle.

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

This checks if the triangle satisfies the Pythagorean theorem.

Test Case

- Input: (5.0, 12.0, 13.0) (Classic Pythagorean triplet).
- Expected Outcome: Right-Angled Triangle.

g) For the non-triangle case, identify test cases to explore the boundary.

This tests the triangle inequality where the sum of two sides is not greater than the third side.

Test Case

- Input: (2.0,3.0, 5.0) (Boundary value where A + B = C).
- Expected Outcome: Invalid Triangle.

h) For non-positive input, identify test points.

This tests cases where one or more sides are non-positive.

Test Case 1

- Input: (0.0, 5.0, 6.0) (Zero side length).
- Expected Outcome: Invalid Triangle.

Test Case 2

- Input: (-4.0, 7.0, 7.0) (Negative side length).
- Expected Outcome: Invalid Triangle.