Lab 8

Functional Testing (Black-Box)

Rishi Shah
202201105

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?

Ans=

No.	**Equivalence Class**	**Validity**
1	Month Value < 1	**Invalid**
2	1 < Month Value <= 12	**Valid**
3	Month Value > 13	**Invalid**
4	Year Value < 1900	**Invalid**
5	1900 <= Year Value <= 2015	**Valid**
6	Year Value > 13	**Invalid**
7	Day Value < 1	**Invalid**
8	1 <= Day Value <= 31	**Valid**
9	Day Value > 31	**Invalid**

Test Cases:

Test Case	**Input (Day, Month, Year)**	**Class Addressed**	**Expected Output**	**Validity**	**Remark**
TC1	(1, 1, 2000)	E2, E5, E8	(31, 12, 1999)	Valid	
TC2	(32, 1, 2000)	E9	Error message	Invalid	Invalid day

Test Case	**Input (Day, Month, Year)**	**Class Addressed**	**Expected Output**	**Validity**	**Remark**
TC3	(15, 1, 2000)	E2, E5 E8	(14, 1, 2000)	Valid	
TC4	(1, 1, 1899)	E4	Error message	Invalid	Invalid year
TC5	(1, 1, 2016)	E6	Error message	Invalid	Invalid year
TC6	(1, 1, 2000)	E2, E5 E8	(31, 12, 1999)	Valid	
TC7	(0, 1, 2000)	E7	Error message	Invalid	Invalid day
TC8	(1, 0, 2000)	E1	Error message	Invalid	Invalid month
TC9	(1, 13, 2000)	E3	Error message	Invalid	Invalid month
TC10	(1, 1, 1899)	E4	Error message	Invalid	Invalid year
TC11	(1, 1, 2016)	E6	Error message	Invalid	Invalid year
TC12	(1, 1, 2000)	E2, E5, E8	(31, 12, 1999)	Valid	

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.

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.

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.

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

Equivalence Class Partitioning:

Equivalence Class Number	Details of Class	Validity
E1	Value v is present in the array	Valid

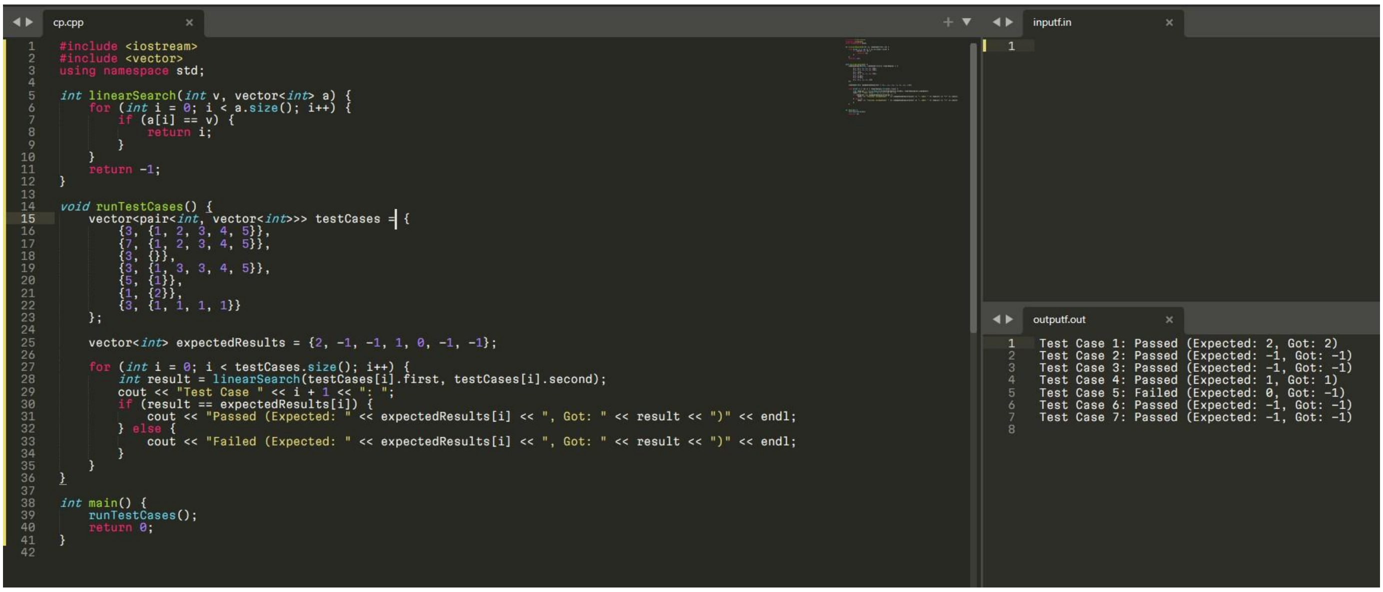
Equivalence Class Number	Details of Class	Validity
E2	Value `v` is not present in the array	Valid
E3	Array is empty	Invalid
E4	Array has duplicate values, value `v` present	Valid
E5	Array has duplicate values, value `v` not present	Valid
E6	Array is not empty	Valid

Test Cases (including Boundary Value Analysis):

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	`v = 3`, `a = [1, 2, 3, 4, 5]`	E1, E6	Index `2`	Value `3` is present
TC2	`v = 7`, `a = [1, 2, 3, 4, 5]`	E2, E6	`-1`	Value `7` not in the array
TC3	`v = 3`, `a = []`	E3	`-1`	Empty array
TC4	`v = 3`, `a = [1, 3, 3, 4, 5]`	E4, E6	Index `1`	First occurrence of `3`
TC5	`v = 5`, `a = [1]`	E1, E6	Index `0`	Boundary value: single element, matches
TC6	`v = 1`, `a = [2]`	E2, E6	`-1`	Boundary value: single element, does not match

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC7	`v = 3`, `a = [1, 1, 1, 1]`	E5, E6	`-1`	Value `3` not in array

Code Output:



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

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


}
```

```
return (count);

}
```

Equivalence Class Partitioning:

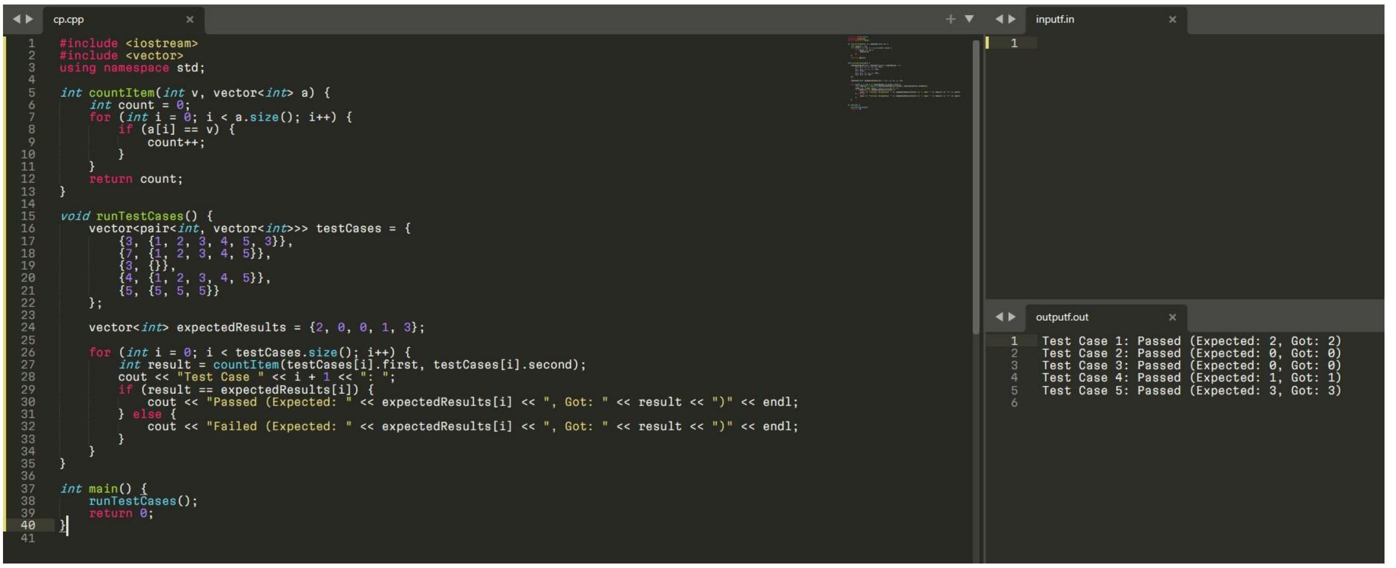
Equivalence Class Number	Details of Class	Validity
E1	Value `v` appears multiple times in the array	Valid
E2	Value `v` does not appear in the array	Valid
E3	Array is empty	Invalid
E4	Value `v` appears exactly once in the array	Valid
E5	Array contains duplicate values, value `v` is present	Valid
E6	Array is not empty	Valid

Test Cases (including Boundary Value Analysis):

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	`v = 3`, `a = [1, 2, 3, 4, 5, 3]`	E1, E6	2	Value `3` appears twice
TC2	`v = 7`, `a = [1, 2, 3, 4, 5]`	E2, E6	0	Value `7` does not appear
TC3	`v = 3`, `a = []`	E3	0	Empty array

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC4	`v = 4, a = [1, 2, 3, 4, 5]`	E4, E6	1	Value `4` appears once
TC5	`v = 5, a = [5, 5, 5]`	E1, E5, E6	3	Value `5` appears three times

Code Output:



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.

```
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);
}
```

Equivalence Class Partitioning:

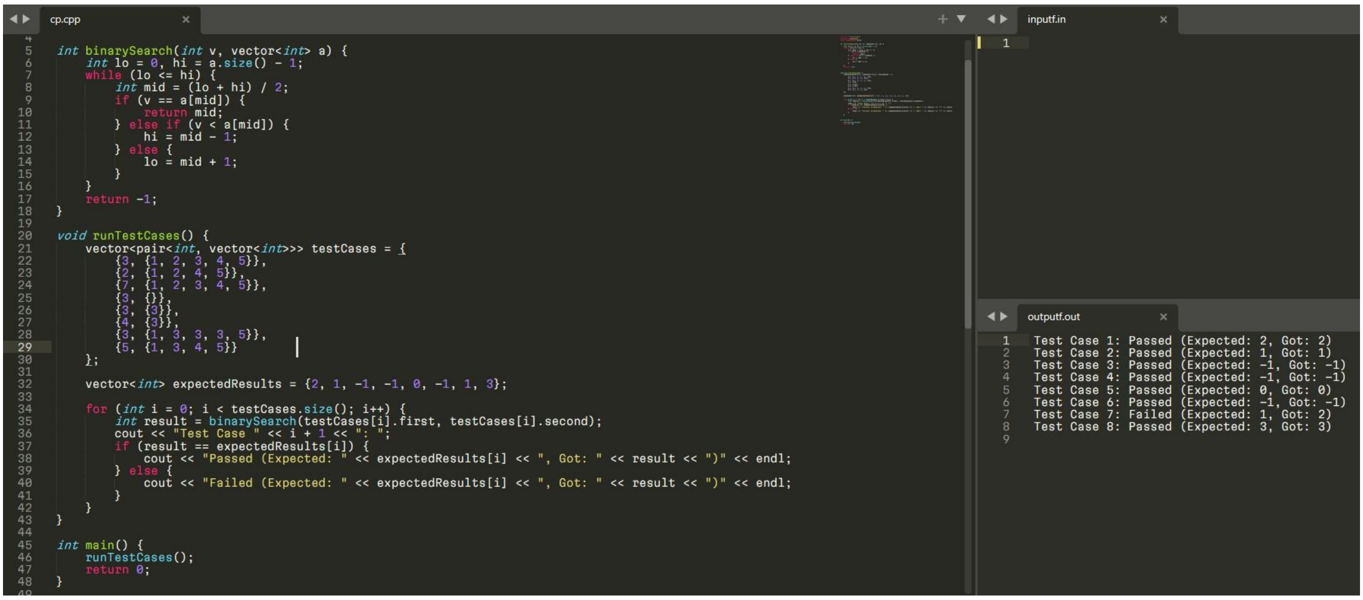
Equivalence Class Number	**Details of Class**	**Validity**
E1	Value `v` is present in the array (multiple occurrences)	Valid
E2	Value `v` is present in the array (single occurrence)	Valid
E3	Value `v` is not present in the array	Valid
E4	Array is empty	Invalid
E5	Array contains a single element	Valid

Equivalence Class Number	Details of Class	Validity
E6	Array contains duplicate values where value `v` is present	Valid
E7	Array is not empty	Valid

Test Cases (including Boundary Value Analysis):

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	`v = 3`, `a = [1, 2, 3, 4, 5]`	E1, E7	2	Value `3` is at index 2
TC2	`v = 2`, `a = [1, 2, 4, 5]`	E2, E7	1	Value `2` is at index 1
TC3	`v = 7`, `a = [1, 2, 3, 4, 5]`	E3, E7	-1	Value `7` is not present
TC4	`v = 3`, `a = []`	E4	-1	Empty array
TC5	`v = 3`, `a = [3]`	E5, E2	0	Value `3` is the only element, at index 0
TC6	`v = 4`, `a = [3]`	E3, E5	-1	Value `4` is not present
TC7	`v = 3`, `a = [1, 3, 3, 3, 5]`	E1, E6, E7	1	Value `3` appears multiple times, first at index 1
TC8	`v = 5`, `a = [1, 3, 4, 5]`	E2, E7	3	Value `5` is at index 3

Code Output:



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).

```
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);
}
```

Equivalence Class Partitioning:

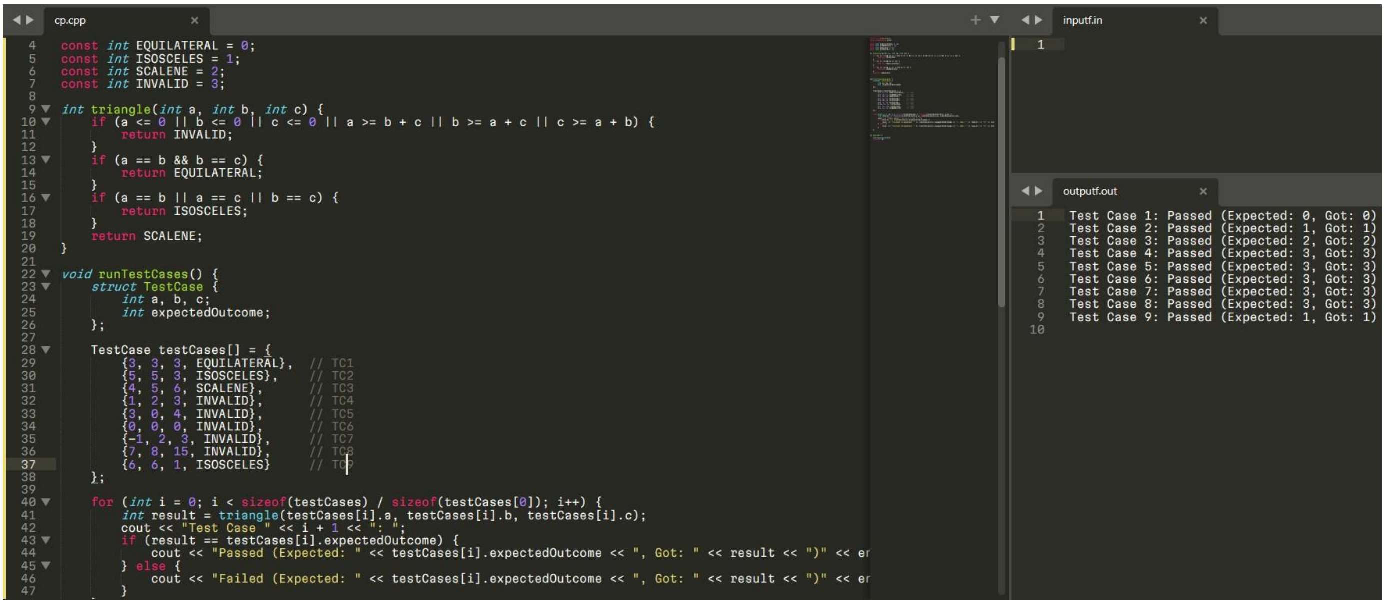
Equivalence Class Number	Details of Class	Validity
E1	Valid equilateral triangle (all sides equal)	Valid
E2	Valid isosceles triangle (two sides equal)	Valid
E3	Valid scalene triangle (no sides equal)	Valid
E4	Invalid triangle (two sides sum less than or equal to third)	Invalid
E5	One side is zero or negative	Invalid
E6	All sides are zero	Invalid

Test Cases (including Boundary Value Analysis):

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	`3, 3, 3`	E1	0	Equilateral triangle
TC2	`5, 5, 3`	E2	1	Isosceles triangle
TC3	`4, 5, 6`	E3	2	Scalene triangle

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC4	`1, 2, 3`	E4	3	Invalid triangle
TC5	`3, 0, 4`	E5	3	Invalid triangle (zero side)
TC6	`0, 0, 0`	E6	3	Invalid triangle (all sides zero)
TC7	`-1, 2, 3`	E5	3	Invalid triangle (negative side)
TC8	`7, 8, 15`	E3	3	Invalid triangle(line)
TC9	`6, 6, 1`	E2	1	Isosceles triangle

Code Output:



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).

```
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;
}
```

Equivalence Class Partitioning:

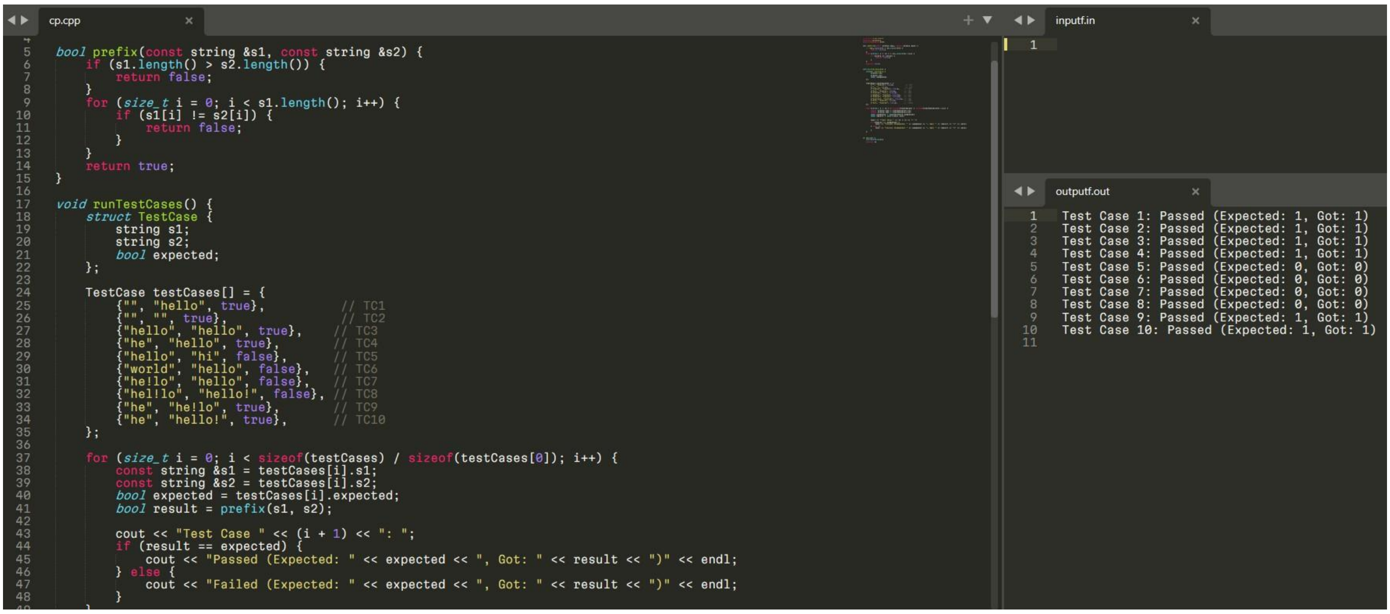
Equivalence Class Number	Details of Class	Validity
E1	`s1` is an empty string (prefix of any string)	Valid
E2	`s2` is an empty string (valid only if `s1` is also empty)	Invalid
E3	`s1` is a prefix of `s2`	Valid
E4	`s1` is equal to `s2`	Valid

Equivalence Class Number	Details of Class	Validity
E5	`s1` is longer than `s2`	Invalid
E6	`s1` is not a prefix of `s2`	Valid

Test Cases (including Boundary Value Analysis):

Test Case Number	Values for Input	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	`""`, `"hello"`	E1, E2	true	Empty string as prefix
TC2	`""`, `""`	E1, E2	true	Both strings are empty
TC3	`"hello"`, `"hello"`	E4	true	Strings are equal
TC4	`"he"`, `"hello"`	E3	true	Valid prefix
TC5	`"hello"`, `"hi"`	E5	false	`s1` is longer than `s2`
TC6	`"world"`, `"hello"`	E6	false	`s1` is not a prefix of `s2`
TC7	`"he!lo"`, `"hello"`	E6	false	Special characters in `s1`
TC8	`"hel!lo"`, `"hello!"`	E6	false	Different strings, `s1` not prefix
TC9	`"he"`, `"he!lo"`	E3	true	Valid prefix with special character
TC10	`"he"`, `"hello!"`	E3	true	Valid prefix with special character

Code Output:



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

Equivalence Class	Description	Validity
E1	Valid input: All three sides can form a triangle.	Valid
E2	Invalid input: At least one side is non-positive.	Invalid
E3	Valid but forms a scalene triangle (no sides equal).	Valid
E4	Valid but forms an isosceles triangle (two sides equal).	Valid

Equivalence Class	Description	Validity
E5	Valid but forms an equilateral triangle (all sides equal).	Valid
E6	Valid but forms a right-angled triangle (satisfies $A^2 + B^2 = C^2$).	Valid
E7	Invalid input: A, B, and C do not satisfy triangle inequality (e.g., $A + B \leq C$).	Invalid

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)

Test Case Number	Values (A, B, C)	Covered Equivalence Classes	Expected Outcome
TC1	(3.0, 4.0, 5.0)	E1, E3	Scalene triangle
TC2	(2.0, 2.0, 2.0)	E1, E5	Equilateral triangle
TC3	(2.0, 2.0, 3.0)	E1, E8	Isosceles triangle
TC4	(5.0, 5.0, 7.0)	E1, E8	Isosceles triangle
TC5	(5.0, 12.0, 13.0)	E1, E6	Right-angled triangle
TC6	(1.0, 1.0, 3.0)	E7	Invalid triangle (not formable)
TC7	(0.0, 2.0, 2.0)	E2	Invalid input
TC8	(-1.0, 2.0, 2.0)	E2	Invalid input
TC9	(3.0, 4.0, 8.0)	E7	Invalid triangle (not formable)
TC10	(3.0, 3.0, 5.0)	E8	Isosceles triangle

Test Case Number	Values (A, B, C)	Covered Equivalence Classes	Expected Outcome
TC11	(2.0, 2.0, 5.0)	E8	Isosceles triangle
TC12	(5.0, 5.0, 10.0)	E9	Invalid triangle (not formable)
TC13	(0.0, 0.0, 1.0)	E2	Invalid input
TC14	(1.0, 1.0, 0.0)	E2	Invalid input
TC15	(1.0, 1.0, -1.0)	E2	Invalid input

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(3.0, 2.0, 6.0)	Invalid triangle
TC2	(4.0, 5.0, 8.0)	Scalene triangle
TC3	(5.0, 5.0, 10.0)	Invalid triangle

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(5.0, 3.0, 5.0)	Isosceles triangle
TC2	(7.0, 15.0, 7.0)	Invalid triangle
TC3	(3.0,4.0,5.0)	Right triangle(Not Isoceles)

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(5.0, 5.0, 5.0)	Equilateral triangle
TC2	(1.0, 2.0, 1.0)	Invalid triangle

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(0.3, 0.4, 0.5)	Right-angled triangle
TC2	(5.0, 12.0, 13.0)	Right-angled triangle

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(2.0, 1.0, 3.0)	Invalid triangle (not formable)
TC2	(1.0, 1.0, 2.0)	Invalid triangle (not formable)

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

Test Case Number	Values (A, B, C)	Expected Outcome
TC1	(0.0, 0.0, 0.0)	Invalid input
TC2	(0.0, 1.0, 1.0)	Invalid input
TC3	(-1.0, -2.0, -3.0)	Invalid input