IT-314: Software Engineering

Lab-08: Functional Testing (Black-Box)

Hitanshu variya – 202201510

BTech, ICT + CS

Section - B

Question 01) Consider a program for determining the previous date. Its input is a 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 the previous date or invalid date.
Design the equivalence class test cases?

Solution:

❖ Derived Equivalence Classes:

No	Equivalence Class	Acceptance
E1	Month value is Non-Numeric.	Invalid
E2	Month value is Numeric.	Invalid
E3	Month value is Decimal.	Invalid
E4	Month value is Natural Number.	Valid
E5	Month value is between 1 and 12 (inclusive)	Valid
E6	Month value is less than 1.	Invalid
E7	Month value is greater than 12.	Invalid
E8	Month value is Empty	Invalid
E9	Month Value contains blanks between digits	Invalid
E10	Month Value does not contains blanks between digits	Valid
E11	Day value is Non-Numeric.	Invalid
E12	Day value is Numeric.	Invalid
E13	Day value is Decimal.	Invalid
E14	Day value is Natural Number.	Valid
E15	Day value is between 1 and 31 (inclusive)	Valid
E16	Day value is less than 1.	Invalid
E17	Day value is greater than 31.	Invalid
E18	Day value is Empty	Invalid
E19	Day Value contains blanks between digits	Invalid

E20	Day Value does not contains blanks between digits	Valid
E21	Year value is Non-Numeric.	Invalid
E22	Year value is Numeric.	Invalid
E23	Year value is Decimal.	Invalid
E24	Year value is Natural Number.	Valid
E25	Year value is between 1900 and 2015 (inclusive)	Valid
E26	Year value is less than 1900.	Invalid
E27	Year value is greater than 2015.	Invalid
E28	Year value is Empty	Invalid
E29	Year Value contains blanks between digits	Invalid
E30	Year Value does not contains blanks between digits	Valid

❖ Black Box Test cases for the Date Tuple Based on the Equivalence classes above:

No	Test Data	Expected Outcome	Classes Covered
1	(5, 5, 1960)	T	E4, E14, E24, E5, E15, E25, E10, E20, E30
2	(abc, 5, 1960)	F	E1
3	(5.3, 5, 1960)	F	E2
4	(5.3333…, 5, 1960)	F	E3
5	(-2, 5, 1960)	F	E6
6	(30, 5, 1960)	F	E7
7	(, 5, 1960)	F	E8
8	(1 2, 5, 1960)	F	E9
9	(5, abc, 1960)	F	E11
10	(5, 5.3, 1960)	F	E12
11	(5, 5.3333…, 1960)	F	E13
12	(5, -2, 1960)	F	E16
13	(5, 64, 1960)	F	E17
14	(5 , , 1960)	F	E18
15	(5, 2 2, 1960)	F	E19
16	(5, 5, abc)	F	E21
17	(5, 5, 1960.33)	F	E22
18	(5, 5, 1960.333…)	F	E23
19	(5, 5, -2)	F	E26
20	(5, 5, 2024)	F	E27
21	(5, 5,)	F	E28
22	(5, 5, 19 60)	F	E29

Question 02) 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.

Problem 01) 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);
}
```

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	Search for the value in the array	First occurrence index of target value
E2	Search for the value not in the array	-1
E3	Search in an empty array	-1

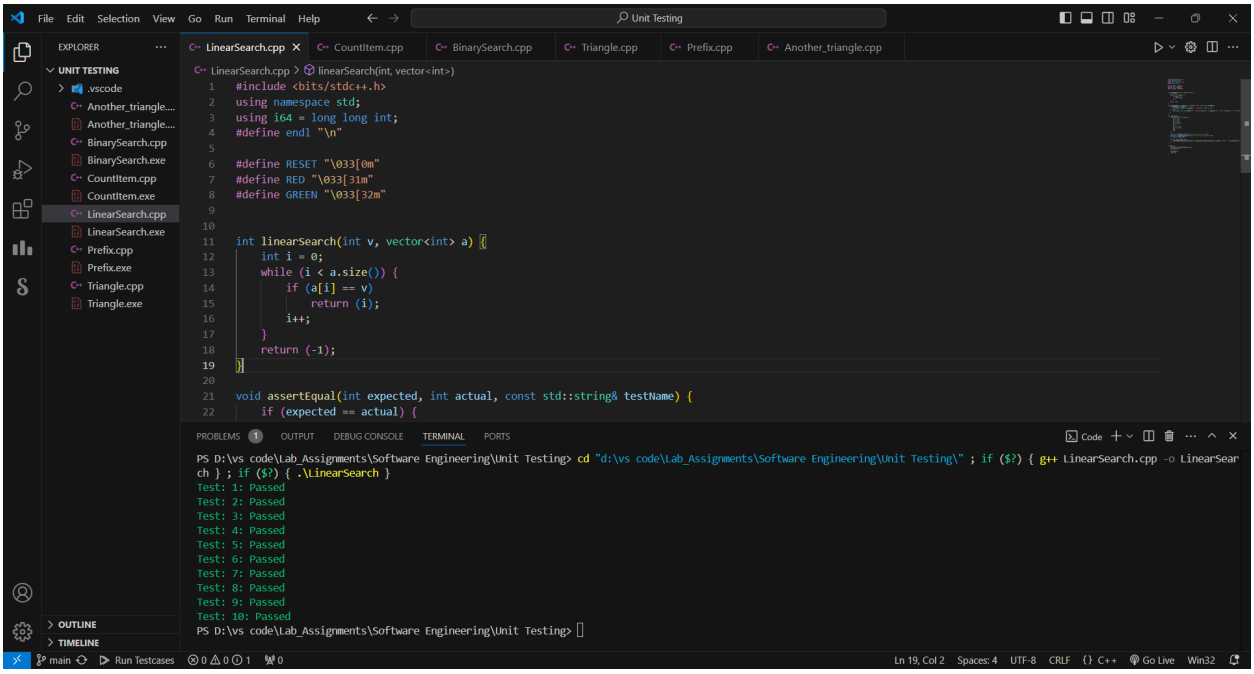
Boundary Value Analysis:

No	Boundary Condition	Expected Outcome
B1	Search for the First element of the array	0
B2	Search for the Last element of the array	size of array - 1
B3	Search for single present element in single sized array	0
B4	Search for element other than single present element in single sized array	-1

Test Suit for Linear Search:

Tester Action and Input Data		Expected Outcome
Equivalence Partitioning		
E1	v = 2, a = [1, 2, 3]	1
E1	v = 1, a = [1, 2, 3]	0
E1	v = 3, a = [1, 2, 3]	2
E1	v = 4, a = [1, 4, 4, 3]	1
E2	v = 2, a = [1, 5, 3]	-1
E3	v = 2, a = []	-1
Boundary Value Analysis		
B1	v = 1, a = [1, 2, 3, 4]	0
B2	v = 4, a = [1, 2, 3, 4]	3
B3	v = 1, a = [1]	0
B4	v = 2, a = [1]	-1

❖ Unit Testing Output:



Problem 02) 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);
}
```

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	Count a value present multiple times in the array	Number of occurrences
E2	Count a value present once in the array	1
E3	Count a value not present in the array	0
E4	Count in an empty array	0

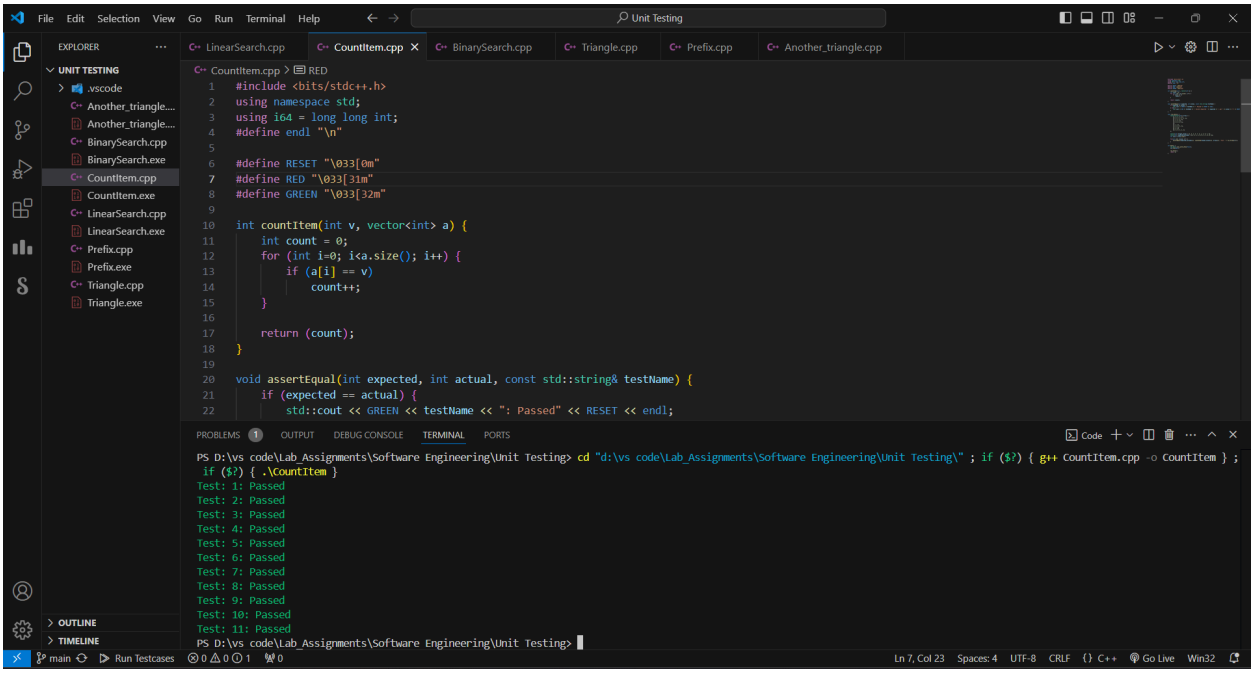
Boundary Value Analysis:

No	Boundary Condition	Expected Outcome
B1	Count the first element in the array	Number of occurrences
B2	Count the last element in the array	Number of occurrences
B3	Count a value one less than the minimum value in the array	0
B4	Count a value one more than the maximum value in the array	0
B5	Count in an array with all elements equal to the search value	size of array

Test Suit for Linear Search:

Tester Action and Input Data		Expected Outcome
Equivalence Partitioning		
E1	v = 2, a = [1, 2, 2, 3, 2]	3
E1	v = 4, a = [4, 4, 1, 2, 4, 4, 3]	4
E2	v = 3, a = [1, 2, 3, 4, 5]	1
E2	v = 5, a = [1, 2, 3, 5]	1
E3	v = 7, a = [1, 2, 3, 4, 5]	0
E4	v = 5, a = []	0
Boundary Value Analysis		
B1	v = 1, a = [1, 2, 3]	1
B2	v = 4, a = [1, 2, 3, 4, 5]	1
B3	v = 0, a = [1, 2, 3]	0
B4	v = 7, a = [4, 5, 6]	0
B5	v = 5, a = [5, 5, 5, 5, 5, 5]	6

❖ Unit Testing Output:



Problem 03) 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);
}
```

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	Search for a value present in the middle of the array	Index of the value
E2	Search for a value present in the first half of the array	Index of the value
E3	Search for a value present in the second half of the array	Index of the value
E4	Search for a value not present in the array, but within its range	-1
E5	Search in an empty array	-1

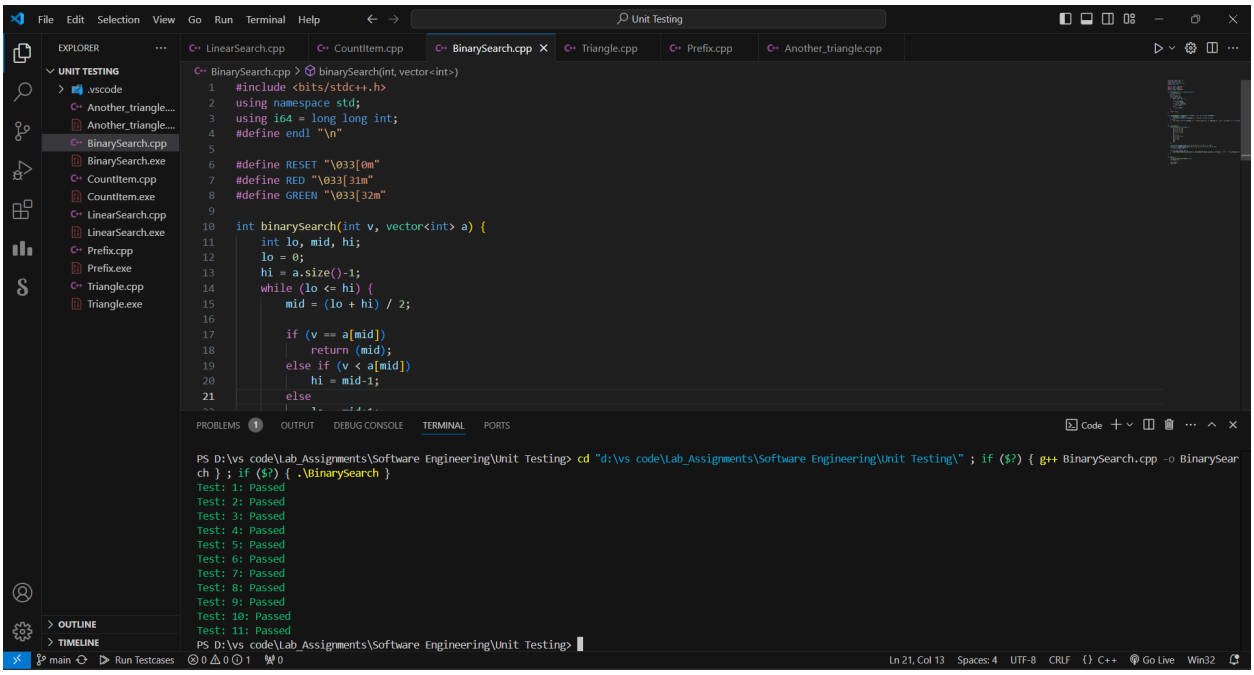
Boundary Value Analysis:

No	**Boundary Condition**	**Expected Outcome**
B1	Search for the first element in the array	0
B2	Search for the last element in the array	Array size - 1
B3	Search for a value one less than the first element	-1
B4	Search for a value one more than the last element	-1
B5	Search in an array with only one element, matching the search value	0
B6	Search in an array with only one element, not matching the search value	-1

Test Suit for Linear Search:

Tester Action and Input Data		**Expected Outcome**
Equivalence Partitioning		
E1	v = 3, a = [1, 2, 3, 4, 5]	2
E2	v = 2, a = [1, 2, 4, 4, 5]	1
E3	v = 8, a = [5, 6, 7, 8, 9]	3
E4	v = 10, a = [1, 2, 3, 4, 5]	-1
E5	v = 5, a = []	-1
Boundary Value Analysis		
B1	v = 1, a = [1, 2, 3]	0
B2	v = 5, a = [1, 2, 3, 4, 5]	4
B3	v = 0, a = [1, 2, 3]	-1
B4	v = 7, a = [4, 5, 6]	-1
B5	v = 1, a = [1]	0
B6	v = 10, a = [1]	-1

❖ Unit Testing Output:



Problem 04) 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).

```
const int EQUILATERAL = 0;
const int ISOSCELES = 1;
const int SCALENE = 2;
const int INVALID = 3;

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

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	All sides of triangle are equal	EQUILATERAL
E2	Two sides of triangle are equal	ISOSCELES
E3	No two sides of triangle are equal	SCALENE
E4	one side of triangle equal to sum of other two	INVALID
E5	one side of triangle greater than sum of other two	INVALID

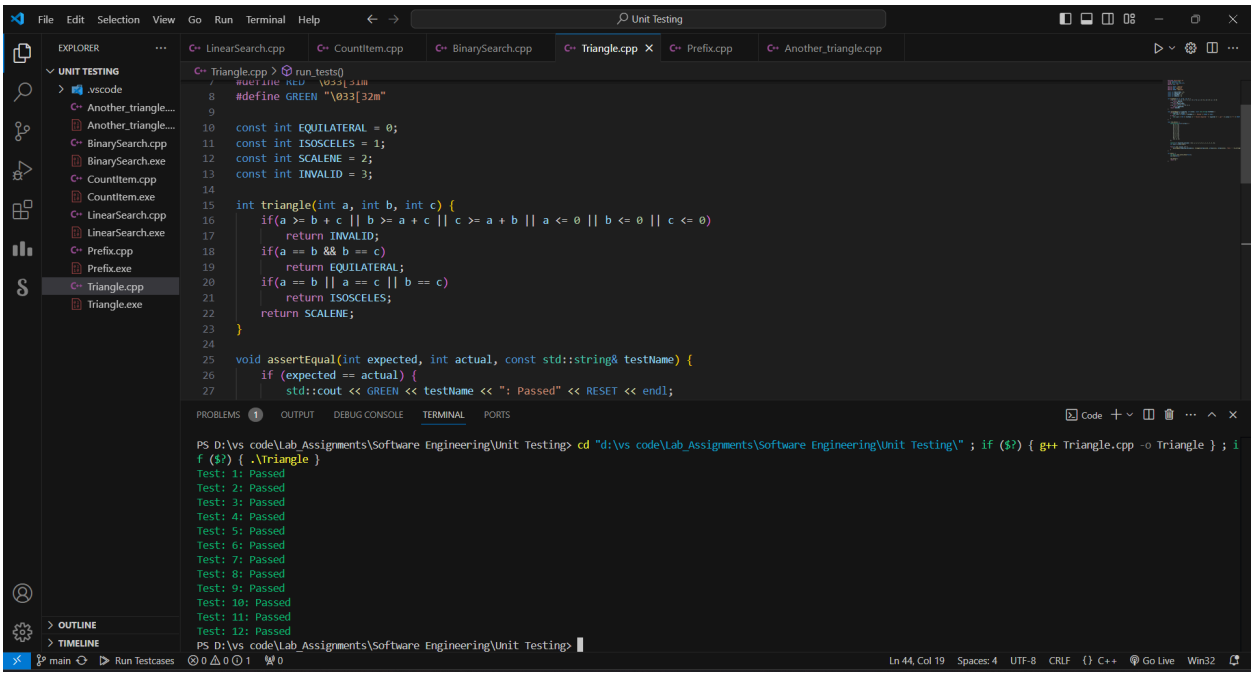
Boundary Value Analysis:

No	**Boundary Condition**	**Expected Outcome**
B1	Minimum valid equilateral triangle	EQUILATERAL
B2	Minimum valid isosceles triangle	ISOSCELES
B3	Minimum valid scalene triangle	SCALENE
B4	Just valid triangle	SCALENE
B5	Just invalid triangle	INVALID
B6	Any number of sides of triangle are zero	INVALID
B7	Any number of sides of triangle are negative	INVALID

Test Suit for Linear Search:

Tester Action and Input Data		**Expected Outcome**
Equivalence Partitioning		
E1	(5, 5, 5)	EQUILATERAL
E2	(5, 5, 3)	ISOSCELES
E3	(3, 4, 5)	SCALENE
E4	(1, 2, 3)	INVALID
E5	(1, 1, 3)	INVALID
Boundary Value Analysis		
B1	(1, 1, 1)	EQUILATERAL
B2	(2, 2, 1)	ISOSCELES
B3	(3, 4, 5)	SCALENE
B4	(3, 4, 6)	SCALENE
B5	(3, 4, 7)	INVALID
B6	(0, 1, 1)	INVALID
B7	(-1, 1, 1)	INVALID

❖ Unit Testing Output:



Problem 05) 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)

```
bool prefix(string &s1, string &s2) {
    if (s1.size() > s2.size())
        return false;
    for (int i = 0; i < s1.size(); i++) {
        if (s1[i] != s2[i])
            return false;
    }
    return true;
}
```

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	s1 is a prefix of s2	true
E2	s1 is not a prefix of s2	false
E3	s1 and s2 are empty strings	true
E4	s1 is longer than s2	false

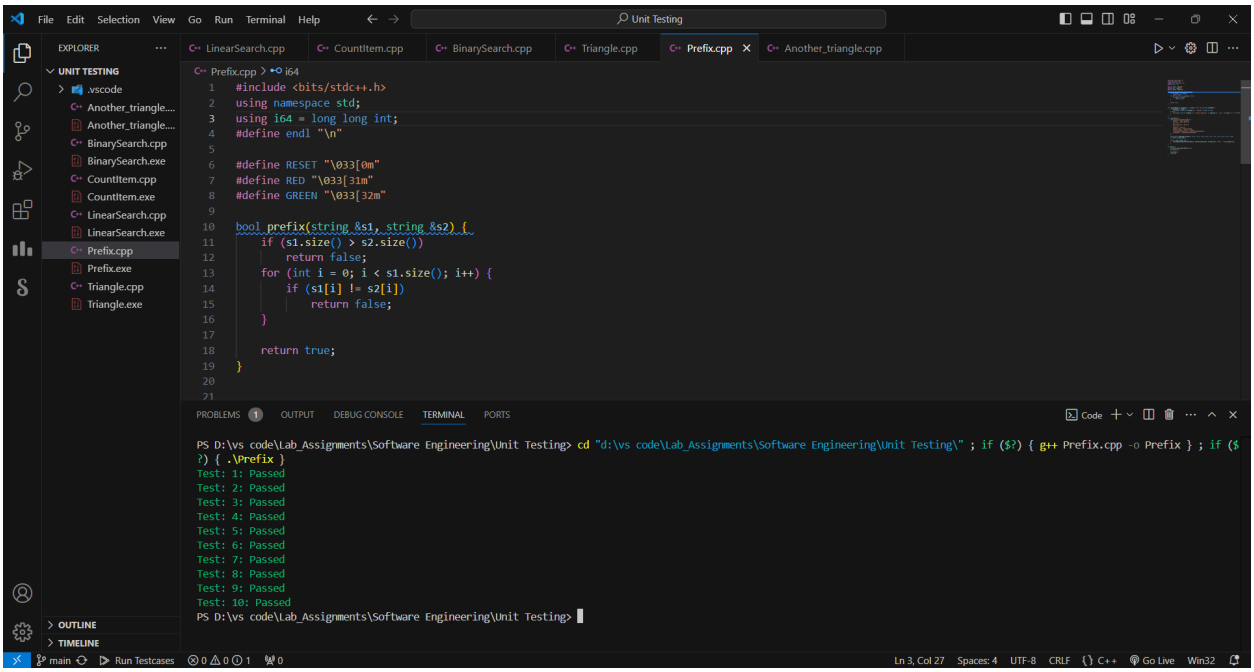
Boundary Value Analysis:

No	Boundary Condition	Expected Outcome
B1	s1 and s2 are both empty strings	true
B2	s1 is the same length as s2	true
B3	s1 is prefix with one character shorter than s2	true
B4	s1 is prefix with one character longer than s2	false
B5	s1 is the maximum possible length (let's say 20) and equal to s2	true
B6	s2 is the maximum possible length (let's say 20) and s1 is prefix	true

Test Suit for Linear Search:

Tester Action and Input Data		**Expected Outcome**
Equivalence Partitioning		
E1	(“hello”, “hello world”)	true
E2	(“world”, “hello world”)	false
E3	(“”, “”)	true
E4	(“hello world”, “hello”)	false
Boundary Value Analysis		
B1	(“”, “”)	true
B2	(“hello”, “hello”)	true
B3	(“hello worl”, “hello world”)	true
B4	(“hello world!”, “hello world”)	false
B5	(“IIIIIIIIIIIIIIIIII”, “IIIIIIIIIIIIIIIIIII”)	true
B6	(“IIIIIIII”, “IIIIIIIIIIIIIIIIIII”)	true

❖ Unit Testing Output:



Problem 06) 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.
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 $A^2 + B^2 = C^2$ 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.

```
const int EQUILATERAL = 0;
const int ISOSCELES = 1;
const int SCALENE = 2;
const int RIGHT_ANGLED = 3;
const int INVALID = 4;

int triangle(double a, double b, double c) {
    if (a >= b + c || b >= a + c || c >= a + b || a <= 0 || b <= 0 || c <= 0)
        return INVALID;
    if (a == b && b == c)
        return EQUILATERAL;
    if (a == b || a == c || b == c)
        return ISOSCELES;
    if (a*a + b*b == c*c || a*a + c*c == b*b || c*c + b*b == a*a)
        return RIGHT_ANGLED;
    return SCALENE;
}
```

Solution:

Equivalence Partitioning:

No	Equivalence Class	Expected Outcome
E1	All sides of triangle are equal	EQUILATERAL
E2	Two sides of triangle are equal	ISOSCELES
E3	No two sides of triangle are equal	SCALENE
E4	Square of sum of any two sides equals the square of third side	RIGHT_ANGLED
E5	one side of triangle equal to sum of other two	INVALID
E6	one side of triangle greater than sum of other two	INVALID
E7	Just valid triangle	SCALENE
E8	Just invalid triangle	INVALID
E9	Any number of sides of triangle are zero	INVALID
E10	Any number of sides of triangle are negative	INVALID

Test Cases for Equivalence Partitioning:

No	Test Data	Expected Outcome	Equivalence Class covered
TC1	(5.0, 5.0, 5.0)	EQUILATERAL	E1
TC2	(5.0, 5.0, 3.0)	ISOSCELES	E2
TC3	(7.0, 10.0, 12.0)	SCALENE	E3
TC4	(3.0, 4.0, 5.0)	RIGHT_ANGLED	E4
TC5	(1.0, 2.0, 3.0)	INVALID	E5
TC6	(1.0, 1.0, 3.0)	INVALID	E6
TC7	(3.0, 4.0, 6.0)	SCALENE	E7
TC8	(3.0, 4.0, 7.0)	INVALID	E8
TC9	(0.0, 1.0, 1.0)	INVALID	E9

No	Test Data	Expected Outcome	Equivalence Class covered
TC10	(-1.0, 1.0, 1.0)	INVALID	E10

Boundary Value Analysis for Scalene Triangle :

No	Boundary Condition	Expected Outcome
B1	Minimum Scalene Triangle	SCALENE
B2	Just Valid Scalene Triangle	SCALENE
B3	Just Invalid Scalene Triangle	INVALID

Test Cases for Boundary Value Analysis for Scalene Triangle :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(7.0, 10.0, 12.0)	SCALENE	B1
TC2	(3.0, 4.0, 6.0)	SCALENE	B2
TC3	(3.0, 4.0, 7.0)	INVALID	B3

Boundary Value Analysis for Isosceles Triangle :

No	Boundary Condition	Expected Outcome
B1	Minimum Isosceles Triangle	ISOSCELES
B2	Just Valid Isosceles Triangle	ISOSCELES
B3	Just Invalid Isosceles Triangle	INVALID

Test Cases for Boundary Value Analysis for Isosceles Triangle :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(2.0, 2.0, 1.0)	ISOSCELES	B1
TC2	(2.0, 2.0, 2.1)	ISOSCELES	B2
TC3	(2.0, 2.0, 5.2)	INVALID	B3

Boundary Value Analysis for Equilateral Triangle :

No	Boundary Condition	Expected Outcome
B1	Minimum Equilateral Triangle	EQUILATERAL
B2	Just Valid Equilateral Triangle	EQUILATERAL
B3	Just Invalid Equilateral Triangle	ISOSCELES

Test Cases for Boundary Value Analysis for Equilateral Triangle :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(1.0, 1.0, 1.0)	EQUILATERAL	B1
TC2	(5.0, 5.0, 5.0)	EQUILATERAL	B2
TC3	(5.0, 5.0, 5.1)	ISOSCELES	B3

Boundary Value Analysis for Right angled Triangle :

No	Boundary Condition	Expected Outcome
B1	Minimum Right angled Triangle	RIGHT_ANGLED
B2	Just Valid Right angled Triangle	RIGHT_ANGLED
B3	Just Invalid Right angled Triangle	SCALENE

Test Cases for Boundary Value Analysis for Right angled Triangle :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(3.0, 4.0, 5.0)	RIGHT_ANGLED	B1
TC2	(5.0, 12.0, 13.0)	RIGHT_ANGLED	B2
TC3	(5.0, 12.0, 13.1)	SCALENE	B3

Boundary Value Analysis for non-triangle :

No	Boundary Condition	Expected Outcome
B1	Just Valid triangle	SCALENE
B2	Just Valid Non-triangle	INVALID

Test Cases for Boundary Value Analysis for non-triangle :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(3.0, 4.0, 6.0)	SCALENE	B1
TC2	(3.0, 4.0, 8.0)	INVALID	B2

Boundary Value Analysis for non-positive inputs :

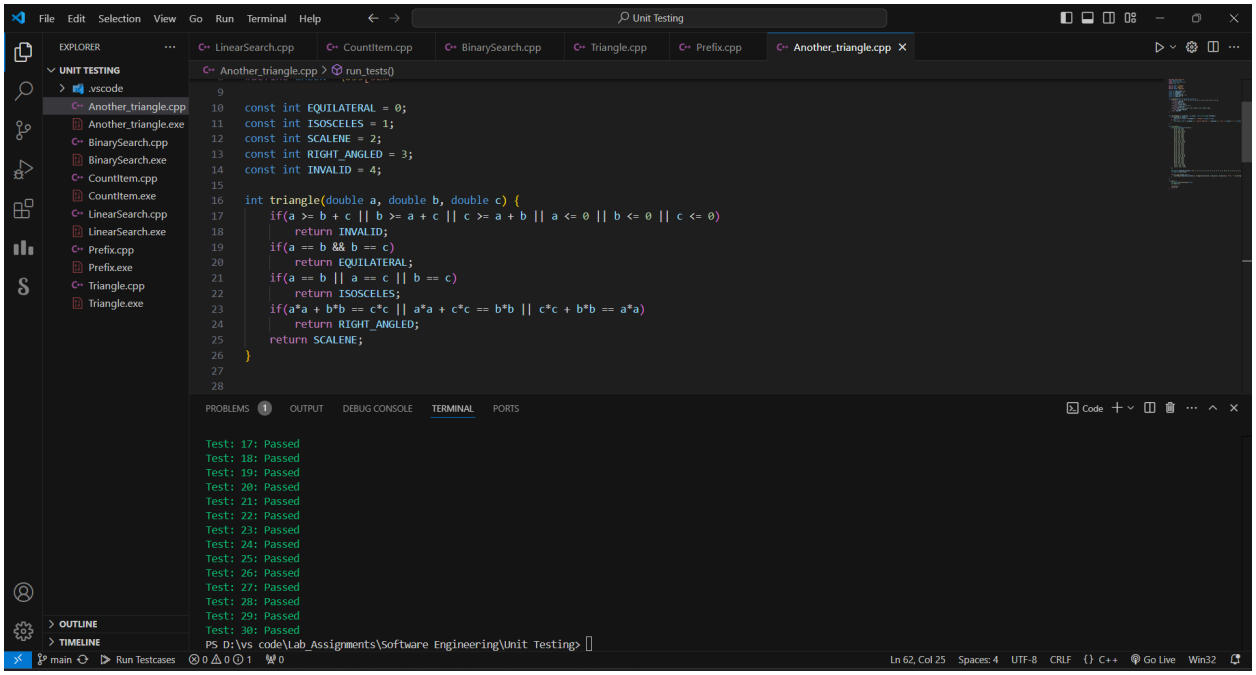
No	Boundary Condition	Expected Outcome
B1	One side zero	INVALID
B2	Two Sides zero	INVALID
B3	All Sides zero	INVALID
B4	One Side Negative	INVALID
B5	Two Sides Negative	INVALID
B6	All Sides Negative	INVALID

Test Cases for Boundary Value Analysis for non-positive inputs :

No	Test Data	Expected Outcome	Boundary Class covered
TC1	(0.0, 4.0, 8.0)	INVALID	B1
TC2	(0.0, 0.0, 7.0)	INVALID	B2
TC3	(0.0, 0.0, 0.0)	INVALID	B3
TC4	(-1.0, 4.0, 8.0)	INVALID	B4

No	Test Data	Expected Outcome	Boundary Class covered
TC5	(-1.0, -2.0, 8.0)	INVALID	B5
TC6	(-1.0, -4.0, -8.0)	INVALID	B6

❖ Unit Testing Output:



Note: Please find all the Unit Testing Codes in the attached folder.