Dhirubhai Ambani Institute of Information and Communication Technology

DA-IICT

IT314 Software Engineering

Name : Rathva sunilkumar

Id : 202201177

Lab 8

Functional Testing (Black-Box)

Q-1

Equivalence Partitioning

In this case, we have three input parameters: day, month, and year. Let's identify the equivalence classes for each parameter:

Day

● Valid days: 1 to 31

● Invalid days: 0, negative numbers, and numbers greater than 31

Month

● Valid months: 1 to 12

● Invalid months: 0, negative numbers, and numbers greater than 12

Year

● Valid years: 1900 to 2015

● Invalid years: Years before 1900 and after 2015

Equivalence Partitioning Test Cases

Tester Action and Input Data	Expected Outcome
Valid day (15), valid month (6), valid year (2000)	Previous date or valid date
Invalid day (0), valid month (6), valid year (2000)	An Error message
Valid day (15), invalid month (0), valid year (2000)	An Error message
Valid day (15), valid month (13), valid year (2000)	An Error message
Valid day (15), valid month (6), invalid year (1899)	An Error message
Valid day (15), valid month (6), invalid year (2016) An Error message	An Error message

Invalid day (0), invalid month (0), valid year (2000)	An Error message
Invalid day (32), invalid month (13), valid year (2000)	An Error message
Valid day (1), valid month (1), invalid year (2016)	An Error message
Valid day (31), valid month (12), invalid year (1899)	An Error message
Invalid day (0), invalid month (0), invalid year (1899)	An Error message
Invalid day (32), invalid month (13), invalid year (2016)	An Error message

Boundary Value Analysis

Boundary Value Analysis focuses on testing at the boundaries between equivalence partitions.

Day

- Test with values: 1, 31, 0, 32

Month

- Test with values: 1, 12, 0, 13

Year

- Test with values: 1900, 2015, 1899, 2016

Boundary Value Analysis Test Cases

Tester Action and Input Data
Expected Outcome

Tester Action and Input Data	Expected Outcome
Valid day (1), valid month (1), valid year (1900)	Previous date or valid date
Valid day (31), valid month (12), valid year (2015)	Previous date or valid date
Invalid day (0), valid month (6), valid year (2000)	An Error message
Invalid day (32), valid month (6), valid year (2000)	An Error message
Valid day (15), invalid month (0), valid year (2000)	An Error message

Valid day (15), invalid month (13), valid year (2000)	**An Error message**
Valid day (15), valid month (6), invalid year (1899)	**An Error message**
Valid day (15), valid month (6), invalid year (2016)	**An Error message**

Question 2

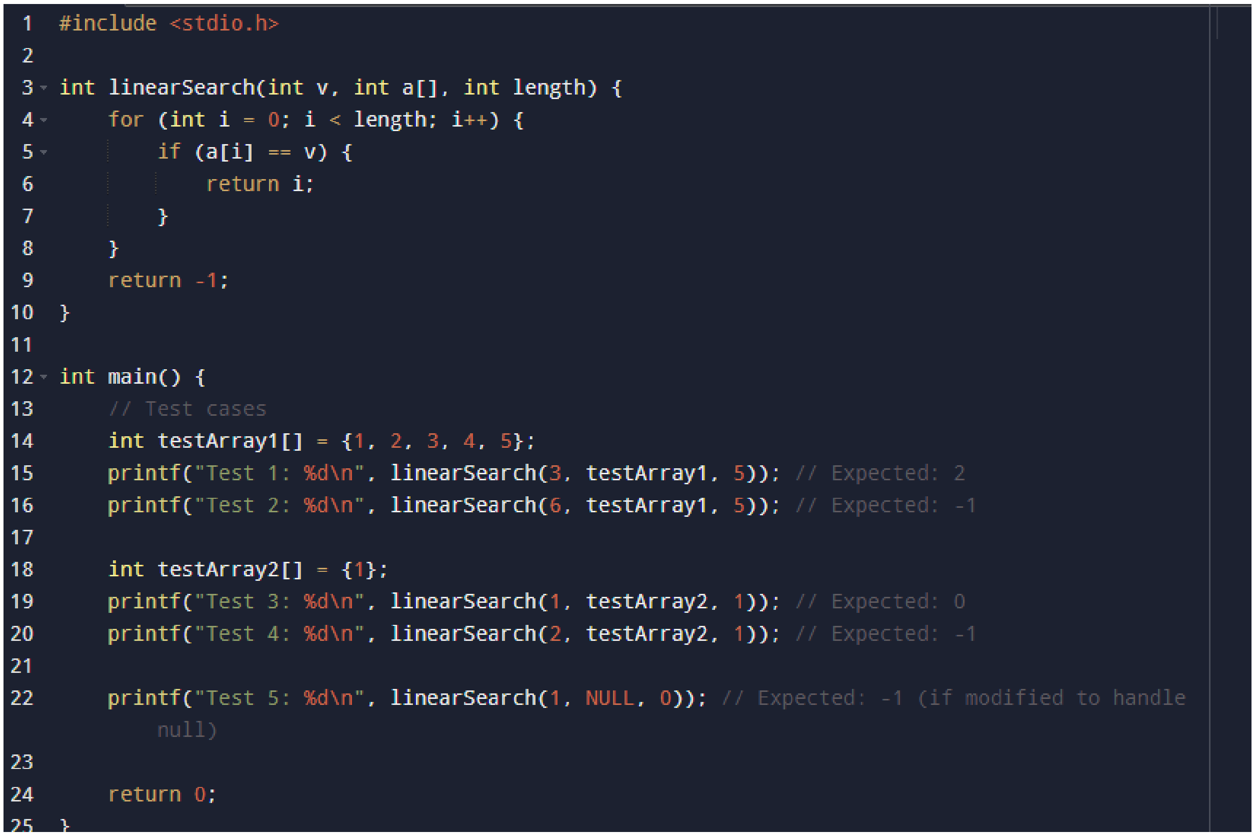
P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
5, [1, 2, 3, 4, 5]	**4**
10, [1, 2, 3, 4, 5]	**-1**
0, []	**-1**
5, null	**Error message**
"a", [1, 2, 3]	**Error message**

5, [1, "2", 3, 4]	Error message

Boundary Value Analysis	
1, [1]	0
2, [1]	-1
1, [1, 2, 3, 4, 5]	0
5, [1, 2, 3, 4, 5]	4

ii) Modified Program



```c
#include <stdio.h>

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

int main() {
    // Test cases
    int testArray1[] = {1, 2, 3, 4, 5};
    printf("Test 1: %d\n", linearSearch(3, testArray1, 5)); // Expected: 2
    printf("Test 2: %d\n", linearSearch(6, testArray1, 5)); // Expected: -1

    int testArray2[] = {1};
    printf("Test 3: %d\n", linearSearch(1, testArray2, 1)); // Expected: 0
    printf("Test 4: %d\n", linearSearch(2, testArray2, 1)); // Expected: -1

    printf("Test 5: %d\n", linearSearch(1, NULL, 0)); // Expected: -1 (if modified to handle null)

    return 0;
}
```

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

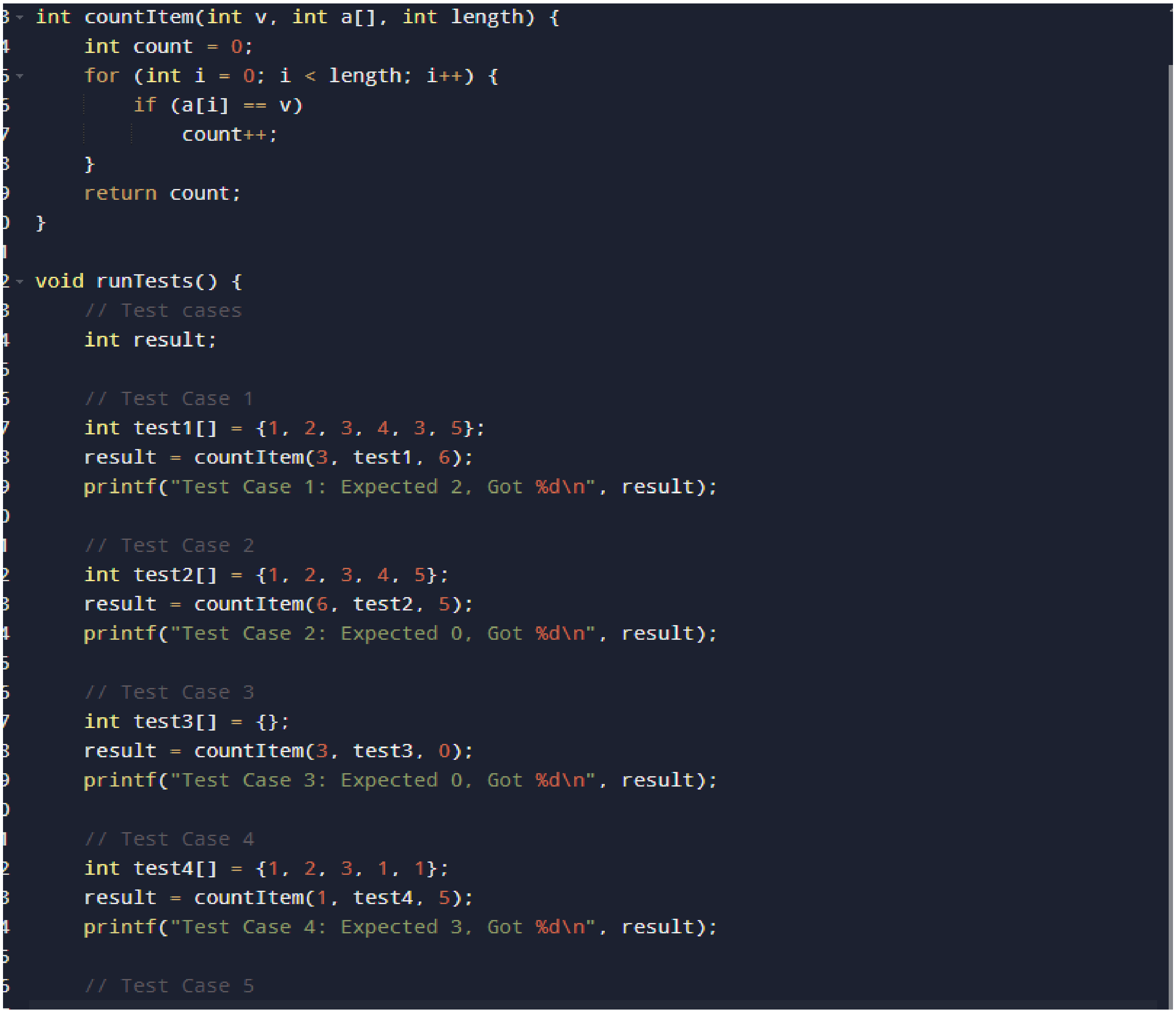
i) Test Suite

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	

3, [1, 2, 3, 4, 3, 5]	2
6, [1, 2, 3, 4, 5]	0
3, []	0
1, [1, 2, 3, 1, 1]	3
1, [2, 3, 4, 1]	1
"a", [1, 2, 3]	Error

Boundary Test Cases	
5, [5]	1
5, [1]	0
2, [2, 1]	1
1, [2, 1]	1

ii) Modified Program



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

void runTests() {
    // Test cases
    int result;

    // Test Case 1
    int test1[] = {1, 2, 3, 4, 3, 5};
    result = countItem(3, test1, 6);
    printf("Test Case 1: Expected 2, Got %d\n", result);

    // Test Case 2
    int test2[] = {1, 2, 3, 4, 5};
    result = countItem(6, test2, 5);
    printf("Test Case 2: Expected 0, Got %d\n", result);

    // Test Case 3
    int test3[] = {};
    result = countItem(3, test3, 0);
    printf("Test Case 3: Expected 0, Got %d\n", result);

    // Test Case 4
    int test4[] = {1, 2, 3, 1, 1};
    result = countItem(1, test4, 5);
    printf("Test Case 4: Expected 3, Got %d\n", result);

    // Test Case 5
```

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.

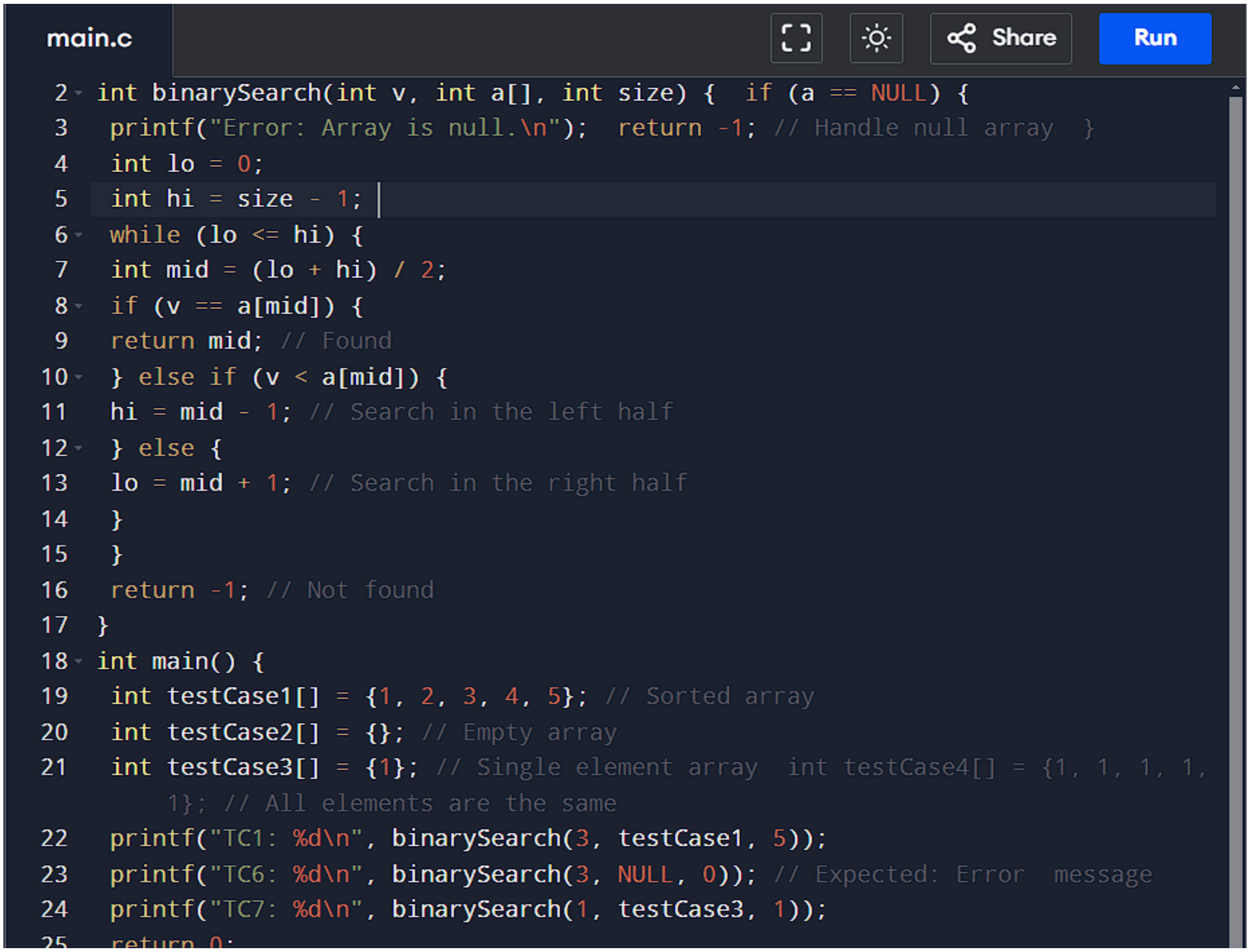
i) Test Suite

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
3, [1, 2, 3, 4, 5]	**2**
10, [1, 2, 3, 4, 5]	**-1**
1, [1, 2, 3, 4, 5]	**0**
5, [1, 2, 3, 4, 5]	**4**
3, []	**-1**
3, null	**Error message**

Boundary Value Analysis	

1, [1]	0
2, [1]	-1
1, [1, 2, 3, 4, 5]	0
5, [1, 2, 3, 4, 5]	4
3, [1, 1, 1, 1, 1]	0

ii) Modified Program



```c
int binarySearch(int v, int a[], int size) {  if (a == NULL) {
 printf("Error: Array is null.\n");  return -1; // Handle null array  }
 int lo = 0;
 int hi = size - 1;
 while (lo <= hi) {
 int mid = (lo + hi) / 2;
 if (v == a[mid]) {
 return mid; // Found
 } else if (v < a[mid]) {
 hi = mid - 1; // Search in the left half
 } else {
 lo = mid + 1; // Search in the right half
 }
 }
 return -1; // Not found
}
int main() {
 int testCase1[] = {1, 2, 3, 4, 5}; // Sorted array
 int testCase2[] = {}; // Empty array
 int testCase3[] = {1}; // Single element array  int testCase4[] = {1, 1, 1, 1, 1}; // All elements are the same
 printf("TC1: %d\n", binarySearch(3, testCase1, 5));
 printf("TC6: %d\n", binarySearch(3, NULL, 0)); // Expected: Error  message
 printf("TC7: %d\n", binarySearch(1, testCase3, 1));
 return 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).

i) Test Suite

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
3, 3, 3	**0**
3, 3, 5	**1**
3, 4, 5	**2**
1, 2, 3	**3**
0, 0, 0	**3**
-1, 2, 3	**3**
Boundary Value Analysis	
1, 1, 1	**0**
2, 2, 3	**1**
2, 2, 5	**3**

1, 2, 2	1
0, 1, 1	3

ii) Modified Program

```
#include <stdio.h>

#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3

int triangle(int a, int b, int c) {
 if (a <= 0 || b <= 0 || c <= 0) {
 return INVALID; // Handle invalid lengths
 }

 if (a >= b + c || b >= a + c || c >= a + b) {
 return INVALID; // Check for triangle inequality  }

 if (a == b && b == c) {
 return EQUILATERAL; // All sides equal
 }

 if (a == b || a == c || b == c) {
 return ISOSCELES; // Two sides equal
 }

 return SCALENE; // No sides equal
}

int main() {
 // Test Cases
 printf("TC1: %d\n", triangle(3, 3, 3)); // Expected: 0 (Equilateral)
printf("TC2: %d\n", triangle(3, 3, 5)); // Expected: 1 (Isosceles)
printf("TC3: %d\n", triangle(3, 4, 5)); // Expected: 2 (Scalene)
printf("TC4: %d\n", triangle(1, 2, 3)); // Expected: 3 (Invalid)
 printf("TC5: %d\n", triangle(0, 0, 0)); // Expected: 3 (Invalid)
printf("TC6: %d\n", triangle(-1, 2, 3)); // Expected: 3 (Invalid)
printf("TC7: %d\n", triangle(1, 1, 1)); // Expected: 0 (Equilateral)
printf("TC8: %d\n", triangle(2, 2, 3)); // Expected: 1 (Isosceles)
printf("TC9: %d\n", triangle(2, 2, 5)); // Expected: 3 (Invalid)
printf("TC10: %d\n", triangle(1, 2, 2)); // Expected: 1 (Isosceles)
printf("TC11: %d\n", triangle(0, 1, 1)); // Expected: 3 (Invalid)

 return 0;
}
```

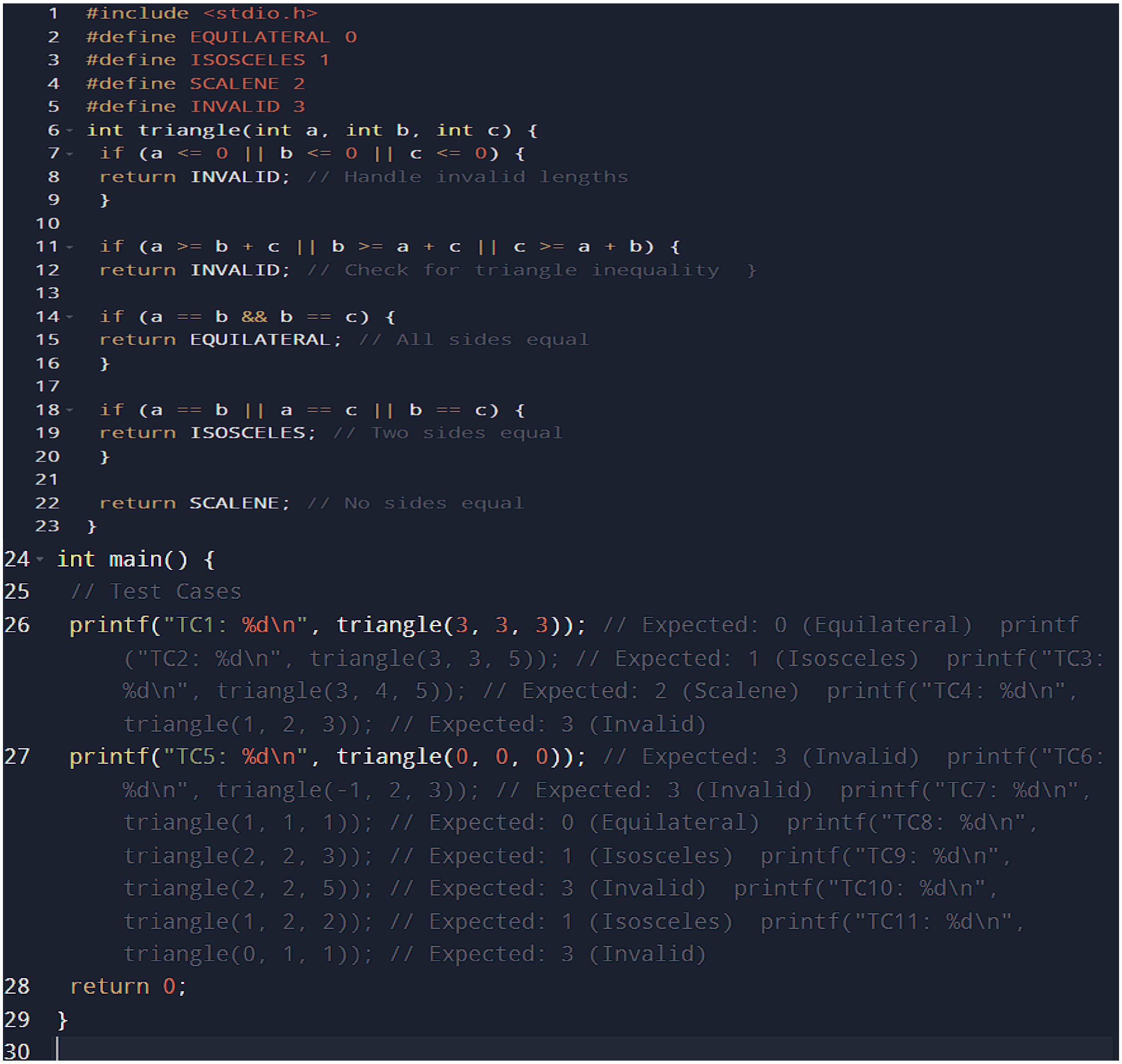
P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths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

i) Test Suite

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
3, 3, 3	**0**
3, 3, 5	**1**
3, 4, 5	**2**

1, 2, 3	3
0, 0, 0	3
-1, 2, 3	3
Boundary Value Analysis	
1, 1, 1	0
2, 2, 3	1
2, 2, 5	3
1, 2, 2	1
0, 1, 1	3

ii) Modified Program



```
#include <stdio.h>
#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3
int triangle(int a, int b, int c) {
 if (a <= 0 || b <= 0 || c <= 0) {
 return INVALID; // Handle invalid lengths
 }

 if (a >= b + c || b >= a + c || c >= a + b) {
 return INVALID; // Check for triangle inequality   }

 if (a == b && b == c) {
 return EQUILATERAL; // All sides equal
 }

 if (a == b || a == c || b == c) {
 return ISOSCELES; // Two sides equal
 }

 return SCALENE; // No sides equal
}
int main() {
 // Test Cases
 printf("TC1: %d\n", triangle(3, 3, 3)); // Expected: 0 (Equilateral)  printf("TC2: %d\n", triangle(3, 3, 5)); // Expected: 1 (Isosceles)  printf("TC3: %d\n", triangle(3, 4, 5)); // Expected: 2 (Scalene)  printf("TC4: %d\n", triangle(1, 2, 3)); // Expected: 3 (Invalid)
 printf("TC5: %d\n", triangle(0, 0, 0)); // Expected: 3 (Invalid)  printf("TC6: %d\n", triangle(-1, 2, 3)); // Expected: 3 (Invalid)  printf("TC7: %d\n", triangle(1, 1, 1)); // Expected: 0 (Equilateral)  printf("TC8: %d\n", triangle(2, 2, 3)); // Expected: 1 (Isosceles)  printf("TC9: %d\n", triangle(2, 2, 5)); // Expected: 3 (Invalid)  printf("TC10: %d\n", triangle(1, 2, 2)); // Expected: 1 (Isosceles)  printf("TC11: %d\n", triangle(0, 1, 1)); // Expected: 3 (Invalid)
 return 0;
}
```

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).

i) Test Suite

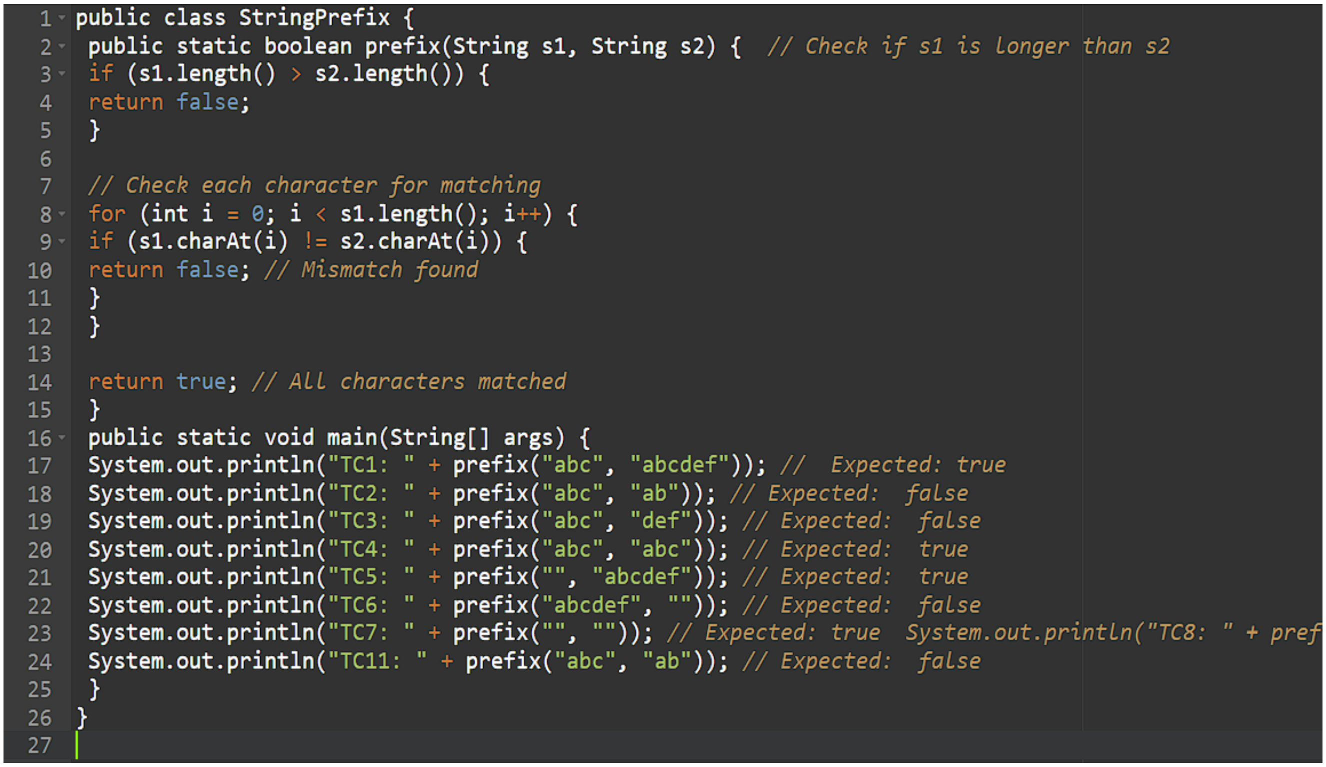
Tester Action and Input Data	Expected Outcome

Equivalence Partitioning	
"abc", "abcdef"	true
"abc", "ab"	false
"abc", "def"	false
"abc", "abc"	true
"", "abcdef"	true
"abcdef", ""	false

Boundary Value Analysis	
"", ""	true
"a", "a"	true
"a", "b"	false
"abc", "abcd"	true

"abc", "ab"	**false**

ii) Modified Program



```
public class StringPrefix {
 public static boolean prefix(String s1, String s2) {  // Check if s1 is longer than s2
 if (s1.length() > s2.length()) {
 return false;
 }

 // Check each character for matching
 for (int i = 0; i < s1.length(); i++) {
 if (s1.charAt(i) != s2.charAt(i)) {
 return false; // Mismatch found
 }
 }

 return true; // All characters matched
 }
 public static void main(String[] args) {
 System.out.println("TC1: " + prefix("abc", "abcdef")); //  Expected: true
 System.out.println("TC2: " + prefix("abc", "ab")); // Expected:  false
 System.out.println("TC3: " + prefix("abc", "def")); // Expected:  false
 System.out.println("TC4: " + prefix("abc", "abc")); // Expected:  true
 System.out.println("TC5: " + prefix("", "abcdef")); // Expected:  true
 System.out.println("TC6: " + prefix("abcdef", "")); // Expected:  false
 System.out.println("TC7: " + prefix("", "")); // Expected: true  System.out.println("TC8: " + pref
 System.out.println("TC11: " + prefix("abc", "ab")); // Expected:  false
 }
}
```

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

1. **Equivalence Class for Valid Triangles**:

   Equilateral: All sides equal (A = B = C).
   Isosceles: Two sides equal (A = B, A = C, or B = C).
   Scalene: All sides different (A ≠ B, A ≠ C, B ≠ C).
   Right-angled: Satisfies Pythagorean theorem ($A^2 + B^2 = C^2$, considering A, B, C as sides).

2. **Equivalence Class for Invalid Triangles**:

   Non-Triangle: Sides do not satisfy triangle inequality (A + B ≤ C, A + C ≤ B, B + C ≤ A).
   Non-positive Values: Any of A, B, or C is less than or equal to zero.

b) Identify Test Cases

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

TC1	Equilateral Triangle	(3.0, 3.0, 3.0)	"Equilateral"	Equilateral
TC2	Isosceles Triangle	(3.0, 3.0, 5.0)	"Isosceles"	Isosceles
TC3	Scalene Triangle	(3.0, 4.0, 5.0)	"Scalene"	Scalene
TC4	Right-Angled Triangle	(3.0, 4.0, 5.0)	"Right angled"	Right-angled
TC5	Non-Triangle	(1.0, 2.0, 3.0)	"Not a triangle"	Non-Triangle
TC6	Non-Triangle	(5.0, 2.0, 3.0)	"Not a triangle"	Non-Triangle
TC7	Non-positive Input	(0.0, 2.0, 3.0)	"Invalid"	Non-positive Values
TC8	Non-positive Input	(-1.0, 2.0, 3.0)	"Invalid"	Non-positive Values

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

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC9	Boundary scalene case	(2.0, 3.0, 4.0)	"Scalene"
TC10	Just not forming scalene case	(2.0, 2.0, 4.0)	"Not a triangle"

d) Boundary Test Cases for Isosceles Triangle (A = C)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC11	Boundary isosceles case	(3.0, 3.0, 5.0)	"Isosceles"
TC12	Just not forming isosceles case	(3.0, 2.0, 5.0)	"Scalene"

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

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC13	Boundary equilateral case	(3.0, 3.0, 3.0)	"Equilateral"

TC14	Just not forming equilateral case	(2.0, 2.0, 3.0)	"Isosceles"

f) Boundary Test Cases for Right-Angled Triangle ($A^2 + B^2 = C^2$)

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC15	Boundary right-angled case	(3.0, 4.0, 5.0)	"Right angled"
TC16	Just not forming right angled	(3.0, 4.0, 6.0)	"Scalene"

g) Boundary Test Cases for Non-Triangle

Test Case ID	Description	Input (A, B, C)	Expected Outcome
TC17	Not satisfying triangle inequality	(1.0, 1.0, 3.0)	"Not a triangle"
TC18	Not satisfying triangle inequality	(1.0, 2.0, 2.0)	"Not a triangle"

h) Test Cases for Non-Positive Input

Test Case ID	**Descriptio n**	**Input (A, B, C)**	**Expected Outcome**
TC19	Zero input	(0.0, 2.0, 3.0)	"Invalid"
TC20	Negative input	(-1.0, 2.0, 3.0)	"Invalid"