IT-304 Software Engineering

Lab session: 8

Zeel Danani (202201507)

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.
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.

Answer:

Equivalence class test cases:

- Day < 1 : Invalid
- Day > 31 : Invalid
- 1 <= Day <= 31 : Valid
- Month < 1 : Invalid
- Month > 12 : Invalid
- 1 <= Month <= 12 : Valid
- Year < 1900 : Invalid
- Year > 2015 : Invalid
- 1900 <= Year <= 2015 : Valid

Equivalence Partitioning Test Cases:

Input	Expected Outcome	Remarks
(2, 11, 1887)	Error message (Year is less than 1900)	Invalid year
(5, 7, 2016)	Previous date (4 July, 2016)	Invalid year (Greater than 2015)
(19, 8, 2020)	Error message (Year is greater than 2015)	Invalid year

(17, 0, 2013)	Error message (Month is less than 1)	Invalid month
(28, 11, 2003)	Previous date (27 November, 2003)	Valid input
(3, 23, 2001)	Error message (Month is greater than 12)	Invalid month
(0, 3, 2005)	Error message (Day is less than 1)	Invalid day
(2, 6, 2007)	Previous date (1 June, 2007)	Valid input
(35, 9, 1999)	Error message (Day is greater than 31)	Invalid day

Boundary Value Analysis Test Cases:

Input	Expected Outcome	Remarks
(1, 1, 1900)	Error message (Month, Year, and Day at min)	Boundary condition (min values)
(31, 12, 2015)	Previous date (30 December, 2015)	Boundary condition (max values)
(2, 1, 1900)	Previous date (1 January, 1900)	Valid input
(1, 1, 1900)	Error message (Month, Day, and Year at min)	Boundary condition (min values)
(31, 12, 2015)	Previous date (30 December, 2015)	Boundary condition (max values)
(1, 12, 2016)	Error message (Year greater than 2015)	Invalid year

(1, 1, 1900)	Error message (Month, Day, and Year at min)	Boundary condition (min values)
(31, 12, 2015)	Previous date (30 December, 2015)	Boundary condition (max values)
(31, 12, 2016)	Error message (Year greater than 2015)	Invalid year
(1, 1, 1901)	Previous date (31 December, 1900)	Valid input (year boundary condition)

CPP Code for the same:

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

// Check if a year is a leap year
bool isLeapYear(int year)
{
    if (year % 400 == 0)
        return true;
    if (year % 100 == 0)
        return false;
    if (year % 4 == 0)
        return true;
    return false;
}

// Get the number of days in a given month
int daysInMonth(int month, int year)
{
    if (month == 2)
    {
        return isLeapYear(year) ? 29 : 28;
    }
```

```
    if (month == 4 || month == 6 || month == 9 || month == 11)
    {
        return 30;
    }
    return 31;
}

// Function to check the previous date
void previousDate(int day, int month, int year)
{
    // Validate the year
    if (year < 1900 || year > 2015)
    {
        cout << "Error: Year is out of valid range (1900-2015)\n";
        return;
    }
    // Validate the month
    if (month < 1 || month > 12)
    {
        cout << "Error: Month is out of valid range (1-12)\n";
        return;
    }
    // Validate the day
    int maxDays = daysInMonth(month, year);
    if (day < 1 || day > maxDays)
    {
        cout << "Error: Day is out of valid range for this month\n";
        return;
    }

    // If the day is valid, compute the previous date
    if (day > 1)
    {
        day--;
    }
    else
    {
        if (month == 1)
        {
            month = 12;
```

```
            year--;
            day = 31;
        }
        else
        {
            month--;
            day = daysInMonth(month, year);
        }
    }

    // Output the previous date
    cout << "Previous Date: " << day << "/" << month << "/" << year
<< endl;
}

int main()
{
    // Test cases for Equivalence Partitioning and Boundary Value
Analysis

    cout << "Equivalence Partitioning Test Cases:\n";

    // Year Test Cases
    previousDate(2, 11, 1887); // Invalid year < 1900
    previousDate(5, 7, 2016);  // Invalid year > 2015
    previousDate(19, 8, 2020); // Invalid year > 2015

    // Month Test Cases
    previousDate(17, 0, 2013);  // Invalid month < 1
    previousDate(28, 11, 2003); // Valid previous date
    previousDate(3, 23, 2001);  // Invalid month > 12

    // Day Test Cases
    previousDate(0, 3, 2005);  // Invalid day < 1
    previousDate(2, 6, 2007);  // Valid previous date
    previousDate(35, 9, 1999); // Invalid day > 31

    cout << "\nBoundary Value Analysis Test Cases:\n";

    // Year Test Cases
```

```
    previousDate(1, 1, 1900);   // Min boundary values for year,
month, day
    previousDate(31, 12, 2015); // Max boundary values for year,
month, day
    previousDate(2, 1, 1900);   // Valid test case, boundary year

    // Month Test Cases
    previousDate(1, 1, 1900);   // Min boundary values
    previousDate(31, 12, 2015); // Max boundary values
    previousDate(1, 12, 2016);  // Invalid year > 2015

    // Day Test Cases
    previousDate(1, 1, 1900);   // Min boundary values
    previousDate(31, 12, 2015); // Max boundary values
    previousDate(31, 12, 2016); // Invalid year > 2015
    previousDate(1, 1, 1901);   // Valid previous date (boundary
condition)

    return 0;
}
```

Output:

```
Equivalence Partitioning Test Cases:
Error: Year is out of valid range (1900-2015)
Error: Year is out of valid range (1900-2015)
Error: Year is out of valid range (1900-2015)
Error: Month is out of valid range (1-12)
Previous Date: 27/11/2003
Error: Month is out of valid range (1-12)
Error: Day is out of valid range for this month
Previous Date: 1/6/2007
Error: Day is out of valid range for this month

Boundary Value Analysis Test Cases:
Previous Date: 31/12/1899
Previous Date: 30/12/2015
Previous Date: 1/1/1900
Previous Date: 31/12/1899
Previous Date: 30/12/2015
Error: Year is out of valid range (1900-2015)
Previous Date: 31/12/1899
Previous Date: 30/12/2015
Error: Year is out of valid range (1900-2015)
Previous Date: 30/12/2015
Previous Date: 1/1/1900
Previous Date: 31/12/1899
Previous Date: 30/12/2015
Error: Year is out of valid range (1900-2015)
Previous Date: 31/12/1899
Previous Date: 30/12/2015
Error: Year is out of valid range (1900-2015)
Previous Date: 31/12/1900
```

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.

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

Equivalence Classes and Test Cases:

Equivalence Class	Description
EC1	Value v exists in the array
EC2	Value v does not exist in the array
EC3	The array is empty
EC4	Value v exists in the array multiple times
EC5	Value v is at the first position in the array
EC6	Value v is at the last position in the array

Test Cases Covering Equivalence Classes:

Test Case	Input (v, a)	Expected Output	Equivalence Class Covered
TC1	(5, [3, 5, 7, 9])	1	EC1: Value v exists in the array
TC2	(10, [2, 4, 6, 8])	-1	EC2: Value v does not exist in the array
TC3	(7, [])	-1	EC3: The array is empty
TC4	(7, [7, 3, 7, 9, 7])	0	EC4: Value v exists multiple times
TC5	(5, [5, 8, 9, 12])	0	EC5: Value v is at the first position
TC6	(12, [3, 5, 7, 12])	3	EC6: Value v is at the last position

Boundary Value Test Cases

Test Case	Input (v, a)	Expected Output	Description
TC1	(3, [3])	0	Minimum input array size, searching for the only element
TC2	(5, [3])	-1	Minimum input array size, element not found
TC3	(7, [3, 5, 7])	2	Searching for the last element in a small array

TC4	(2, [3, 5, 7])	-1	Searching for a non-existing element in a small array
TC5	(5, [3, 5, 7, 9, 5])	1	Searching for the first occurrence of a repeated element
TC6	(9, [3, 5, 7, 9, 12])	3	Searching for the last element in a larger array
TC7	(1, [])	-1	Searching in an empty array
TC8	(0, [0])	0	Searching for the boundary value in an array of size 1
TC9	(10, [10, 20, 30])	0	Searching for the first element in an array
TC10	(30, [10, 20, 30])	2	Searching for the last element in a small array

Code:

```cpp
#include <iostream>

using namespace std;

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

```cpp
int main() {
    // Test cases
    int arr1[] = {3, 5, 7, 9};
    int arr2[] = {2, 4, 6, 8};
    int arr3[] = {};
    int arr4[] = {7, 3, 7, 9, 7};
    int arr5[] = {5, 8, 9, 12};
    int arr6[] = {3, 5, 7, 12};

    // Running test cases
    cout << "Test Case 1: " << linearSearch(5, arr1, 4) << " (Expected: 1)" << endl;
    cout << "Test Case 2: " << linearSearch(10, arr2, 4) << " (Expected: -1)" << endl;
    cout << "Test Case 3: " << linearSearch(7, arr3, 0) << " (Expected: -1)" << endl;
    cout << "Test Case 4: " << linearSearch(7, arr4, 5) << " (Expected: 0)" << endl;
    cout << "Test Case 5: " << linearSearch(5, arr5, 4) << " (Expected: 0)" << endl;
    cout << "Test Case 6: " << linearSearch(12, arr6, 4) << " (Expected: 3)" << endl;

    return 0;
}
```

OutPut:

```
Test Case 1: 1 (Expected: 1)
Test Case 2: -1 (Expected: -1)
Test Case 3: -1 (Expected: -1)
Test Case 4: 0 (Expected: 0)
Test Case 5: 0 (Expected: 0)
Test Case 6: 3 (Expected: 3)
```

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);

}
```

1. **Equivalence Class Identification:**

Equivalence Class	Description	Test Case (v, a)	Expected Output
Valid Equivalence Classes			
Value exists in the array	The value v is present in the array.	(3, [3, 5, 7, 3, 9, 3])	3
Value does not exist in the array	The value v is not present in the array.	(10, [3, 5, 7, 3, 9, 3])	0
Value appears multiple times	The value v appears multiple times in the array.	(5, [5, 5, 5, 5, 5])	5

Array with distinct elements	The value v is present in an array of distinct elements.	(1, [1, 2, 3, 4])	1
Empty array	The array is empty.	(0, [])	0

Invalid Equivalence Classes

Null array (not directly testable in C++)	Not applicable, using empty array as representative.	(0, [])	0

2. Boundary Value Test Cases:

Test Case	Input (v, a)	Expected Output	Description
TC1	(3, [3, 3, 3])	3	Count of 3 in a small array with all identical values
TC2	(4, [1, 2, 3, 4, 4, 4])	3	Count of 4 in an array with multiple occurrences
TC3	(5, [1, 2, 3, 4, 5, 6])	1	Count of 5 in an array with distinct elements
TC4	(7, [1, 2, 3, 4, 5, 6])	0	Count of 7 in an array with distinct elements (not present)

Code:

```cpp
#include <iostream>

using namespace std;

int countItem(int v, int a[], int length) {
    int count = 0;
    for (int i = 0; i < length; i++) {
        if (a[i] == v) {
            count++;  // Increment count if v is found
        }
    }
    return count;  // Return the total count
```

```
}

void runEquivalenceClassTests() {
    // Equivalence Class Test Cases
    int arr1[] = {3, 5, 7, 3, 9, 3};  // Test case for value exists
    int arr2[] = {3, 5, 7, 3, 9, 3};  // Test case for value does not
exist
    int arr3[] = {5, 5, 5, 5, 5};     // Test case for value appears
multiple times
    int arr4[] = {1, 2, 3, 4};        // Test case for array with distinct
elements
    int arr5[] = {};                  // Test case for empty array

    // Running Equivalence Class test cases
    cout << "Equivalence Class Test Case 1: " << countItem(3, arr1, 6) <<
" (Expected: 3)" << endl; // Value exists
    cout << "Equivalence Class Test Case 2: " << countItem(10, arr1, 6) <<
" (Expected: 0)" << endl; // Value does not exist
    cout << "Equivalence Class Test Case 3: " << countItem(5, arr3, 5) <<
" (Expected: 5)" << endl; // Value appears multiple times
    cout << "Equivalence Class Test Case 4: " << countItem(1, arr4, 4) <<
" (Expected: 1)" << endl; // Distinct elements
    cout << "Equivalence Class Test Case 5: " << countItem(0, arr5, 0) <<
" (Expected: 0)" << endl; // Empty array
}

void runBoundaryValueTests() {
    // Boundary Value Test Cases
    int arr6[] = {3, 3, 3};            // Test case with identical
elements
    int arr7[] = {1, 2, 3, 4, 4, 4};   // Test case with multiple
occurrences
    int arr8[] = {1, 2, 3, 4, 5, 6};    // Test case with distinct
elements

    // Running Boundary Value test cases
    cout << "Boundary Value Test Case 1: " << countItem(3, arr6, 3) << "
(Expected: 3)" << endl; // Identical values
    cout << "Boundary Value Test Case 2: " << countItem(4, arr7, 6) << "
(Expected: 3)" << endl; // Multiple occurrences
```

```
    cout << "Boundary Value Test Case 3: " << countItem(5, arr8, 6) << "
(Expected: 1)" << endl; // Distinct elements
    cout << "Boundary Value Test Case 4: " << countItem(7, arr8, 6) << "
(Expected: 0)" << endl; // Not present in array
}

int main() {
    cout << "Running Equivalence Class Tests:" << endl;
    runEquivalenceClassTests();

    cout << "\nRunning Boundary Value Tests:" << endl;
    runBoundaryValueTests();

    return 0;
}
```

Output:

```
Running Equivalence Class Tests:
Equivalence Class Test Case 1: 3 (Expected: 3)
Equivalence Class Test Case 2: 0 (Expected: 0)
Equivalence Class Test Case 3: 5 (Expected: 5)
Equivalence Class Test Case 4: 1 (Expected: 1)
Equivalence Class Test Case 5: 0 (Expected: 0)

Running Boundary Value Tests:
Boundary Value Test Case 1: 3 (Expected: 3)
Boundary Value Test Case 2: 3 (Expected: 3)
Boundary Value Test Case 3: 1 (Expected: 1)
Boundary Value Test Case 4: 0 (Expected: 0)
PS C:\Users\ZEEL\OneDrive\Desktop\cpp>
```

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);
}
```

1. Equivalence Class Identification

Equivalence Class	Description	Test Case (v, a)	Expected Output
Valid Equivalence Classes			
Value exists in the array	The value v is present in the ordered array.	(5, [1, 2, 3, 4, 5, 6])	4
Value does not exist in the array	The value v is not present in the ordered array.	(10, [1, 2, 3, 4, 5, 6])	-1
Value is less than the first element	The value v is less than the smallest element.	(0, [1, 2, 3, 4, 5, 6])	-1
Value is greater than the last element	The value v is greater than the largest element.	(7, [1, 2, 3, 4, 5, 6])	-1
Array with one element	The array contains only one element.	(3, [3])	0
Value equals the only element in the array	The value v equals the single element.	(3, [5])	-1

Invalid Equivalence Classes			
Null array (not directly testable in C++)	Not applicable, using an empty array as representative.	(5, [])	-1

2. Boundary Value Test Cases

Test Case	Input (v, a)	Expected Output	Description
TC1	(5, [1, 2, 3, 4, 5, 6])	4	Value exists at index 4
TC2	(10, [1, 2, 3, 4, 5, 6])	-1	Value does not exist in the array
TC3	(0, [1, 2, 3, 4, 5, 6])	-1	Value is less than the smallest element
TC4	(7, [1, 2, 3, 4, 5, 6])	-1	Value is greater than the largest element
TC5	(3, [3])	0	Value exists in an array with one element
TC6	(5, [])	-1	Value does not exist in an empty array

Code:

```
#include <iostream>

using namespace std;

int binarySearch(int v, int a[], int length) {
    int lo = 0, hi = length - 1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (v == a[mid])
            return mid;  // Value found at index mid
        else if (v < a[mid])
            hi = mid - 1;  // Search in the left half
        else
```

```
            lo = mid + 1;  // Search in the right half
    }
    return -1;  // Value not found
}

void runEquivalenceClassTests() {
    // Equivalence Class Test Cases
    int arr1[] = {1, 2, 3, 4, 5, 6};  // Ordered array
    int arr2[] = {3};                  // Array with one element
    int arr3[] = {};                   // Empty array

    // Running Equivalence Class test cases
    cout << "Equivalence Class Test Case 1: " << binarySearch(5, arr1, 6)
<< " (Expected: 4)" << endl; // Value exists
    cout << "Equivalence Class Test Case 2: " << binarySearch(10, arr1, 6)
<< " (Expected: -1)" << endl; // Value does not exist
    cout << "Equivalence Class Test Case 3: " << binarySearch(0, arr1, 6)
<< " (Expected: -1)" << endl; // Less than first element
    cout << "Equivalence Class Test Case 4: " << binarySearch(7, arr1, 6)
<< " (Expected: -1)" << endl; // Greater than last element
    cout << "Equivalence Class Test Case 5: " << binarySearch(3, arr2, 1)
<< " (Expected: 0)" << endl; // Value equals the only element
    cout << "Equivalence Class Test Case 6: " << binarySearch(5, arr3, 0)
<< " (Expected: -1)" << endl; // Empty array
}

void runBoundaryValueTests() {
    // Boundary Value Test Cases
    int arr1[] = {1, 2, 3, 4, 5, 6};  // Ordered array

    // Running Boundary Value test cases
    cout << "Boundary Value Test Case 1: " << binarySearch(5, arr1, 6) <<
" (Expected: 4)" << endl; // Value exists
    cout << "Boundary Value Test Case 2: " << binarySearch(10, arr1, 6) <<
" (Expected: -1)" << endl; // Value does not exist
    cout << "Boundary Value Test Case 3: " << binarySearch(0, arr1, 6) <<
" (Expected: -1)" << endl; // Less than first element
    cout << "Boundary Value Test Case 4: " << binarySearch(7, arr1, 6) <<
" (Expected: -1)" << endl; // Greater than last element
}
```

```cpp
int main() {
    cout << "Running Equivalence Class Tests:" << endl;
    runEquivalenceClassTests();

    cout << "\nRunning Boundary Value Tests:" << endl;
    runBoundaryValueTests();

    return 0;
}
```

Output:

```
Running Equivalence Class Tests:
Equivalence Class Test Case 1: 4 (Expected: 4)
Equivalence Class Test Case 2: -1 (Expected: -1)
Equivalence Class Test Case 3: -1 (Expected: -1)
Equivalence Class Test Case 4: -1 (Expected: -1)
Equivalence Class Test Case 5: 0 (Expected: 0)
Equivalence Class Test Case 6: -1 (Expected: -1)

Running Boundary Value Tests:
Boundary Value Test Case 1: 4 (Expected: 4)
Boundary Value Test Case 2: -1 (Expected: -1)
Boundary Value Test Case 3: -1 (Expected: -1)
Boundary Value Test Case 4: -1 (Expected: -1)
PS C:\Users\ZEEL\OneDrive\Desktop\cpp>
```

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 lengthsequal), scalene (no lengths equal), or invalid (impossible lengths).

```cpp
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 Identification

Equivalence Class	Description	Test Case (a, b, c)	Expected Output
Valid Equivalence Classes			
Equilateral triangle	All sides are equal.	(3, 3, 3)	0
Isosceles triangle	Two sides are equal, one is different.	(3, 3, 4)	1
Scalene triangle	All sides are different.	(3, 4, 5)	2
Invalid Equivalence Classes			
Invalid triangle	One or more sides cannot form a triangle.	(1, 2, 3)	3
Invalid lengths (negative values)	At least one side length is negative.	(-1, 2, 3)	3
Zero-length side	At least one side length is zero.	(0, 2, 3)	3

2. Boundary Value Test Cases

Test Case	Input (a, b, c)	Expected Output	Description
TC1	(3, 3, 3)	0	Equilateral triangle
TC2	(3, 3, 4)	1	Isosceles triangle

TC3	(3, 4, 5)	2	Scalene triangle
TC4	(1, 2, 3)	3	Invalid triangle (triangle inequality)
TC5	(-1, 2, 3)	3	Invalid triangle (negative side length)
TC6	(0, 2, 3)	3	Invalid triangle (zero-length side)

code:

```
#include <iostream>

using namespace std;

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 <= 0 || b <= 0 || c <= 0 || a >= b + c || b >= a + c || c >= a +
b)
        return INVALID;  // Invalid triangle condition
    if (a == b && b == c)
        return EQUILATERAL;  // Equilateral triangle
    if (a == b || a == c || b == c)
        return ISOSCELES;    // Isosceles triangle
    return SCALENE;        // Scalene triangle
}

void runEquivalenceClassTests() {
    // Equivalence Class Test Cases
    cout << "Equivalence Class Test Case 1: " << triangle(3, 3, 3) << "
(Expected: 0)" << endl; // Equilateral
    cout << "Equivalence Class Test Case 2: " << triangle(3, 3, 4) << "
(Expected: 1)" << endl; // Isosceles
    cout << "Equivalence Class Test Case 3: " << triangle(3, 4, 5) << "
(Expected: 2)" << endl; // Scalene
```

```
    cout << "Equivalence Class Test Case 4: " << triangle(1, 2, 3) << "
(Expected: 3)" << endl; // Invalid
    cout << "Equivalence Class Test Case 5: " << triangle(-1, 2, 3) << "
(Expected: 3)" << endl; // Invalid (negative length)
    cout << "Equivalence Class Test Case 6: " << triangle(0, 2, 3) << "
(Expected: 3)" << endl; // Invalid (zero length)
}

void runBoundaryValueTests() {
    // Boundary Value Test Cases
    cout << "Boundary Value Test Case 1: " << triangle(3, 3, 3) << "
(Expected: 0)" << endl; // Equilateral
    cout << "Boundary Value Test Case 2: " << triangle(3, 3, 4) << "
(Expected: 1)" << endl; // Isosceles
    cout << "Boundary Value Test Case 3: " << triangle(3, 4, 5) << "
(Expected: 2)" << endl; // Scalene
    cout << "Boundary Value Test Case 4: " << triangle(1, 2, 3) << "
(Expected: 3)" << endl; // Invalid
    cout << "Boundary Value Test Case 5: " << triangle(-1, 2, 3) << "
(Expected: 3)" << endl; // Invalid (negative length)
    cout << "Boundary Value Test Case 6: " << triangle(0, 2, 3) << "
(Expected: 3)" << endl; // Invalid (zero length)
}

int main() {
    cout << "Running Equivalence Class Tests:" << endl;
    runEquivalenceClassTests();

    cout << "\nRunning Boundary Value Tests:" << endl;
    runBoundaryValueTests();

    return 0;
}
```

Output:

```
Running Equivalence Class Tests:
Equivalence Class Test Case 1: 0 (Expected: 0)
Equivalence Class Test Case 2: 1 (Expected: 1)
Equivalence Class Test Case 3: 2 (Expected: 2)
Equivalence Class Test Case 4: 3 (Expected: 3)
Equivalence Class Test Case 5: 3 (Expected: 3)
Equivalence Class Test Case 6: 3 (Expected: 3)

Running Boundary Value Tests:
Boundary Value Test Case 1: 0 (Expected: 0)
Boundary Value Test Case 2: 1 (Expected: 1)
Boundary Value Test Case 3: 2 (Expected: 2)
Boundary Value Test Case 4: 3 (Expected: 3)
Boundary Value Test Case 5: 3 (Expected: 3)
Boundary Value Test Case 6: 3 (Expected: 3)
PS C:\Users\ZEEL\OneDrive\Desktop\cpp>
```

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;
}
```

1. Equivalence Class Identification:

Equivalence Class	Description	Test Case (s1, s2)	Expected Output
Valid Equivalence Classes			
s1 is a prefix of s2	The entire string s1 matches the beginning of s2.	("abc", "abcde")	true
s1 is equal to s2	The strings are identical.	("abc", "abc")	true
s1 is not a prefix of s2	The beginning of s2 does not match s1.	("abc", "defabc")	false
s1 is longer than s2	s1 cannot be a prefix if it is longer than s2.	("abcd", "abc")	false
Invalid Equivalence Classes			
s1 is an empty string	An empty string is a prefix of any non-empty string.	("", "abc")	true
s2 is an empty string	A non-empty string cannot be a prefix of an empty string.	("abc", "")	false
Both s1 and s2 are empty	Both are empty strings; they are equal.	("", "")	true

2. Boundary Value Test Cases

Test Case	Input (s1, s2)	Expected Output	Description
TC1	("abc", "abcde")	true	s1 is a prefix of s2
TC2	("abc", "abc")	true	s1 is equal to s2

TC3	("abc", "defabc")	false	s1 is not a prefix of s2
TC4	("abcd", "abc")	false	s1 is longer than s2
TC5	("", "abc")	true	Empty s1 is a prefix of non-empty s2
TC6	("abc", "")	false	Non-empty s1 is not a prefix of empty s2
TC7	("", "")	true	Both s1 and s2 are empty strings; they are equal.

Code:

```
#include <iostream>
#include <string>

using namespace std;

bool prefix(const string& s1, const string& s2) {
    if (s1.length() > s2.length()) {
        return false;  // s1 cannot be a prefix if it's longer than s2
    }
    for (size_t i = 0; i < s1.length(); i++) {
        if (s1[i] != s2[i]) {
            return false;  // characters do not match
        }
    }
    return true;  // s1 is a prefix of s2
}

void runEquivalenceClassTests() {
    // Equivalence Class Test Cases
    cout << "Equivalence Class Test Case 1: " << prefix("abc", "abcde") <<
" (Expected: 1)" << endl; // Prefix
    cout << "Equivalence Class Test Case 2: " << prefix("abc", "abc") << "
(Expected: 1)" << endl; // Equal
    cout << "Equivalence Class Test Case 3: " << prefix("abc", "defabc")
<< " (Expected: 0)" << endl; // Not a prefix
    cout << "Equivalence Class Test Case 4: " << prefix("abcd", "abc") <<
" (Expected: 0)" << endl; // s1 longer than s2
    cout << "Equivalence Class Test Case 5: " << prefix("", "abc") << "
(Expected: 1)" << endl; // Empty s1
```

```
    cout << "Equivalence Class Test Case 6: " << prefix("abc", "") << "
(Expected: 0)" << endl; // Empty s2
    cout << "Equivalence Class Test Case 7: " << prefix("", "") << "
(Expected: 1)" << endl; // Both empty
}

void runBoundaryValueTests() {
    // Boundary Value Test Cases
    cout << "Boundary Value Test Case 1: " << prefix("abc", "abcde") << "
(Expected: 1)" << endl; // Prefix
    cout << "Boundary Value Test Case 2: " << prefix("abc", "abc") << "
(Expected: 1)" << endl; // Equal
    cout << "Boundary Value Test Case 3: " << prefix("abc", "defabc") << "
(Expected: 0)" << endl; // Not a prefix
    cout << "Boundary Value Test Case 4: " << prefix("abcd", "abc") << "
(Expected: 0)" << endl; // s1 longer than s2
    cout << "Boundary Value Test Case 5: " << prefix("", "abc") << "
(Expected: 1)" << endl; // Empty s1
    cout << "Boundary Value Test Case 6: " << prefix("abc", "") << "
(Expected: 0)" << endl; // Empty s2
    cout << "Boundary Value Test Case 7: " << prefix("", "") << "
(Expected: 1)" << endl; // Both empty
}

int main() {
    cout << "Running Equivalence Class Tests:" << endl;
    runEquivalenceClassTests();

    cout << "\nRunning Boundary Value Tests:" << endl;
    runBoundaryValueTests();

    return 0;
}
```

Output:

```
Running Equivalence Class Tests:
Equivalence Class Test Case 1: 1 (Expected: 1)
Equivalence Class Test Case 2: 1 (Expected: 1)
Equivalence Class Test Case 3: 0 (Expected: 0)
Equivalence Class Test Case 4: 0 (Expected: 0)
Equivalence Class Test Case 5: 1 (Expected: 1)
Equivalence Class Test Case 6: 0 (Expected: 0)
Equivalence Class Test Case 7: 1 (Expected: 1)

Running Boundary Value Tests:
Boundary Value Test Case 1: 1 (Expected: 1)
Boundary Value Test Case 2: 1 (Expected: 1)
Boundary Value Test Case 3: 0 (Expected: 0)
Boundary Value Test Case 4: 0 (Expected: 0)
Boundary Value Test Case 5: 1 (Expected: 1)
Boundary Value Test Case 6: 0 (Expected: 0)
Boundary Value Test Case 7: 1 (Expected: 1)
PS C:\Users\ZEEL\OneDrive\Desktop\cpp>
```

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
Valid Classes	
Valid scalene triangle	All sides are different, and the triangle inequality holds.
Valid isosceles triangle	Exactly two sides are equal, and the triangle inequality holds.

Valid equilateral triangle	All sides are equal, and the triangle inequality holds.
Valid right-angled triangle	Follows the Pythagorean theorem; one angle is 90 degrees.
Invalid Classes	
Invalid triangle (non-positive sides)	At least one side is less than or equal to zero.
Invalid triangle (fails triangle inequality)	The sum of the lengths of any two sides is less than or equal to the third side.

b) Identify Test Cases to Cover the Identified Equivalence Classes

Test Case	Input (A, B, C)	Expected Output	Equivalence Class Covered
TC1	(3.0, 4.0, 5.0)	"Scalene"	Valid scalene triangle
TC2	(5.0, 5.0, 3.0)	"Isosceles"	Valid isosceles triangle
TC3	(6.0, 6.0, 6.0)	"Equilateral"	Valid equilateral triangle
TC4	(3.0, 4.0, 5.0)	"Right-angled"	Valid right-angled triangle ($3^2 + 4^2 = 5^2$)
TC5	(1.0, 2.0, 3.0)	"Not a triangle"	Invalid triangle (fails triangle inequality)
TC6	(-1.0, 2.0, 2.0)	"Not a triangle"	Invalid triangle (non-positive sides)
TC7	(0.0, 1.0, 1.0)	"Not a triangle"	Invalid triangle (non-positive sides)
TC8	(1.0, 2.0, 2.0)	"Isosceles"	Valid isosceles triangle (two sides are equal)

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

Test Case	Input (A, B, C)	Expected Output	Description
TC9	(2.0, 3.0, 4.0)	"Scalene"	A + B > C (valid scalene triangle)
TC10	(1.0, 1.0, 1.9)	"Scalene"	A + B > C (valid scalene triangle)

d) Boundary Condition for A = C (Isosceles Triangle)

Test Case	Input (A, B, C)	Expected Output	Description
TC11	(2.0, 3.0, 2.0)	"Isosceles"	A = C (valid isosceles triangle)
TC12	(3.0, 3.0, 2.0)	"Isosceles"	A = B (valid isosceles triangle)

e) Boundary Condition for A = B = C (Equilateral Triangle)

Test Case	Input (A, B, C)	Expected Output	Description
TC13	(3.0, 3.0, 3.0)	"Equilateral"	A = B = C (valid equilateral triangle)
TC14	(0.1, 0.1, 0.1)	"Equilateral"	A = B = C with small positive values

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

Test Case	Input (A, B, C)	Expected Output	Description
TC15	(3.0, 4.0, 5.0)	"Right-angled"	$A^2 + B^2 = C^2$ ($3^2 + 4^2 = 5^2$)
TC16	(5.0, 12.0, 13.0)	"Right-angled"	$A^2 + B^2 = C^2$ ($5^2 + 12^2 = 13^2$)

g) Test Cases for Non-Triangle Case

Test Case	Input (A, B, C)	Expected Output	Description
TC17	(1.0, 2.0, 3.0)	"Not a triangle"	Fails triangle inequality: A + B ≤ C
TC18	(1.0, 0.5, 1.0)	"Not a triangle"	Fails triangle inequality: A + B ≤ C
TC19	(1.0, -1.0, 1.0)	"Not a triangle"	Non-positive side length

h) Non-Positive Input Test Cases

Test Case	Input (A, B, C)	Expected Output	Description
TC20	(0.0, 1.0, 1.0)	"Not a triangle"	Non-positive side length
TC21	(-1.0, 2.0, 2.0)	"Not a triangle"	Non-positive side length
TC22	(1.0, 1.0, 0.0)	"Not a triangle"	Non-positive side length

Code:

```
#include <iostream>
#include <cmath> // For sqrt function

using namespace std;

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

int classifyTriangle(float a, float b, float c) {
    // Check for non-positive lengths
    if (a <= 0 || b <= 0 || c <= 0) {
        return INVALID; // Non-positive side lengths
    }

    // Check for triangle inequality
    if (a + b <= c || a + c <= b || b + c <= a) {
        return INVALID; // Not a triangle
    }

    // Check for equilateral
    if (a == b && b == c) {
        return EQUILATERAL;
    }

    // Check for isosceles
    if (a == b || a == c || b == c) {
```

```
        return ISOSCELES;
    }

    // Check for right-angled triangle using Pythagorean theorem
    float a2 = a * a;
    float b2 = b * b;
    float c2 = c * c;

    if (fabs(a2 + b2 - c2) < 1e-6 || fabs(a2 + c2 - b2) < 1e-6 || fabs(b2
+ c2 - a2) < 1e-6) {
        return RIGHT_ANGLED;
    }

    return SCALENE; // All sides different
}

void runTestCases() {
    // Test Cases
    cout << "Test Case 1 (3.0, 4.0, 5.0): " << classifyTriangle(3.0, 4.0,
5.0) << " (Expected: 2 - Scalene)" << endl; // Scalene
    cout << "Test Case 2 (5.0, 5.0, 3.0): " << classifyTriangle(5.0, 5.0,
3.0) << " (Expected: 1 - Isosceles)" << endl; // Isosceles
    cout << "Test Case 3 (6.0, 6.0, 6.0): " << classifyTriangle(6.0, 6.0,
6.0) << " (Expected: 0 - Equilateral)" << endl; // Equilateral
    cout << "Test Case 4 (3.0, 4.0, 5.0): " << classifyTriangle(3.0, 4.0,
5.0) << " (Expected: 3 - Right-angled)" << endl; // Right-angled
    cout << "Test Case 5 (1.0, 2.0, 3.0): " << classifyTriangle(1.0, 2.0,
3.0) << " (Expected: 4 - Invalid)" << endl; // Invalid
    cout << "Test Case 6 (-1.0, 2.0, 2.0): " << classifyTriangle(-1.0,
2.0, 2.0) << " (Expected: 4 - Invalid)" << endl; // Invalid
    cout << "Test Case 7 (0.0, 1.0, 1.0): " << classifyTriangle(0.0, 1.0,
1.0) << " (Expected: 4 - Invalid)" << endl; // Invalid
    cout << "Test Case 8 (1.0, 2.0, 2.0): " << classifyTriangle(1.0, 2.0,
2.0) << " (Expected: 1 - Isosceles)" << endl; // Isosceles
}

int main() {
    runTestCases();
    return 0;
}
```

Output:

```
Test Case 1 (3.0, 4.0, 5.0): 3 (Expected: 2 - Scalene)
Test Case 2 (5.0, 5.0, 3.0): 1 (Expected: 1 - Isosceles)
Test Case 3 (6.0, 6.0, 6.0): 0 (Expected: 0 - Equilateral)
Test Case 4 (3.0, 4.0, 5.0): 3 (Expected: 3 - Right-angled)
Test Case 5 (1.0, 2.0, 3.0): 4 (Expected: 4 - Invalid)
Test Case 6 (-1.0, 2.0, 2.0): 4 (Expected: 4 - Invalid)
Test Case 7 (0.0, 1.0, 1.0): 4 (Expected: 4 - Invalid)
Test Case 8 (1.0, 2.0, 2.0): 1 (Expected: 1 - Isosceles)
PS C:\Users\ZEEL\OneDrive\Desktop\cpp>
```