IT314: Lab 8

Functional Testing (BlackBox)

Name-Akhil Rachhadia

Student Id-202201005

Question 1

Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
15, 6, 2000	Yes (14/6/2000)
1, 7, 2000	Yes (30/6/2000)
31, 12, 2000	Yes (30/12/2000)
0, 6, 2000	Error
32, 6, 2000	Error
15, 0, 2000	Error
29, 2, 2000	Yes (28/2/2000)
31, 3, 2000	Yes (30/3/2000)
30, 4, 2000	Yes (29/4/2000)

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
1, 6, 2000	Yes (31/5/2000)
31, 12, 2000	Yes (30/12/2000)
0, 6, 2000	Error
15, 12, 2000	Yes (14/12/2000)
15, 0, 2000	Error
15, 13, 2000	Error
15, 6, 1900	Yes (14/6/1900)
28, 2, 2000	Yes (27/2/2000)

Modified Program

```
#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) { 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;



    // 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));
    }
  }


voidrunBoundaryValueTests() { 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, "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;
}
```

Outputs :

```
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
--------------------------------------------------
```

```
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: An Error message
Actual: An Error message
Status: PASS
Output: Invalid date
```

Question 2

P1. Linear Search Test Cases Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
v=5, a=[5,2,8,1,9]	0
v=8, a=[5,2,8,1,9]	2
v=3, a=[5,2,8,1,9]	-1
v=5, a=[]	-1

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
v=5, a=[5]	0
v=5, a=[1,5]	1
v=5, a=[5,5,5]	0
v=1, a=[1]	0
v=1, a=[]	-1

Modified Program

```
#include <iostream>
 #include <string>
 #include <vector>
#include <tuple>
using namespace std;
class LinearSearch {

public:

int search(int v, int a[], int length) { int i = 0;
while (i < length) { if (a[i] == v) return i; i++;
}

return -1; }
};

class TestRunner { private:
LinearSearch searcher;

void runTestCase(int value, int arr[], int length, int
expectedOutcome, string testType, string description) { int
result = searcher.search(value, arr, length);
```

```
string status = (result == expectedOutcome) ? "PASS" : "FAIL";
cout << testType << ": " << description << endl;
cout << "Input: value=" << value << ", array=["; for(int i = 0; i < length; i++) { cout << arr[i];

if(i < length-1) cout << ",";

}

cout << "]" << endl;

cout << "Expected: " << expectedOutcome << endl; cout << "Actual: " << result << endl; cout << "Status: " << status << endl;
cout << string(50, '-') << endl;
}

public:

void runEquivalencePartitioningTests() {

cout << "\nEQUIVALENCE PARTITIONING TEST CASES" << endl;

cout << string(50, '=') << endl; int test1[] = {5,2,8,1,9}; int test2[] = {1};

int test3[] = {1,1,1,1,1};
```

runTestCase(5, test1, 5, 0, "EP", "Value at start"); runTestCase(8, test1, 5, 2, "EP", "Value in middle"); runTestCase(9, test1, 5, 4, "EP", "Value at end"); runTestCase(3, test1, 5, -1, "EP", "Value not present"); runTestCase(1, test2, 1, 0, "EP", "Single element array - value present"); runTestCase(2, test2, 1, -1, "EP", "Single element array - value not present"); runTestCase(1, test3, 5, 0, "EP", "All elements same - value present");

}

void runBoundaryValueTests() {

cout << "\nBOUNDARY VALUE ANALYSIS TEST CASES" << endl;

cout << string(50, '=') << endl; int test1[] = {5};

int test2[] = {5,5};

int test3[] = {1,2,3,4,5};

runTestCase(5, test1, 1, 0, "BVA", "Single element - found"); runTestCase(1, test1, 1, -1, "BVA", "Single element - not found"); runTestCase(5, test2, 2, 0, "BVA", "Two identical elements"); runTestCase(1, test3, 5, 0, "BVA", "First element"); runTestCase(5, test3, 5, 4, "BVA", "Last element");

}

};

int main() {
TestRunner runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests(); *return*
0;

} Output :

```
EQUIVALENCE PARTITIONING TEST CASES
====================================================
EP: Value at start
Input: value=5, array=[5,2,8,1,9]
Expected: 0
Actual: 0
Status: PASS
----------------------------------------------------
EP: Value in middle
Input: value=8, array=[5,2,8,1,9]
Expected: 2
Actual: 2
Status: PASS
----------------------------------------------------
EP: Value at end
Input: value=9, array=[5,2,8,1,9]
Expected: 4
Actual: 4
Status: PASS
----------------------------------------------------
EP: Value not present
Input: value=3, array=[5,2,8,1,9]
Expected: -1
Actual: -1
Status: PASS
```

```
BOUNDARY VALUE ANALYSIS TEST CASES
================================================
BVA: Single element - found
Input: value=5, array=[5]
Expected: 0
Actual: 0
Status: PASS
------------------------------------------------
BVA: Single element - not found
Input: value=1, array=[5]
Expected: -1
Actual: -1
Status: PASS
------------------------------------------------
BVA: Two identical elements
Input: value=5, array=[5,5]
Expected: 0
Actual: 0
Status: PASS
```

P2. Count Item Test Cases

Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
v=5, a=[5,2,5,1,5]	3
v=2, a=[5,2,5,1,5]	1
v=3, a=[5,2,5,1,5]	0
v=5, a=[]	0

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
v=5, a=[5]	1
v=5, a=[5,5]	2
v=5, a=[5,5,5]	3

v=1, a=[1]	1
v=1, a=[]	0

Modified Program

```
#include <iostream>
 #include <string>
 #include <vector>
#include <tuple>  using
namespace std;  class
CountItem {  public:
int count(int v, int a[], int length) { int count = 0; for (int i
= 0; i < length; i++) { if (a[i] == v) count++;

}

return count;

}

};

class TestRunner { private:
CountItem counter;
```

```
void runTestCase(int value, int arr[], int length, int
expectedOutcome, string testType, string description) { int result
= counter.count(value, arr, length);


string status = (result == expectedOutcome) ? "PASS" : "FAIL";
cout << testType << ": " << description << endl; cout << "Input:
value=" << value << ", array=["; for(int i = 0; i < length; i++) { cout
<< arr[i];


if(i < length-1) cout << ",";


}


cout << "]" << endl;


cout << "Expected: " << expectedOutcome << endl; cout
<< "Actual: " << result << endl; cout << "Status: " << status <<
endl; cout << string(50, '-') << endl;
}


public:


void runEquivalencePartitioningTests() {


cout << "\nEQUIVALENCE PARTITIONING TEST CASES"
<< endl;
```

```
cout << string(50, '=') << endl; int test1[] = {5,2,5,1,5}; int test2[]
= {1};

int test3[] = {1,1,1,1,1};

runTestCase(5, test1, 5, 3, "EP", "Multiple occurrences");
runTestCase(2, test1, 5, 1, "EP", "Single occurrence");
runTestCase(3, test1, 5, 0, "EP", "No occurrence"); runTestCase(1,
test2, 1, 1, "EP", "Single element array - value present");
runTestCase(2, test2, 1, 0, "EP", "Single element array - value not
present"); runTestCase(1, test3, 5, 5, "EP", "All elements same -
value present");
}

void runBoundaryValueTests() {

cout << "\nBOUNDARY VALUE ANALYSIS TEST CASES"
<< endl;

cout << string(50, '=') << endl; int test1[] = {5};
int test2[] = {5,5};

int test3[] = {1,2,3,4,5};

runTestCase(5, test1, 1, 1, "BVA", "Single element - found");
runTestCase(1, test1, 1, 0, "BVA", "Single element - not found");
runTestCase(5, test2, 2, 2, "BVA", "Two identical elements");
```

```
runTestCase(1, test3, 5, 1, "BVA", "Single occurrence at start");
runTestCase(5, test3, 5, 1,
"BVA", "Single occurrence at end");


}


};


int main() {  TestRunner
runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests(); return 0;
 }
```

```
EQUIVALENCE PARTITIONING TEST CASES
================================================
EP: Multiple occurrences
Input: value=5, array=[5,2,5,1,5]
Expected: 3
Actual: 3
Status: PASS
------------------------------------------------
EP: Single occurrence
Input: value=2, array=[5,2,5,1,5]
Expected: 1
Actual: 1
Status: PASS
------------------------------------------------
EP: No occurrence
Input: value=3, array=[5,2,5,1,5]
Expected: 0
Actual: 0
Status: PASS
```

```
BOUNDARY VALUE ANALYSIS TEST CASES
==================================================
BVA: Single element - found
Input: value=5, array=[5]
Expected: 1
Actual: 1
Status: PASS
--------------------------------------------------
BVA: Single element - not found
Input: value=1, array=[5]
Expected: 0
Actual: 0
Status: PASS
--------------------------------------------------
BVA: Two identical elements
Input: value=5, array=[5,5]
Expected: 2
Actual: 2
Status: PASS
```

P3. Binary Search Test Cases Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
v=5, a=[1,3,5,7,9]	2
v=1, a=[1,3,5,7,9]	0
v=9, a=[1,3,5,7,9]	4
v=4, a=[1,3,5,7,9]	-1
v=5, a=[]	-1

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
v=1, a=[1]	0
v=5, a=[5]	0
v=1, a=[1,3]	0
v=3, a=[1,3]	1
v=0, a=[1]	-1

Modified Program :

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

class BinarySearch { public:
int search(int v, int a[], int length)
{ int lo = 0; int hi = length - 1;
while (lo <= hi) { int mid = (lo +
hi) / 2;
if (v == a[mid]) return mid; else if (v < a[mid]) hi = mid - 1; else lo =
mid + 1;
}
return -1;
}

};

class TestRunner { private:
BinarySearch searcher;

void runTestCase(int value, int arr[], int length, int
expectedOutcome, string testType, string description) {
int result = searcher.search(value, arr, length);
string status = (result == expectedOutcome) ? "PASS" : "FAIL";

cout << testType << ": " << description << endl; cout << "Input:
value=" << value << ", array=["; for(int i = 0; i < length; i++) {
cout << arr[i];
if(i < length-1) cout << ",";
}
cout << "]" << endl;
```

```
cout << "Expected: " << expectedOutcome << endl; cout <<
"Actual: " << result << endl; cout << "Status: " << status << endl;
cout << string(50, '-') << endl;
}

public:
void runEquivalencePartitioningTests() {
cout << "\nEQUIVALENCE PARTITIONING TEST
CASES" << endl; cout << string(50, '=') << endl;

int test1[] = {1,3,5,7,9}; int
test2[] = {1}; int test3[] =
{1,1,1,1,1};

runTestCase(5, test1, 5, 2, "EP", "Value in middle");
runTestCase(1, test1, 5, 0, "EP", "Value at start"); runTestCase(9,
test1, 5, 4, "EP", "Value at end"); runTestCase(4, test1, 5, -1, "EP",
"Value not present"); runTestCase(1, test2, 1, 0, "EP", "Single
element array - found");
runTestCase(2, test2, 1, -1, "EP", "Single element array - not
found");
}

void runBoundaryValueTests() {
cout << "\nBOUNDARY VALUE ANALYSIS TEST
CASES" << endl; cout << string(50, '=') << endl;

int test1[] =
{1}; int test2[]
= {1,2};
int test3[] = {1,2,3,4,5};
```

```
runTestCase(1, test1, 1, 0, "BVA", "Single element - found");
runTestCase(0, test1, 1, -1, "BVA", "Single element - not found");
runTestCase(1, test2, 2, 0, "BVA", "Two elements - found first");
runTestCase(2, test2, 2, 1, "BVA", "Two elements - found second");
runTestCase(1, test3, 5, 0, "BVA", "Found at first position");
runTestCase(5, test3, 5, 4, "BVA", "Found at last position");
}
};

int main() { TestRunner runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests(); return 0;
}
```

```
EQUIVALENCE PARTITIONING TEST CASES
==========================================
EP: Value in middle
Input: value=5, array=[1,3,5,7,9]
Expected: 2
Actual: 2
Status: PASS
------------------------------------------
EP: Value at start
Input: value=1, array=[1,3,5,7,9]
Expected: 0
Actual: 0
Status: PASS
------------------------------------------
EP: Value at end
Input: value=9, array=[1,3,5,7,9]
Expected: 4
Actual: 4
Status: PASS
------------------------------------------
EP: Value not present
Input: value=4, array=[1,3,5,7,9]
Expected: -1
Actual: -1
```

```
BOUNDARY VALUE ANALYSIS TEST CASES
==================================================
BVA: Single element - found
Input: value=1, array=[1]
Expected: 0
Actual: 0
Status: PASS
--------------------------------------------------
BVA: Single element - not found
Input: value=0, array=[1]
Expected: -1
Actual: -1
Status: PASS
--------------------------------------------------
BVA: Two elements - found first
Input: value=1, array=[1,2]
Expected: 0
Actual: 0
Status: PASS
--------------------------------------------------
BVA: Two elements - found second
Input: value=2, array=[1,2]
Expected: 1
Actual: 1
Status: PASS
```

P4. Triangle Test Cases

Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
a=3, b=3, c=3	EQUILATERAL
a=3, b=3, c=4	ISOSCELES
a=3, b=4, c=5	SCALENE
a=1, b=1, c=3	INVALID
a=0, b=0, c=0	INVALID

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
a=1, b=1, c=1	EQUILATERAL
a=2, b=2, c=3	ISOSCELES
a=3, b=4, c=5	SCALENE
a=1, b=2, c=3	INVALID
a=0, b=1, c=1	INVALID

Modified Program

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

class Triangle { public:
static const int EQUILATERAL = 0; static const int ISOSCELES =
1; static const int SCALENE = 2; static const int INVALID = 3;

int classify(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;
}
};

class TestRunner { private:
Triangle triangle;

void runTestCase(int a, int b, int c, int expectedOutcome, string
testType, string description) {
int result = triangle.classify(a, b, c);
string status = (result == expectedOutcome) ? "PASS" : "FAIL";

cout << testType << ": " << description << endl;
cout << "Input: a=" << a << ", b=" << b << ", c=" << c << endl; cout
<< "Expected: " << expectedOutcome << endl;
cout << "Actual: " << result << endl; cout << "Status: " << status <<
endl; cout << string(50, '-') << endl;
}
```

```
public:
void runEquivalencePartitioningTests() {
cout << "\nEQUIVALENCE PARTITIONING TEST CASES" <<
endl;
cout << string(50, '=') << endl;

runTestCase(5, 5, 5, Triangle::EQUILATERAL, "EP", "Equilateral
triangle"); runTestCase(5, 5, 3, Triangle::ISOSCELES, "EP",
"Isosceles triangle"); runTestCase(3, 4, 5, Triangle::SCALENE,
"EP", "Scalene triangle"); runTestCase(1, 1, 3, Triangle::INVALID,
"EP", "Invalid triangle"); runTestCase(0, 0, 0, Triangle::INVALID,
"EP", "Zero sides");
}

void runBoundaryValueTests() {
cout << "\nBOUNDARY VALUE ANALYSIS TEST CASES" <<
endl; cout << string(50, '=') << endl;

runTestCase(1, 1, 1, Triangle::EQUILATERAL, "BVA", "Minimum
equilateral"); runTestCase(2, 2, 3, Triangle::ISOSCELES, "BVA",
"Minimum isosceles"); runTestCase(3, 4, 5, Triangle::SCALENE,
"BVA", "Minimum scalene"); runTestCase(1, 2, 3,
Triangle::INVALID, "BVA", "Just invalid (sum)"); runTestCase(1, 1,
2, Triangle::INVALID, "BVA", "Boundary of invalid");
}
};

int main() { TestRunner runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests();
return 0;
}
```

```
EQUIVALENCE PARTITIONING TEST CASES
================================================
EP: Equilateral triangle
Input: a=5, b=5, c=5
Expected: 0
Actual: 0
Status: PASS
------------------------------------------------
EP: Isosceles triangle
Input: a=5, b=5, c=3
Expected: 1
Actual: 1
Status: PASS
------------------------------------------------
EP: Scalene triangle
Input: a=3, b=4, c=5
Expected: 2
Actual: 2
Status: PASS
------------------------------------------------
EP: Invalid triangle
Input: a=1, b=1, c=3
Expected: 3
Actual: 3
```

```
BOUNDARY VALUE ANALYSIS TEST CASES
================================================
BVA: Minimum equilateral
Input: a=1, b=1, c=1
Expected: 0
Actual: 0
Status: PASS
------------------------------------------------
BVA: Minimum isosceles
Input: a=2, b=2, c=3
Expected: 1
Actual: 1
Status: PASS
------------------------------------------------
BVA: Minimum scalene
Input: a=3, b=4, c=5
Expected: 2
Actual: 2
Status: PASS
------------------------------------------------
BVA: Just invalid (sum)
Input: a=1, b=2, c=3
Expected: 3
Actual: 3
Status: PASS
```

P5. Prefix Test Cases

Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
s1="he", s2="hello"	true
s1="hi", s2="hello"	false
s1="", s2="hello"	true
s1="hello", s2="he"	false
s1="hello", s2="hello"	true

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
s1="", s2=""	true
s1="a", s2="a"	true
s1="a", s2="ab"	true
s1="ab", s2="a"	false
s1="abc", s2="abcd"	true

Modified Program

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

class Prefix { public: bool isPrefix(string s1, string s2)
{ if (s1.length() > s2.length())

return false;
for (int i = 0; i < s1.length(); i++) { if (s1[i] !=
s2[i]) return false;
}
return true;
}
};

class TestRunner { private:
Prefix prefix;

void runTestCase(string s1, string s2, bool
expectedOutcome, string testType, string description) { bool
result = prefix.isPrefix(s1, s2);
string status = (result == expectedOutcome) ? "PASS" : "FAIL";

cout << testType << ": " << description << endl;
cout << "Input: s1=\"" << s1 << "\", s2=\"" << s2 << "\"" << endl;
cout << "Expected: " << (expectedOutcome ? "true" : "false") <<
endl; cout
<< "Actual: " << (result ? "true" : "false") << endl;
cout << "Status: " << status << endl; cout << string(50, '-') << endl;
}
```

```
public:
void runEquivalencePartitioningTests() {
cout << "\nEQUIVALENCE PARTITIONING TEST CASES" <<
endl; cout << string(50, '=') << endl;

runTestCase("", "abc", true, "EP", "Empty string prefix");
runTestCase("abc",
"abc", true, "EP", "Equal strings"); runTestCase("ab", "abc", true,
"EP", "Proper prefix"); runTestCase("abc", "ab", false, "EP", "First
longer than second"); runTestCase("abc", "def", false, "EP", "No
match");
}

void runBoundaryValueTests() {
cout << "\nBOUNDARY VALUE ANALYSIS TEST CASES" <<
endl; cout << string(50, '=') << endl;

runTestCase("", "", true, "BVA", "Both empty strings");
runTestCase("a", "a", true, "BVA", "Single character - match");

runTestCase("a", "b", false, "BVA", "Single character - no match");
runTestCase("a", "ab", true, "BVA", "Single char prefix");
runTestCase("ab", "abc", true, "BVA", "All but last char");
}
};

int main() { TestRunner runner;
runner.runEquivalencePartitioningTests();
runner.runBoundaryValueTests(); return 0;
}
```

```
EQUIVALENCE PARTITIONING TEST CASES
========================================
EP: Empty string prefix
Input: s1="", s2="abc"
Expected: true
Actual: true
Status: PASS
----------------------------------------
EP: Equal strings
Input: s1="abc", s2="abc"
Expected: true
Actual: true
Status: PASS
----------------------------------------
EP: Proper prefix
Input: s1="ab", s2="abc"
Expected: true
Actual: true
Status: PASS
----------------------------------------
EP: First longer than second
Input: s1="abc", s2="ab"
Expected: false
Actual: false
```

```
BOUNDARY VALUE ANALYSIS TEST CASES
==========================================
BVA: Both empty strings
Input: s1="", s2=""
Expected: true
Actual: true
Status: PASS
------------------------------------------
BVA: Single character - match
Input: s1="a", s2="a"
Expected: true
Actual: true
Status: PASS
------------------------------------------
BVA: Single character - no match
Input: s1="a", s2="b"
Expected: false
Actual: false
Status: PASS
------------------------------------------
BVA: Single char prefix
Input: s1="a", s2="ab"
Expected: true
Actual: true
Status: PASS
```

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:

Identify the equivalence classes for the system:

Valid Triangle Classes:
EC1: Equilateral triangles (all sides equal)
EC2: Isosceles triangles (exactly two sides equal)
EC3: Scalene triangles (no sides equal)
EC4: Right-angled triangles ($a^2 + b^2 = c^2$)
Invalid Triangle Classes:
EC5: Non-triangle (sum of any two sides ≤ third side)
EC6: Negative numbers for any side
EC7: Zero value for any side
EC8: Numbers too large for float representation

Test cases covering identified equivalence classes:

EC1 (Equilateral):
Test case: (5.0, 5.0, 5.0) Test
case: (10.0, 10.0, 10.0) EC2
(Isosceles):

Test case: (5.0, 5.0, 3.0)
Test case: (4.0, 3.0, 4.0)
Test case: (3.0, 4.0, 4.0) EC3
(Scalene):
Test case: (3.0, 4.0, 6.0)
Test case: (5.0, 7.0, 9.0) EC4
(Right-angled):
Test case: (3.0, 4.0, 5.0) Test
case: (6.0, 8.0, 10.0) EC5
(Invalid - sum):
Test case: (1.0, 1.0, 3.0)
Test case: (2.0, 3.0, 6.0) EC6
(Negative):
Test case: (-1.0, 2.0, 3.0) Test
case: (1.0, -2.0, 3.0)

EC7 (Zero):
Test case: (0.0, 2.0, 3.0)
Test case: (1.0, 0.0, 3.0)

Boundary test cases for A + B > C (scalene triangle):

Just below boundary (invalid):
Test case: (3.0, 4.0, 7.001)
Test case: (5.0, 6.0, 11.001) On
boundary (invalid):
Test case: (3.0, 4.0, 7.0)
Test case: (5.0, 6.0, 11.0) Just
above boundary (valid): Test
case: (3.0, 4.0, 6.999)
Test case: (5.0, 6.0, 10.999)

Boundary test cases for A = C (isosceles triangle):

Just below equality: Test
case: (5.0, 3.0, 4.999)
Exact equality: Test case:
(5.0, 3.0, 5.0) Just above
equality:
Test case: (5.0, 3.0, 5.001)

Boundary test cases for A = B = C (equilateral triangle):

Just below equality: Test
case: (5.0, 4.999, 5.0)
Exact equality: Test case:
(5.0, 5.0, 5.0) Just above
equality:
Test case: (5.0, 5.001, 5.0)

Boundary test cases for $A^2 + B^2 = C^2$ (right-angle triangle):

Just below right angle:
Test case: (3.0, 4.0, 4.999)
Exact right angle: Test
case: (3.0, 4.0, 5.0) Just
above right angle:
Test case: (3.0, 4.0, 5.001)

Boundary test cases for non-triangle:

Just below triangle inequality:

Test case: (2.0, 3.0, 5.001) On
triangle inequality:

Test case: (2.0, 3.0, 5.0) Just above triangle inequality:
Test case: (2.0, 3.0, 4.999)

Test points for non-positive input:

Zero values:
Test case: (0.0, 4.0, 5.0)
Test case: (3.0, 0.0, 5.0)
Test case: (3.0, 4.0, 0.0) Negative values:
Test case: (-0.001, 4.0, 5.0) Test case: (3.0, -0.001, 5.0) Test case: (3.0, 4.0, -0.001) Just above zero:
Test case: (0.001, 4.0, 5.0) Test case: (3.0, 0.001, 5.0) Test case: