It-314

Software Engineering

Meet Katharotiya - 202201157

Lab08: Black Box testing

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.

- **Equivalence class partitioning :**

Equivalence Class	**Input Type**	**description**	**Valid /invalid**
E1	Day	day<1	Invalid
E2	Day	1 ≤ day ≤ 31	Valid
E3	Day	day > 31	Invalid
E4	Month	month < 1	Invalid
E5	Month	1 ≤ month ≤ 12	Valid
E6	Month	month > 12	Invalid
E7	Year	year < 1900	Invalid
E8	Year	1900 ≤ year ≤ 2015	Valid
E9	Year	year > 2015	Invalid

- **Boundary Test-Cases analysis:**

Test Case No	Test Case (Day, Month, Year)	Valid/Invalid	Covered Classes
TC1	(0, 5, 1990)	Invalid	E1
TC2	(1, 5, 1990)	Valid	E2, E5, E8
TC3	(31, 5, 1990)	Valid	E2, E5, E8
TC4	(32, 5, 1990)	Invalid	E1
TC5	(15, 0, 1990)	Invalid	E4
TC6	(15, 1, 1990)	Valid	E2, E5, E8
TC7	(15, 12, 1990)	Valid	E2, E5, E8
TC8	(15, 13, 1990)	Invalid	E6
TC9	(15, 5, 1899)	Invalid	E7
TC10	(15, 5, 1900)	Valid	E2, E5, E8
TC11	(15, 5, 2015)	Valid	E2, E5, E8
TC12	(15, 5, 2016)	Invalid	E9

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.

```
#include <iostream>
#include <string>
using namespace std;
bool isLeapYear(int year)
{
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 ==
0);
}
int getDaysInMonth(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;
}
string getPreviousDate(int day, int month, int year)
{
    if (year < 1900 || year > 2015)
        return "Invalid";
    if (month < 1 || month > 12)
        return "Invalid";
    if (day < 1 || day > getDaysInMonth(month, year))
        return "Invalid";
    if (day > 1)
```

```
    {
        return to_string(day - 1) + "," + to_string(month) +
"," + to_string(year);
    }
    else
    {
        if (month > 1)
        {
            month--;
            day = getDaysInMonth(month, year);
        }
        else
        {
            year--;
            month = 12;
            day = 31;
        }
        return to_string(day) + "," + to_string(month) + "," +
to_string(year);
    }
}
```

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.

Equivalence Class Description:

- **E1**: Array is empty → Invalid
- **E2**: Value v is in the array → Valid

- **E3**: Value v is not in the array → Invalid
- **E4**: Array contains one element that is equal to v → Valid
- **E5**: Array contains one element that is not equal to v → Invalid
- **E6**: Array contains duplicate elements, and v is among them → Valid
- **E7**: Array contains duplicate elements, and v is not among them → Invalid

Equivalence Class Test Cases:

Test Case	Input Data (Array a, Value v)	Expected Outcome	Covered Equivalence Class
TC1	([], 5)	-1	E1
TC2	([1, 2, 3, 4, 5], 3)	2	E2
TC3	([1, 2, 3, 4, 5], 6)	-1	E3
TC4	([5], 5)	0	E4
TC5	([5], 3)	-1	E5
TC6	([1, 2, 3, 2, 1], 2)	1	E6
TC7	([1, 2, 3, 4, 5], 0)	-1	E7

Boundary Conditions:

- **B1**: Array has 0 elements (empty)
- **B2**: Array has 1 element (equal to v)
- **B3**: Array has 1 element (not equal to v)
- **B4**: Array has 2 elements (first is v)
- **B5**: Array has 2 elements (second is v)
- **B6**: Array has 2 elements (v is not present)

Boundary Value Test Cases:

Test Case	Input Data (Array a, Value v)	Expected Outcome	Covered Boundary Condition
TC1	([], 5)	-1	B1
TC2	([5], 5)	0	B2
TC3	([5], 3)	-1	B3
TC4	([3, 5], 3)	0	B4
TC5	([5, 3], 3)	1	B5
TC6	([1, 2], 3)	-1	B6

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

```
    return (-1);
}
```

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

Equivalence Class Description:

- **E1**: Array is empty → Invalid
- **E2**: Value v is present in the array (at least once) → Valid
- **E3**: Value v is not present in the array → Invalid
- **E4**: Array contains one element that is equal to v → Valid
- **E5**: Array contains one element that is not equal to v → Invalid
- **E6**: Array contains multiple elements, and v appears multiple times → Valid
- **E7**: Array contains multiple elements, and v appears once → Valid
- **E8**: Array contains multiple elements, and v does not appear at all → Invalid

Equivalence Class Test Cases:

Test Case	Input Data (Value v, Array a)	Expected Outcome
TC1	(5, [])	0
TC2	(3, [1, 2, 3, 4, 3, 5])	2
TC3	(6, [1, 2, 3, 4, 5])	0
TC4	(5, [5])	1
TC5	(3, [5])	0

Boundary Conditions:

- **B1**: Array has 0 elements (empty)
- **B2**: Array has 1 element (equal to v)
- **B3**: Array has 1 element (not equal to v)
- **B4**: Array has 2 elements (one is v)
- **B5**: Array has 2 elements (both are v)
- **B6**: Array has 2 elements (none is v)

Boundary Value Test Cases:

Test Case	Input Data (Value v, Array a)	Expected Outcome	Covered Boundary Condition
TC1	(5, [])	0	B1
TC2	(5, [5])	1	B2
TC3	(3, [5])	0	B3
TC4	(2, [1, 2])	1	B4
TC5	(2, [2, 2])	2	B5
TC6	(3, [1, 2])	0	B6

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

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.

Equivalence Class Description:

- **E1**: Array is empty → Invalid
- **E2**: Value v is in the array → Valid
- **E3**: Value v is not in the array → Invalid
- **E4**: Array contains one element that is equal to v → Valid
- **E5**: Array contains one element that is not equal to v → Invalid
- **E6**: Array contains multiple elements, and v is among them → Valid
- **E7**: Array contains multiple elements, and v is not among them → Invalid
- **E8**: Array has only one element which is less than v → Invalid
- **E9**: Array has only one element which is greater than v → Invalid

Equivalence Class Test Cases:

Test Case	Input Data (Value v, Array a)	Expected Outcome	Covered Equivalence Class
TC1	(5, [])	-1	E1
TC2	(3, [1, 2, 3, 4, 5])	2	E2
TC3	(6, [1, 2, 3, 4, 5])	-1	E3
TC4	(5, [5])	0	E4
TC5	(3, [5])	-1	E5

Boundary Conditions:

- **B1**: Array has 0 elements (empty)
- **B2**: Array has 1 element (equal to v)
- **B3**: Array has 1 element (not equal to v)
- **B4**: Array has 2 elements (one is v)
- **B5**: Array has 2 elements (both are v)
- **B6**: Array has 2 elements (none is v)

Boundary Value Test Cases:

Test Case	Input Data (Value v, Array a)	Expected Outcome	Covered Boundary Condition
TC1	(5, [])	-1	B1
TC2	(5, [5])	0	B2
TC3	(3, [5])	-1	B3
TC4	(2, [1, 2])	1	B4
TC5	(2, [2, 2])	0	B5
TC6	(3, [1, 2])	-1	B6

```
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);
}
```

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).

Equivalence Class Description:

- **E1**: All sides are equal → Equilateral → Valid
- **E2**: Two sides are equal, and one is different → Isosceles → Valid
- **E3**: All sides are different → Scalene → Valid
- **E4**: Any side is greater than or equal to the sum of the other two sides → Invalid → Invalid
- **E5**: One or more sides are negative → Invalid → Invalid
- **E6**: All sides are zero → Invalid → Invalid

Equivalence Class Test Cases:

Test Case	Input Data (Sides a, b, c)	Expected Outcome	Covered Equivalence Class
TC1	(3, 3, 3)	0	E1
TC2	(5, 5, 3)	1	E2
TC3	(4, 5, 6)	2	E3
TC4	(1, 2, 3)	3	E4
TC5	(5, 2, 10)	3	E4

TC6	(0, 0, 0)	3	E6
TC7	(4, -2, 5)	3	E5
TC8	(5, 5, 5)	0	E1
TC9	(7, 3, 7)	1	E2
TC10	(1, 1, 2)	3	E4

Boundary Conditions:

- **B1**: All sides are equal and positive
- **B2**: One side is zero, and the others are positive
- **B3**: One or more sides are negative
- **B4**: One side equals the sum of the other two
- **B5**: Two sides equal the sum of the third side (edge case)

Boundary Value Test Cases:

Test Case	Input Data (Sides a, b, c)	Expected Outcome	Covered Boundary Condition
TC1	(3, 3, 3)	0	B1
TC2	(0, 1, 1)	3	B2
TC3	(-1, 1, 1)	3	B3
TC4	(3, 4, 7)	3	B4
TC5	(3, 3, 6)	3	B5
TC6	(0, 0, 0)	3	B6 (covers E6)

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).

Equivalence Class Description:

- E1: s1 is a non-empty string and is a prefix of s2 → Valid
- E2: s1 is a non-empty string but not a prefix of s2 → Invalid
- E3: s1 is an empty string, and s2 is a non-empty string → Valid (an empty string is a prefix of any string)
- E4: s1 is a non-empty string, and s2 is empty → Invalid (a non-empty string cannot be a prefix of an empty string)
- E5: Both s1 and s2 are empty → Valid (an empty string is a prefix of another empty string)

Equivalence Class Test Cases:

Test Case	Input Data (String s1, String s2)	Expected Outcome	Covered Equivalence Class
TC1	("pre", "prefix")	TRUE	E1
TC2	("test", "testing")	FALSE	E2
TC3	("", "hello")	TRUE	E3
TC4	("hello", "")	FALSE	E4
TC5	("", "")	TRUE	E5

Boundary Conditions:

- **B1**: s1 is empty, and s2 is non-empty
- **B2**: s1 is non-empty, and s2 is empty
- **B3**: Length of s1 is 1, and s2 is longer than s1
- **B4**: Length of s1 is equal to the length of s2

- **B5**: Length of s1 is greater than the length of s2

Boundary Value Test Cases:

Test Case	Input Data (String s1, String s2)	Expected Outcome	Covered Boundary Condition
TC1	("", "a")	TRUE	B1
TC2	("a", "")	FALSE	B2
TC3	("a", "ab")	TRUE	B3
TC4	("abc", "abc")	TRUE	B4
TC5	("abc", "ab")	FALSE	B5

```
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;
}
```

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
b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence classes)
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 A2 + B2 = C2 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.

a) Equivalence Class Identification

1. **E1**: All sides are equal (A = B = C) → Equilateral → Valid
2. **E2**: Two sides are equal (A = B or B = C or A = C) but not all → Isosceles → Valid
3. **E3**: All sides are different (A ≠ B, B ≠ C, A ≠ C) → Scalene → Valid
4. **E4**: A + B > C, A + C > B, B + C > A → Valid (triangle can be formed)
5. **E5**: A + B = C, A + C = B, or B + C = A → Invalid (degenerate triangle)
6. **E6**: A + B < C, A + C < B, or B + C < A → Invalid (triangle cannot be formed)

7. **E7**: One or more sides are zero or negative → Invalid (non-positive input)

b) Test Cases for Identified Equivalence Classes

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Equivalence Class
TC1	(3.0, 3.0, 3.0)	"Equilateral"	E1
TC2	(4.0, 4.0, 5.0)	"Isosceles"	E2
TC3	(3.0, 4.0, 5.0)	"Scalene"	E3
TC4	(2.0, 2.0, 5.0)	"Invalid"	E5
TC5	(1.0, 2.0, 3.0)	"Invalid"	E6
TC6	(-1.0, 2.0, 3.0)	"Invalid"	E7
TC7	(0.0, 3.0, 4.0)	"Invalid"	E7

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

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(2.0, 3.0, 4.0)	"Scalene"	A + B > C
TC2	(3.0, 4.0, 7.0)	"Invalid"	A + B = C (boundary)
TC3	(4.0, 5.0, 8.0)	"Scalene"	A + B > C

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

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(5.0, 3.0, 5.0)	"Isosceles"	A = C
TC2	(5.0, 5.0, 5.0)	"Equilateral"	A = C (valid)

TC3	(5.0, 3.0, 7.0)	"Invalid"	A + C < B (boundary)

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

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(3.0, 3.0, 3.0)	"Equilateral"	A = B = C
TC2	(0.0, 0.0, 0.0)	"Invalid"	A = B = C (non-positive)

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

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(3.0, 4.0, 5.0)	"Right Angled"	$A^2 + B^2 = C^2$
TC2	(5.0, 12.0, 13.0)	"Right Angled"	$A^2 + B^2 = C^2$
TC3	(3.0, 5.0, 7.0)	"Scalene"	$A^2 + B^2 > C^2$

g) Boundary Test Cases for Non-Triangle Case

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(1.0, 1.0, 3.0)	"Invalid"	A + B < C
TC2	(2.0, 2.0, 5.0)	"Invalid"	A + B = C
TC3	(0.0, 2.0, 2.0)	"Invalid"	Non-positive input

h) Boundary Test Cases for Non-Positive Input

Test Case	Input Data (A, B, C)	Expected Outcome	Covered Boundary Condition
TC1	(-1.0, 2.0, 3.0)	"Invalid"	Non-positive input
TC2	(0.0, 3.0, 4.0)	"Invalid"	Non-positive input
TC3	(2.0, 0.0, 3.0)	"Invalid"	Non-positive input
TC4	(3.0, 4.0, -5.0)	"Invalid"	Non-positive input