IT – 314

Software Engineering

Lab – 08 :

Software Testing

Lab Session - Functional Testing (Black-Box)

Student Name - ID :

Dhruvin Akhaja - 202201172

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.

Equivalence class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Month	1≤month≤12	Valid months from 1 to 12	Valid
E2	Month	month<1	Months less than 1 (e.g., 0, -1)	Invalid
E3	Month	month>12	Months greater than 12 (e.g., 13, 14)	Invalid
E4	Day	1≤day≤max_day	Valid days from 1 to the maximum day for each month (e.g., 31 for January, 28 for February, etc.)	Valid
E5	Day	day<1	Days less than 1 (e.g., 0, -1)	Invalid
E6	Day	day>max_day	Days greater than the maximum valid day for each month (e.g., 32 in January, 31 in February)	Invalid
E7	Year	1900≤year≤2015	Valid years between 1900 and 2015	Valid
E8	Year	year<1900	Years less than 1900 (e.g., 1899, 1800)	Invalid
E9	Year	year>2015	Years greater than 2015 (e.g., 2016, 2020)	Invalid

Test Cases:

Test Case	Input (Day, Month, Year)	Expected Outcome	Classes Covered
TC1	(1, 5, 2010)	Valid	E1, E4, E7
TC2	(15, 10, 2015)	Valid	E1, E4, E7
TC3	(31, 12, 2010)	Valid	E1, E4, E7
TC4	(1, 3, 2010)	Valid	E1, E4, E7
TC5	(29, 2, 2012)	Valid (leap year)	E1, E4, E7
TC6	(28, 2, 2011)	Valid	E1, E4, E7
TC7	(1, 0, 2010)	Invalid	E2
TC8	(1, 13, 2010)	Invalid	E3
TC9	(0, 5, 2010)	Invalid	E5
TC10	(32, 5, 2010)	Invalid	E6
TC11	(1, 5, 1899)	Invalid	E8
TC12	(1, 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 <tuple>
using namespace std;
string prev_date(int d, int m, int y)
{
    if (m < 1 || m > 12 || y < 1900 || y > 2015 || d < 1 || d > 31)
    {
        return "Invalid";
    }
    return "Valid";
}
```

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 class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Array a[]	Array is empty	The array has no elements, so a.length == 0	Invalid
E2	Array a[]	Array has elements	The array has elements, a.length > 0	Valid
E3	Value v	v exists in the array	The search value v is present in the array	Valid
E4	Value v	v does not exist in the array	The search value v is not present in the array	Invalid
E5	Value v	v is a boundary element	The search value v is at the first or last index of the array	Valid

Test Cases :

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

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);
}
```

Equivalence class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Array a[]	Array is empty	The array has no elements, so a.length == 0	Invalid
E2	Array a[]	Array has elements	The array has elements, a.length > 0	Valid
E3	Value v	v exists in the array	The search value v is present in the array	Valid
E4	Value v	v does not exist in the array	The search value v is not present in the array	Invalid
E5	Value v	Multiple occurrences of v	The search value v appears multiple times in the array	Valid

E6	Value v	v is a boundary element	The search value v is at the first or last index of the array	Valid

Test cases :

Test Case	Input (v, a[])	Expected Outcome	Classes Covered
TC1	(5, [])	0	E1, E4
TC2	(3, [1, 2, 3, 4, 5])	1	E2, E3
TC3	(5, [1, 2, 3, 4, 5])	0	E2, E4
TC4	(3, [1, 2, 3, 3, 4, 5])	2	E2, E3, E5
TC5	(1, [1, 2, 3, 1, 4, 5, 1])	3	E2, E3, E5
TC6	(6, [6, 2, 3, 4, 6])	2	E2, E3, E5

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);
}
```

Equivalence class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Array a[]	Array is empty	The array has no elements, so a.length == 0	Invalid
E2	Array a[]	Array has elements	The array has elements, a.length > 0	Valid
E3	Value v	v exists in the array	The search value v is present in the array	Valid
E4	Value v	v does not exist in the array	The search value v is not present in the array	Invalid
E5	Value v	v is less than the first element	The search value v is smaller than the first element in the array	Invalid
E6	Value v	v is greater than the last element	The search value v is larger than the last element in the array	Invalid
E7	Value v	v is a boundary element	The search value v is at the first or last index of the array	Valid

Test cases :

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

TC8	(10, [1, 2, 3, 4, 5])	-1	E2, E6
TC9	(2, [1, 2, 3, 4, 5])	1	E2, E3
TC10	(5, [1, 2, 3, 5])	3	E2, E3, E7

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).

```
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);
}
```

Equivalence class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Side lengths	Invalid triangle	The provided lengths do not satisfy the triangle inequality	Invalid
E2	Side lengths	Equilateral triangle	All three sides are equal	Valid
E3	Side lengths	Isosceles triangle	Exactly two sides are equal	Valid

E4	Side lengths	Scalene triangle	All three sides are different	Valid

Test cases :

Test Case	Input (a, b, c)	Expected Outcome	Classes Covered
TC1	(1, 1, 1)	0	E2
TC2	(2, 2, 3)	1	E3
TC3	(3, 4, 5)	2	E4
TC4	(1, 2, 3)	3	E1
TC5	(5, 5, 5)	0	E2
TC6	(2, 2, 2)	0	E2
TC7	(2, 2, 5)	1	E3
TC8	(5, 5, 10)	3	E1
TC9	(7, 8, 9)	2	E4
TC10	(1, 3, 4)	3	E1

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;
```

```
}
```

Equivalence class partitioning:

Class Name	Input Type	Equivalence Class	Description	Validity
E1	String s1	s1 is empty	The first string is empty	Valid
E2	String s2	s2 is empty	The second string is empty	Invalid
E3	String s1	s1 length > s2 length	The first string is longer than the second string	Invalid
E4	String s1	s1 is a prefix of s2	The first string is an exact prefix of the second string	Valid
E5	String s1	s1 is not a prefix of s2	The first string is not a prefix of the second string	Invalid
E6	String s1	s1 is identical to s2	Both strings are exactly the same	Valid

Test cases :

Test Case	Input (s1, s2)	Expected Outcome	Classes Covered
TC1	("", "hello")	TRUE	E1
TC2	("hello", "")	FALSE	E2
TC3	("hello", "hello world")	TRUE	E4
TC4	("hello", "world")	FALSE	E5
TC5	("hello world", "hello")	FALSE	E3
TC6	("abc", "abc")	TRUE	E4, E6
TC7	("abc", "abcd")	TRUE	E4
TC8	("abc", "ab")	FALSE	E5
TC9	("abcd", "abc")	FALSE	E3
TC10	("", "")	TRUE	E1

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) Identify the Equivalence Classes

Class Name	Input Type	Equivalence Class	Description	Validity
E1	Length A	$A \leq 0$	Length A is non-positive (0 or negative).	Invalid
E2	Length B	$B \leq 0$	Length B is non-positive (0 or negative).	Invalid
E3	Length C	$C \leq 0$	Length C is non-positive (0 or negative).	Invalid
E4	Length A, B, C	$A + B > C$	A triangle can be formed (valid triangle condition for scalene	Valid

			triangle).	
E5	Length A, B, C	A + B = C	Degenerate triangle (valid but does not form a proper triangle).	Invalid
E6	Length A, B, C	A + B < C	Cannot form a triangle.	Invalid
E7	Length A, B, C	A = B	Isosceles triangle condition (two sides equal).	Valid
E8	Length A, B, C	A = C	Isosceles triangle condition (two sides equal).	Valid
E9	Length A, B, C	B = C	Isosceles triangle condition (two sides equal).	Valid
E10	Length A, B, C	A = B = C	Equilateral triangle condition (all sides equal).	Valid
E11	Length A, B, C	$A^2 + B^2 = C^2$	Right-angled triangle condition (Pythagorean theorem).	Valid
E12	Length A, B, C	$A^2 + B^2 < C^2$	Not a right-angled triangle.	Invalid
E13	Length A, B, C	$A^2 + B^2 > C^2$	Not a right-angled triangle.	Invalid

b) Identify Test Cases to Cover the Identified Equivalence Classes

Test Case	Input (A, B, C)	Expected Outcome	Classes Covered
TC1	(0, 5, 5)	Invalid (A ≤ 0)	E1
TC2	(5, 0, 5)	Invalid (B ≤ 0)	E2
TC3	(5, 5, 0)	Invalid (C ≤ 0)	E3
TC4	(2, 3, 4)	Scalene	E4
TC5	(3, 3, 6)	Invalid (A + B = C)	E5
TC6	(2, 3, 6)	Invalid (A + B < C)	E6
TC7	(4, 4, 5)	Isosceles	E7
TC8	(5, 5, 5)	Equilateral	E10
TC9	(3, 4, 5)	Right-angled	E11
TC10	(2, 2, 3)	Isosceles	E8
TC11	(2, 3, 4)	Not right-angled	E12
TC12	(1, 1, 1)	Equilateral	E10
TC13	(1, 1, Math.sqrt(2))	Right-angled	E11

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

Test Case	Input (A, B, C)	Expected Outcome	Description
BC1	(2, 3, 4)	Scalene	Valid triangle
BC2	(3, 5, 6)	Scalene	Valid triangle
BC3	(1, 2, 2.5)	Scalene	Valid triangle

d) Boundary Condition A = C (Isosceles Triangle)

Test Case	Input (A, B, C)	Expected Outcome	Description
BC1	(2, 2, 3)	Isosceles	Valid triangle
BC2	(3, 4, 3)	Isosceles	Valid triangle

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

Test Case	Input (A, B, C)	Expected Outcome	Description
BC1	(1, 1, 1)	Equilateral	Valid triangle
BC2	(3, 3, 3)	Equilateral	Valid triangle

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

Test Case	Input (A, B, C)	Expected Outcome	Description
BC1	(3, 4, 5)	Right-angled	Valid triangle
BC2	(5, 12, 13)	Right-angled	Valid triangle
BC3	(1, 1, Math.sqrt(2))	Right-angled	Valid triangle

g) Non-Triangle Case

Test Case	Input (A, B, C)	Expected Outcome	Description
TC1	(1, 1, 3)	Invalid	Cannot form a triangle
TC2	(2, 2, 5)	Invalid	Cannot form a triangle
TC3	(1, 2, 4)	Invalid	Cannot form a triangle

h) Non-Positive Input

Test Case	Input (A, B, C)	Expected Outcome	Description
TC1	(0, 2, 3)	Invalid	A is non-positive
TC2	(-1, 5, 3)	Invalid	A is negative
TC3	(2, 0, -3)	Invalid	B is non-positive
TC4	(1, 1, 0)	Invalid	C is non-positive
TC5	(-1, -1, -1)	Invalid	All sides negative