IT-314
Software Engineering

Jal Dani
202201315

Lab_08 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 dates or invalid dates. Design the equivalence class test cases?

- Equivalence Classes:
    1. Year < 1900 (Invalid)
    2. Year > 2015 (Invalid)
    3. 1900<=Year<=2015 (Valid)
    4. Month < 1 (Invalid)
    5. Month > 12 (Invalid)
    6. 1<=Month<=12 (Valid)
    7. Day < 1 (Invalid)
    8. Day > 31 (Invalid)
    9. 1 <= Day <= 31 (Valid)

In Boundary Value Analysis, the emphasis is on testing values at the extremes of defined ranges:

- **Month boundaries:** Check edge cases such as the first and last months (1 and 12, valid) and values slightly outside this range, like 0 and 13 (invalid).
- **Day boundaries:** Evaluate days at the start (1) and the maximum possible (31), as well as out-of-range values like 0 and 32.
- **Year boundaries:** Test the lower and upper limits, such as the years 1900 and 2015 (valid), and years just below or above, like 1899 and 2016 (invalid).

TestCase No.	Day	Month	Year	Expected Outcome	Remark
1	15	5	2004	Valid	
2	31	12	2005	Valid	
3	1	1	2000	Valid	
4	32	1	2000	Invalid	Day>32
5	0	1	2000	Invalid	Day<1
6	7	0	2000	Invalid	Month<1
7	7	10	2000	Invalid	Month>12
8	7	10	1977	Invalid	Year<1990
9	7	1	2020	Invalid	Year>2015
10	30	2	2000	Invalid	Feb month
11	29	2	2001	Invalid	Not leap Year
12	31	6	2004	Invalid	31 day in June
13	32	10	2004	Invalid	32 Day in Oct

Equivalence Class Test Cases:

Number	Day	Month	Year	Covered equivalence class	Valid/Invalid
1.	2	3	1901	E1, E4, E7	Valid Output - 1/3/1901
2.	-2	3	2000	E2	Invalid
3.	40	3	2000	E3	Invalid
4.	12	-1	2004	E5	Invalid
5.	1	3	2001	E1,E4,E7	Valid Output - 28/2/2001
6.	1	3	2001	E1,E4,E7	Valid Output - 29/2/2001
7.	12	15	2002	E6	Invalid
8.	12	12	1801	E8	Invalid
9.	10	1	2020	E9	Invalid

- 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;

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 > 31) return "Invalid";

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

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 Class Description:

- E1: The array is empty.
- E2: The value v is present in the array.
- E3: The value v is not present in the array.
- E4: The array contains only one element which is equal to v.
- E5: The array contains only one element which is not equal to v

Test Cases:

Number	Input Data(Array a, Value v)	Expected Outcome	Covered Equivalence Class
1.	a = [ ], v = 5	-1	E1
2.	a = [1,2,3,4,5], v = 5	4	E2
3.	a = [1,2,3,4,6], v = 5	-1	E3
4.	a = [5], v = 5	0	E4
5.	a = [4], v = 5	-1	E5

Boundary Value Analysis:

- The array size is at its minimum size, either empty or contains just one element.
- The value is at the start or end of the array.
- The value is not present, but close to elements in the array.

Input data	Description	Expected Outcome
v = 5, a = {5}	Array has only one element and v is present	0
v = 3, a = {5}	Array has only one element and v is not present	-1
v = 10, a = {1, 2, 3, 4, 5}	v is not in the array but outside the range	-1
v = 5, a = {}	Empty array	-1

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:-**

1. Array contains multiple occurrences of the value.
2. Array does not contain the value.
3. Empty array.
4. Single element array (element is the value).
5. Single element array (element is not the value).

Equivalence Partitioning:

Input data	Description	Expected Outcome
v = 3, a = {1, 2, 3, 4, 5}	v is present 1 time in array a[]	1 (valid)
v = 6, a = {1, 2, 3, 4, 5}	v is not present in array a[]	0 (valid)
v = 1, a = {}	Array a[] is empty	0 (valid)
v = 2, a = {1, 2, 2, 3, 4}	v is present 2 times in a[]	2 (valid)
v = 5, a = null	a[] is null	Error
v = 3, a = {1, 'a', 3, 4}	Array a[] contains non-integer values	Error
v = 'b', a = {1, 2, 3, 4,5}	v is a non-integer value	Error

- Boundary Class Values:

Test Cases	Expected Outcome
v=1, a[] = [1,2,3]	1
v=2, a[] = [1,2,3]	1
v=1, a[] = []	0
v=2, a[] = [2]	0
v=3, a[] = [1,2]	0

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

```cpp
#include <iostream>

int binarySearch(int v, int a[], int size) {
    int lo = 0;
    int hi = size - 1;

    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (v == a[mid]) {
            return mid; // Found the value, return index
        } 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
}
```

Equivalence Class Description:

- E1: The array is empty.
- E2: The value v is present in the array.
- E3: The value v is not present in the array.
- E4: The array contains only one element which is equal to v.
- E5: The array contains only one element which is not equal to v.

Test Case	Input Data (Array a,Value v)	Expected Outcome	Equivalence Class
1.	a = [ ], v = 5	-1	E1
2.	a = [1,2,3,4,5], v = 5	4	E2
3.	a = [1,2,3,4,6], v = 5	-1	E3
4.	a = [5], v = 5	0	E4
5.	a = [4], v = 5	-1	E5

Boundary Conditions:

- The array size is at its minimum size, either empty or contains just one element.
- The value is at the start, middle, or end of the array.
- The value is not present but close to elements in the array.

Test Case	Input Data (Array a,Value v)	Expected Outcome	Boundary Condition
1.	a = [5], v = 5	0	Single element array, value present
2.	a = [5], v = 6	-1	Single element array, value absent
3.	a = [1,2,3,4,5], v = 1	0	Value is at the start of the array
4.	a = [1,2,3,4,5], v = 3	2	Value is in middle of array
5.	a = [1,2,3,4,5], v = 5	4	Value is at end of array
6.	A = [1,2,3,4,5], v = 6	-1	Value absent but close to elements in array

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:
1. Valid equilateral triangle:
2. Valid isosceles triangle:
3. Valid scalene triangle:
4. Invalid triangle (sum of any two sides must be greater than the third):
5. Negative lengths:
6. Zero lengths:

Test Case	Expected Outcomes
a=3 ,b=3,c=3	Equilateral
a=4,b=4,c=5	Isosceles
a=3,b=4,c=5	Scalene
a=1,b=2,c=3	InvaliD
a=-1,b=3,c=4	Invalid
a=0,b=3,c=4	Invalid

- **Boundary Class:**

Test-Case	Expected Outcomes
a = 2, b = 2, c = 2	Invalid
a = 1, b = 1, c = 2	Equilateral
a = -1, b = 1, c = 1	Invalid
a = 0, b = 1, c = 1	Invalid
a = 1, b = 1, c = 1	Equilateral

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 Description:

- E1: s1 is longer than s2 (impossible to be a prefix).
- E2: s1 is a valid prefix of s2.
- E3: s1 is not a prefix of s2.
- E4: s1 is an empty string (edge case).
- E5: s2 is an empty string (edge case).

Test Case	Input Data (s1, s2)	Expected Outcome	Covered Equivalence Class
TC1	"abcdef", "abc"	false	E1
TC2	"abc", "abcdef"	true	E2
TC3	"xyz", "abcdef"	false	E3
TC4	" ", "abcdef"	true	E4
TC5	"abc", ""	false	E5

Boundary Conditions:

- Length of s1 is greater than the length of s2.
- s1 is an empty string, s2 is a non-empty string.
- s2 is an empty string, s1 is non-empty.
- s1 equals s2.

Test Case	Input Data (s1, s2)	Expected Outcome	Boundary Condition
TC1	“a”,” ”	false	S2 is empty
TC2	"abcdef", "abcdef"	true	s1 equals s2
TC3	"abc", "abc"	true	Shorter but equal strings
TC4	" ", " "	true	Both strings are empty

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.

● **By Equivalence Class:**
1. Valid equilateral triangle: All sides are equal.
2. Valid isosceles triangle: Exactly two sides are equal.
3. Valid scalene triangle: All sides are different.
4. Valid right-angled triangle: Follows the Pythagorean theorem.
5. Invalid triangle (non-triangle): Sides do not satisfy triangle inequalities.
6. Invalid input (non-positive values): one or more sides are non-positive.

a) Identify the equivalence classes for the system:

1. Valid Equivalence Classes
2. Equilateral Triangle a. All sides are equal.
3. Isosceles Triangle a. Exactly two sides are equal.
4. Scalene Triangle a. All sides are different.
5. Right-Angled Triangle a. Satisfies the Pythagorean theorem ($A^2 + B^2 = C^2$).
6. Invalid Equivalence Classes
7. Non-Triangle a. The sum of the lengths of any two sides is not greater than the third side.
8. Negative Values a. One or more sides have negative lengths.
9. Zero Values a. One or more sides are zero.

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)

Side Lengths	Description	Triangle Class
2.0, 2.0, 2.0	All sides are equal	Equilateral Triangle
3.0, 3.0, 4.0	Two sides are equal	Isosceles Triangle
7.0, 9.0, 10.0	All sides are different	Scalene Triangle
3.0, 4.0, 5.0	Satisfies the Pythagorean theorem	Right-Angled Triangle
1.0, 2.0, 3.0	The sum of the two shorter sides is equal to the longest	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Negative Values
0.0, 1.0, 1.0	One side is zero	Zero Values

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Class
2.0, 3.0, 4.0	Valid scalene triangle	Scalene Triangle
2.0, 2.5, 5.0	Just below boundary	Non-Triangle
1.0, 1.0, 2.0	Exactly on the boundary	Non-Triangle

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Class
5.0, 5.0, 3.0	Two sides equal, valid isosceles	Isosceles Triangle
3.0, 3.0, 6.0	Just below boundary	Non-Triangle
2.0, 2.0, 4.0	Exactly on the boundary	Non-Triangle

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

Input Data	Description	Covered Class
3.0, 3.0, 3.0	All sides equal, valid equilateral	Equilateral Triangle
2.0, 2.0, 2.0	Valid equilateral	Equilateral Triangle
2.0, 2.0, 3.0	Just isosceles	Isosceles Triangle

f)) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary

Input Data	Description	Covered Class
3.0, 4.0, 5.0	Valid right-angled triangle	Right-Angled Triangle
5.0, 12.0, 13.0	Valid right-angled triangle	Right-Angled Triangle

g) For the non-triangle case, identify test cases to explore the boundary.

Input Data	Description	Covered Class
1.0, 2.0, 3.0	Sum of two sides equals the third	Non-Triangle
2.0, 5.0, 3.0	Valid triangle	Scalene
10.0, 1.0, 1.0	Impossible lengths	Non-Triangle

h) For non-positive input, identify test points.

Input Data	Description	Covered Class
0.0, 0.0, 0.0	All sides are zero	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Non-Triangle
2.0, 0.0, 2.0	One side is zero	Non-Triangle