Software Engineering : Lab-8

Name – Patel Aryan
Id – 202201511

Question - 1

Equivalence Classes :

(1) Valid Classes
  (a) Month range is in between 1 <= month <= 12.
  (b) Day range is in between 1<= day <= 31.
  (c) Year range is in between 1900 <= year <= 2015
(2) Invalid Classes
  (a) Month is less than 1
  (b) Month is greater than 12
  (c) Year is less than 1900
  (d) Year is greater than 2015
  (e) Day is less than 1
  (f) Day is greater than 31
  (g) Day is 30 or 31, when month is 2(February)

Test Cases :

(1) Equivalence class partitioning :

No of Test Cases	Tester Action and input data	Expected Output	Reason

1	(1, 1, 1900)	(31, 12, 1899)	-
2	(1, 3, 2000)	(29, 2, 2000)	-
3	(1, 5, 2015)	(30, 4, 2015)	-
4	(31, 4, 2000)	Error	April has 30 days
5	(30, 2, 2015)	Error	February has 28 days
6	(0, 1, 2015)	Error	Day out of range
7	(1, 13, 2015)	Error	Month out of range
8	(1, 1, 1899)	Error	Year out of range
9	(1, 1, 2016)	Error	Year out of range

(2) Boundary Value Analysis :

No of Test Cases	Tester Action and input data	Expected Output	Reason
1	(1, 1, 1900)	(31, 12, 1899)	-
2	(31, 12, 2015)	(30, 12, 2015)	-
3	(1, 1, 1899)	Error	Year out of range
4	(1, 1, 2016)	Error	Year out of range
5	(1, 2, 2015)	(31, 1, 2015)	-
6	(1, 1, 2015)	(31, 12, 2014)	-

(3) Program Code :

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

```cpp
bool is_leap_year(int year) {
    return year % 4 == 0 && (year % 100 != 0 || year % 400 == 0);
}

std::string get_previous_date(int day, int month, int year) {
    if (year < 1900 || year > 2015) {
        return "Error: Invalid year";
    }
    if (month < 1 || month > 12) {
        return "Error: Invalid month";
    }

    std::vector<int> days_in_month = {31, 28 + is_leap_year(year), 31, 30, 31,
30, 31, 31, 30, 31, 30, 31};

    if (day < 1 || day > days_in_month[month - 1]) {
        return "Error: Invalid day";
    }

    if (day == 1) {
        if (month == 1) {
            year -= 1;
            month = 12;
            day = 31;
        } else {
            month -= 1;
            day = days_in_month[month - 1];
        }
    } else {
        day -= 1;
    }

    return "Previous date: " + std::to_string(day) + "/" + std::to_string(month)
+ "/" + std::to_string(year);
}

int main() {
    std::vector<std::tuple<int, int, int>> test_cases = {
        {12, 5, 2015},
        {1, 1, 1900},
        {32, 1, 2010},
        {15, 13, 2010},
        {29, 2, 2012},
        {29, 2, 2013},
        {31, 4, 2010},
```

```
        {0, 5, 2000},
        {10, 0, 2010},
        {25, 6, 1800},
        {25, 6, 2016},
        {1, 1, 1900},
        {31, 12, 2015},
        {31, 1, 2015},
        {1, 2, 2015},
        {1, 3, 2015},
        {29, 2, 2012},
        {1, 5, 2015},
    };

    for (const auto& [day, month, year] : test_cases) {
        std::string result = get_previous_date(day, month, year);
        std::cout << "Input: " << day << "/" << month << "/" << year << " -> " <<
result << std::endl;
    }

    return 0;
}
```

Question – 2

Problem-1 :

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

Equivalence Classes (Valid and Invalid):

Class 1: The array is non-empty, and the value v is present in the array.
Class 2: The array is non-empty, and the value v is not present in the array.
Class 3: The array contains multiple occurrences of the value v.
Class 4: The array has a single element, and that element is equal to v.
Class 5: The array has a single element, and that element is not equal to v.
Class 6: The array is empty.
Class 7: The array is non-integer or non-numeric or the value v is of an invalid type.

Test cases:

Test Case	Input v	Input a[]	Valid/Invalid
TC1	3	{1, 2, 3, 4, 5}	Valid
TC2	6	{1, 2, 3, 4, 5}	Valid
TC3	2	{1, 2, 2, 2, 5}	Valid
TC4	2	{2}	Valid
TC5	1	{2}	Valid
TC6	3	{}	Invalid (edge case)
TC7	3	{3.5, 1, 2}	Invalid
TC8	a'	{1, 2, 3}	Invalid
TC9	5	{1, 2, 3, 4, 5}	Valid
TC10	0	{0, 1, 2, 3}	Valid

Problem-2 :

```
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 Classes:

1. Class 1: The array is non-empty, and the value v appears multiple times.
2. Class 2: The array is non-empty, and the value v does not appear at all.
3. Class 3: The array has only one element, and that element is equal to v.
4. Class 4: The array has only one element, and that element is not equal to v.
5. Class 5: The array is non-empty, and all elements are equal to v.
6. Class 6: The array is empty.
7. Class 7: The array or value v is of an invalid type.

Test cases :

Test Case	Input v	Input a[]	Valid/Invalid
TC1	2	{1, 2, 2, 2, 3}	Valid
TC2	5	{1, 2, 3, 4}	Valid
TC3	2	{2}	Valid
TC4	1	{2}	Valid
TC5	7	{7, 7, 7, 7}	Valid
TC6	3	{}	Invalid
TC7	a'	{1, 2, 3}	Invalid
TC8	2	{2.5, 1, 2}	Invalid

Problem-3 :

```
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 Classes:

1. Class 1: The array is non-empty, and the value v exists in the array.
2. Class 2: The array is non-empty, and the value v does not exist in the array.
3. Class 3: The array contains only one element, and that element is equal to v.
4. Class 4: The array contains only one element, and that element is not equal to v.
5. Class 5: The array is empty.
6. Class 6: The array is not sorted in non-decreasing order.
7. Class 7: The array or value v is of an invalid type.

Test Case	Input v	Input a[]	Valid/Invalid
TC1	4	{1, 2, 3, 4, 5, 6}	Valid
TC2	7	{1, 2, 3, 4, 5, 6}	Valid
TC3	2	{2}	Valid
TC4	3	{2}	Valid
TC5	4	{}	Invalid
TC6	4	{5, 4, 3, 2, 1}	Invalid
TC7	a'	{1, 2, 3, 4, 5}	Invalid
TC8	3	{1, 2.5, 3}	Invalid

Problem-4 :

```
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 Classes:

1. Class 1: The triangle is equilateral (all three sides are equal).

2. Class 2: The triangle is isosceles (two sides are equal, one is different).
3. Class 3: The triangle is scalene (all three sides are different).
4. Class 4: The sides do not satisfy the triangle inequality (a + b <= c or similar).
5. Class 5: One or more sides are non-positive (a <= 0, b <= 0, or c <= 0).

Test cases:

Test Case	Input (a, b, c)	Valid/Invalid
TC1	(3, 3, 3)	Valid
TC2	(5, 5, 8)	Valid
TC3	(4, 5, 6)	Valid
TC4	(10, 3, 3)	Invalid
TC5	(0, 4, 5)	Invalid
TC6	(-1, 2, 2)	Invalid
TC7	(1, 1, 2)	Invalid
TC8	(2, 2, 5)	Invalid
TC9	(1, 2, 3)	Invalid
TC10	(3, 3, 1)	Valid

Problem-5 :

```
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 Classes:

1. Class 1: s1 is a prefix of s2.
2. Class 2: s1 is not a prefix of s2 (but s1.length() <= s2.length()).s1.
3. Class 3: s1 is an empty string.
4. Class 4: s1 is longer than s2.

Test cases:

Test Case	Input s1	Input s2	Valid/Invalid
TC1	"pre"	"prefix"	Valid
TC2	"fix"	"prefix"	Valid
TC3	""	"prefix"	Valid
TC4	"longer"	"short"	Invalid
TC5	"pre"	"pre"	Valid
TC6	"abc"	"a"	Invalid

Problem-6 :

(a) Equivalence classes :

Valid triangle types :

- ➔ Equilateral: All sides are equal (A = B = C).
- ➔ *Isosceles: Exactly two sides are equal (A = B ≠ C or A = C ≠ B or B = C ≠ A).*
- ➔ *Scalene: All sides are different (A ≠ B ≠ C).*
- ➔ Right-angled: Follows Pythagorean theorem ($A^2 + B^2 = C^2$).

Invalid triangle types

- ➔ Not a triangle: When the sum of any two sides is not greater than *the third (A + B ≤ C or A + C ≤ B or B + C ≤ A).*
- ➔ Non-positive input: At least one side is less than or equal to zero.

(b) Identify Test Cases :

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	Reason
1	5.0, 5.0, 5.0	Equilateral Triangle	-
2	5.0, 5.0, 8.0	Isosceles Triangle	-
3	5.0, 6.0, 7.0	Scalene Triangle	-
4	3.0, 4.0, 5.0	Right-Angle Triangle	-
5	1.0, 2.0, 10.0	Invalid	Not Possible
6	-1.0, 3.0, 4.0	Invalid	Negative Values
7	0.0, 0.0, 0.0	Invalid	Zero values

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

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	(2.0, 3.0, 5.0)	Invalid	Boundary where A + B = C
2	2.0, 3.0, 4.9999	Scalene	Slightly less than A + B = C

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

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	4.0, 6.0, 4.0	Iso Scalene	Boundary where A = C
2	4.0, 6.0, 3.9999	Scalene	Slightly less than A = C

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

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	3.0, 3.0, 3.0	Equilateral	Perfect boundary for equilateral
2	3.0, 3.0, 3.001	Isosceles	Slightly more than equilateral

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

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	3.0, 4.0, 5.0	Right Angled	Perfect boundary for right-angle
2	3.0, 4.0, 5.001	Scalene	Slightly more than right-angle

(g) Boundary Condition for Non-Triangle :

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	1.0, 1.0, 2.0	Invalid	A + B = C
2	1.0, 1.0, 2.1	Scalene	Slightly greater than invalid

(h) Non-Positive Input Test Points :

No of Test Cases	Tester Inputs (A,B,C)	Expected Output	
1	-1.0, 2.0, 3.0	Invalid	A < 0
2	2.0, -1.0, 3.0	Invalid	B < 0
3	2.0, 3.0, 0.0	Invalid	C = 0