IT-314 Software Engineering

Lab 8

Functional Testing (Black-Box)

ID:-202201176

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.

Answer:

Equivalence Class Partitioning Test Case

Input	Expected Outcome	Reasoning
15, 5, 2000	valid	A typical valid date within the allowable ranges.
29, 2, 2000	valid	February 29 is valid in leap years like 2000.
32, 5, 2000	Invalid	May has only 31 days, so 32 is invalid.
31, 6, 2010	Invalid	June has only 30 days, so 31 is invalid.
1, 1, 2016	Invalid	Year exceeds the upper boundary (2015).
15, 8, 1899	Invalid	Year is below the lower boundary (less than 1900).
28, 2, 2001	valid	February 28 is valid in a non-leap year like 2001.
31, 12, 2015	valid	End of year date at upper boundary of 2015.
30, 2, 2000	Invalid	February never has 30 days, even in a leap year.

15, 11, 2010	Valid	A typical valid date in November, which has 30 days.

Boundary Value Analysis Test Cases

Input	Expected Outcome	Reasoning
1, 1, 1900	valid	Lower boundary for day, month, and year.
31, 12, 2015	valid	Upper boundary for day, month, and year.
1, 3, 2000	valid	Leap year boundary case (Feb 29 → March 1).
0, 1, 2000	Invalid	Day below lower boundary (less than 1).
32, 1, 2000	Invalid	Day above upper boundary (greater than 31 in January).
1, 0, 2000	Invalid	Month below lower boundary (less than 1).
1, 13, 2000	Invalid	Month above upper boundary (greater than 12).
31, 4, 2000	Invalid	April has only 30 days, so 31 is an invalid boundary day.
29, 2, 2001	Invalid	February 29 is invalid in a non-leap year.
28, 2, 1900	valid	February 28 is valid in 1900, which is not a leap year (divisible by 100).

Code:-

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

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

int daysInMonth(int month, int year) {
    if (month == 2) {
        return isLeap(year) ? 29 : 28;
    } else if (month == 4 || month == 6 || month == 9 || month == 11) {
        return 30;
```

```
    } else {
        return 31;
    }
}

int main() {
    int day, month, year;

    day = 4; month = 5; year = 1900;
    // Input validation
    if (month < 1 || month > 12 || day < 1 || year < 1900 || year > 2015)
{
        cout << "Invalid Date" << endl;
    } else {
        int maxDaysInMonth = daysInMonth(month, year);
        if (day > maxDaysInMonth) {
            cout << "Invalid Date" << endl;
        } else {
            if (day > 1) {
                cout << day - 1 << ", " << month << ", " << year << endl;
            } else if (month == 1) {
                cout << 31 << ", " << 12 << ", " << (year - 1) << endl;
            } else {
                int prevMonth = month - 1;
                int lastDayOfPrevMonth = daysInMonth(prevMonth, year);
                cout << lastDayOfPrevMonth << ", " << prevMonth << ", " <<
year << endl;
            }
        }
    }

    return 0;
}
```

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);
    }
```

Answer:-

Value Present:
- E1: The array contains the value v, and it appears exactly one time.
- E2: The value v is present in the array and shows up multiple times.
- E3: The value v is not present in the array.

Array Edge Cases:
- E4: The array is empty.
- E5: The value v is located either at the beginning or at the end of the array.

Equivalence Classes:

Test Case	Input	Expected output	Equivalence Boundary
TC1	v=10, [10,20,30,40,50]	0	E1
TC2	v=7, [5,7,7,8,9]	1	E2
TC3	v=0, [1,2,3,4,5]	-1	E3
TC4	v=4, []	-1	E4
TC5	v=50, [10,20,30,40,50]	4	E5

Boundary Points

BP1: The array has only one element, and that element is v.
BP2: The array has just one element, but it is not v.
BP3: The value v appears at the start of the array.
BP4: The value v is located at the end of the array.
BP5: The array includes negative numbers, and v is a negative number.

Test Case	Input	Expected output	Equivalence Boundary
BP1	v=7, [7]	0	BP1
BP2	v=4, [5]	-1	BP2
BP3	v=2, [2,4,6,8,10]	0	BP3
BP4	v=10, [1,3,5,7,10]	4	BP4
BP5	v=-2, [-6,-4,-2,0,2]	2	BP5

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);
}
```

Answer:-

Value Present:
E1: The array contains the value v, and it appears exactly one time.
E2: The value v is present in the array and shows up multiple times.
E3: The value v is not present in the array.

Array Edge Cases:
E4: The array is empty.
E5: The value v is located either at the beginning or at the end of the array.

Equivalence Classes:

Test Case	**Input**	**Expected output**	**Equivalence Boundary**
TC1	v=10, [1, 10, 2, 3]	1	E1

TC2	v=4, [4, 4, 4, 4, 5]	4	E2
TC3	v=7, [1, 2, 3, 5, 6]	0	E3
TC4	v=8, []	0	E4
TC5	v=3, [3, 2, 1, 4, 5]	1	E5

Boundary Points

BP1: The array has only one element, and that element is v.
BP2: The array has just one element, but it is not v.
BP3: The value v appears at the start of the array.
BP4: The value v is located at the end of the array.
BP5: The array includes negative numbers, and v is a negative number.

Test Case	**Input**	**Expected output**	**Equivalence Boundary**
BP1	v=4, [4, 5, 6]	1	BP1
BP2	v=0, [-1, 0, 1]	0	BP2
BP3	v=2, [1, 2, 3]	1	BP3
BP4	v=6, [1, 2, 3, 4, 5, 6]	1	BP4
BP5	v=-4, [-5, -4, -3, -2]	1	BP5

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);
}
```

Answer:-

Equivalence Classes:

Value Found:

- **E1:** The target value v exists in the array and is positioned at the first index.
- **E2:** The target value v exists in the array and is positioned at the last index.
- **E3:** The target value v exists in the array and is located somewhere in the middle.

Value Not Found:

- **E4:** The target value v is less than the smallest element in the array.

E5: The target value v is greater than the largest element in the array.
E6: The target value v is absent from the array but lies within the range of two adjacent elements.

Edge Cases for the Array:

E7: The array is empty (contains no elements).
E8: The array contains a single element, which may or may not match the target value v.

Equivalence Classes:

Test Case	**Input**	**Expected output**	**Equivalence Boundary**
TC1	v=2, [2, 4, 6, 8, 10]	0	E1
TC2	v=10, [2, 4, 6, 8, 10]	4	E2
TC3	v=6, [2, 4, 6, 8, 10]	2	E3
TC4	v=1, [2, 4, 6, 8, 10]	-1	E4
TC5	v=12, [2, 4, 6, 8, 10]	-1	E5
TC6	v=5, [2, 4, 6, 8, 10]	-1	E6
TC7	v=3, []	-1	E7
TC8	v=4, [4]	0	E8
TC9	v=5, [4]	-1	E9

Boundary Points

BP1: An array with one element where the target value v matches that element.
BP2: An array with one element where the target value v does not match the element.
BP3: The target value v is located at the first index in a sorted array with multiple elements.
BP4: The target value v is located at the last index in a sorted array with multiple elements.
BP5: The array contains several occurrences of the target value v.

Test Case	Input	Expected output	Equivalence Boundary
BP1	v=4, [4]	0	BP1
BP2	v=5, [4]	-1	BP2
BP3	v=1, [1, 2, 3, 4, 5]	0	BP3
BP4	v=5, [1, 2, 3, 4, 5]	4	BP4
BP5	v=2, [1, 2, 2, 3, 4, 5]	1	BP5

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);
    }
```

Answer:

EP1: Invalid triangle (one or more sides have zero or negative length) → Output: INVALID
EP2: Invalid triangle (sum of any two sides is not greater than the third side) → Output: INVALID
EP3: Equilateral triangle (all three sides are of equal length) → Output: EQUILATERAL
EP4: Isosceles triangle (two sides are of equal length, while the third is different) → Output: ISOSCELES
EP5: Scalene triangle (all sides have different lengths) → Output: SCALENE

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	-1, 2, 3	INVALID	EP1 (Invalid triangle: side length is negative)
TC2	6, 2, 3	INVALID	EP2 (Invalid triangle: violates triangle inequality)
TC3	7, 7, 7	EQUILATERAL	EP3 (Equilateral triangle: all sides are the same)
TC4	6, 6, 4	ISOSCELES	EP4 (Isosceles triangle: two sides are equal)
TC5	5, 7, 9	SCALENE	EP5 (Scalene triangle: all sides are different)

Boundary value analysis

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	2, 2, 2	EQUILATERAL	BVA1 (Minimum valid sides for an equilateral triangle)
TC2	2, 2, 5	INVALID	BVA2 (Invalid triangle: sum of two sides not greater than the third)
TC3	2, 3, 3	ISOSCELES	BVA3 (Minimum valid sides for an isosceles triangle)
TC4	1, 3, 3	ISOSCELES	BVA4 (Valid isosceles triangle with smallest integer values)
TC5	4, 5, 7	SCALENE	BVA5 (Valid scalene triangle with mixed sides)
TC6	-2, 2, 2	INVALID	BVA6 (Invalid triangle: one side is negative)
TC7	0, 4, 4	INVALID	BVA7 (Invalid triangle: contains a non-positive side)
TC8	5, 5, 10	INVALID	BVA8 (Invalid triangle: sum of two sides not greater than the third)

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;
}
```

Answer:

Valid Prefix Cases:

E1: The string s1 is a non-empty string that serves as a prefix of the string s2.
E2: The string s1 is empty, which qualifies it as a prefix for any non-empty string s2.
E3: The string s1 is identical to the string s2.

Invalid Prefix Cases:

E4: The string s1 has a greater length than s2.

E5: The string s1 does not match the beginning of s2(they diverge at some point).

Equivalence Classes

Test Case	Input	Expected Output	Equivalence Class
TC1	s1 = "pre", s2 = "prefix"	true	E1
TC2	s1 = "", s2 = "anything"	true	E2
TC3	s1 = "test", s2 = "test"	true	E3
TC4	s1 = "longer", s2 = "shorter"	false	E4
TC5	s1 = "prefix", s2 = "pre"	false	E5
TC6	s1 = "abc", s2 = "abcdef"	true	E1
TC7	s1 = "xyz", s2 = "abcxyz"	false	E5
TC8	s1 = "test", s2 = "testing"	true	E1

Boundary Points :

BP1: The string s1 consists of a single character that matches the prefix of the string s2.
BP2: The string s1 is a single character that does not match the beginning of the string s2.
BP3: The string s1 is empty, while s2 contains one or more characters.
BP4: The string s1 is identical to s2, with both strings containing a single character.
BP5: The string s1 matches the initial segment of s2 but does not encompass the entire string s2.

Boundary Points

Test Case	Input	Expected Output	Boundary Point
BP1	s1 = "p", s2 = "prefix"	true	BP1
BP2	s1 = "z", s2 = "prefix"	false	BP2
BP3	s1 = "", s2 = "hello"	true	BP3
BP4	s1 = "x", s2 = "x"	true	BP4
BP5	s1 = "prefix", s2 = "prefix_test"	true	BP5
BP6	s1 = "ab", s2 = "abcd"	true	BP1
BP7	s1 = "c", s2 = "abc"	true	BP1
BP8	s1 = "d", s2 = "abc"	false	BP2

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.

Answer:-

Valid Triangle:

- Equilateral Triangle (E1): All sides are equal: A=B=C
- Isosceles Triangle (E2): Two sides are equal: A=B, B=C, or C=A
- Scalene Triangle (E3): All sides are different: A≠B, B≠C, A≠C
- Right-angled Triangle (E4): Follows the Pythagorean theorem A 2+B 2=C 2 with A ≤ B ≤ C

Invalid Triangle:

- Non-Triangle Case (I1): Sum of two sides is less than or equal to the third side: A + B ≤ C or A + C ≤ B or B + C ≤ A
- Non-Positive Inputs(I2):One or more sides have non-positive values: A≤ 0 ,B ≤ 0,C ≤ 0

b) Identify test cases to cover the identified equivalence classes.

Test Case	Input	Expected output	Equivalence Class
TC1	6, 6, 2	Isosceles	E2
TC2	4, 4, 4	Equilateral	E1
TC3	7, 5, 3	Scalene	E3
TC4	8, 15, 17	Right Angled	E4
TC5	2, 2, 5	Invalid	I1
TC6	0, 5, 5	Invalid	I2
TC7	-3, 3, 4	Invalid	I2
TC8	10, 10, 20	Invalid	I1
TC9	5, 12, 13	Scalene	E3
TC10	9, 9, 12	IsosceleS	E2

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

Test Case	**Input**	**Expected Output**	**Boundary Condition**
BC1	5, 6, 11	Invalid	(A + B = C)
BC2	4, 5, 8	Scalene	(A + B > C)
BC3	3, 4, 8	Invalid	(A + B = C)

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

Test Case	Input	Expected Output	Boundary Condition
BC4	7, 7, 4	Isosceles	(A = B)
BC5	5, 8, 5	Isosceles	(A = C)
BC6	9, 2, 9	Isosceles	(B = C)
BC7	5, 5, 10	Invalid	(A + B = C)

e) For the boundary condition A = B = C case (equilateral triangle).

Test Case	Input	Expected Output	Boundary Condition
BC8	3.5, 3.5, 3.5	Equilateral	(A = B = C)
BC9	6.1, 6.1, 6.1	Equilateral	(A = B = C)

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle).

Test Case	Input	Expected Output	Boundary Condition
BC10	5, 12, 13	Right Angled	(A^2 + B^2 = C^2)
BC11	9, 12, 15	Right Angled	(A^2 + B^2 = C^2)
BC12	8, 15, 17	Right Angled	(A^2 + B^2 = C^2)

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

Test Case	Input	Expected Output	Boundary Condition
BC13	1, 1, 2	Invalid	(A + B = C)
BC14	2, 3, 6	Invalid	(A + B < C)
BC15	4, 4, 8	Invalid	(A + B < C)

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

Test Case	Input	Expected Output	Boundary Condition
BC16	0, 1, 1	Invalid	(A = 0)
BC17	-1, 2, 3	Invalid	(A = -1)
BC18	5, 0, 5	Invalid	(B = 0)
BC19	4, 3, -2	Invalid	(C = -2)