IT314 - Software Engineering

Lab - 8

Functional Testing (Black-Box)

Prof. Saurabh Tiwari
ID: 202201142

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?

Valid Input Classes (EP1):

1. **Valid days for months** (e.g., 30 days in April, 31 days in July, 28 or 29 days in February).
2. **Valid months**: (1 to 12).
3. **Valid years**: (1900 to 2015).

Invalid Input Classes (EP2):

1. **Invalid days**: e.g., day < 1 or day > 31.
2. **Invalid months**: month < 1 or month > 12.
3. **Invalid years**: year < 1900 or year > 2015.
4. **Invalid date combinations**: e.g., February 30, April 31.

Boundary Value Analysis (BVA)

Month boundaries:
- Test with months like 1 and 12 (valid) and just outside, like 0 and 13 (invalid).

Day boundaries:
- Test with days at the beginning (1) and end (31) and just beyond these values, like 0 and 32.

Year boundaries:
- Test with years 1900 and 2015 (valid), and years just below and above this rang

Test Case	Input (Day, Month, Year)	Test Type	Expected Outcome	Reason
Case 1	(15, 6, 2005)	Equivalence Partitioning	(14, 6, 2005)	Valid mid-range values for day, month, and year.
Case 2	(1, 3, 2000)	Equivalence Partitioning	(29, 2, 2000)	Valid leap year transition (March 1 to Feb 29).
Case 3	(1, 1, 1900)	Equivalence Partitioning	(31, 12, 1899)	Valid start boundary for year.
Case 4	(31, 12, 2015)	Equivalence Partitioning	(30, 12, 2015)	Valid end boundary for year.
Case 5	(32, 1, 2005)	Equivalence Partitioning	Error (Invalid day)	Day exceeds valid range for January.
Case 6	(29, 2, 1900)	Equivalence Partitioning	Error (Invalid date)	February 29 does not exist in the year 1900 (non-leap year).
Case 7	(15, 13, 2005)	Equivalence Partitioning	Error (Invalid month)	Month exceeds valid range (1-12).
Case 8	(15, 6, 2016)	Equivalence Partitioning	Error (Year out of range)	Year exceeds upper boundary (2015).
Case 9	(1, 1, 2000)	Boundary Value Analysis	(31, 12, 1999)	Transition from Jan 1 to Dec 31 of the previous year.
Case 10	(1, 12, 2000)	Boundary Value Analysis	(30, 11, 2000)	Transition from Dec 1 to Nov 30.
Case 11	(31, 12, 2015)	Boundary Value Analysis	(30, 12, 2015)	Valid boundary for end of year 2015.
Case 12	(1, 13, 2005)	Boundary Value Analysis	Error (Invalid month)	Invalid boundary for month (13 out of range).

Case 13	(31, 1, 2000)	Boundary Value Analysis	(30, 1, 2000)	Transition within January.
Case 14	(1, 2, 2000)	Boundary Value Analysis	(31, 1, 2000)	Transition from Feb 1 to Jan 31.
Case 15	(29, 2, 2000)	Boundary Value Analysis	(28, 2, 2000)	Leap year boundary in February.
Case 16	(0, 1, 2000)	Boundary Value Analysis	Error (Invalid day)	Day below valid range.
Case 17	(32, 1, 2000)	Boundary Value Analysis	Error (Invalid day)	Day exceeds valid range.
Case 18	(1, 1, 1900)	Boundary Value Analysis	(31, 12, 1899)	Boundary at start of valid year range.
Case 19	(31, 12, 2015)	Boundary Value Analysis	(30, 12, 2015)	Boundary at end of valid year range.
Case 20	(15, 6, 1899)	Boundary Value Analysis	Error (Year below range)	Year is below valid boundary.

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 Partitioning:

1. The value v is present in the array.
2. The value v is absent from the array.
3. The array is empty.
4. The value v appears multiple times in the array.

Boundary Value Analysis:

1. Conduct a test with an array containing only one element.
2. Conduct a test with an array of larger size.
3. Conduct a test when v is located at either the first or last position in the array.

Test Case	Input (v, a[])	Test Type	Expected Outcome	Reason
Case 1	(5, [1, 2, 3, 4, 5])	Equivalence Partitioning	4	Array contains v at index 4.
Case 2	(3, [1, 2, 3, 4, 5])	Equivalence Partitioning	2	Array contains v at index 2.

Case 3	(6, [1, 2, 3, 4, 5])	Equivalence Partitioning	-1	Array does not contain v.
Case 4	(3, [])	Equivalence Partitioning	-1	Array is empty, so v cannot be found.
Case 5	(3, [3, 1, 3, 4, 5])	Equivalence Partitioning	0	Array contains v multiple times, first occurrence at index 0.
Case 6	(5, [5])	Boundary Value Analysis	0	Single element array where v is the first and only element.
Case 7	(10, [5, 6, 7, 8, 9, 10])	Boundary Value Analysis	5	v is at the last index of the array.
Case 8	(1, [1, 2, 3, 4, 5])	Boundary Value Analysis	0	v is at the first index of the array.
Case 10	(7, [-5, -3, 0, 7, 10])	Boundary Value Analysis	3	Array contains negative, zero, and positive values; v is present.
Case 11	(-3, [-5, -3, 0, 7, 10])	Boundary Value Analysis	1	v is a negative number in the array.
Case 12	(100, [5, 10, 20, 30, 40])	Boundary Value Analysis	-1	v is much larger than any element in the array.
Case 13	(-100, [5, 10, 20, 30, 40])	Boundary Value Analysis	-1	v is much smaller than any element in the array.

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:

- The value v is absent from the array.
- The value v is present exactly once in the array.
- The value v occurs multiple times within the array.
- The array is completely empty.

Boundary Value Testing:

- The array contains just a single element.
- The array has a larger number of elements.
- The value v is positioned at either the first or last element of the array.

Test Case	Input (v, a[])	Test Type	Expected Outcome	Reason
Case 1	(5, [1, 2, 3, 4, 5])	Equivalence Partitioning	1	Array contains v exactly once at the last index.
Case 2	(3, [1, 2, 3, 4, 5])	Equivalence Partitioning	1	Array contains v exactly once at the middle.
Case 3	(6, [1, 2, 3, 4, 5])	Equivalence Partitioning	0	Array does not contain v.
Case 4	(3, [])	Equivalence Partitioning	0	Array is empty, so v cannot be found.

Case 5	(3, [3, 1, 3, 4, 5])	Equivalence Partitioning	2	Array contains v multiple times (two occurrences).
Case 6	(5, [5])	Boundary Value Analysis	1	Single element array where v is the first and only element.
Case 7	(3, [3, 3, 3, 3, 3])	Boundary Value Analysis	5	Array contains v at all positions (5 occurrences).
Case 8	(10, [5, 6, 7, 8, 9, 10])	Boundary Value Analysis	1	v is at the last index of the array.
Case 9	(1, [1, 2, 3, 1, 1])	Boundary Value Analysis	3	v appears multiple times, including the first index.
Case 10	(-3, [-5, -3, 0, 7, 10])	Boundary Value Analysis	1	v is a negative number present in the array.
Case 12	(100, [5, 10, 20, 30, 40])	Boundary Value Analysis	0	v is much larger than any element in the array, not found.

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 Partitioning:

- The value **v** is present in the array.
- The array does **not** include the value **v**.
- The array has no elements.
- The array includes **v** more than once.

Boundary Value Analysis :

- Try an array with only a single element.
- Test when the first or last element equals **v**.
- Verify if **v** equals the element in the center.

Test Case	Input (v, a[])	Test Type	Expected Outcome	Reason
Case 1	(5, [1, 2, 3, 4, 5])	Equivalence Partitioning	4	Array contains v at index 4 (last element).
Case 2	(3, [1, 2, 3, 4, 5])	Equivalence Partitioning	2	Array contains v at index 2 (middle element).
Case 3	(6, [1, 2, 3, 4, 5])	Equivalence Partitioning	-1	Array does not contain v.
Case 4	(3, [])	Equivalence Partitioning	-1	Array is empty, so v cannot be found.
Case 5	(3, [3, 3, 3, 3, 3])	Equivalence Partitioning	2 or 3	Array contains multiple occurrences of v (all elements are v).
Case 6	(5, [5])	Boundary Value Analysis	0	Single element array where v matches the first and only element.
Case 7	(1, [1, 2, 3, 4, 5])	Boundary Value Analysis	0	v is the first element in the array.
Case 8	(10, [5, 6, 7, 8, 9, 10])	Boundary Value Analysis	5	v is the last element in the array.

Case 9	(7, [5, 6, 7, 8, 9, 10])	Boundary Value Analysis	2	v is the middle element of the array.
Case 10	(-3, [-5, -3, 0, 7, 10])	Boundary Value Analysis	1	v is a negative number present at index 1 in the sorted array.
Case 11	(100, [10, 20, 30, 40, 50])	Boundary Value Analysis	-1	v is much larger than any element in the array, not found.
Case 12	(-100, [-50, -30, -10, 0, 20])	Boundary Value Analysis	-1	v is much smaller than any element in the array, not found.

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 Partitioning:

- The three sides are the same length.
- Exactly two sides have the same length, while the third is different.
- All three sides are distinct in length.
- The side lengths violate the triangle inequality condition.

Boundary Value Analysis:

- The smallest possible positive side lengths that still form a triangle (e.g., sides with length 1).
- Use very large numbers to represent the maximum possible side lengths.
- At least one of the sides has a length of zero.
- Negative values for one or more sides.
- Situations where the sum of two sides is exactly equal to the length of the third side.

Test Case	Input (v, a[])	Test Type	Expected Outcome	Reason
Case 1	(3, 3, 3)	Equivalence Partitioning	0	Equilateral
Case 2	(3, 3, 4)	Equivalence Partitioning	1	Isosceles
Case 3	(3, 4, 5)	Equivalence Partitioning	2	Scalene
Case 4	(1, 2, 3)	Equivalence Partitioning	3	inequality fails
Case 5	(0, 0, 0)	Equivalence Partitioning	3	Side cannot be 0
Case 6	(1, 1, 1)	Boundary Value Analysis	0	Equilateral
Case 7	(1, 1, 2)	Boundary Value Analysis	3	inequality fails

Case 8	(3, -1, 4)	Boundary Value Analysis	3	Negative side
Case 9	(3, 5, -2)	Boundary Value Analysis	3	Negative side
Case 10	(2, 2, 5)	Boundary Value Analysis	3	inequality fails
Case 11	(1, 2, 1)	Boundary Value Analysis	3	inequality fails
Case 12	(1000000, 10000000, 10000000)	Boundary Value Analysis	0	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 Partitioning:

- The sequence of characters in s1 perfectly aligns with the beginning of s2.
- The characters in s1 fail to match the initial portion of s2.
- It's impossible for s1 to be a prefix of s2 based on their contents.
- A string that's empty serves as a valid prefix for any other string.
- A non-empty s1 cannot act as a prefix for an empty s2.

Boundary Value Analysis (BVA):

- Special case where both s1 and s2 are empty strings.
- Valid scenario where s1 is an actual prefix of s2.
- Invalid case where s1 is not a prefix of s2.
- Test where s1 and s2 are exactly the same.
- One-character difference between the lengths of s1 and s2 as a test case.

Test Case	Input (s1, s2)	Test Type	Expected Outcome	Reason
Case 1	("pre", "pre")	Equivalence Partitioning	true	s1 is a prefix of s2.
Case 2	("pre", "postfix")	Equivalence Partitioning	false	s1 does not match the start of s2.
Case 3	("prefix", "pre")	Equivalence Partitioning	false	s1 is longer than s2.
Case 4	("", "hello")	Equivalence Partitioning	true	Empty s1 is a prefix of any non-empty s2.
Case 5	("hello", "")	Equivalence Partitioning	false	Non-empty s1 cannot be a prefix of an empty s2.
Case 6	("", "")	Boundary Value Analysis	true	Both strings are empty.
Case 7	("test", "test")	Boundary Value Analysis	true	s1 equals s2

Case 8	("123", "123456")	Equivalence Partitioning	true	s1 is a prefix of s2.
Case 9	("pre", "prefixe ")	Boundary Value Analysis	true	s1 is a prefix of s2.
Case 10	("abc", "abcd")	Boundary Value Analysis	true	s1 is a prefix of s2
Case 11	("abc", "ab")	Boundary Value Analysis	false	s1 is not a prefix of s2.
Case 12	("abcdefg", "abc")	Boundary Value Analysis	false	s1 is longer than s2.

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 $A_2 + B_2 = C_2$ 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 Classes Identification

Valid Triangle Cases

- Equilateral Triangle: All sides are equal, meaning A=B=CA = B = CA=B=C.
- Isosceles Triangle: Two sides have the same length, while the third is different (e.g., A=B≠CA = B \neq CA=B=C).
- Scalene Triangle: Each side has a distinct length, so A≠B≠CA \neq B \neq CA=B=C.
- Right-Angled Triangle: Satisfies the Pythagorean theorem A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (or any permutation of this formula).

Invalid Triangle Cases (Non-Triangle)

- Triangle Inequality Violation: The sum of two sides is less than or equal to the third side (e.g., A+B≤CA + B \leq CA+B≤C).
- Negative or Zero Values: At least one side is zero or negative (e.g., A≤0A \leq 0A≤0).

B. Test Cases for Equivalence Classes

Test Case	Input	Expected Outcome	Reason
TC1	3,3,3	Equilateral	Equilateral Triangle
TC2	5, 5, 2008	Isosceles	Isosceles Triangle
TC3	4, 6, 2007	Scalene	Scalene Triangle
TC4	3, 4, 2005	Right-angled	Right-Angled Triangle
TC5	1, 2, 2003	Not a Triangle	Violates Triangle Inequality
TC6	-3, 4, 5	Invalid Input	Non-positive Input
TC7	2000, 5, 7	Invalid Input	Non-positive Input

C. Boundary Condition: A+B>C (Scalene Triangle)

Test Case	Input (A, B, C)	Expected Result	Remark
TC8	10, 11, 2	2	Scalene triangle
TC9	5, 6, 10	2	Scalene Triangle

D. Boundary Condition: A=C (Isosceles Triangle)

Test Case	Input (A, B, C)	Expected Result	Remarks
TC9	5.1, 5.1, 10	1	Isosceles triangle
TC10	10000, 10000, 1	1	Isosceles triangle

E. Boundary Condition: A=B=C (Equilateral Triangle)

Test Case	Input (A, B, C)	Expected Result	Remarks
TC11	1, 1, 1	0	Equilateral triangle
TC12	1000, 1000, 1000	0	Equilateral triangle

F.Boundary Condition: A2+B2=C2 (Right-angled Triangle)

Test Case	Input (A, B, C)	Expected Result	Remarks
TC13	3, 4, 5	2	Right-angled scalene triangle
TC14	5, 12, 13	2	Right-angled scalene triangle

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

Test Case	Input (A, B, C)	Expected Result	Remarks
TC15	1, 2, 3	3	Invalid triangle
TC16	1, 2, 10	3	Invalid triangle