IT314 : Software Engineering

Lab Session 8 - Functional Testing (Black-Box)

ID : 202201154
Name : Rishi Godhasara
Group no : 2

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?

Equivalence Class Partitioning (EP)

Valid Input Classes

EP1. Valid day for months with 31 days: 1 <= day <= 31 (January, March, May, July, August, October, December)

EP2. Valid day for months with 30 days: 1 <= day <= 30 (April, June, September, November)

EP3. Valid day for February: 1 <= day <= 28 (for non-leap years) and 1 <= day <= 29 (for leap year)

EP4. Valid months (1 to 12).

EP5. Valid year (1900 to 2015).

Invalid Input Classes

EP6. Invalid day for months with 31 days : e.g., day < 1 or day > 31.

EP7. Invalid day for months with 30 days : e.g., day < 1 or day > 30.

EP8. Invalid day for February: day<1 or day>28 (for non-leap years) and day <1 or day>29 (for leap year)

EP9. Invalid months: month < 1 or month > 12.
EP10. Invalid years: year < 1900 or year > 2015.

Boundary Value Analysis (BVA)
- Day boundaries: 1, 2, 30, 31 (or 28/29 for February).
- Month boundaries: 1, 2, 11, 12.
- Year boundaries: 1900, 1901, 2014, 2015.

Equivalence Partitioning (EP) Test Cases

Test Case No.	**Tester Action (Input Data)**	**Expected Outcome**	**Derived From**
TC1	15, 01, 2010	Valid Previous Date (14/01/2010)	**EP1 (Valid day for months with 31 days)**
TC2	29, 04, 2005	Valid Previous Date (28/04/2005)	**EP2 (Valid day for months with 30 days)**
TC3	28, 02, 2003	Valid Previous Date (27/02/2003)	**EP3 (Valid day for February non-leap year)**
TC4	29, 02, 2012	Valid Previous Date (28/02/2012) (Leap Year)	**EP3 (Valid day for February leap year)**
TC5	25, 05, 2015	Valid Previous Date (24/05/2015)	**EP4 (Valid months)**
TC6	32, 12, 2010	Invalid Date (day > 31)	**EP6 (Invalid day for months with 31 days)**
TC7	31, 04, 2010	Invalid Date (day > 30 for April)	**EP7 (Invalid day for months with 30 days)**
TC8	30, 02, 2013	Invalid Date (day > 28 for non-leap year February)	**EP8 (Invalid day for February non-leap year)**
TC9	30, 02, 2012	Invalid Date (day > 29 for leap year February)	**EP8 (Invalid day for February leap year)**
TC10	15, 13, 2015	Invalid Date (month > 12)	**EP9 (Invalid month)**
TC11	15, 00, 2015	Invalid Date (month < 1)	**EP9 (Invalid month)**

Test Case No.	Tester Action (Input Data)	Expected Outcome	Derived From
TC12	15, 05, 1899	Invalid Date (year < 1900)	**EP10 (Invalid year)**
TC13	15, 05, 2016	Invalid Date (year > 2015)	**EP10 (Invalid year)**

Boundary Value Analysis (BVA) Test Cases

Test Case No.	Tester Action (Input Data)	Expected Outcome	Derived From
TC14	1, 1, 1900	Valid Previous Date (31/12/1899)	**BVA (Lower Bound Year, Month, Day)**
TC15	31, 12, 2015	Valid Previous Date (30/12/2015)	**BVA (Upper Bound Year, Month, Day)**
TC16	30, 11, 2015	Valid Previous Date (29/11/2015)	**BVA (Upper Bound Day in 30-day Month)**
TC17	1, 2, 2012	Valid Previous Date (31/01/2012)	**BVA (Lower Bound Day in February)**
TC18	1, 3, 2015	Valid Previous Date (28/02/2015)	**BVA (Upper Bound Day in February non-leap year)**
TC19	29, 2, 2012	Valid Previous Date (28/02/2012) (Leap Year)	**BVA (Upper Bound Day in February leap year)**
TC20	0, 1, 2000	Invalid Date (day < 1)	**BVA (Day Below Lower Bound)**
TC21	1, 1, 2016	Invalid Date (year > 2015)	**BVA (Year Beyond Upper Bound)**

```cpp
#include <iostream>
#include <string>

using namespace std;

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

int getDaysInMonth(int month, int year) {
    if (month == 2) {
        return isLeapYear(year) ? 29 : 28;
    } else if (month == 4 || month == 6 || month == 9 || month == 11) {
        return 30;
    } else {
        return 31;
    }
}

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

    // Example input
    day = 1; month = 1; year = 1900; // Replace with any test case values
    // Check for invalid inputs
    if (month < 1 || month > 12 || day < 1 || year < 1900 || year > 2015)
{
        cout << "Invalid Date" << endl;
    } else {
        int maxDays = getDaysInMonth(month, year);
        if (day > maxDays) {
            cout << "Invalid Date" << endl;
        } else {
            // Compute previous date
            if (day > 1) {
                cout << day - 1 << ", " << month << ", " << year << endl;
// Previous day in same month
```

```
            } else if (month == 1) {
                cout << 31 << ", " << 12 << ", " << (year - 1) << endl; //
            } else {
                int previousMonth = month - 1;
                int previousDay = getDaysInMonth(previousMonth, year);
                cout << previousDay << ", " << previousMonth << ", " <<
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);
}
```

Equivalence Class Partitioning (EP)

EP1: The value v is present in the array and occurs once.
EP2: The value v is present in the array and occurs multiple times.

EP3: The value v is not present in the array.
EP4 : The array is empty.

Boundary Value Analysis (BVA)

BVA1:Single-element array where v is present.
BVA2:Single-element array where v is not present.
BVA3:v is at the first position.
BVA4:v is at the last position.
BVA5:Array contains negative numbers and v is a negative number.

Test Cases for Linear Search Function

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	5, [1, 2, 3, 5, 4]	3 (value found once at index 3)	EP1 (value present, occurs once)
TC2	5, [5, 1, 2, 5, 4]	0 (value found first at index 0)	EP2 (value present, occurs multiple times)
TC3	10, [1, 2, 3, 4]	-1 (value not found)	EP3 (value not present)
TC4	5, []	-1 (empty array)	EP4 (array is empty)

Boundary Value Analysis (BVA) Test Cases

Test Case No.	Input Data	Expected Outcome	Derived From
BVA1	5, [5]	0 (value found at index 0)	Single-element array where v is present
BVA2	5, [1]	-1 (value not found)	Single-element array where v is not present
BVA3	1, [1, 2, 3, 4]	0 (value found at first position)	v is at the first position
BVA4	4, [1, 2, 3, 4]	3 (value found at last position)	v is at the last position
BVA5	-5, [-5, -1, 0, 1]	0 (value found at index 0)	Array contains negative numbers and v is a negative number

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 Classes

EP1: Empty array ⟶ Output: 0
EP2: Value exists once ⟶ Output: 1

EP3: Value exists multiple times ⟶ Output: Count of occurrences(e.g., n if v appears n times)

EP4: Value does not exist ⟶ Output: 0

EP5: All elements are equal to v ⟶ Output: Length of the array

Equivalence Class Partitioning (EP) Test Cases

Test Case No.	**Input Data**	**Expected Outcome**	**Derived From**
TC1	5, []	0 (Empty array)	**EP1 (Empty array)**
TC2	5, [1, 2, 3, 5, 4]	1 (Value exists once)	**EP2 (Value exists once)**
TC3	5, [5, 1, 2, 5, 4]	2 (Value exists multiple times)	**EP3 (Value exists multiple times)**
TC4	10, [1, 2, 3, 4]	0 (Value does not exist)	**EP4 (Value does not exist)**
TC5	1, [1, 1, 1, 1]	4 (All elements are equal to v)	**EP5 (All elements are equal to v)**

Boundary Value Analysis for `countItem`

Test Case No.	**Input Values (v, a)**	**Expected Output**	**Boundary Condition**
TC1	1, [1]	1	Single-element array with matching value
TC2	2, [1]	0	Single-element array with non-matching value
TC3	1, []	0	Empty array
TC4	1, [1, 1, 1]	3	Multiple elements, all matching
TC5	2, [1, 1, 1]	0	Multiple elements, none matching
TC6	1, [1, 2, 3]	1	Multiple elements, one matching
TC7	2, [1, 2, 3]	1	Multiple elements, one matching

Test Case No.	**Input Values (v, a)**	**Expected Output**	**Boundary Condition**
TC8	2, [2, 2, 2, 2, 2]	5	Multiple elements, all matching
TC9	2, [1, 1, 1, 1, 1]	0	Multiple elements, none matching
TC10	1, [0, 1, 2, 3, 4]	1	Includes zero, one matching

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 Classes for binarySearch:

1. Value Present

E1:The value v is present in the array and is located at the first position.
E2:The value v is present in the array and is located at the last position.
E3:The value v is present in the array and is located somewhere in the middle.

2. Value Not Present:
E4:The Value Is Less than the smallest element in the array.
E5:The value v is greater than the largest element in the array.
E6:The value v is not in the array but falls between two elements.

3. Array EdgeCases:
E7:The Array Isempty.
E8:The Array Contains one element, which may or maynot be equal to v.

Equivalence Classes for binarySearch

Test Case No.	**Input Data**	**Expected Outcome**	**Derived From**
TC1	`v`, `[v, 2, 3, 4]`	Index `0` (Value present at the first position)	**E1 (Value Present - First Position)**
TC2	`4`, `[1, 2, 3, 4]`	Index `3` (Value present at the last position)	**E2 (Value Present - Last Position)**
TC3	`2`, `[1, 2, 3, 4]`	Index `1` (Value present in the middle)	**E3 (Value Present - Middle Position)**
TC4	`0`, `[1, 2, 3, 4]`	`-1` (Value is less than the smallest element)	**E4 (Value Not Present - Less than Minimum)**
TC5	`5`, `[1, 2, 3, 4]`	`-1` (Value is greater than the largest element)	**E5 (Value Not Present - Greater than Maximum)**
TC6	`2.5`, `[1, 2, 3, 4]`	`-1` (Value is not present but falls between two elements)	**E6 (Value Not Present - Between Two Elements)**

Test Case No.	Input Data	Expected Outcome	Derived From
TC7	`5, []`	`-1` (Array is empty)	**E7 (Array Edge Case - Empty Array)**
TC8	`5, [v]`	Index `0` (Array contains one element equal to v)	**E8 (Array Edge Case - One Element)**
TC9	`1, [v]`	`-1` (Array contains one element not equal to v)	**E8 (Array Edge Case - One Element)**

Boundary Points for binarySearch:

1. BP1: Single-element array where v is equal to the element.
2. BP2: Single-element array where v is not equal to the element.
3. BP3: The value v is at the first position in a multi-element sorted array.
4. BP4: The value v is at the last position in a Multi-element sorted array.
5. BP5: The Array Contains Duplicate Values Of v.

Boundary Points for binarySearch

Test Case No.	Input Data	Expected Outcome	Derived From
BP1	`v, [v]`	Index `0` (Single-element array, v is equal to the element)	**BP1 (Single-element Array - Equal)**
BP2	`v, [x]`	`-1` (Single-element array, v is not equal to the element)	**BP2 (Single-element Array - Not Equal)**
BP3	`1, [1, 2, 3, 4]`	Index `0` (Value v is at the first position)	**BP3 (First Position in Multi-element Array)**

Test Case No.	Input Data	Expected Outcome	Derived From
BP4	4, `[1, 2, 3, 4]`	Index 3 (Value v is at the last position)	**BP4 (Last Position in Multi-element Array)**
BP5	2, `[1, 2, 2, 2, 3]`	Index 1 (Value v exists in an array with duplicate values)	**BP5 (Array Contains Duplicates)**

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 lengthsequal), 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 Classes :

EP1: Invalid triangle (non-positive sides) ⟶ Output: INVALID
Ep2: Invalid triangle (triangle inequality not satisfied) ⟶ Output: INVALID
EP3: Equilateral triangle (all sides equal) ⟶ Output: EQUILATERAL
EP4: Isosceles triangle (two sides equal) ⟶ Output: ISOSCELES
EP5: Scalene triangle (all sides different) ⟶ Output: SCALENE

Equivalence Class Test Cases for `triangle`

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	`0, 0, 0`	`INVALID`	**EP1 (Invalid triangle: non-positive sides)**
TC2	`3, 4, 8`	`INVALID`	**EP2 (Invalid triangle: triangle inequality not satisfied)**
TC3	`5, 5, 5`	`EQUILATERAL`	**EP3 (Equilateral triangle)**
TC4	`3, 3, 4`	`ISOSCELES`	**EP4 (Isosceles triangle)**
TC5	`3, 4, 5`	`SCALENE`	**EP5 (Scalene triangle)**

Boundary value analysis

Boundary Value Analysis Test Cases for `triangle`

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	`1, 1, 1`	`EQUILATERAL`	**BVA1 (Minimum positive sides: equilateral)**
TC2	`1, 1, 2`	`INVALID`	**BVA2 (Invalid triangle: non-positive sides)**
TC3	`1, 2, 2`	`ISOSCELES`	**BVA3 (Minimum sides for isosceles)**

Test Case No.	Input Data	Expected Outcome	Derived From
TC4	`1, 2, 3`	`INVALID`	**BVA4 (Invalid triangle: triangle inequality)**
TC5	`3, 4, 5`	`SCALENE`	**BVA5 (Valid scalene triangle)**
TC6	`0, 5, 5`	`INVALID`	**BVA6 (Invalid triangle: non-positive sides)**
TC7	`2, 3, 4`	`SCALENE`	**BVA7 (Valid scalene triangle)**

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 Classes :

EP1: s1 is longer than s2 ⟶ Output: false
EP2: s1 is an exact prefix of s2 ⟶ Output: true
EP3: s1 is a partial prefix of s2 ⟶ Output: false
EP4: s1 is empty ⟶ Output: true
EP5: s2 is empty and s1 is not ⟶ Output: false
EP6: s1 is equal to s2 ⟶ Output: true

Equivalence Classes for Prefix Checking

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	`s1: "hello", s2: "hello world"`	`true`	**EP2 (s1 is an exact prefix of s2)**
TC2	`s1: "hello", s2: "hello"`	`true`	**EP6 (s1 is equal to s2)**
TC3	`s1: "hello", s2: "hell"`	`false`	**EP1 (s1 is longer than s2)**
TC4	`s1: "hello", s2: "world"`	`false`	**EP3 (s1 is a partial prefix of s2)**
TC5	`s1: "", s2: "hello"`	`true`	**EP4 (s1 is empty)**
TC6	`s1: "hello", s2: ""`	`false`	**EP5 (s2 is empty and s1 is not)**
TC7	`s1: "hello world", s2: "hello world"`	`true`	**EP6 (s1 is equal to s2)**

Boundary value analysis

Boundary Value Analysis for Prefix Checking

Test Case No.	Input Data	Expected Outcome	Derived From
TC1	`s1: "a", s2: "a"`	`true`	**BVA (Single character, both equal)**
TC2	`s1: "a", s2: "b"`	`false`	**BVA (Single character, not equal)**
TC3	`s1: "a", s2: ""`	`false`	**BVA (s1 is not empty, s2 is empty)**
TC4	`s1: "", s2: "a"`	`true`	**BVA (s1 is empty, s2 is not empty)**
TC5	`s1: "hello", s2: "he"`	`false`	**BVA (s1 longer than s2)**
TC6	`s1: "he", s2: "hello"`	`true`	**BVA (s1 is a prefix of s2)**
TC7	`s1: "hello", s2: "hello world"`	`true`	**BVA (s1 is an exact prefix of s2)**
TC8	`s1: "hello world", s2: "hello"`	`false`	**BVA (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 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. Equivalence Classes Identification

Class Type	**Description**
1. Valid Triangle Cases	
Equilateral Triangle	(A = B = C)
Isosceles Triangle	Two sides are equal (e.g., (A = B \neq C))
Scalene Triangle	All three sides are different (e.g., (A \neq B \neq C))
Right-Angled Triangle	(A^2 + B^2 = C^2) (or permutations of this relation)
2. Invalid Triangle Cases (Non-triangle)	
Violates Triangle Inequality	(A + B \leq C)

Class Type	Description
Negative or Zero Inputs	Any side is non-positive (e.g., (A \leq 0))

B. Test Cases for Equivalence Classes

Test Case No.	Input Values (A, B, C)	Expected Output	Equivalence Class Covered
TC1	3, 3, 3	Equilateral	Equilateral Triangle
TC2	5, 5, 8	Isosceles	Isosceles Triangle
TC3	4, 6, 7	Scalene	Scalene Triangle
TC4	3, 4, 5	Right-angled	Right-Angled Triangle
TC5	1, 2, 3	Not a Triangle	Violates Triangle Inequality
TC6	-3, 4, 5	Invalid Input	Non-positive Input
TC7	0, 5, 7	Invalid Input	Non-positive Input

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

Test Case No.	Input Values (A, B, C)	Expected Output	Boundary Condition
TC8	4.9, 5.0, 9.8	Scalene	Just satisfies (A + B > C)
TC9	5.0, 5.0, 10.0	Not a Triangle	Exactly (A + B = C)

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

Test Case No.	**Input Values (A, B, C)**	**Expected Output**	**Boundary Condition**
TC10	7.0, 5.0, 7.0	Isosceles	Boundary for Isosceles
TC11	7.0, 7.0, 7.1	Scalene	Just beyond boundary

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

Test Case No.	**Input Values (A, B, C)**	**Expected Output**	**Boundary Condition**
TC12	6.0, 6.0, 6.0	Equilateral	All sides equal
TC13	6.0, 6.0, 6.1	Isosceles	Just beyond boundary

F. Boundary Condition: (A^2 + B^2 = C^2) (Right-angled Triangle)

Test Case No.	**Input Values (A, B, C)**	**Expected Output**	**Boundary Condition**
TC14	3.0, 4.0, 5.0	Right-angled	Exact Pythagorean triplet
TC15	3.0, 4.0, 5.1	Scalene	Slightly beyond boundary

G. Boundary Condition: Non-triangle Case

Test Case No.	**Input Values (A, B, C)**	**Expected Output**	**Boundary Condition**
TC16	1.0, 2.0, 3.1	Not a Triangle	Just beyond triangle inequality
TC17	2.0, 2.0, 3.9	Not a Triangle	Just fails triangle inequality

---

H. Non-Positive Input Cases

Test Case No.	**Input Values (A, B, C)**	**Expected Output**	**Non-positive Input Case**
TC18	0, 4, 5	Invalid Input	Zero side
TC19	-1, 5, 7	Invalid Input	Negative side