Lab -8 - Functional Testing (Black-Box)

IT-314 Software Engineering

Rutvik Vegad - 202201143
21th October, 2024

Q1. 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.

Ans :-

- **Equivalence Class Partitioning** :-

Test Case ID	Input (day, month, year)	Tester Action	Expected Outcome	Reasoning
1.	15, 5, 2000	Valid	14/5/2000	Valid date (in valid range)
2.	31, 12, 2015	Valid	30/12/2015	Valid date (in valid range)
3.	29, 2, 2004	Valid	28/02/2004	Valid date (in valid range)
4.	0, 5, 2005	Invalid	Error	Day < 1 is

				invalid
5.	32, 6, 2010	Invalid	Error	Day > 31 is invalid
6.	25, 0, 2014	Invalid	Error	Month < 1 is invalid
7.	4, 15, 2007	Invalid	Error	Month > 12 is invalid
8.	4, 10, 1864	Invalid	Error	Year < 1900 is invalid
9.	4, 8, 2024	Invalid	Error	Year> 2015 is invalid

- **Boundary Value Analysis** :-

Test Case ID	Input (day, month, year)	Reasoning	Expected Outcome
1.	1, 2, 1900	Lower Boundary for date	31, 1, 1900
2.	31, 3, 2012	Upper Boundary for date	30, 3, 2012
3.	15, 1, 2010	Lower Boundary for month	14, 1, 2010
4.	31, 12, 2004	Upper Boundary for	30, 12, 2004

		month	
5.	1, 1, 1900	Lower Boundary for year	31, 12, 1899
6.	1, 1, 2015	Upper Boundary for year	31, 12, 2014
7.	31, 6, 2004	Invalid day (June only has 30 days)	Error
8.	30, 6, 2004	Valid day boundary for a 30-day month	29, 6, 2004
9.	0, 1, 2000	Invalid(day<1)	Error
10.	32, 1, 2000	Invalid(day>31)	Error
11.	29, 2, 1900	Invalid(1900 not a leap year)	Error
12.	1, 3, 2000	Valid	29, 2, 2000
13.	28, 2, 2001	Valid boundary for non-leap year	27, 2, 2001

- **Code** :-

```cpp
#include <iostream>

using namespace std;

// Function to check if a year is a leap year

bool isLeapYear(int year) {

    if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)) {

        return true;

    }

    return false;

}

// Function to return the number of days in a given month and year

int daysInMonth(int month, int year) {

    if (month == 2) {

        return isLeapYear(year) ? 29 : 28; // Handle February (leap year)

    }

    // Return 30 for April, June, September, November

    if (month == 4 || month == 6 || month == 9 || month == 11) {

        return 30;

    }

    // Return 31 for all other months

    return 31;

}
```

```
// Function to print the previous date
void previousDate(int day, int month, int year) {
    // Validate input date range
    if (year < 1900 || year > 2015 || month < 1 || month > 12
            || day < 1 || day > daysInMonth(month, year)) {
        cout << "Error: Invalid date" << endl;
        return;
    }
    // Decrease the day by 1
    day--;
    // If the day becomes 0, move to the previous month
    if (day == 0) {
        month--;
        // If the month becomes 0, move to the previous year and set month
to December
        if (month == 0) {
            month = 12;
            year--;
        }
        // Set the day to the last day of the previous month
        day = daysInMonth(month, year);
    }
    // Ensure the year is still within the valid range after decrementing
```

```
    if (year < 1900) {

        cout << "Error: Invalid date" << endl;

        return;

    }

    // Print the previous date

    cout << "Previous date is: " << day << "/" << month << "/" << year <<
endl;

}

int main() {

    // Example test case

    previousDate(1, 1, 2001); // Expected output: 31/12/2000

    return 0;

}
```

Q2. Programs:

a. 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.

Ans.

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

    return (-1);
}
```

- **Equivalence Class** :-
  - E1 : the value is present in the array.
  - E2: the value is not present in the array.
  - E3: the array is non-empty.
  - E4: the array is empty.
  - E5: the array contains invalid data-types.
  - E6: Invalid search element.

Test Case	Equivalence Class covered	Array(a)	Value(v)	Expected Output
1.	E1,E3	[1, 2, 3, 4, 5]	4	3
2.	E2,E3	[1, 2, 3, 4, 5]	10	-1
3.	E4	[]	6	-1
4.	E1,E3	[1]	1	0

5.	E1,E3	[1, 2, 3, -1]	-1	3
6.	E2,E4	[]	-1	-1
7.	E1,E3	[5, 5, 5]	5	0
8.	E2,E3	[1, 2, 3, 4]	8	-1
9.	E6,E3	[1, 2, 3, 4]	NULL	Error
10.	E5,E3	[1, 'a', 2]	2	Error

- **Boundary Value Analysis** :-

Test Case	Description	Array(a)	Value(v)	Expected Output
1.	Single element array, present.	[1]	1	0
2.	Search value at boundary	[1, 2, 3]	1	0
3.	Search value at upper boundary	[10, 20, 30]	30	2
4.	Search value at lower boundary	[10, 20, 30]	10	0
5.	Empty array	[]	10	-1

	(boundary)			
6.	Search value just above highest value	[1, 2, 3, 4, 5]	10	-1
7.	Search value at lower negative boundary	[-1, 0, 1, 2]	-1	0
8.	All identical values, v at boundary	[5, 5, 5, 5]	5	0
9.	Null value as boundary for invalid input	[1, 2, 3]	NULL	Error
10.	Array contains invalid data type	[1, 2, 'a']	2	Error

b. The function countItem returns the number of times a value v appears in an array of integers a.

Ans.

```
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** :-
  - E1: the value v is only present once.
  - E2: the value v is present multiple times.
  - E3: the value v is not present in the array.
  - E4: array a is non-empty.
  - E5:array a is empty.
  - E6: array contains invalid data types (not integers).
  - E7: invalid search value (e.g., null, if applicable).

Test Case ID	Search Value (v)	Array (a)	Expected Result	Covers Equivalence Class
1.	5	[1, 2, 3, 4, 5]	4	E1, E3
2.	5	[1, 2, 3, 4, 5, 5]	4	E1, E3
3.	10	[1, 2, 3, 4, 5]	-1	E2, E3
4.	3	[]	-1	E4
5.	1	[1]	0	E1, E3
6.	3	[3, 3, 3]	0	E1, E3
7.	2	[]	-1	E4
8.	1	[1, 'a', 2]	Error	E5, E3

9.	NULL	[1, 2, 3, 4]	Error	E6, E3
10.	6	[1, 2, 3, 4]	-1	E2, E3

- **Boundary Value Analysis** :-

Test Case ID	Search Value (v)	Array (a)	Expected Result	Covers Boundary
1.	5	[1, 2, 3, 4, 5]	4	Search value at upper boundary of array
2.	5	[1, 2, 3, 4, 5, 5]	4	Search value equal to multiple elements
3.	10	[1, 2, 3, 4, 5]	-1	Search value just above upper boundary
4.	3	[]	-1	Empty array (boundary for array size)
5.	1	[1]	0	Single-element array (boundary size 1)
6.	3	[3, 3, 3]	0	Search value equal to all elements

7.	2	[]	-1	Empty array (boundary)
8.	1	[1, 'a', 2]	Error	Array with invalid data type (boundary)
9.	NULL	[1, 2, 3, 4]	Error	Null search value (boundary for invalid input)
10.	6	[1, 2, 3, 4]	-1	Search value just above highest element

c. The function countItem returns the number of times a value v appears in an array of integers a.

Ans.

```
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** :-
  - E1: the search value v is present in the array.
  - E2: the search value v is present at the first index in the array.
  - E3: the search value v is present at the last index in the array.
  - E4: the search value v is present at multiple locations.

- E5: the search value v is less than the smallest element of the array.
- E6: the search value v is greater than the largest element of the array.
- E7: the array is empty.
- E8: the array contains invalid data types.

Test Case ID	Search Value (v)	Array (a)	Expected Result	Covers Equivalence Class
1.	5	[1, 2, 3, 4, 5]	4	E1, E3, E4
2.	1	[1, 2, 3, 4, 5]	0	E1, E2
3.	5	[1, 2, 3, 4, 5]	4	E1, E3
4.	3	[1, 2, 3, 3, 4, 5]	2	E1, E4
5.	0	[1, 2, 3, 4, 5]	-1	E5
6.	6	[1, 2, 3, 4, 5]	-1	E6
7.	2	[]	-1	E7
8.	‘a’	[1, 2, 3, 4]	Error	E8
9.	4	[2, 3, 4, 4, 4, 5]	2	E1, E4
10.	7	[1, 2, 3, 4, 5]	-1	E6

- **Boundary Value analysis** :-

Test Case ID	Search Value (v)	Array (a)	Expected Result	Covers Boundary
1.	5	[1, 2, 3, 4, 5]	4	Upper boundary for search value (last element)
2.	1	[1, 2, 3, 4, 5]	0	Lower boundary for search value (first element)
3.	5	[1, 2, 3, 4, 5]	4	Upper boundary for array size and search value
4.	3	[1, 2, 3, 3, 4, 5]	2	Boundary for search value with duplicates
5.	0	[1, 2, 3, 4, 5]	-1	Below lower boundary of search value (smaller than smallest element)

6.	6	[1, 2, 3, 4, 5]	-1	Above upper boundary of search value (larger than largest element)
7.	2	[]	-1	Empty array (boundary for array size)
8.	‘a’	[1, 2, 3, 4]	Error	Invalid data type (boundary for invalid input)
9.	4	[2, 3, 4, 4, 4, 5]	2	Search value at middle boundary, with duplicates
10.	7	[1, 2, 3, 4, 5]	-1	Above upper boundary of array elements

d. 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).

Ans.

```
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** :-
  - E1: all three sides of a triangle are of equal length.
  - E2: any two sides of a triangle are of equal length.
  - E3: all three sides of a triangle are of different lengths.
  - E4: the sides do not satisfy the triangle inequality.
  - E5: one or more sides are non-positive(invalid entries)

Test Case ID	Sides (a, b, c)	Expected Result	Covers Equivalence Class
1.	(3, 3, 3)	0	E1

2.	(3, 3, 4)	1	E2
3.	(3, 4, 5)	2	E3
4.	(1, 2, 3)	3	E4
5.	(1, 1, 0)	3	E5
6.	(5, 5, 5)	0	E1
7.	(4, 4, 2)	1	E2
8.	(5, 7, 9)	2	E3
9.	(1, 1, 3)	3	E4
10.	(0, 2, 2)	3	E5

- **Boundary Value Analysis** :-

Test Case ID	Sides (a, b, c)	Expected Result	Covers Boundary
1.	(3, 3, 3)	0	Boundary for equal sides (equilateral)
2.	(3, 3, 4)	1	Boundary for isosceles triangle
3.	(3, 4, 5)	2	Boundary for scalene triangle
4.	(1, 2, 3)	3	Boundary for triangle inequality violation
5.	(1, 1, 0)	3	Boundary for

			non-positive side length
6.	(5, 5, 5)	0	Upper boundary for equal sides (equilateral)
7.	(4, 4, 2)	1	Boundary for isosceles with smaller third side
8.	(5, 7, 9)	2	Upper boundary for valid scalene triangle
9.	(1, 1, 3)	3	Boundary for triangle inequality violation
10.	(0, -1, 2)	3	Boundary for invalid side (zero/neg value)

e. 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).

Ans.

```java
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;
}
```

- **<u>Equivalence Classes</u>** :-
    - E1: s1 is anexact prefix of s2.
    - E2: s1 is not a prefix of s2,but s1 is shorter than s2.
    - E3: s1 is not the exact prefix of s2,but s1 is longer than s2.
    - E4: s1 is equal to s2.
    - E5: s1 is an empty string.
    - E6: s2 is an empty string.
    - E7: both s1 and s2 are empty string.

Test Case ID	String s1	String s2	Expected Result	Covers Equivalence Class
1.	"pre"	"prefix"	true	E1
2.	“fix”	"prefix"	false	E2
3.	"prefixes"	"prefix"	false	E3
4.	"prefix"	"prefix"	true	E4
5.	“”	"prefix"	true	E5

6.	"pre"	“”	false	E6
7.	“”	“”	true	E7

- **Boundary value analysis** :-

Test Case ID	String s1	String s2	Expected Result	Covers Boundary
1.	"p”	"prefix"	true	Lower boundary: single character prefix
2.	"pre"	"prefix"	true	Boundary: s1 is a valid prefix but shorter than s2
3.	"prefix"	"prefix"	true	Boundary: s1 equals s2 (same length, identical)
4.	"prefixes"	"prefix"	false	Boundary: s1 is longer than s2
5.	“”	"prefix"	true	Boundary: s1 is empty (empty string

				is a prefix)
6.	"prefix"	“”	false	Boundary: s2 is empty (non-empty string cannot be a prefix)
7.	“”	“”	true	Boundary: both s1 and s2 are empty
8.	"prex"	"prefix"	false	Boundary: s1 differs from s2 at the last character

f. 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:
   i. Identify the equivalence classes for the system
   ii. 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)

iii. For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.
iv. For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.
v. For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.
vi. For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.
vii. For the non-triangle case, identify test cases to explore the boundary.
viii. For non-positive input, identify test points.

Ans.

- **Equivalence Classes** :-
    - E1: A + B > C, A + C > B, and B + C > A (valid triangle).
    - E2: A = B = C (equilateral triangle).
    - E3: A = B, A ≠ C or B = C, B ≠ A, or A = C, A ≠ B (isosceles triangle).
    - E4: A ≠ B ≠ C (scalene triangle).
    - E5: $A^2 + B^2 = C^2$ or $B^2 + C^2 = A^2$ or $A^2 + C^2 = B^2$ (right-angled triangle).
    - E6: A + B ≤ C, A + C ≤ B, or B + C ≤ A (not a triangle).
    - E7: Any side is non-positive (e.g., A ≤ 0, B ≤ 0, C ≤ 0).

Test Case ID	A	B	C	Expected Output	Covers Equivalence Class
1.	3	3	3	Equilateral	E2
2.	4	4	6	Isosceles	E3
3.	3	4	5	Right-angled	E5
4.	4.5	5.5	6.5	Scalene	E4
5.	1	2	3.5	Not a triangle	E6
6.	0	3	4	Invalid (non-positive)	E7
7.	-1	3	3	Invalid (non-positive)	E7

- **Boundary condition A = B = C (Equilateral Triangle)** :-

Test Case ID	A	B	C	Expected Output	Covers Boundary
1.	3	3	3	Equilateral	Boundary where all

					sides are equal.

- **Boundary Condition $A^2 + B^2 = C^2$ (Right-Angled Triangle)** :-

Test Case ID	A	B	C	Expected Output	Covers Boundary
1.	3	4	5	Right-angled	Boundary where $A^2 + B^2 = C^2$ (Pythagorean triplet).
2.	5	12	13	Right-angled	Boundary where another Pythagorean triplet applies.

- **Boundary Condition $A + B \leq C$** :-

Test Case ID	A	B	C	Expected Output	Covers Boundary
1.	1	1	2	Not a	Boundary

				triangle	where A + B equals C.
2.	1	2	3.1	Not a triangle	Boundary where A + B is less than C.

- **Boundary Condition for Non-Positive input** :-

Test Case ID	A	B	C	Expected Output	Covers Boundary
1.	0	3	4	Invalid (non-positive)	Non-positive value for A.
2.	-1	3	3	Invalid (non-positive)	Negative value for A.
3.	3	0	4	Invalid (non-positive)	Non-positive value for B.
4.	3	3	-1	Invalid (non-positive)	Negative value for C.