Software Engineering(IT314)

Lab-8- Functional Testing (Black-Box)

NAME:Moradiya Rajan Rakeshbhai
Student ID:202201063

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 Partitioning Test Cases**

Test Case ID	Input (day, month, year)	Tester Action	Expected Outcome	Reasoning
TC01	(15, 5, 2015)	Check previous date	(14, 5, 2015)	Valid input, should return the previous date.
TC02	(1, 1, 2000)	Check previous date	(31, 12, 1999)	Valid input at the beginning of the year, should wrap to last day of previous year.
TC03	(29, 2, 2000)	Check previous date	(28, 2, 2000)	Valid leap year date, should return previous date in February.
TC04	(31, 4, 2015)	Check previous date	An Error message	April has 30 days, invalid day input.
TC05	(32, 1, 2000)	Check previous date	An Error message	Invalid day input (day > 31).
TC06	(15, 13, 2015)	Check previous date	An Error message	Invalid month input (month > 12).
TC07	(15, 5, 2016)	Check previous date	An Error message	Year out of valid range (2016 is not allowed).
TC08	(15, 5, 1899)	Check previous date	An Error message	Year out of valid range (1899 is not allowed).

TC09	(30, 2, 2001)	Check previous date	An Error message	Invalid day in non-leap year February (30 is invalid).
TC10	(29, 2, 2016)	Check previous date	(28, 2, 2016)	Valid leap year date, should return previous date in February.

- **Test Cases for Boundary Value Analysis:**

Test Case ID	Input (day, month, year)	Reasoning	xpected Outcom
TC01	(1, 1, 1900)	Testing the lowest boundary for day, month, and year.	(31, 12, 1899)
TC02	(1, 12, 2015)	Testing the first day of the last month within valid year.	(30, 11, 2015)
TC03	(31, 12, 2015)	Testing the last day of the year, should return last day of previous month.	(30, 12, 2015)
TC04	(29, 2, 2015)	Testing a non-leap year (February should only have 28 days).	(28, 2, 2015)
TC05	(29, 2, 2016)	Testing a leap year (February should have 29 days).	(28, 2, 2016)
TC06	(30, 4, 2015)	Testing the last valid day in a month with 30 days (April).	(29, 4, 2015)
TC07	(1, 2, 1900)	Testing the first day of February in a non-leap year (1900 is not a leap year).	(31, 1, 1900)
TC08	(31, 1, 2015)	Testing the last day of January, should return last day of December.	(30, 12, 2014)
TC09	(28, 2, 1900)	Testing the last valid day in February for a non-leap year.	(27, 2, 1900)
TC10	(1, 2, 2015)	Testing the first day of February in a non-leap year (2015).	(31, 1, 2015)

- **Code:**

```
#include <iostream>
 using namespace std;
 bool isLeapYear(int year) {
 if (year % 400 == 0 ||
 (year % 4 == 0 && year % 100 != 0)){
```

```
 return true;
 }
 return false;
 }
 int daysInMonth(int month, int year) {
 if (month == 2){
 return isLeapYear(year) ? 29 : 28;
 }
 if (month == 4 || month == 6
 || month == 9 || month == 11){
 return 30;
 }
 return 31;
 }
 void previousDate(int day, int month, int year) {
if (year < 1900 || year > 2015 || month < 1
 || month > 12 || day < 1 ||
 day > daysInMonth(month, year)) {
 cout << "Error: Invalid date" << endl;
 return;
 }
 day--;
 if (day == 0) {
 month--;
 if (month == 0) {
 month = 12;
 year--;
 }
 day = daysInMonth(month, year);
 }
 if (year < 1900) {
 cout << "Error: Invalid date" << endl;
 return;
 }
 cout << "Previous date is: " << day
 << "/" << month << "/" << year << endl;
 }
int main() {
```

```
previousDate(1, 1, 2001);
return 0;
}
```

Q2. 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.

Ans.

Equivalence Class Definitions:

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 (non-integer).

E6: The search element is invalid (e.g., NULL).

- **Equivalence Partitioning Test Cases**

Test Case Description	Equivalence Classes Covered	Array (a)	Value (v)	Expected Outpu
1. Value present in the array	E1, E3	[1, 2, 3, 4, 5]	3	2
2. Value not present in the array	E2, E3	[1, 2, 3, 4, 5]	10	-1
3. Searching in an empty array	E4	[]	6	-1
4. Single element array, value present	E1, E3	[1]	1	0
5. Negative number present in the array	E1, E3	[1, 2, 3, -1]	-1	3

6. Searching in an empty array	E2, E4	[]	-1	-1
7. All identical values, search for one	E1, E3	[5, 5, 5]	5	0
8. Value not present in non-empty array	E2, E3	[1, 2, 3, 4]	8	-1
9. Invalid search element (NULL)	E6, E3	[1, 2, 3, 4]	NULL	Error
10. Array contains invalid data type	E5, E3	[1, 'a', 2]	2	Error

- **Boundary Value Analysis Test Cases**

Test Case Description	Array (a)	Value (v)	Expected Output
1. Single element array, value present	[1]	1	0
2. Search value at lower 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. Search in an empty array (boundary case)	[]	10	-1
6. Search value just above the 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 at the 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
Summary			

- **Code:**

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

```
{
if (a[i] == v)
return(i);
i++;
}
return (-1);
}
```

P2. The function countItem returns the number of times a value v appears in an array of integers a.

Ans

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	Array (a)	Value (v)	Expected Output
EP1	Value is present	[1, 2, 3, 1, 4, 1]	1	3
EP2	Value is present	[5, 5, 5]	5	3
EP3	Value is not present	[2, 3, 4]	5	0
EP4	Array is non-empty	[1, 2, 3]	3	1
EP5	Array is empty	[]	3	0
EP6	Array contains invalid data-types	[1, "two", 3]	1	Error message (invalid data type)
EP7	Invalid search element	[1, 2, 3]	"two"	Error message (invalid search element)
EP8	Value is not present	[-1, -2, -3]	0	0
EP9	Value is present (negative)	[-1, -2, -3, -1]	-1	2
EP10	Array contains invalid data-types	[1, 2, 3, 3.14]	3	Error message (invalid data type)

- **Boundary Value Analysis Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Output
BVA1	Single element in array	[1]	1	1
BVA2	Single element (no match)	[2]	1	0
BVA3	Array length at upper	[1, 1, 1, 1, 1]	1	5

	limit			
BVA4	Array length at lower limit	[]	1	0
BVA5	Value at boundary of int range	[Integer.MAX_VALUE]	Integer.MAX_VALUE	1
BVA6	Negative boundary values	[-1, -2, -3]	-1	1
BVA7	Zero value	[0, 0, 0]	0	3
BVA8	Empty array	[]	0	0
BVA9	Large array with no matching value	[2] * 1000	3	0
BVA10	Large array with all matching value	[3] * 1000	3	1000

- **Code**

```
int countItem(int v, int a[])
{
int count = 0;
for (int i = 0; i < a.length; i++)
{
if (a[i] == v)
count++;

}
return (count);

}
```

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.

ans

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	Array (a)	Value (v)	Expected Output

TC1	Value present in array	[1, 3, 5, 7, 9]	5	2 (index)
TC2	Value present in array	[-3, 0, 2, 4, 8]	4	3 (index)
TC3	Value not in array	[1, 3, 5, 7, 9]	6	-1
TC4	Value not in array	[-3, 0, 2, 4, 8]	10	-1
TC5	Non-empty array	[10, 20, 30, 40, 50]	40	3 (index)
TC6	Non-empty array	[2, 4, 6, 8, 10]	7	-1
TC7	Empty array	[]	5	-1
TC8	Invalid data types	[1, 2, "three", 4]	4	Error
TC9	Invalid data types	[1, 2, 3.5, 4]	4	Error
TC10	Invalid search element	[1, 2, 3, 4, 5]	"five"	Error

- **Boundary Value Analysis Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Output
TC11	Minimum boundary for array	[1]	1	0 (index)
TC12	Just above minimum boundary	[1, 2]	2	1 (index)
TC13	Just below maximum boundary	[1, 3, 5, 7, 9]	9	4 (index)
TC14	Maximum boundary for array	[1, 3, 5, 7, 9]	10	-1
TC15	Array of size 0 (empty)	[]	1	-1
TC16	Single-element array	[5]	5	0 (index)
TC17	Large array	[1, 2, 3,...,10000]	9999	9998 (index)
TC18	Value just below midpoint	[1, 2, 3, 4, 5]	2	1 (index)
TC19	Value just above midpoint	[1, 2, 3, 4, 5]	4	3 (index)
TC20	Midpoint boundary	[1, 2, 3, 4, 5]	3	2 (index)

- **Code**

```
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);
}
```

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).

Ans:

E1: All sides are equal → Equilateral triangle.
E2: Two sides are equal → Isosceles triangle.
E3: No sides are equal → Scalene triangle.
E4: Invalid triangle (sum of any two sides must be greater than the third side).
E5: Negative or zero-length sides → Invalid triangle.
E6: Non-integer input → Invalid (if the program handles such cases).

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class	Input (a, b, c)	Expected Outcome
TC1	E1	(3, 3, 3)	Equilateral
TC2	E2	(5, 5, 8)	Isosceles
TC3	E2	(7, 10, 10)	Isosceles
TC4	E3	(4, 6, 7)	Scalene
TC5	E4	(1, 2, 10)	Invalid
TC6	E4	(5, 3, 9)	Invalid
TC7	E5	(0, 5, 5)	Invalid

TC8	E5	(-2, 5, 5)	Invalid
TC9	E6	(3.5, 4, 4)	Invalid
TC10	E6	("a", 5, 6)	Invalid

- **Boundary Value Analysis Test Cases**

Test Case	Description	Input (a, b, c)	xpected Outcom
TC11	Minimum valid triangle	(1, 1, 1)	Equilateral
TC12	Almost invalid triangle	(1, 1, 2)	Invalid
TC13	Just valid isosceles	(2, 2, 3)	Isosceles
TC14	Just invalid triangle	(1, 2, 3)	Invalid
TC15	Scalene boundary	(3, 4, 5)	Scalene
TC16	Zero side	(0, 1, 2)	Invalid
TC17	Negative side	(-1, 1, 1)	Invalid
TC18	Large valid triangle	(1000, 1000, 1000)	Equilateral
TC19	Large almost invalid	(1000, 1, 999)	Invalid
TC20	Very small valid values	(1, 1, 1)	Equilateral

- **Code**

```
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);
}
```

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).

Ans

E1: s1 is a valid prefix of s2.
E2: s1 is not a valid prefix of s2.
E3: s1 is longer than s2.
E4: s1 is an empty string.
E5: s2 is an empty string.
E6: s1 and s2 are both empty strings.
E7: s1 contains invalid characters or symbols.

- **Equivalence Partitioning Test Cases**

Test Case #	Equivalence Class	Input s1	Input s2	Expected Outpu
TC1	E1	"pre"	"prefix"	TRUE
TC2	E2	"post"	"prefix"	FALSE
TC3	E3	"longstring"	"short"	FALSE
TC4	E4	""	"prefix"	TRUE
TC5	E5	"prefix"	""	FALSE
TC6	E6	""	""	TRUE
TC7	E7	"@pre!"	"prefix"	FALSE
TC8	E1	"apple"	"applepie"	TRUE
TC9	E2	"banana"	"apple"	FALSE
TC10	E1	"car"	"cart"	TRUE

- **Boundary Value Analysis Test Cases**

Test Case #	Description	Input s1	Input s2	Expected Outpu
TC1	s1 and s2 have the same length (boundary)	"abc"	"abc"	TRUE
TC2	s1 is one character less than s2	"abc"	"abcd"	TRUE

TC3	s1 is one character more than s2	"abcd"	"abc"	FALSE
TC4	s1 is empty, s2 has a single character	""	"a"	TRUE
TC5	s2 is empty, s1 has a single character	"a"	""	FALSE
TC6	s1 and s2 are both empty strings	""	""	TRUE
TC7	s1 is one character, s2 is one character	"a"	"b"	FALSE
TC8	s1 is equal to s2 except last character	"prefix"	"prefiy"	FALSE
TC9	s1 has the same first 5 characters as s2	"apple"	"applesauce"	TRUE
TC10	s1 has a different first character than s2	"orange"	"apple"	FALSE

- **Code**

```
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;
}
```

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.**

Ans-
E1: Valid triangle sides forming a scalene triangle.
E2: Valid triangle sides forming an isosceles triangle.
E3: Valid triangle sides forming an equilateral triangle.
E4: Valid triangle sides forming a right-angled triangle.
E5: Invalid triangle sides (e.g., A + B ≤ C, A or B or C ≤ 0).
E6: Non-triangle case (sides do not form a triangle).
E7: Non-positive inputs (negative or zero values).

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	A	B	C	Expected Output
TC1	E1 (Scalene)	3	4	5	Right-angled Triangle
TC2	E2 (Isosceles)	5	5	7	Isosceles Triangle
TC3	E3 (Equilateral)	6	6	6	Equilateral Triangle
TC4	E4 (Right-angled)	8	6	10	Right-angled Triangle

TC5	E5 (Invalid triangle)	1	2	3	Not a triangle
TC6	E6 (Non-triangle case)	2	2	8	Not a triangle
TC7	E7 (Non-positive input)	-1	2	3	Error (invalid input)
TC8	E7 (Non-positive input)	0	0	0	Error (invalid input)
TC9	E1 (Scalene)	5.5	7.2	8.1	Scalene Triangle
TC10	E1 (Scalene)	2.3	3.7	4.5	Scalene Triangle

- **Boundary Value Analysis Test Cases**

B1: Boundary for scalene triangle: A + B > C, A + C > B, B + C > A.
B2: Boundary for isosceles triangle: A = B or B = C or A = C.
B3: Boundary for equilateral triangle: A = B = C.
B4: Boundary for right-angled triangle: A2+B2=C2A^2 + B^2 = C^2A2+B2=C2.
B5: Boundary for invalid triangle: A + B = C, A + C = B, B + C = A.
B6: Non-positive inputs: A ≤ 0, B ≤ 0, C ≤ 0.

Test Case	Description	A	B	C	Expected Output
TC11	B1 (Scalene boundary A+B=C)	3	4	7	Not a triangle
TC12	B1 (Scalene boundary A+C=B)	2	5	7	Not a triangle
TC13	B1 (Scalene boundary B+C=A)	4	3	7	Not a triangle
TC14	B2 (Isosceles boundary A=B)	6	6	5	Isosceles Triangle
TC15	B2 (Isosceles boundary B=C)	7	8	8	Isosceles Triangle
TC16	B3 (Equilateral boundary A=B=C)	9	9	9	Equilateral Triangle
TC17	B4 (Right-angled triangle)	9	12	15	Right-angled Triangle
TC18	B5 (Invalid triangle A+B=C)	5	7	12	Not a triangle
TC19	B6 (Non-positive input)	-5	3	4	Error (invalid input)
TC20	B6 (Zero input)	0	4	5	Error (invalid input)

- **Non-triangle and Non-positive Input Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Output
(1, 2, 3)	Non-triangle (boundary case)	[1, 2, 3]	3	Error message (not a triangle)
(0, 4, 5)	Non-triangle (one side is zero)	[0, 4, 5]	0	Error message (non-positive input)
(-1, 1, 2)	Non-triangle (negative side)	[-1, 1, 2]	-1	Error message (non-positive input)
(5, 5, -5)	Non-triangle (one side is negative)	[5, 5, -5]	-5	Error message (non-positive input)
(5, 5, 0)	Non-triangle (one side is zero)	[5, 5, 0]	0	Error message (non-positive input)
(3, 4, "b")	Invalid data type (string input)	[3, 4, "b"]	"b"	Error message (invalid data type)
(null, 4, 5)	Invalid input (null value)	[null, 4, 5]	null	Error message (invalid data type)
(5, "a", 5)	Invalid input (string and number mix)	[5, "a", 5]	"a"	Error message (invalid data type)
(5.5, 2, -1)	Non-positive input (negative)	[5.5, 2, -1]	-1	Error message (non-positive input)
(5.5, "hello", 2)	Invalid data type (non-numeric)	[5.5, "hello", 2]	"hello"	Error message (invalid data type)