Software Engineering(IT314)

Lab-8- Functional Testing (Black-Box)

NAME:RAMANI JANVI BIPINBHAI
STUDENT ID:202201158

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	Equivalence Class Covered	Array (a)	Value (v)	Expected Output
1	Valid Date	[1, 1, 2000]	(1, 1, 2000)	31-12-1999
2	Valid Date	[15, 8, 2001]	(15, 8, 2001)	14-08-2001
3	Valid Date (Leap Year)	[29, 2, 2004]	(29, 2, 2004)	28-02-2004
4	Valid Date	[1, 3, 2015]	(1, 3, 2015)	29-02-2015
5	Invalid Month	[31, 13, 2000]	(31, 13, 2000)	Invalid Date
6	Invalid Day	[32, 1, 2001]	(32, 1, 2001)	Invalid Date
7	Invalid Day for February	[30, 2, 2001]	(30, 2, 2001)	Invalid Date
8	Invalid Year	[1, 1, 1899]	(1, 1, 1899)	Invalid Date
9	Valid Date (End of Year)	[1, 1, 2015]	(1, 1, 2015)	31-12-2014
10	Invalid Month	[1, 0, 2000]	(1, 0, 2000)	Invalid Date

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

Test Case	Description	Array (a)	Value (v)	Expected Output
1	Minimum day in January	[1, 1, 1900]	(1, 1, 1900)	31-12-1899
2	Maximum day in January	[31, 1, 1900]	(31, 1, 1900)	30-01-1900

3	Minimum day in February (non-leap)	[1, 2, 1900]	(1, 2, 1900)	31-01-1900
4	Maximum day in February (leap year)	[29, 2, 2004]	(29, 2, 2004)	28-02-2004
5	Maximum day in February (non-leap)	[29, 2, 2001]	(29, 2, 2001)	Invalid Date
6	Minimum day in March	[1, 3, 1900]	(1, 3, 1900)	29-02-1900
7	Maximum day in December	[31, 12, 2015]	(31, 12, 2015)	30-12-2015
8	Minimum day in a leap year (Feb)	[1, 2, 2000]	(1, 2, 2000)	31-01-2000
9	Maximum valid year	[1, 1, 2015]	(1, 1, 2015)	31-12-2014
10	Year boundary (lower limit)	[1, 1, 1900]	(1, 1, 1900)	31-12-1899

- **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 (e.g., strings, objects).

E6: Invalid search element (e.g., negative number, or special characters).

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	Array (a)	Value (v)	Expected Outpu
TC1	E1	[1, 2, 3, 4, 5]	3	2
TC2	E1	[10, 20, 30]	10	0
TC3	E1	[5, 10, 15, 20]	15	2
TC4	E2	[1, 2, 3]	4	-1
TC5	E2	[5, 6, 7]	8	-1
TC6	E3	[1]	1	0
TC7	E3	[1, 2, 3, 4]	1	0
TC8	E4	[]	1	-1
TC9	E5	[1, 'a', 3]	a'	-1
TC10	E6	[1, 2, 3]	-1	-1

- **Boundary Value Analysis Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Outpu
TC1	Search first element	[1, 2, 3, 4, 5]	1	0
TC2	Search last element	[1, 2, 3, 4, 5]	5	4
TC3	Search non-existent before	[1, 2, 3, 4, 5]	0	-1
TC4	Search non-existent after	[1, 2, 3, 4, 5]	6	-1
TC5	Single element found	[10]	10	0
TC6	Single element not found	[10]	5	-1
TC7	Empty array	[]	1	-1
TC8	Invalid search on empty array	[]	-1	-1
TC9	Value at first index	[1, 2, 3, 4]	1	0
TC10	Value at second index	[1, 2, 3, 4]	2	1

- **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

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.

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	Array (a)	Value (v)	Expected Outpu
1	E1	[1, 2, 3, 4, 5]	3	1
2	E1	[1, 1, 2, 3, 1]	1	3
3	E2	[1, 2, 3, 4, 5]	6	0
4	E2	[2, 2, 2]	1	0
5	E3	[5, 3, 1, 2]	2	1
6	E4	[]	1	0
7	E4	[]	0	0
8	E5	[1, 2, 'a', 4]	a'	0
9	E5	[1, 'b', 3, 4]	1	0
10	E6	[1, 2, 3]	null	0

- **Boundary Value Analysis Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Outpu
1	Lower boundary, empty array	[]	1	0
2	Lower boundary, one element	[1]	1	1
3	Lower boundary, two elements	[1, 2]	1	1
4	Upper boundary, five elements	[1, 2, 3, 4, 5]	5	1
5	Upper boundary, repeated value	[1, 2, 3, 4, 5, 5]	5	2
6	Value less than lower bound	[1, 2, 3]	-1	0
7	Value equal to upper bound	[1, 2, 3]	3	1
8	Value beyond upper bound	[1, 2, 3]	4	0
9	Large array	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*	5	1
10	Value as the first element	[1, 2, 3]	1	1

- **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
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.

- **Equivalence Partitioning Test Cases**

Test Case #	Equivalence Class Covered	Array (a)	Value (v)	Expected Outpu
1	E1	[1, 2, 3, 4, 5]	3	2
2	E1	[10, 20, 30, 40]	20	1
3	E2	[1, 2, 3, 4, 5]	6	-1
4	E2	[10, 20, 30, 40]	15	-1
5	E3	[1, 2, 3]	2	1
6	E3	[10, 20, 30]	10	0
7	E4	[]	5	-1
8	E5	[1, "two", 3]	"two"	Error message
9	E5	[1, 2, 3, null]	2	Error message
10	E6	[1, 2, 3, 4]	-5	Error message

- **Boundary Value Analysis Test Cases**

Test Case #	Description	Array (a)	Value (v)	Expected Outpu
1	Value is the first element	[1, 2, 3, 4, 5]	1	0
2	Value is the last element	[1, 2, 3, 4, 5]	5	4
3	Value is just below the first	[1, 2, 3, 4, 5]	0	-1
4	Value is just above the last	[1, 2, 3, 4, 5]	6	-1
5	Search in an empty array	[]	1	-1
6	Single element array, present	[1]	1	0
7	Single element array, not present	[1]	2	-1
8	Two elements array, first present	[1, 2]	1	0

9	Two elements array, second present	[1, 2]	2	1
10	Two elements array, not present	[1, 2]	3	-1

- **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: Valid lengths that can form a triangle.
E2: Invalid lengths that cannot form a triangle.
E3: Equal lengths for an equilateral triangle.
E4: Two lengths equal for an isosceles triangle.
E5: All lengths different for a scalene triangle.
E6: Non-integer values (invalid data types).

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	Input (a, b, c)	Expected Output
TC1	E1	(3, 4, 5)	SCALENE (2)
TC2	E1	(5, 5, 8)	ISOSCELES (1)
TC3	E1	(7, 7, 7)	EQUILATERAL (0)
TC4	E2	(1, 2, 3)	INVALID (3)
TC5	E2	(1, 1, 2)	INVALID (3)
TC6	E3	(4, 4, 4)	EQUILATERAL (0)
TC7	E4	(6, 6, 10)	ISOSCELES (1)
TC8	E5	(10, 15, 20)	SCALENE (2)
TC9	E6	(3.5, 4.5, 5.5)	Error message
TC10	E6	(3, "a", 5)	Error message

- **Boundary Value Analysis Test Cases**

Test Case	Description	Input (a, b, c)	Expected Output
TC1	Valid triangle (just valid)	(3, 4, 5)	SCALENE (2)
TC2	Minimum edge case (invalid)	(1, 1, 2)	INVALID (3)
TC3	Equilateral (minimum edge case)	(1, 1, 1)	EQUILATERAL (0)
TC4	Isosceles at boundary	(2, 2, 3)	ISOSCELES (1)
TC5	Invalid triangle at boundary	(5, 5, 10)	INVALID (3)
TC6	Just over boundary for isosceles	(2, 3, 3)	ISOSCELES (1)
TC7	Just under boundary for scalene	(1, 2, 2)	ISOSCELES (1)
TC8	Just invalid triangle	(1, 2, 3)	INVALID (3)
TC9	Large triangle	(10000, 10001, 20000)	SCALENE (2)
TC10	Large triangle (almost invalid)	(9999, 10000, 19999)	INVALID (3)

- **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: 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.

- **Equivalence Partitioning Test Cases**

Test Case	Equivalence Class Covered	String 1 (s1)	String 2 (s2)	Expected Outcome
TC1	E1	"pre"	"prefix"	TRUE
TC2	E1	"prefix"	"prefix"	TRUE
TC3	E2	"suffix"	"prefix"	FALSE
TC4	E3	"pre"	""	FALSE
TC5	E4	""	""	TRUE
TC6	E5	"pre"	null	FALSE
TC7	E6	"pre"	"pr"	FALSE
TC8	E6	""	"prefix"	TRUE

TC9	E1	"abc"	"abcde"	TRUE
TC10	E2	"xyz"	"abcd"	FALSE

- **Boundary Value Analysis Test Cases**

Test Case	Description	String 1 (s1)	String 2 (s2)	Expected Outcome
B1	Normal case	"abc"	"abcdef"	TRUE
B2	Normal case, prefix is same	"def"	"def"	TRUE
B3	Prefix is longer than string	"abcdef"	"abc"	FALSE
B4	One character prefix	"a"	"abc"	TRUE
B5	One character, mismatch	"b"	"abc"	FALSE
B6	Empty prefix	""	"anything"	TRUE
B7	Empty string, empty prefix	""	""	TRUE
B8	One character prefix, edge case	"a"	""	FALSE
B9	Long prefix	"prefix"	"prefixabcd"	TRUE
B10	Mismatch at the boundary	"prefix"	"prefixed"	TRUE

- **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: The values represent a scalene triangle (all sides different).
E2: The values represent an isosceles triangle (two sides equal).
E3: The values represent an equilateral triangle (all sides equal).
E4: The values represent a right-angled triangle (Pythagorean triplet).
E5: The values do not form a triangle (triangle inequality not satisfied).
E6: The values are non-positive (zero or negative).
E7: The values are of invalid data types (e.g., strings, special characters).

- **Equivalence Partitioning Test Cases**

Tester Action and Input Data	Expected Outcome
(3, 4, 5)	Right-angled triangle
(2, 2, 3)	Isosceles triangle
(3, 4, 6)	Scalene triangle
(1, 1, 1)	Equilateral triangle

(2, 2, 2)	Equilateral triangle
(1, 2, 3)	Error message (not a triangle)
(0, 4, 5)	Error message (non-positive input)
(-1, -2, -3)	Error message (non-positive input)
(3, "a", 5)	Error message (invalid data type)
(4, 4, 8)	Error message (not a triangle)

- **Boundary Value Analysis Test Cases**

Test Case	Description	Array (a)	Value (v)	Expected Output
(3, 4, 5)	Right-angled triangle (boundary check)	[3, 4, 5]	5	Right-angled triangle
(3, 4, 5)	Scalene triangle, just above boundary	[3, 4, 6]	6	Scalene triangle
(2, 2, 3)	Isosceles triangle, just at the boundary	[2, 2, 3]	3	Isosceles triangle
(1, 1, 1)	Equilateral triangle at the lower boundary	[1, 1, 1]	1	Equilateral triangle
(3, 4, 0)	Non-triangle (non-positive)	[3, 4, 0]	0	Error message (non-positive input)
(4, 4, 8)	Non-triangle at the edge (invalid triangle)	[4, 4, 8]	8	Error message (not a triangle)
(3, 3, 3.0001)	Close to equilateral triangle	[3, 3, 3.0001]	3.0001	Equilateral triangle
(3, 4, 5.0001)	Just above right-angle boundary	[3, 4, 5.0001]	5.0001	Scalene triangle
(3, 4.9999, 5)	Close to right-angled triangle	[3, 4.9999, 5]	5	Right-angled triangle
(5, 12, 13)	Perfect Pythagorean triplet (right triangle)	[5, 12, 13]	13	Right-angled triangle

- **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)