Software Testing
Lab Session - Functional Testing (Black-Box)

NAME : Zenith Zinzuvadia
ID : 202201082

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? 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.

→

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
15, 6, 2000	14, 6, 2000
1, 1, 2000	31, 12, 1999
29, 2, 2004	28, 2, 2004 (Leap Year)
31, 6, 2000	Invalid Date
0, 5, 2000	Invalid Date
1,3,2004	29,2,2004
Boundary Value Analysis	
1, 1, 1900	31, 12, 1899

31, 12, 2015	30, 12, 2015
32, 5, 2000	Invalid Date
0, 7, 2005	Invalid Date

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.

→

```
bool isLeapYear(int year) {
   if (year % 400 == 0) return true; if
   (year % 100 == 0) return false;
   return year % 4 == 0;
}

string previousDate(int day, int month, int year) {
   int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

   if (isLeapYear(year)) {
      daysInMonth[2] = 29;
   }

   if (month < 1 || month > 12 || day < 1 || day > daysInMonth[month] || year < 1900 || year
> 2015) {
      return "Invalid Date";
   }

   if (day > 1) { day--;
   } else { if (month >
      1) {
         month--;
      } else { month
         = 12;
         year--;
```

```
            if (year < 1900) return "Invalid Date";
        }
        day = daysInMonth[month];
    }

    return to_string(day) + ", " + to_string(month) + ", " + to_string(year);
}
```

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.

→

Equivalence Partitioning	
Input (Array `a[]`, Value `v`)	Expected Output
{1, 2, 3, 4, 5}, 3	2
{1, 2, 3, 4, 5}, 6	-1
{}, 3	-1
Boundary Value Analysis	
{3, 2, 3, 4, 5}, 3	0
{1, 2, 3, 4, 5}, 5	4
{3}, 3	0
{3}, 2	-1

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

```
    return -1;
}
```

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

→

Equivalence Partitioning	
Input (Array `a[]`, Value `v`)	Expected Output
{1, 2, 3, 2, 4, 2}, 2	3
{5, 6, 7, 8, 9}, 6	1
{10, 11, 12, 13}, 14	0
{}, 3	0
Boundary Value Analysis	
{2}, 2	1
{2}, 1	0
{1, 1, 1, 1, 1}, 1	5

→

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

→

Equivalence Partitioning	
Input (Array `a[]`, Value `v`)	Expected Output
{1, 2, 3, 4, 5}, 3	2
{1, 2, 3, 4, 5}, 6	-1
{}, 3	-1
Boundary Value Analysis	
{1, 2, 3, 4, 5}, 1	0
{1, 2, 3, 4, 5}, 5	4
3,{2}	-1
{3},3	0

→

```
int binarySearch(int v, int a[], int length) {
   int lo = 0, hi = length - 1;

   while (lo <= hi) { int 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 0 (three

lengths equal), isosceles 1 (two lengths equal), scalene 2 (no lengths equal), or invalid 3 (impossible lengths).

→

Equivalence Partitioning	
Input (a,b,c)	Expected Output
(3,3,3)	0
(3,3,4)	1
(3,4,5)	2
(1,2,3)	3
(0,1,1)	3
(-1,2,2)	3
(2,2,3)	1
Boundary Value Analysis	
(1,1,2)	3
(2,2,3)	1
(1,2,3)	3
(-1,0,1)	3

→

```
const int EQUILATERAL = 0;
const int ISOSCELES = 1; const
int SCALENE = 2; const int
INVALID = 3;

int triangle(int a, int b, int c) {
   if (a <= 0 || b <= 0 || c <= 0 || 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).
→ **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).
→

Input Strings S1,S2	Expected Output
Equivalence Partitioning	
`"abc", "abcdef"`	true
`"abc", "abddef"`	false
`"abc", "abc"`	true
`"", "abc"`	true
`"abc", ""`	false
Boundary Value Analysis	
`"abcdefg", "abc"`	false
`"a", "apple"`	true
`"abc", "abc"`	true
`"b", "apple"`	false

→

```
bool prefix(string s1, string s2) {
    if (s1.length() > s2.length()) {
    return false;
    }
```

```
        for (int i = 0; i < s1.length(); i++) {
            if (s1[i] != s2[i]) { return false;
            }
        }
        return true;
    }
```

P6: Consider the triangle classification program (P4) with a slightly altered specification: The program reads floating-point values from standard input. The three values A,B, and C are interpreted as the lengths of the sides of a triangle. The program then prints a message to standard output that indicates 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.

1. Valid Triangle Classes:

- Equilateral Triangle: $A = B = C$

- Isosceles Triangle: $A = B \neq C$ or $A = C \neq B$ or $B = C \neq A$

- Scalene Triangle: $A \neq B \neq C$ and $A + B > C$, $A + C > B$, $A + B > C$

- Right-angled Triangle: $A^2 + B^2 = C^2$ (or any permutation of this condition)

2. Invalid Triangle Classes:

- Non-Triangle: $A + B \leq C$ or $A + C \leq B$ or $B + C \leq A$

- Non-positive input: $A \leq 0$, $B \leq 0$, $C \leq 0$

b) Identify Test Cases to Cover the Identified Equivalence Classes

Input (a,b,c)	Expected Outcome
(3,4,5)	Right Angle

(2,2,3)	Isosceles
(1,1,1)	Equilateral
(3,4,6)	Scalene
(1,1,3)	Non-Triangle
(0,0,5)	Non-Positive Input

c) Boundary Condition A + B > C Case (Scalene Triangle)

(3,4,6)	Valid
(2,3,5)	Invalid
(2,4,5)	Valid

d) Boundary Condition A = C Case (Isosceles Triangle)

(3,3,5)	Valid
(4,4,8)	Valid
(1,1,2)	Invalid

e) Boundary Condition A = B = C Case (Equilateral Triangle)

(3,3,3)	Valid
(4,4,4)	Valid

f) Boundary Condition $A^2 + B^2 = C^2$ Case (Right-Angle Triangle

(3,4,5)	Valid
(6,8,10)	Valid
(1,1,2)	Invalid

g) Non-Triangle Case

(1,2,3)	Invalid

(2,3,6)	Invalid
(4,1,3)	Invalid

h) Non-Positive Input

(0,4,5)	Invalid
(-1,3,5)	Invalid
(2,-2,0)	Invalid