Software Testing
Functional Testing (Black-Box)
Lab-8

Malay Sidapara
202201488

Q1. Consider a program for determining the previous date. Its input is a 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 the previous date or invalid date. Design the equivalence class test cases?

Solution: The ranges for the input data are given as –
1 <= month <= 12
1 <= day <= 31
1900 <= year <= 2015

Hence the Equivalence Classes are –
E1 : Month value is alphabetic (Invalid)
E2 : Month value is numeric (Valid)
E3 : Month value is decimal (Invalid)
E4 : Month value is non-alphabetic (Invalid)
E5 : Month value is empty(Invalid)
E6 : Month value is less than 1 (Invalid)
E7 : Month value is the range 1 to 12 (Valid)
E8 : Month value is more than 12 (Invalid)
E9 : Day value is alphabetic (Invalid)
E10 : Day value is numeric (Valid)
E11 : Day value is decimal (Invalid)
E12 : Day value is non-alphabetic (Invalid)
E13 : Day value is empty (Invalid)
E14 : Day value is less than 1 (Invalid)
E15 : Day value is in the range 1 to 31 (Valid)
E16 : Day value is more than 31 (Invalid)
E17 : Year value is less than 1900 (Invalid)
E18 : Year value is in the range 1900 to 2015 (Valid)
E19 : Year value is more than 2015 (Invalid)
E20 : Year value is alphabetic (Invalid)
E21 : Year value is numeric (Valid)
E22 : Year value is decimal (Invalid)
E23 : Year value is non-alphabetic (Invalid)

E24 : Year value is empty (Invalid)

Test Cases with format (month, day, year) for the Equivalence Classes above are –

Test Case No.	Input Values	Expected Outcome	Classes Covered
1	(7, 19, 2004)	Previous Date	E2, E7, E10, E15, E18, E21
2	(two, 2, 1945)	Invalid Date	E1
3	(2.3, 4, 2003)	Invalid Date	E3
4	(%, 24, 2003)	Invalid Date	E4
5	(,12, 1967)	Invalid Date	E5
6	(0, 12, 1999)	Invalid Date	E6
7	(14, 25, 2006)	Invalid Date	E8
8	(11, three, 1988)	Invalid Date	E9
9	(10, 14.6, 1932)	Invalid Date	E11
10	(2, &, 1922)	Invalid Date	E12
11	(4, , 2007)	Invalid Date	E13
12	(5, 0, 2000)	Invalid Date	E14
13	(11, 35, 1992)	Invalid Date	E16
14	(8, 22, twenty)	Invalid Date	E20
15	(5, 19, 1987.6)	Invalid Date	E22
16	(12, 13, $)	Invalid Date	E23
17	(4, 20,)	Invalid Date	E24
18	(9, 17, 1899)	Invalid Date	E17
19	(1, 23, 2016)	Invalid Date	E19

Q2. 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.

Solution:

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 –
E1 : Element exists in the middle of the array
E2 : Element does not exists in the array
E3 : The array is empty
E4 : Element occurs more then one time in the array

Tester Action and Input Data	Expected Outcome	Classes Covered

v = 3
a[] = [1, 2, 3, 4, 5]
	2	E1

v = 7
a[] = [1, 2, 3, 4, 5]
	-1	E2

v = 5
a[] = []
	-1	E3

v = 4
a[] = [2, 3, 4, 7, 1, 4, 2]
	2	E4

Boundary Value Analysis –
C1 : Element exists in a single element array
C2 : Element does not exists in a single element array
C3 : Element occurs at the first position in the array
C4 : Element occurs at the last position in the array

Tester Action and Input Data	Expected Outcome	Cases Covered

v = 3
a[] = [3]
	0	C1

v = 6
a[] = [5]
	-1	C2

v = 9
a[] = [9, 10, 3, 2, 5]
	0	C3

v = 8
a[] = [0, 2, 4, 6, 8]
	4	C4

Modified Code –

```
public class SearchFunctions {
    // Modified linearSearch function to handle null arrays
    public static int linearSearch(int v, int[] a) {
        if (a == null || a.length == 0) {
            return -1; // Return -1 if the array is null or empty
        }

        for (int i = 0; i < a.length; i++) {
            if (a[i] == v) {
                return i; // Return the index if the value is found
            }
        }
        return -1; // Return -1 if the value is not found
    }
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

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 Class Partitioning –
E1 : Element appears multiple times in the array
E2 : Element does not appear in the array
E3 : The array is empty

Tester Action and Input Data	Expected Outcome	Classes Covered

v = 3
a[] = [3, 2, 3, 4, 7, 6, 3, 2]
	3	E1

v = 7
a[] = [1, 2, 3, 4, 5]
	0	E2

v = 5
a[] = []
	0	E3

v = 4
a[] = [-1, -2, -3, -4]
	0	E2

Boundary Value Analysis –
C1 : Element appears in a single element array
C2 : Element does not exists in a single element array
C3 : Element occurs at the first position in the array
C4 : Element occurs at the last position in the array

Tester Action and Input Data	Expected Outcome	Cases Covered

v = 3
a[] = [3]
	1	C1

v = 6
a[] = [5]
	0	C2

v = 9
a[] = [9, 10, 3, 2, 5]
	1	C3

v = 8
a[] = [0, 2, 4, 6, 8]
	1	C4

Modified Code –

```
#include <iostream>
using namespace std;

// Modified countItem function
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;
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

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 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 Class Partitioning –
E1 : Element exists in the array
E2 : Element does not exists in the array
E3 : The array is empty
E4 : Element occurs more then one time in the array

Tester Action and Input Data	Expected Outcome	Classes Covered

v = 5
a[] = [1, 2, 3, 4, 5, 6, 7, 8]
	4	E1

v = 7
a[] = [1, 2, 3, 4, 5]
	-1	E2

v = 2
a[] = []
	-1	E3

v = 4
a[] = [1, 3, 4, 4, 5, 6, 9]
	3	E4

Boundary Value Analysis –
C1 : Element exists in a single element array
C2 : Element does not exists in a single element array
C3 : Element occurs at the first position in the array
C4 : Element occurs at the last position in the array
C5 : Element is greater than the greatest element in the array
C6 : Element is smaller than the smallest element in the array

Tester Action and Input Data	Expected Outcome	Cases Covered
v = 3	0	C1

a[] = [3]		

v = 6
a[] = [5]
	-1	C2

v = 9
a[] = [9, 10, 11, 12]
	0	C3

v = 8
a[] = [0, 2, 4, 6, 8]
	4	C4

v = 0
a[] = [1, 3, 5, 7, 9]
	-1	C5

v = 10
a[] = [1, 3, 5, 7, 9]
	-1	C6

Modified Code –

```
#include <iostream>
using namespace std;

// Modified binarySearch function to accept array size
int binarySearch(int v, int a[], int length) {
   int lo = 0, hi = length - 1, mid;
   while (lo <= hi) {
      mid = lo + (hi - lo) / 2;
      if (v == a[mid])
         return mid;
      else if (v < a[mid])
         hi = mid - 1;
      else
         lo = mid + 1;
   }
   return -1;
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

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).

```
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 Class Partitioning –
E1 : Three equal sides lengths to form an equilateral triangle
E2 : Two equal sides lengths to form an isosceles triangle
E3 : Three unequal sides lengths to form an scalene triangle
E4 : One or more negative side lengths
E5 : Entering zero for one of the side length
E6 : Entering side lengths of a valid triangle
E7 : Sum of two sides is not greater than the third

Tester Action and Input Data	Expected Outcome	Classes Covered
a = 4, b = 4, c = 4	EQUILATERAL (0)	E1, E6
a = 3, b = 3, c = 4	ISOSCELES (1)	E2, E6
a = 4, b = 6, c = 5	SCALENE (2)	E3, E6
a = 1, b = 2, c = 3	INVALID (3)	E7
a = 0, b = 4, c = 4	INVALID (3)	E5
a = 3, b = 0, c = 4	INVALID (3)	E5
a = 9, b = 4, c = 0	INVALID (3)	E5
a = -6, b = 4, c = 4	INVALID (3)	E4

Boundary Value Analysis –
C1 : Enter side lengths of the smallest valid triangle

C2 : Sum of two sides is equal to the third
C3 : Triangle with one side close to zero

Tester Action and Input Data	Expected Outcome	Cases Covered
a = 1, b = 1, c = 1	EQUILATERAL (0)	C1
a = 1, b = 1, c = 3	ISOSCELES (1)	C1
a = 1, b = 2, c = 4	SCALENE (2)	C1
a = 5, b = 4, c = 9	INVALID (3)	C2
a = 100, b = 100, c = 1	INVALID (3)	C3

Modified Code –

```
public class TriangleType {
    final int EQUILATERAL = 0;
    final int ISOSCELES = 1;
    final int SCALENE = 2;
    final int INVALID = 3;

    public int triangle(int a, int b, int c) {
        // Check if the triangle is invalid
        if (a <= 0 || b <= 0 || c <= 0 || a >= b + c || b >= a + c || c >= a + b) {
            return INVALID;
        }
        // Check if the triangle is equilateral
        if (a == b && b == c) {
            return EQUILATERAL;
        }
        // Check if the triangle is isosceles
        if (a == b || a == c || b == c) {
            return ISOSCELES;
        }
        // Otherwise, it must be scalene
        return SCALENE;
    }
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

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 Class Partitioning –

E1 : String s1 is a valid prefix of the string s2
E2 : String s1 is not a valid prefix of the string s2
E3 : String s1 is longer than the string s2
E4 : String s1 is empty string
E5 : String s2 is empty string

Tester Action and Input Data	Expected Outcome	Classes Covered

s1 = “abc”
s2 = “abcdef”
	True	E1

s1 = “mnl”
s2 = “abcdef”
	False	E2

s1 = “abcdef”
s2 = “abc”
	False	E3

s1 = “”
s2 = “abcdef”
	True	E4

s1 = “abc”
s2 = “”
	False	E5

Boundary Value Analysis –

C1 : Both the strings are of the same size
C2 : String s1 is almost a prefix of the string s2 except the last character
C3 : String s1 is one character long and matches the first character of the string s2
C4 : String s1 is one character long and doesn’t match the first character of the string s2
C5 : Both the strings are empty strings

Tester Action and Input Data	Expected Outcome	Cases Covered

s1 = “abc”
s2 = “abc”
	True	C1

s1 = “abcd”
s2 = “abce”
	False	C2

s1 = “a”
s2 = “abcdef”
	True	C3

s1 = “c”
s2 = “abcdef”
	False	C4

s1 = “”
s2 = “”
	True	C5

Modified Code –

```
public class StringPrefix {
    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;
    }
}
```

After executing the test suite on the modified program, the identified expected outcome turns out to be correct.

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

➔ The Equivalence Classes are –
E1 : All sides are positive (Valid)
E2 : One or more sides are negative (Invalid)
E3 : Valid triangle inequality i.e. sum of two sides is greater than the third (Valid)
E4 : Invalid triangle inequality (Invalid)
E5 : All the sides are equal to form an Equilateral triangle (Valid)
E6 : Two sides are equal to form an Isosceles triangle (Valid)
E7 : All the sides are unequal to form an Scalene triangle (Valid)
E8 : Sides are entered to form a Right-angled triangle (Valid)
E9 : One of the sides has length 0 (Invalid)

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)

➔ The Test Cases are –

Test Case No.	Input Values	Expected Outcome	Covered Equivalence Class
1	3, 4, 5	Right-angled Triangle	E1, E3, E8
2	3, 3, 3	Equilateral Triangle	E1, E3, E5
3	4, 5, 4	Isosceles Triangle	E1, E3, E6
4	2, 3, 4	Scalene Triangle	E1, E3, E7
5	1, 2, 3	Invalid Triangle	E1, E4
6	0, 2, 3	Invalid Input	E9
7	-1, 2, 3	Invalid Input	E2

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome

1	2.9999, 4, 7	Scalene Triangle
2	3, 4, 7.0001	Scalene Triangle

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	5, 7.12, 5	Isosceles Triangle
2	7, 7, 13,2	Isosceles Triangle

e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	8, 8, 8	Equilateral Triangle
2	2.0, 2.0, 2.0	Equilateral Triangle

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	5, 12, 13	Right-angled Triangle
2	6, 8, 10	Right-angled Triangle

g) For the non-triangle case, identify test cases to explore the boundary.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome
1	1, 2, 3	Invalid Triangle
2	4, 4, 8	Invalid Triangle

h) For non-positive input, identify test points.

➔ The Test Case are –

Test Case No.	Input Values	Expected Outcome

1	0, 5, 3	Invalid Input
2	-1, -5, 3	Invalid Input