IT314: Software Engineering

Lab 8

Functional Testing (Black-Box)

Student ID: **202201207**
Name: **Swayam Hingu**

Q. 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 the previous dates or invalid dates. Design the equivalence class test cases?

Soln:

To design the equivalence class test cases for the program, we need to consider the input ranges and identify the valid and invalid equivalence classes. Here are the equivalence classes for the input parameters:

Valid equivalence classes:

Valid day, month, and year
Valid day, month, and minimum year (1900)
Valid day, month, and maximum year (2015)

Invalid equivalence classes:

Invalid day, month, and year (e.g., day 0, day 32, month 0, month 13, year < 1900 or year > 2015)
Invalid day for a given month and year (e.g., Feb 29 in a non-leap year, Apr 31, Jun 31, Sep 31, Nov 31)

Based on these equivalence classes, here are the test cases that should be considered:

Valid equivalence class test cases:

Test case 1: Valid day, month, and year (e.g., 15, 7, 2005)

Test case 2: Valid day, month, and minimum year (e.g., 1, 1, 1900)
Test case 3: Valid day, month, and maximum year (e.g., 31, 12, 2015)

Invalid equivalence class test cases:

Test case 4: Invalid day, month, and year (e.g., 0, 0, 1899)

Test case 5: Invalid day, month, and year (e.g., 32, 13, 2016)

Test case 6: Invalid day for a given month and year (e.g., Feb 29 in a non-leap year, such as 29, 2, 2001)

Test case 7: Invalid day for a given month and year (e.g., Apr 31, such as 31, 4, 2005)

Test case 8: Invalid day for a given month and year (e.g., Jun 31, such as 31, 6, 2005)

Test case 9: Invalid day for a given month and year (e.g., Sep 31, such as 31, 9, 2005)

Test case 10: Invalid day for a given month and year (e.g., Nov 31, such as 31, 11, 2005)

The above test cases cover all the equivalence classes for the input parameters and should be sufficient to test the program for determining the previous date.

Test Case ID	Day	Month	Year	Expected Output
1	1	6	2000	31-5-2000
2	2	6	2015	1-6-2015
3	2	6	2016	Invalid
4	1	1	1900	31-12-1899
5	31	12	1899	Invalid
6	31	12	1900	30-12-1900
7	29	2	2012	28-2-2012
8	1	3	2012	29-2-2012
9	29	2	2011	Invalid
10	30	2	2020	Invalid

Equivalence Class Partitions:

Day:

Partition ID	Range	Status
E1	Between 1 and 28	Valid
E2	Less than 1	Invalid
E3	Greater than 31	Invalid
E4	Equals 30	Valid
E5	Equals 29	Valid for leap year
E6	Equals 31	Valid

Month:

Partition ID	Range	Status
E7	Between 1 and 12	Valid
E8	Less than 1	Invalid
E9	Greater than 12	Invalid

Year:

Partition ID	Range	Status
E10	Between 1900 and 2015	Valid
E11	Less than 1	Invalid
E12	Greater than 2015	Invalid

Q. 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 on Eclipse IDE, 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.

Program 1:

```
package tests;
public class Programs
{
  int linearSearch(int v, int a[])
  {
    int i = 0;
    while (i < a.length)
    {
      if (a[i] == v)
        return (i);
      i++;
    }
    return (-1);
  }
}
```

Equivalence Partitioning:

- If a is empty, an error message should be returned. If v is not in a, -1 should be returned.
- If v is a's first element, the function should return 0.
- If v is in the middle of a, the function should return the first index i such that a[i] == v, where i is greater than 0 and less than a.length - 1.
- If v is the last element of a, the function should return a.length - 1

Tester Action and Input Data	Expected Outcome
[2, 4, 6, 8, 10], v = 2	0
[1, 3, 5, 7, 9], v = 2	-1
[2, 4, 6, 8, 10], v = 11	-1
[-100, 100]	0
[1,2,3,4,5,6], v = 6	5
[], v = 10	-1
NULL, v = 111	-1

Boundary Value Analysis:

- If a has one element, and it's not v, -1 should be returned.
- If a has one element, and it's v, the function should return 0.
- If a has two elements and v is the first element, the function should return 0.
- If a has two elements and v is the second element, the function should return 1.
- If a has n elements, and v is the first element, the function should return 0.
- If a has n elements, and v is the last element, the function should return n-1.
- If a has n elements, and v is not in a, -1 should be returned.

Tester Action and Input Data	Expected Outcome

NULL	-1
[], v = 5	-1
[5], v = 5	0
[5], v = 60	-1
[3,5], v = 3	0
[3,5], v = 5	1
[3,5], v = 14	-1
[1,3,5], v = 1	0
[1,3,5], v = 5	2
[1,3,5], v = 10	-1
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 1	0
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 111	12
[1,2,3,4,5,6,7,8,9,1,0,11,111], v = 1111	-1

Test Cases in eclipse:

```
public class UnitTesting1
{
  @Test public void test1()
  {
    int arr[] = {1, 2, 3, 4, 5};

    Programs program = new Programs();
    int output = program.linearSearch(1, arr);
```

```
    System.out.println(output);
    assertEquals(0, output);
  }

  @Test public void test2()
  {
    int arr[] = {};

    Programs program = new Programs();
    int output = program.linearSearch(5, arr);

    System.out.println(output);
    assertEquals(-1, output);
  }

  @Test public void test3()
  {
    int arr[] = {5};

    Programs program = new Programs();
    int output = program.linearSearch(5, arr);

    System.out.println(output);
    assertEquals(0, output);
  }
}
```

Program 2

```
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 Partitioning:

- If a is empty, the function should return 0. If v is not in a, the function should return 0.
- If v appears once in a, the function should return 1.

- If v appears multiple times in a, the function should return the number of occurrences of v.

Tester Action and Input Data	Expected Outcome
[265, 41, 60, 80, 100],v = 100	1
[2655, 451, 6560, 1050, 1050],v = 1050	2
[[265545, 451, 65460, 1050, 105024]],v = 1	0
[[10,10,10,10]],v = 11	0
[],v = 100	0
NULL,v = 51	0
[0],v = 0	1
[-89,-89],v = -89	2

Boundary Value Analysis:

- If a has one element and it's not v, the function should return 0. If a has one element and it's v, the function should return 1.
- If a has two elements and v is the first element, the function should return 1.
- If a has two elements and v is the second element, the function should return 1.
- If a has n elements and v appears once, the function should return 1.
- If a has n elements and v appears multiple times, the function should return the number of occurrences of v.
- If a has n elements and v is not in a, the function should return 0.

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4],v = 2	2
15, 10, 15, 15],v = 15	3
[],v = 100	0
NULL,v = 51	0
[-100,100,100,100],v = 10000	0
[-89,89],v = -89	1
[-890,890],v = 890	1

Test Cases in Eclipse:

```
public
class UnitTesting2
{
  @Test
  public void test1()
  {
    int input[] = {};
    Programs program = new Programs();
    int output = program.countItem(0, input);

    assertEquals(output, 0);
  }

  @Test
  public void test2()
  {
    int input[] = {1, 1, 2, 3, 1};
    Programs program = new Programs();
    int output = program.countItem(10, input);
```

```
  assertEquals(output, 0);
}

@Test
public void test3()
{
  int input[] = {1, 1, 2, 3, 1};
  Programs program = new Programs();
  int output = program.countItem(1, input);

  assertEquals(output, 3);
}


@Test
public void test4()
{
  int input[] = {1, 1, 2, 3, 1};
  Programs program = new Programs();
  int output = program.countItem(2, input);

  assertEquals(output, 1);
}

@Test
public void test5()
{
  int input[] = {1};
  Programs program = new Programs();
  int output = program.countItem(1, input);

  assertEquals(output, 1);
}

@Test
public void test6()
{
  int input[] = {1};
  Programs program = new Programs();
  int output = program.countItem(2, input);

  assertEquals(output, 0);
```

```
  }

  @Test
  public void test7()
  {
    int input[] = {1, 1, 2, 3, 1, 1, 1};
    Programs program = new Programs();
    int output = program.countItem(1, input);

    assertEquals(output, 5);
  }
}
```

Program 3:

```
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 Partitioning:

- If a is empty, the function should return -1. If v is not in a, the function should return -1.
- If v is the first element in a, the function should return 0.
- If v is the last element in a, the function should return a.length-1. If v appears once in a, the function should return the index of v.
- If v appears multiple times in a, the function should return the index of the first occurrence of v.

Tester Action and Input Data	Expected Outcome
[10, 20, 30, 40, 50, 60],v = 30	2
[10,100,1000,10000],v = 100000	-1
[,11,22,33,44],v = 444	-1

[11,20,200,300],v=11	0
[-100,-90,-80,100,1000],v = 10000	4
[],v = 12	-1
NULL,v = 168	-1
[1,2],v = 3	-1
[1,3],v=3	1

Boundary Value Analysis:

- If a has one element and it's not v, the function should return -1. If a has one element and it's v, the function should return 0.
- If a has two elements and v is the first element, the function should return 0.
- If a has two elements and v is the second element, the function should return 1.
- If a has n elements and v is the first element, the function should return 0.
- If a has n elements and v is the last element, the function should return n-1. If a has n elements and v appears once, the function should return the index of v.
- If a has n elements and v appears multiple times, the function should return the index of the first occurrence of v.
- If a has n elements and v is not in a, the function should return -1.

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4, 5],v = 2	1
[1, 2, 2, 351, 551],v = 2	2
[1,22,33,44,55],v = 66	-1
[2, 4, 6, 8, 10],v = 51	-1
[-100, 0, 1000],v = -100	0
[-100, 0, 1000],v = 1000	2
[],v=0	-1
NULL,v = 4	-1

Test cases in eclipse:

```
public
class UnitTesting3
{
  @Test
  public void test1()
  {
    int input[] = {2, 4, 6, 8, 10};
    Programs program = new Programs();
    int output = program.binarySearch(6, input);

    assertEquals(2, output);
  }

  @Test
  public void test2()
  {
    int input[] = {2, 4, 6, 8, 10};
    Programs program = new Programs();
```

```
  int output = program.binarySearch(2, input);

  assertEquals(0, output);
}

@Test
public void test3()
{
  int input[] = {2, 4, 6, 8, 10};
  Programs program = new Programs();
  int output = program.binarySearch(10, input);

  assertEquals(4, output);
}

@Test
public void test4()
{
  int input[] = {2, 4, 6, 8, 10};
  Programs program = new Programs();
  int output = program.binarySearch(1, input);

  assertEquals(-1, output);
}

@Test
public void test5()
{
  int input[] = {2, 4, 6, 8, 10};
  Programs program = new Programs();
  int output = program.binarySearch(12, input);

  assertEquals(-1, output);
}

@Test
public void test6()
{
  int input[] = {10};
  Programs program = new Programs();
  int output = program.binarySearch(10, input);

  assertEquals(0, output);
```

```
  }

  @Test
  public void test7()
  {
    int input[] = {};
    Programs program = new Programs();
    int output = program.binarySearch(5, input);

    assertEquals(-1, output);
  }

  @Test
  public void test8()
  {
    int input[] = {5, 7, 9};
    Programs program = new Programs();
    int output = program.binarySearch(5, input);

    assertEquals(0, output);
  }

  @Test
  public void test9()
  {
    int input[] = {1, 3, 5};
    Programs program = new Programs();
    int output = program.binarySearch(5, input);

    assertEquals(2, output);
  }
}
```

Program 4:

```
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 Partitioning:

- Equilateral triangle (a=a, b=a, c=a): expected outcome is EQUILATERAL (0)
- Isosceles triangle (a=a, b=b, c=c): expected outcome is ISOSCELES (1) Scalene triangle (a=b, b=c, c=a): expected outcome is SCALENE (2) Invalid triangle (a=b+c): expected outcome is INVALID (3)
- Invalid triangle (a=b-c): expected outcome is INVALID (3) Invalid triangle (a=b+c-1): expected outcome is INVALID (3)
- Invalid triangle (a=-1, b=-1, c=-1): expected outcome is INVALID (3) Invalid triangle (a=0, b=0, c=0): expected outcome is INVALID (3) Invalid triangle (a=1, b=2, c=4): expected outcome is INVALID (3)

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=1,b=1,c=1	EQUILATERAL
a=0,b=0,c=0	INVALID
a=-1,b=-1,c=-1	INVALID
a=10,b=10,c=0	INVALID
a=17,b=17,c=5	ISOSCELES
a=15,b=2,c=15	ISOSCELES
a=6,b=11,c=5	SCALENE
a=16,b=21,c=25	SCALENE
a=-1,b=21,c=25	INVALID
a=2,b=3,c=4	SCALENE

```
public
class UnitTesting4
{
  @Test public void test1()
  {
    int a = 2, b = 2, c = 2;
    Programs program = new Programs();
    int output = program.triangle(a, b, c);

    assertEquals(output, 0);
  }
  @Test public void test2()
```

```
{
  int a = 3, b = 3, c = 4;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 1);
}
@Test public void test3()
{
  int a = 6, b = 5, c = 4;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 2);
}

@Test public void test4()
{
  int a = 0, b = 0, c = 0;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test5()
{
  int a = -1, b = -1, c = 5;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test6()
{
  int a = 2, b = 2, c = 1;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 1);
}
@Test public void test7()
{
  int a = 0, b = 1, c = 1;
```

```
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test8()
{
  int a = 1, b = 0, c = 1;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test9()
{
  int a = 1, b = 1, c = 0;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test10()
{
  int a = 1, b = 2, c = 3;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 3);
}
@Test public void test11()
{
  int a = 3, b = 1, c = 3;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);

  assertEquals(output, 1);
}
@Test public void test12()
{
  int a = 5, b = 4, c = 2;
  Programs program = new Programs();
  int output = program.triangle(a, b, c);
```

```
    assertEquals(output, 2);
  }
  @Test public void test13()
  {
    int a = Integer.MAX_VALUE, b = Integer.MAX_VALUE, c = Integer.MAX_VALUE;
    Programs program = new Programs();
    int output = program.triangle(a, b, c);

    assertEquals(output, 3);
  }
  @Test public void test14()
  {
    int a = Integer.MIN_VALUE, b = Integer.MIN_VALUE, c = Integer.MIN_VALUE;
    Programs program = new Programs();
    int output = program.triangle(a, b, c);

    assertEquals(output, 3);
  }
}
```

Program 5

```
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 Partitioning:

- s1 and s2 are both empty strings: false s1 is empty and s2 is non-empty: true s1 is non-empty and s2 is empty: false s1 is a proper prefix of s2: true
- s1 is not a prefix of s2: false s1 and s2 are equal: true

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1 = "",s2 = ""	true
s1 = "po",s2 = "poojan"	true
s1 = "poo",s2 = "po"	false
s1 = "abc",s2 = ""	false
s1 = "",s2 = "abc"	true
s1 = "o",s2 = "ott"	true
s1 = "abc",s2 = "def"	false
s1 = "deg",s2 = "def"	false

Boundary Value Analysis:

- s1 is one character shorter than s2: true s1 is one character longer than s2: false s1 and s2 have the same length: true

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1= "",s2 = ""	true

s1= "abcd",s2 = ""	false
s1= "",s2 = "abcd"	true
s1 = "aef",s2 = "def"	false
s1 = "def",s2 = "deg"	false
s1 = "a",s2 = "att"	true
s1 = "poojan",s2 = "patel"	false

```
ublic class UnitTesting6
{
  @Test public void test1()
  {
    String str1 = "good", str2 = "good morning";

    Programs program = new Programs();
    boolean output = program.prefix(str1, str2);

    assertEquals(output, true);
  }
  @Test public void test2()
  {
    String str1 = "a", str2 = "abc";

    Programs program = new Programs();
    boolean output = program.prefix(str1, str2);

    assertEquals(output, true);
  }
  @Test public void test3()
  {
    String str1 = "", str2 = "good morning";

    Programs program = new Programs();
    boolean output = program.prefix(str1, str2);
```

```
  assertEquals(output, true);
}
@Test public void test4()
{
  String str1 = "morning", str2 = "good morning";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, false);
}
@Test public void test5()
{
  String str1 = "soft", str2 = "software";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
@Test public void test6()
{
  String str1 = "software", str2 = "soft";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, false);
}
@Test public void test7()
{
  String str1 = "a", str2 = "ab";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
@Test public void test8()
{
  String str1 = "software", str2 = "softwareee";

  Programs program = new Programs();
```

```
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
@Test

    public void
    test9()
{
  String str1 = "abc", str2 = "abc";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
@Test public void test10()
{
  String str1 = "a", str2 = "b";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, false);
}
@Test public void test11()
{
  String str1 = "a", str2 = "a";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
@Test public void test12()
{
  String str1 = "", str2 = "";

  Programs program = new Programs();
  boolean output = program.prefix(str1, str2);

  assertEquals(output, true);
}
```

Program 6

Equivalence classes:

- A, B, and C form a valid triangle
- A, B, and C do not form a valid triangle

Class ID	Class
E1	All sides are positive
E2	two of its sides are zero
E3	One of its sides are negative
E4	Sum of two sides is less than third side
E5	Any of the side/sides is negative

Test cases:

- A=4, B=4, C=4 (Equilateral triangle) A=4, B=4, C=5 (Isosceles triangle) A=4, B=5, C=6 (Scalene triangle) A=3, B=4, C=5 (Right-angle triangle)
- A=1, B=2, C=3 (Does not form a valid triangle)

Test Case ID	Class ID	Test Case
T1	E1	A = 1,B = 1,C = 1
T2	E1	A = 3, B = 4, C= 5
T3	E2	A = 0,B = 0,C = 1
T4	E3	A = 0,B = 1,C = 2
T5	E4	A = 1, B = 3, C = 8
T6	E5	A = -1,C = 1,D = 5

- Test cases for boundary condition A+B>C:

A=0.1, B=0.2, C=0.3 (Smallest valid scalene triangle)
A=0.1, B=0.1, C=0.2 (Smallest invalid triangle)

- Test cases for boundary condition A=C:

A=3, B=4, C=3 (Isosceles triangle with equal sides A and C)
A=0.1, B=0.2, C=0.1 (Smallest isosceles triangle with equal sides A and C)
A=1, B=2, C=1 (Smallest invalid triangle with equal sides A and C)

- Test cases for boundary condition
  A=B=C: A=5, B=5, C=5 (Equilateral
  triangle)

A=0.1, B=0.1, C=0.1 (Smallest equilateral triangle)
A=1, B=2, C=3 (Smallest invalid triangle with equal sides A, B, and C)

- Test cases for boundary condition A^2 + B^2 = C^2: A=3, B=4, C=5 (Right-angle triangle)

A=0.1, B=0.2, C=0.22361 (Smallest right-angle triangle)
A=1, B=1, C=1.41421 (Smallest invalid right-angle triangle)

- Test cases for non-triangle case: A=1, B=2, C=10 (A + B< C)

A=1, B=10, C=2 (A + C < B) A=10, B=1, C=2 (B + C < A)

- Test cases for non-positive input: A=-1, B=2, C=3

A=1, B=-2, C=3
A=1, B=2, C=-3
A=-1, B=-2, C=-3