Dhirubhai Ambani
Institute of Information and Communication Technology

Lab 8

Subject: Software Engineering

Subject code: IT-314

Student Name: Tanmay Singh

Student Id: 202201135

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?

Answer:

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

Equivalence Classes:

- 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 in 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	(6, 18, 2011)	Previous Date	E2, E7, E10, E15, E18, E21
2	(April, 5, 2005)	Invalid Date	E1
3	(3.5, 10, 1997)	Invalid Date	E3
4	(@, 15, 2010)	Invalid Date	E4
5	(, 29, 2012)	Invalid Date	E5
6	(0, 8, 2001)	Invalid Date	E6
7	(13, 30, 2007)	Invalid Date	E8
8	(8, three, 1988)	Invalid Date	E9
9	(10, 9.6, 1932)	Invalid Date	E11
10	(9, *, 1925)	Invalid Date	E12
11	(11, , 2013)	Invalid Date	E13
12	(10, 0, 1980)	Invalid Date	E14
13	(2, 32, 1930)	Invalid Date	E16
14	(5, 11, two thousand)	Invalid Date	E20
15	(4, 17, 1999.8)	Invalid Date	E22
16	(7, 12, &)	Invalid Date	E23
17	(3, 21,)	Invalid Date	E24
18	(1, 6, 1888)	Invalid Date	E17
19	(12, 25, 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.

P1: The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, the function returns the first index i such that a[i] == v; otherwise, -1 is returned.

Answer:

Code:

```
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 somewhere in the middle of the array

- E2: Element does not exist in the array
- E3: The array is empty

- E4: Element occurs more than once in the array

Tester Action and Input Data	Expected Outcome	Classes Covered
v = 12, a[] = [5, 9, 12, 18, 22]	2	E1
v = 20, a[] = [3, 8, 15, 21, 27]	-1	E2
v = 10, a[] = []	-1	E3
v = 18, a[] = [12, 18, 7, 18, 5]	2	E4

Boundary Value Analysis:

- C1: Element exists in a single-element array
- C2: Element does not exist 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 = 7, a[] = [7]	0	C1
v = 8, a[] = [1]	-1	C2
v = 14, a[] = [14, 20, 25, 30, 35]	0	C3
v = 42, a[] = [10, 20, 30, 35, 42]	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.

Answer:

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;
}
```

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 = 5, a[] = [5, 1, 5, 7, 5, 8]	3	E1
v = 9, a[] = [2, 4, 6, 8, 10]	0	E2
v = 12, a[] = []	0	E3

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

Boundary Value Analysis:

- C1: Element appears in a single-element array
- C2: Element does not exist 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 = 7, a[] = [7]	1	C1
v = 2, a[] = [1]	0	C2
v = 5, a[] = [5, 9, 12, 15, 18]	1	C3
v = 20, a[] = [10, 15, 17, 19, 20	1	C4

Modified code:

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

// Modified countItem function to handle
null or empty arrays 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 is found in the array, the function returns an index i such that a[i] == v; otherwise, it returns -1.

Answer:

Code:

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

Equivalence Class Partitioning:

- E1: Element exists in the array
- E2: Element does not exist in the array
- E3: The array is empty
- E4: Element occurs more than once in the array

Tester Action and Input Data	Expected Outcome	Classes Covered
v = 8, a[] = [2, 4, 6, 8, 10, 12, 14]	4	E1
v = 1, a[] = [3, 5, 7, 9, 11]	-1	E2
v = 4, a[] = []	-1	E3

v = 6, a[] = [1, 3, 6, 6, 7, 9, 10]	3	E4

Boundary Value Analysis:

- C1: Element exists in a single-element array
- C2: Element does not exist 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 = 4, a[] = [4]	0	C1
v = 2, a[] = [5]	-1	C2
v = 7, a[] = [7, 8, 9, 10]	0	C3
v = 15, a[] = [5, 7, 9, 11, 15]	4	C4
v = 20, a[] = [2, 4, 6, 8, 10]	-1	C5
v = -5, a[] = [0, 2, 4, 6, 8]	-1	C6

Modified code:

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

 // Modified binarySearch function to
 handle the 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).

Answer:

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 for invalid triangles: non-positive sides or

    triangle inequality violation 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;

  }

}
```

Equivalence Class Partitioning:

- E1: All three sides are equal (Equilateral triangle).
- E2: Exactly two sides are equal (Isosceles triangle).
- E3: All three sides are different (Scalene triangle).
- E4: One or more negative sides (Invalid triangle).
- E5: One side length is zero (Invalid triangle).
- E6: Valid side lengths for a valid triangle.
- E7: Sum of two sides is not greater than the third side (Invalid triangle).

Tester Action and Input Data	Expected Outcome	Classes Covered
a = 7, b = 7, c = 7	EQUILATERAL (0)	E1, E6
a = 5, b = 5, c = 9	ISOSCELES (1)	E2, E6
a = 3, b = 4, c = 5	SCALENE (2)	E3, E6
a = 10, b = 5, c = 3	INVALID (3)	E7
a = 0, b = 6, c = 6	INVALID (3)	E5
a = -1, b = 3, c = 4	INVALID (3)	E5

Boundary Value Analysis:

- C1: Smallest valid triangle (all sides = 1).
- C2: Sum of two sides equals the third.
- C3: One side is very close to zero but valid.

Tester Action and Input Data	Expected Outcome	Cases Covered
a = 1, b = 1, c = 1	EQUILATERAL (0)	C1
a = 2, b = 2, c = 4	INVALID (3)	C2
a = 1000, b = 1, c = 1	INVALID (3)	C3

Modified code:

```
public class TriangleType {
```

```
// Constants representing different

triangle types public static final int

EQUILATERAL = 0;

public static final int

ISOSCELES = 1; public static

final int SCALENE = 2;

public static final int

INVALID = 3;


public int triangle(int a, int b, int c) {

  // Check for invalid triangles: non-positive sides or

  invalid triangle inequality 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;

  }
```

```
    // If it’s neither equilateral nor isosceles,

    it must be scalene return SCALENE;

  }


  public static void main(String[] args) {

    TriangleType triangleType = new TriangleType();


    // Example test cases

    System.out.println(triangleType.triangle(7, 7, 7)); // Output: 0
    (Equilateral)

    System.out.println(triangleType.triangle(5, 5, 9)); // Output: 1
    (Isosceles)

    System.out.println(triangleType.triangle(3, 4, 5)); // Output: 2
    (Scalene)

    System.out.println(triangleType.triangle(10, 5, 3)); // Output: 3
    (Invalid)

    System.out.println(triangleType.triangle(-1, 3, 4)); // Output: 3
    (Invalid)
    }
}
```

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).

Answer:

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;
  }
}
```

Equivalence Class Partitioning:

1. E1: s1 is a valid prefix of s2.
2. E2: s1 is not a valid prefix of s2.
3. E3: s1 exceeds the length of s2.
4. E4: s1 is an empty string.
5. E5: s2 is an empty string.

Tester Action and Input Data	Expected Outcome	Classes Covered

s1 = "hello"
s2 = "hello world"
	True	E1

s1 = "xyz" s2 = "abcdef"	False	E2

s1 = "abcdefgh"
s2 = "abc"
	False	E3

s1 = ""
s2 = "test"
	True	E4

s1 = "abc"
s2 = ""
	False	E5

Boundary Value Analysis:

1. C1: Both strings are of equal length.
2. C2: s1 is nearly a prefix of s2, differing only at the last character.
3. C3: s1 is a single character that matches the beginning of s2.
4. C4: s1 is a single character that does not match the start of s2.
5. C5: Both strings are empty.

Tester Action and Input Data	Expected Outcome	Cases Covered
s1 = "test" s2 = "test"	True	C1
s1 = "test1" s2 = "test2"	False	C2
s1 = "t" s2 = "test"	True	C3
s1 = "x" s2 = "test"	False	C4
s1 = "" s2 = ""	True	C5

P6: Triangle Classification: The program takes floating-point numbers as input, treating them as the side lengths of a triangle. It then outputs a message stating whether a valid triangle can be formed and identifies its type: scalene, isosceles, equilateral, or right-angled.

Answer:

a) Equivalence Classes Identification

The identified Equivalence Classes are:

- E1: All sides are positive (Valid)
- E2: One or more sides are negative (Invalid)
- E3: Valid triangle inequality (sum of two sides greater than the third) (Valid)
- E4: Invalid triangle inequality (Invalid)
- E5: All sides equal, forming an Equilateral triangle (Valid)
- E6: Two sides equal, forming an Isosceles triangle (Valid)
- E7: All sides unequal, forming a Scalene triangle (Valid)
- E8: Sides 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