IT314 - Software Engineering

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

Krutarth Kadia - 202201475

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.

Input: Triple of day, month, and year

Input ranges:
1 <= month <= 12
1 <= day <= 31
1900 <= year <= 2015

Output: Previous date or Invalid date

1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

Equivalence Partitioning

Valid Equivalence Classes:

- Days: 1 <= day <= 31
- Months: 1 <= month <= 12

- Years: 1900 <= year <= 2015

Invalid Equivalence Classes:

- Days: < 1, > 31
- Months: < 1, > 12
- Years: < 1900, > 2015

Tester Action and Input Data	Expected Outcome
a, b, c	An error message
(15, 8, 2015)	Previous date: 14-08-2015
(1, 1, 1900)	Previous date: 31-12-1899
(31, 12, 2015)	Previous date: 30-12-2015
(32, 8, 2015)	Error: Invalid day
(0,7,2014)	Error: Invalid day
(15, 13, 2010)	Error: Invalid month
(15, 0, 2010)	Error: Invalid month
(15, 8, 1899)	Error: Invalid year
(15, 8, 2016)	Error: Invalid year
(29, 2, 2015)	Error: Invalid date (non-leap year)

Boundary Value Analysis

Valid Boundary Cases:

- Days: 1, 31
- Months: 1, 12
- Years: 1900, 2015

Invalid Boundary Cases:

- Days: 0, 32
- Months: 0, 13
- Years: 1899, 2016

Tester Action and Input Data	Expected Outcome
(1, 1, 1900)	Previous date: 31-12-1899
(31, 12, 2015)	Previous date: 30-12-2015
(1, 0, 2000)	Error: Invalid month
(0, 1, 2000)	Error: Invalid day
(31, 13, 2000)	Error: Invalid month
(31,0,2000)	Error: Invalid month
(32, 12, 2000)	Error: Invalid day
(1, 1, 1899)	Error: Invalid year
(1, 1, 2016)	Error: Invalid year

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.

C++ Implementation:

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

// Function to check if a given year is a leap year
bool isLeapYear(int year)
{
    if (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))
```

```
        return true;
    return false;
}

// Function to get the number of days in a month
int daysInMonth(int month, int year)
{
    switch (month)
    {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        return 31;
    case 4:
    case 6:
    case 9:
    case 11:
        return 30;
    case 2:
        return (isLeapYear(year)) ? 29 : 28;
    default:
        return -1; // Invalid month
    }
}

// Function to determine the previous date
void previousDate(int day, int month, int year)
{
    // Check for valid date input
    if (year < 1900 || year > 2015 || month < 1 || month > 12 ||
day < 1 || day > daysInMonth(month, year))
```

```
    {
        cout << "Invalid Date" << endl;
        return;
    }

    // Decrement the day
    day--;

    // If day goes to 0, adjust the month and day accordingly
    if (day == 0)
    {
        month--; // Move to previous month
        if (month == 0)
        { // Move to previous year if needed
            month = 12;
            year--;
            if (year < 1900)
            {
                cout << "Invalid Date" << endl;
                return;
            }
        }
        day = daysInMonth(month, year); // Set day to the last
day of the previous month
    }

    // Output the previous date
    cout << "Previous Date: " << day << "/" << month << "/" <<
year << endl;
}

int main()
{
    int day, month, year;
    cout << "Enter day: ";
    cin >> day;
    cout << "Enter month: ";
    cin >> month;
    cout << "Enter year: ";
    cin >> year;

    previousDate(day, month, year);

    return 0;
}
```

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.

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

Equivalence Partitioning

- v is present in the array (valid).
- v is not present in the array (valid).
- The array is empty (invalid).
- The value v is negative, but the array contains non-negative integers (valid).
- The array contains negative integers, and v is negative (valid).

Tester Action and Input Data	Expected Outcome
linearSearch(3, [1, 2, 3, 4])	Returns 2 (index of 3)
linearSearch(5, [1, 2, 3, 4])	Returns -1 (not found)
linearSearch(1, [])	Returns -1 (empty array)
linearSearch(-2, [1, 2, 3, 4])	Returns -1 (not found)
linearSearch(-2, [-1, -2, -3])	Returns 1 (index of -2)
linearSearch(null,[1,2,3])	An error message
linearSearch(5,[])	An error message

Boundary Value Analysis

- The array has only one element, and the element matches v.
- The array has only one element, but the element does not match v.
- The array has multiple elements, and v is the first element.
- The array has multiple elements, and v is the last element.
- The array has multiple elements, and v is not present.

Tester Action and Input Data	Expected Outcome
linearSearch(1, [1])	Returns 0 (single element)
linearSearch(5, [1])	Returns -1 (not found)
linearSearch(1, [1, 2, 3, 4])	Returns 0 (first element)
linearSearch(4, [1, 2, 3, 4])	Returns 3 (last element)
linearSearch(5, [1, 2, 3, 4])	Returns -1 (not found)
linearSearch(null,[1,2,3])	An error message

linearSearch(5,[])	An error message

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 Partitioning

- v is present in the array one or more times (valid).
- v is not present in the array (valid).
- The array is empty (invalid).
- The array contains only negative integers, and v is negative (valid).
- The array contains positive integers, and v is positive (valid).

Tester Action and Input Data	Expected Outcome
countItem(3, [1, 2, 3, 4, 3])	Returns 2
countItem(5, [1, 2, 3, 4])	Returns 0
countItem(1, [])	Returns 0
countItem(-2, [-1, -2, -2, -3])	Returns 2

countItem(2, [1, 2, 2, 3, 2])	Returns 3
countItem(null,[1,2,3])	An error message
countItem(5,[])	An error message

Boundary Value Analysis

- The array has only one element, and the element matches v.
- The array has only one element, and the element does not match v.
- The array has multiple elements, and v appears at the beginning.
- The array has multiple elements, and v appears at the end.
- The array has multiple elements, and v appears multiple times.

Tester Action and Input Data	Expected Outcome
countItem(1, [1])	Returns 1
countItem(5, [1])	Returns 0
countItem(1, [1, 2, 3, 1])	Returns 2
countItem(4, [1, 2, 3, 4])	Returns 1
countItem(2, [1, 2, 2, 2, 3, 2])	Returns 4
countItem(null,[1,2,3])	An error message
countItem(5,[])	An error message

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.

```
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

- v is present in the array (valid).
- v is not present in the array (valid).
- The array is empty (invalid).
- The value v is negative, and the array contains negative numbers (valid).
- The value v is positive, and the array contains positive numbers (valid).

Tester Action and Input Data	Expected Outcome
binarySearch(3, [1, 2, 3, 4])	Returns 2
binarySearch(5, [1, 2, 3, 4])	Returns -1
binarySearch(1, [])	Returns -1
binarySearch(-2, [-3, -2, -1])	Returns 1
binarySearch(3, [1, 2, 3, 4, 5])	Returns 2

binarySearch(null,[1,2,3])	An error message

Boundary Value Analysis

- The array has only one element, and the element matches v.
- The array has only one element, but the element does not match v.
- The array has multiple elements, and v is the first element.
- The array has multiple elements, and v is the last element.
- The array has multiple elements, and v is not present.

Tester Action and Input Data	Expected Outcome
binarySearch(1, [1])	Return 0
binarySearch(5, [1])	Returns -1
binarySearch(1, [1, 2, 3, 4])	Returns 0
binarySearch(4, [1, 2, 3, 4])	Returns 3
binarySearch(5, [1, 2, 3, 4])	Returns -1
binarySearch(null,[1,2,3])	An error message
binarySearch(1, [])	Returns -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).

```
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

- Valid triangle, equilateral (all sides equal).
- Valid triangle, isosceles (two sides equal).
- Valid triangle, scalene (no sides equal).
- Invalid triangle (sum of two sides is less than or equal to the third side).

Tester Action and Input Data	Expected Outcome
triangle(3, 3, 3)	Returns `EQUILATERAL`
triangle(3, 3, 4)	Returns `ISOSCELES`
triangle(3, 4, 5)	Returns `SCALENE`
triangle(1, 2, 3)	Returns `INVALID`
triangle(0,1,0)	An error message

Boundary Value Analysis

- a + b = c (invalid triangle).

- a = b = c (equilateral triangle).
- a = b or a = c (isosceles triangle).
- Close to the boundary of being an invalid triangle but still valid.

Tester Action and Input Data	Expected Outcome
triangle(1, 2, 3)	Returns `INVALID`
triangle(4, 4, 4)	Returns `EQUILATERAL`
triangle(5, 5, 3)	Returns `ISOSCELES`
triangle(3, 4, 6)	Returns `INVALID`
triangle(0,1,0)	An error message

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 Partitioning

- s1 is a prefix of s2 (valid).
- s1 is not a prefix of s2 (valid).
- s1 is longer than s2 (valid).
- s1 is empty (valid).

Tester Action and Input Data	Expected Outcome
prefix("pre", "prefix")	Returns true
prefix("suf", "prefix")	Returns `false`
prefix("prefixextra", "prefix")	Returns `false`
prefix("", "prefix")	Returns `true`
prefix("prefix", null)	An error message

Boundary Value Analysis

- s1 and s2 are identical strings.
- s1 has one character less than s2, and all characters match.
- s1 has one character more than s2.

Tester Action and Input Data	Expected Outcome
prefix("test", "test")	Returns `true`
prefix("tes", "test")	Returns `true`
prefix("tests", "test")	Returns `false`
prefix("anything", null)	An error message

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

- EC1 : Valid triangle, equilateral (A = B = C)
- EC2 : Valid triangle, isosceles (A = B, or A = C, or B = C)
- EC3 : Valid triangle, scalene (A ≠ B ≠ C)
- EC4 : Valid triangle, right-angled (A^2 + B^2 = C^2 or its permutations)
- EC5 : Invalid triangle (A + B ≤ C, A + C ≤ B, or B + C ≤ A)
- EC6 : Non-positive values (invalid) (Any side A, B, or C is less than or equal to zero)

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)

Tester Action and Input Data	Expected Outcome	Equivalence Class
triangle(3.0, 3.0, 3.0)	Equilateral	EC1
triangle(3.0, 3.0, 4.0)	Isosceles	EC2
triangle(3.0, 4.0, 5.0)	Scalene	EC3
triangle(3.0, 4.0, 5.0)	Right-angled	EC4
triangle(1.0, 2.0, 3.0)	Invalid	EC5
triangle(-1.0, 2.0, 3.0)	Invalid	EC6

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

Tester Action and Input Data	Expected Outcome

triangle(2.0, 2.0, 3.0)	Scalene
triangle(1.0, 1.0, 2.0)	Invalid

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

Tester Action and Input Data	Expected Outcome
triangle(3.0, 2.0, 3.0)	Isosceles
triangle(3.0, 2.0, 5.0)	Invalid

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

Tester Action and Input Data	Expected Outcome
triangle(3.0, 3.0, 3.0)	Equilateral
triangle(3.0, 2.0, 5.0)	Invalid

f) For the boundary condition A^2 + B^2 = C^2 case (right-angle triangle), identify test cases to verify the boundary.

Tester Action and Input Data	Expected Outcome
triangle(3.0, 4.0, 5.0)	Right-angled
triangle(3.0, 2.0, 5.0)	Invalid

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

Tester Action and Input Data	Expected Outcome
triangle(1.0, 2.0, 3.0)	Invalid

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

Tester Action and Input Data	Expected Outcome
triangle(-1.0, 2.0, 3.0)	Invalid
triangle(0.0, 0.0, 0.0)	Invalid