Lab08

IT-314 Software Engineering

Rushi makadiya
202201117

Q1. 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.

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 :-

➜ Equivalence Class Partitioning Test Case

INPUT	**EXPECTED OUTPUT**	**REASONING**
10, 6, 2005	Valid	Typical valid date.
30, 11, 2014	Valid	Date on upper month boundary.
1, 3, 2004	Valid	Leap year transition (Feb 29).
31, 4, 2002	Invalid	April has only 30 days.
32, 8, 2010	Invalid	Day exceeds valid

		range.
29, 2, 1901	Invalid	1901 is not a leap year.
15, 13, 2003	Invalid	Month exceeds valid range.
-1, 5, 2012	Invalid	Negative day input.
1, 0, 1999	Invalid	Zero is not a valid month.
15, 2, 2017	Invalid	Year exceeds valid range.

➜ **Boundary Value Analysis Test Cases :**

Input	**Expected Outcome**	**Reasoning**
1, 1, 1900	Valid	Lower year boundary.
31, 12, 2015	Valid	Upper year boundary.
1, 3, 2000	Valid	Leap year Feb-to-March transition.
0, 1, 2000	Invalid	Day below minimum range.
32, 1, 2000	Invalid	Day exceeds valid range.
30, 4, 2001	Valid	Valid boundary for April.
29, 2, 1900	Invalid	1900 not a leap year.

Code:-

```
#include <iostream>

using namespace std;


bool isLeapYear(int year) {

    return (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0));

}


int daysInMonth(int month, int year) {

    if (month == 2)

        return isLeapYear(year) ? 29 : 28;

    if (month == 4 || month == 6 || month == 9 || month == 11)

        return 30;

    return 31;

}


void previousDate(int day, int month, int year) {

    if (year < 1900 || year > 2015 || month < 1 || month > 12 ||

        day < 1 || day > daysInMonth(month, year)) {

        cout << "Error: Invalid date" << endl;

        return;

    }
```

```
    day--;
    if (day == 0) {
        month--;
        if (month == 0) {
            month = 12;
            year--;
        }
        day = daysInMonth(month, year);
    }

    if (year < 1900) {
        cout << "Error: Invalid date" << endl;
        return;
    }

    cout << "Previous date is: " << day << "/" << month << "/" << year << endl;
}

int main() {
    previousDate(1, 1, 2001);
    previousDate(1, 3, 2004);
    previousDate(1, 1, 1900);
```

```
    previousDate(15, 7, 2015);

    previousDate(29, 2, 2000);



    return 0;

}
```

Q2.

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);
}
```

➔ **Value Present:**

E1: Thevaluevispresent in the array and occurs once.

E2: Thevaluevispresent in the array and occurs multiple times.

E3: Thevaluevisnotpresent in the array.

➔ **Array Edge Cases:**

E4 : The array is empty.

Test Case	**Input**	**Expected Output**	**Equivalence Boundary**
TC1	v = 8, [3, 7, 8, 10, 12]	2	E1
TC2	v = 4, [4, 4, 4, 4, 4]	0	E2
TC3	v = 15, [5, 10, 20, 30]	-1	E3
TC4	v = 0, [ ]	-1	E4

➔ **Boundry Points:-**

BP1:Single-element array where v is present.

BP2:Single-element array where v is not present.

BP3:visatthefirstposition.

BP4:visatthelast position.

BP5:Arraycontains negative numbers and v is a negative number.

Test Case	Input	Expected Output	Equivalence Boundary
BP1	v=3,[3]	1	BP1
BP2	v=2, [1]	0	BP2
BP3	v=1, [1,2,3,4,5]	1	BP3
BP4	v =5, [1,2,3,4,5]	1	BP4
BP5	v=-3,[-5,-4,-3,-2,-1]	1	BP5

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 Classes:

Class 1: Empty array ➜ Output: 0

Class 2: Value exists once ➜ Output: 1

Class 3: Value exists multiple times ➜ Output: Count of occurrences

(e.g., n if v appears n times)

Class 4: Value does not exist ➜ Output: 0

Class 5: All elements are equal to v ➜ Output: Length of the array

Test Cases :

Input (v, a)	Expected Output	Covers Equivalence Class
(7, [])	0	1
(6, [3, 6, 9])	1	2
(10, [1, 2, 4])	0	4
(2, [2, 2, 2, 2])	4	5
(4, [4, 4, 4, 5])	3	3
(12, [5, 6, 8])	0	4
(11, [11, 11, 11])	3	5
(5, [0, 1, 5, 5])	2	3
(6, [6, 7, 8, 9])	1	2

(3, [1, 2, 3, 3, 3])	3	3

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.

```
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 Classes for binarySearch:

1. <u>Value Present:</u>

○ E1:Thevaluevispresent in the array and is located at the first position.

○ E2:Thevaluevispresent in the array and is located at the last position.

○ E3:Thevaluevispresent in the array and is located somewhere in the middle.

2. Value Not Present:

○ E4:The Value Is Less than the smallest element in the array.

○ E5:Thevaluevisgreater than the largest element in the array.

○ E6:Thevaluevisnotinthearray but falls between two elements.

3. Array EdgeCases:

○ E7:The Array Isempty.

○ E8:The Array Contains one element, which may or maynot be equal to v.

Equivalence Classes :-

Test Case	Input	Expected Output	Equivalence Boundary
TC1	v = 7, [1, 7, 3, 4, 5]	1	E1
TC2	v = 10, [2, 4, 6, 8, 10]	4	E2
TC3	v = 3, [3, 3, 3]	0	E3
TC4	v = 0, [1, 2, 3, 4, 5]	-1	E4
TC5	v = 9, [1, 2, 3, 4, 5]	-1	E5
TC6	v = 2.5, [1, 2, 4, 5]	-1	E6

TC7	v = 1, []	-1	E7

Boundary Points for binarySearch:

1. BP1: Single-element array where v is equal to the element.
2. BP2: Single-element array where v is not equal to the element.
3. BP3: The value v is at the first position in a multi-element sorted array.
4. BP4: The value v is at the last position in a Multi-element sorted array.
5. BP5: The Array Contains Duplicate Values Of v.

Test Case	**Input**	**Expected Output**	**Equivalence Boundary**
BP1	v=3,[3]	0	BP1
BP2	v=2, [3]	-1	BP2
BP3	v=1, [1,2,3,4,5]	0	BP3
BP4	v =5, [1,2,3,4,5]	4	BP4
BP5	v=-3,[-5,-4,-3,-2,-1]	2	BP5

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 Classes :

Class 1: Invalid triangle (non-positive sides) → Output: `INVALID`

Class 2: Invalid triangle (triangle inequality not satisfied) → Output: `INVALID`

Class 3: Equilateral triangle (all sides equal) → Output: `EQUILATERAL`

Class 4: Isosceles triangle (two sides equal) → Output: `ISOSCELES`

Class 5: Scalene triangle (all sides different) → Output: `SCALENE`

Test Cases :

Input (a, b, c)	Expected Outcome	Covers Equivalence Class
(2, 2, 2)	EQUILATERAL	3
(3, 3, 6)	ISOSCELES	4
(5, 7, 9)	SCALENE	5
(1, 1, 3)	INVALID	1
(0, 4, 5)	INVALID	1
(-1, 2, 2)	INVALID	1
(6, 6, 12)	INVALID	2
(8, 10, 12)	SCALENE	5
(7, 7, 14)	ISOSCELES	4
(4, 5, 9)	INVALID	2

P5 - The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2.

```
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 Classes :

- **Class 1**: s1 is longer than s2 → Output: `false`
- **Class 2**: s1 is an exact prefix of s2 → Output: `true`
- **Class 3**: s1 is a partial prefix of s2 → Output: `false`
- **Class 4**: s1 is empty → Output: `true`
- **Class 5**: s2 is empty and s1 is not → Output: `false`
- **Class 6**: s1 is equal to s2 → Output: `true`

Test Cases:

Input (s1, s2)	Expected Outcome	Covers Equivalence Class
("long", "short")	false	1
("hello", "hello!")	true	2
("greeting", "")	false	5
("ab", "abc")	true	6
("longerPrefix", "short")	false	1
("abc", "abcde")	true	2
("match", "mat")	false	3
("empty", "empty")	true	6

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

>> Valid Triangle Types:

Equilateral Triangle: Side A = Side B = Side C

Isosceles Triangle: Side A = Side B, or Side A = Side C, or Side B = Side C

Scalene Triangle: All sides unequal (A ≠ B ≠ C)

Right-Angled Triangle: $A^2 + B^2 = C^2$ (Pythagorean theorem) or its permutations

>> Invalid Triangle Cases:

Not a Triangle: A + B ≤ C, A + C ≤ B, or B + C ≤ A

Non-positive Input: 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.

Input (A, B, C)	Expected Output	Equivalence Classes Covered
(3.0, 3.0, 3.0)	Equilateral Triangle	Equilateral Triangle
(6, 6, 10)	Isosceles Triangle	Isosceles Triangle
(7, 8, 9)	Scalene Triangle	Scalene Triangle
(9, 12, 15)	Right-Angled Triangle	Right-Angled Triangle
(2, 3, 10)	Not a Triangle	Not a Triangle
(-1, 5, 7)	Invalid	Non-positive Input

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

Input (A, B, C)	Expected Output
(3, 4, 5)	Right-Angled Triangle
(8, 10, 12)	Scalene Triangle
(2, 2, 5)	Not a Triangle
(7, 7, 10)	Isosceles Triangle

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

Input (A, B, C)	Expected Output
(7.0, 8.0, 7.0)	Isosceles Triangle
(39.0, 40.0, 40.0)	Isosceles Triangle
(5, 9, 14)	Not a Triangle
(9, 9, 9)	Equilateral Triangle

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

Input (A, B, C)	Expected Output
(6, 6, 6)	Equilateral Triangle
(3, 3, 3)	Equilateral Triangle
(7, 8, 14)	Not a Triangle
(6, 9, 13)	Scalene Triangle

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

Input (A, B, C)	Expected Output
(6, 8, 10)	Right-Angled Triangle
(5, 12, 13)	Right-Angled Triangle
(6, 9, 14)	Not a Triangle
(7, 10, 12)	Scalene Triangle

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

Input (A, B, C)	Expected Output
(5, 6, 7)	Scalene Triangle
(1.0, 2.0, 3.0)	Not a Triangle
(10.0, 1.0, 1.0)	Not a Triangle
(6, 8, 14)	Scalene Triangle

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

Input (A, B, C)	Expected Output
(0.0, 1.0, 1.0)	Invalid
(5, 7, -3)	Invalid
(1.0, 0.0, 1.0)	Invalid
(-4, 6, 9)	Invalid