IT314 Software Engineering

202201512 - Chirag Patel

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.

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.

->For month : 1 > month , 12 < month is invalid.
1 <= month <= 12 is valid.
->For day : 1 > day, 31 < day is invalid.
1 <= day<= 31 is valid.
->For year : 9000 > Year, 2015 < Year is not valid.
9000 <= year <= 2015 is valid.

Test Case	Day	Month	Year	Expected Outcome
TC1	15	5	2010	Previous date: 14/05/2010
TC2	32	5	2010	Invalid date
TC3	15	13	2010	Invalid date
TC4	15	5	1899	Invalid date
TC5	32	13	2010	Invalid date
TC6	15	13	1899	Invalid date
TC7	32	5	1899	Invalid date
TC8	32	13	1899	Invalid date
TC9	1	1	1900	Previous date: 31/12/1899

TC10	31	12	2015	Previous date: 30/12/2015
TC11	29	2	2000	Previous date: 28/02/2000
TC12	30	2	2001	Invalid date
TC13	1	1	1900	Previous date: 31/12/1899
TC14	31	12	2015	Previous date: 30/12/2015
TC15	1	3	2000	Previous date: 29/02/2000
TC16	29	2	2001	Invalid date
TC17	30	4	2010	Invalid date
TC18	31	1	2015	Previous date: 30/01/2015

Code :

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

bool isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

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 0;
    }
}

string validateDate(int day, int month, int year) {
```

```
    if (year < 1900 || year > 2015) {
        return "Invalid date: Year out of range.";
    }
    if (month < 1 || month > 12) {
        return "Invalid date: Month out of range.";
    }
    int maxDays = daysInMonth(month, year);
    if (day < 1 || day > maxDays) {
        return "Invalid date: Day out of range.";
    }
    return "Valid date";
}

int main() {
    int date, month, year;
    cout << "Enter date as following 'date month year' : "
    cin >> date >> month >> year;

    cout << validateDate(date, 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 Classes (Valid and Invalid):

1. Class 1: The array is non-empty, and the value v is present in the array.
2. Class 2: The array is non-empty, and the value v is not present in the array.
3. Class 3: The array contains multiple occurrences of the value v.
4. Class 4: The array has a single element, and that element is equal to v.
5. Class 5: The array has a single element, and that element is not equal to v.
6. Class 6: The array is empty.
7. Class 7: The array is non-integer or non-numeric or the value v is of an invalid type.

Test cases:

Test Case	Input v	Input a[]	Valid/Invalid
TC1	3	{1, 2, 3, 4, 5}	Valid
TC2	6	{1, 2, 3, 4, 5}	Valid
TC3	2	{1, 2, 2, 2, 5}	Valid
TC4	2	{2}	Valid
TC5	1	{2}	Valid
TC6	3	{}	Invalid (edge case)
TC7	3	{3.5, 1, 2}	Invalid
TC8	a'	{1, 2, 3}	Invalid
TC9	5	{1, 2, 3, 4, 5}	Valid
TC10	0	{0, 1, 2, 3}	Valid

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:

1. Class 1: The array is non-empty, and the value v appears multiple times.
2. Class 2: The array is non-empty, and the value v does not appear at all.
3. Class 3: The array has only one element, and that element is equal to v.
4. Class 4: The array has only one element, and that element is not equal to v.
5. Class 5: The array is non-empty, and all elements are equal to v.
6. Class 6: The array is empty.
7. Class 7: The array or value v is of an invalid type.

Test cases :

Test Case	Input v	Input a[]	Valid/Invalid
TC1	2	{1, 2, 2, 2, 3}	Valid
TC2	5	{1, 2, 3, 4}	Valid
TC3	2	{2}	Valid
TC4	1	{2}	Valid
TC5	7	{7, 7, 7, 7}	Valid
TC6	3	{}	Invalid
TC7	a'	{1, 2, 3}	Invalid
TC8	2	{2.5, 1, 2}	Invalid

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 Classes:

1. Class 1: The array is non-empty, and the value v exists in the array.
2. Class 2: The array is non-empty, and the value v does not exist in the array.
3. Class 3: The array contains only one element, and that element is equal to v.
4. Class 4: The array contains only one element, and that element is not equal to v.
5. Class 5: The array is empty.
6. Class 6: The array is not sorted in non-decreasing order.
7. Class 7: The array or value v is of an invalid type.

Test Case	Input v	Input a[]	Valid/Invalid
TC1	4	{1, 2, 3, 4, 5, 6}	Valid
TC2	7	{1, 2, 3, 4, 5, 6}	Valid
TC3	2	{2}	Valid
TC4	3	{2}	Valid
TC5	4	{}	Invalid
TC6	4	{5, 4, 3, 2, 1}	Invalid
TC7	a′	{1, 2, 3, 4, 5}	Invalid
TC8	3	{1, 2.5, 3}	Invalid

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:

1. Class 1: The triangle is equilateral (all three sides are equal).
2. Class 2: The triangle is isosceles (two sides are equal, one is different).
3. Class 3: The triangle is scalene (all three sides are different).
4. Class 4: The sides do not satisfy the triangle inequality (a + b <= c or similar).
5. Class 5: One or more sides are non-positive (a <= 0, b <= 0, or c <= 0).

Test cases:

Test Case	Input (a, b, c)	Valid/Invalid
TC1	(3, 3, 3)	Valid
TC2	(5, 5, 8)	Valid
TC3	(4, 5, 6)	Valid
TC4	(10, 3, 3)	Invalid
TC5	(0, 4, 5)	Invalid
TC6	(-1, 2, 2)	Invalid
TC7	(1, 1, 2)	Invalid
TC8	(2, 2, 5)	Invalid
TC9	(1, 2, 3)	Invalid
TC10	(3, 3, 1)	Valid

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 Classes:

1. Class 1: s1 is a prefix of s2.
2. Class 2: s1 is not a prefix of s2 (but s1.length() <= s2.length()).s1.
3. Class 3: s1 is an empty string.
4. Class 4: s1 is longer than s2.

Test cases:

Test Case	Input s1	Input s2	Valid/Invalid
TC1	"pre"	"prefix"	Valid
TC2	"fix"	"prefix"	Valid
TC3	""	"prefix"	Valid
TC4	"longer"	"short"	Invalid
TC5	"pre"	"pre"	Valid
TC6	"abc"	"a"	Invalid

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

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)

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

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

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

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

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

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

a) Identify the Equivalence Classes for the System2
Valid Input Classes:

- Class 1: Equilateral Triangle: A = B = C
- Class 2: Isosceles Triangle: Two sides are equal (A=B≠C , A=C≠B, B=C≠A)
- Class 3: Scalene Triangle: All sides are different (A≠B≠CA)
- Class 4: Right-Angled Triangle: Satisfies A^2+B^2=C^2 (with sides arranged such that CCC is the longest)
- Class 5: Non-Triangle: Does not satisfy triangle inequality (A+B≤C)
- Class 6: Non-Positive Input: Any side is less than or equal to zero (A≤0A, B≤0B, or C≤0C)

b) Identify Test Cases to Cover the Identified Equivalence Classes

Test Case	Input (A, B, C)	Valid/Invalid
TC1	(3.0, 3.0, 3.0)	Valid
TC2	(5.0, 5.0, 8.0)	Valid
TC3	(4.0, 5.0, 6.0)	Valid
TC4	(3.0, 4.0, 5.0)	Valid
TC5	(10.0, 3.0, 3.0)	Invalid
TC6	(0.0, 4.0, 5.0)	Invalid
TC7	(-1.0, 2.0, 2.0)	Invalid
TC8	(1.0, 1.0, 2.0)	Invalid
TC9	(2.0, 2.0, 2.0)	Valid
TC10	(5.0, 12.0, 13.0)	Valid

c) Boundary Condition: A + B > C (Scalene Triangle)

Test Case	Input (A, B, C)	Condition
BC1	(3.0, 4.0, 5.0)	A + B > C (valid scalene triangle)
BC2	(2.0, 2.0, 4.0)	A + B = C (boundary for non-triangle)
BC3	(3.0, 3.0, 5.0)	A + B > C (valid scalene triangle)

d) Boundary Condition: A = C (Isosceles Triangle)

Test Case	Input (A, B, C)	Condition
BC1	(3.0, 3.0, 4.0)	A = B (valid isosceles triangle)
BC2	(5.0, 5.0, 10.0)	A = B, but A + B = C (invalid)
BC3	(6.0, 6.0, 9.0)	A = B (valid isosceles triangle)

e) Boundary Condition: A = B = C (Equilateral Triangle)

Test Case	Input (A, B, C)	Condition
BC1	(3.0, 3.0, 3.0)	A = B = C (valid equilateral triangle)
BC2	(4.0, 4.0, 4.0)	A = B = C (valid equilateral triangle)

f) Boundary Condition: $A^2 + B^2 = C^2$ (Right-Angled Triangle)

Test Case	Input (A, B, C)	Condition
BC1	(3.0, 4.0, 5.0)	$A^2 + B^2 = C^2$ (valid right-angled triangle)
BC2	(5.0, 12.0, 13.0)	$A^2 + B^2 = C^2$ (valid right-angled triangle)
BC3	(1.0, 1.0, sqrt(2))	$A^2 + B^2 = C^2$ (valid right-angled triangle)

g) Non-Triangle Case: Identify Test Cases to Explore the Boundary

Test Case	Input (A, B, C)	Condition
TC1	(1.0, 2.0, 3.0)	A + B = C (non-triangle)
TC2	(5.0, 2.0, 3.0)	A + B < C (non-triangle)
TC3	(10.0, 5.0, 4.0)	A + B > C (valid triangle, but check other inputs)

h) Non-Positive Input: Identify Test Points

Test Case	Input (A, B, C)	Condition
TC1	(0.0, 4.0, 5.0)	A = 0 (invalid)
TC2	(4.0, 0.0, 5.0)	B = 0 (invalid)
TC3	(4.0, 5.0, -1.0)	C < 0 (invalid)
TC4	(-3.0, 4.0, 5.0)	A < 0 (invalid)