Dhirubhai Ambani
Institute of Information and Communication Technology

Lab 08

IT314 Software Engineering

Divyarajsinh Chundavat - 202201155

Que-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.

Ans:

<u>Derived equivalent classes:</u>

1. Input is triple of day, month and year.(Valid)
2. Input is not triple of day, month and year (i.e. Either day, month or year is missing). (Invalid)
3. Day contains only numbers. (Valid)
4. Day contains alphabets. (Invalid)
5. Day value is between 1 to 31 included. (Valid)
6. Day value is smaller than one. (Invalid)
7. Day value is greater than thirty one. (Invalid)
8. Month contains only numbers. (Valid)
9. Month contains alphabets. (Invalid)
10. Month value is between 1 to 12 included. (Valid)
11. Month value is smaller than one. (Invalid)
12. Month value is greater than twelve. (Invalid)
13. Year contains only numbers. (Valid)
14. Year contains alphabets. (Invalid)
15. Year value is between 1900 to 2015 included. (Valid)
16. Year value is smaller than nineteen hundred. (Invalid)
17. Year value is greater than twenty fifteen. (Invalid)
18. Date is greater than twenty eight and month is two in case of non-leap year. (???)
19. Date is greater than twenty nine and month is two in case of leap year. (???)

- Equivalence class partitioning

Sr. No.	Input	Expected Outcome	Classes Covered	Remarks
1	xy,12,1877	Invalid	4, 15, 16	Day should be number and year should be b/w 1900 to 2015
2	1,xz,1900	Invalid	8, 9	Month should be number
3	_, _, 1995	Invalid	2	No value should be missing
4	30,2,2012	Valid	1, 3, 5, 8, 10, 13, 15, 19	Valid input provided
5	0,14,1990	Invalid	1, 3, 8, 13, 6, 12, 15	Day<1 & Month>12
6	55,-1,2050	Invalid	1, 3, 8, 13, 7, 11, 17	Day>31, Month<1 & Year>2015
7	4,5,20xy	Invalid	14	Year contain alphabets
8	4,5,2005	Valid	1, 3, 5, 8, 10, 13, 15	Perfect input
9	_, pz, 101	Invalid	2, 9, 13, 16	Day not given, Month value is NaN, Year < 1900

- **<u>Boundary value analysis</u>**

Sr. No.	Input	Exp Outcome	Remarks
1	0,10,1990	Invalid	Day<1
2	2,10,1999	Valid	-
3	29,2,2001	Invalid	February has 28 days in 2001
4	29,2,2020	Valid	-
5	32,10,2001	Invalid	Boundary value for day
6	1,13,1899	Invalid	Boundary exceed for month and year
7	1,0,2016	Invalid	Boundary exceed for month and year
8	31,4,2005	Valid	Valid boundaries for 31 day month

<u>Code:</u>

```
#include <bits/stdc++.h>
using namespace std;

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

   return false;
}

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() {
   int DAY, MONTH, YEAR;
   scanf("%d%d%d",&DAY,&MONTH,&YEAR);
   previousDate(DAY, MONTH, YEAR);
   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: The value v is present in the array and occurs once.
- E2: The value v is present in the array and occurs multiple times.
- E3: The value v is not present in the array.

Array Edge Cases:

- E4: The array is empty.

Equivalent Classes:

Test Case	Input	Expected Outcome	Equivalence Boundary
TC1	v=5,[1,2,3,4,5]	4	E1
TC2	v=3,[1,3,3,3,5]	1	E2
TC3	v=-1,[]	-1	E3
TC4	v=0,[1,1,1,2]	-1	E4
TC5	v=2,[2,1,4,5]	0	E5

Boundary Points:

- BP1: Single-element array where v is present
- BP2: Single-element array where v is not present
- BP3: v is at the first position
- BP4: v is at the last position
- BP5: Array contains negative numbers and v is a negative number

Test Case	Input	Expected Outcome	Equivalence Boundary
BP1	v=3,[3]	0	BP1
BP2	v=2,[1]	1	BP2
BP3	v=-1,[-1,-1,2,3]	0	BP3
BP4	v=0,[1,1,1,2,0]	4	BP4
BP5	v=-4,[2,1,-4,5]	2	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);
}
```

Value Present:

- E1: The value v is present in the array and occurs once.
- E2: The value v is present in the array and occurs multiple times.
- E3: The value v is not present in the array.

Array Edge Cases:

- E4: The array is empty.
- E5: The value v is at the first or last position in the array.

Equivalence Classes:

Test Case	Input	Expected Outcome	Equivalence Boundary
TC1	v=5,[1,2,3,4,5]	1	E1
TC2	v=3,[1,3,3,3,5]	3	E2
TC3	v=-1,[]	0	E3
TC4	v=0,[1,1,1,2]	0	E4
TC5	v=2,[2,1,4,5]	1	E5

Boundary Points for countItem:

- BP1: Single-element array where v is present.
- BP2: Single-element array where v is not present.
- BP3: v is at the first position.
- BP4: v is at the last position.
- BP5: Array contains negative numbers and v is a negative number

Test Case	Input	Expected Outcome	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=-4,[-5,-4,-3,-2,-1]	1	BP5

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 for binarySearch:

1. Value Present:

- E1: The value v is present in the array and is located at the first position.
- E2: The value v is present in the array and is located at the last position.
- E3: The value v is present in the array and is located somewhere in the middle.

2. Value Not Present:

- E4: The value v is less than the smallest element in the array.
- E5: The value v is greater than the largest element in the array.
- E6: The value v is not in the array but falls between two elements.

3. Array Edge Cases:

- E7: The array is empty.
- E8: The array contains one element, which may or may not be equal to v.

Test Case	Input	Expected Outcome	Equivalence Boundary
TC1	v=1,[1,2,3,4,5]	0	E1
TC2	v=5,[1,2,3,4,5]	4	E2
TC3	v=3,[1,2,3,4,5]	2	E3
TC4	v=0,[1,2,3,4,5]	-1	E4
TC5	v=6,[1,2,3,4,5]	-1	E5
TC6	v=2.5,[1,2,3,4,5]	-1	E6
TC7	v=3,[]	-1	E7
TC8	v=1,[1]	0	E8
TC9	v=2,[1]	-1	E9

<u>Boundary Points for binarySearch:</u>

- BP1: Single-element array where v is equal to the element.
- BP2: Single-element array where v is not equal to the element.
- BP3: The value v is at the first position in a multi-element sorted array.
- BP4: The value v is at the last position in a multi-element sorted array.
- BP5: The array contains duplicate values of v.

Test Case	Input	Expected Outcome	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,[1,2,3,3,4,5]	2	BP5

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).

Valid Prefix Cases:

- E1: s1 is a non-empty string and is a prefix of s2.
- E2: s1 is an empty string, which is considered a prefix of any string s2.
- E3: s1 is equal to s2.

Invalid Prefix Cases:

- E4: s1 is longer than s2.
- E5: s1 is not a prefix

Equivalence Classes:

Test Case	Input	Expected Outcome	Equivalence Boundary
TC1	s1=”pre”, s2=”prefix”	true	E1
TC2	s1=””, s2=”anything”	true	E2
TC3	s1=”same”, s2=”same”	true	E3
TC4	s1=”longer”, s2=”shorter”	false	E4
TC5	s1=”prefix”, s2=”pre”	false	E5

<u>Boundary Points for prefix Function:</u>

- BP1: s1 is a single character and is a prefix of s2.
- BP2: s1 is a single character and is not a prefix of s2.
- BP3: s1 is an empty string and s2 is a non-empty string.
- BP4: s1 is equal to s2, which also has one character.
- BP5: s1 is the same as the first few characters of s2, but does not cover the entire s2.

Test Case	Input	Expected Outcome	Equivalence Boundary
BP1	s1=”p”, s2=”prefix”	true	BP1
BP2	s1=”x”, s2=”prefix”	false	BP2
BP3	s1=””, s2=”hello”	true	BP3
BP4	s1=”a”, s2=”a”	true	BP4
BP5	s1=”pre”, s2=”prefix”	true	BP5

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.

Ans:
a) Identify the equivalence classes for the system

The equivalence classes for this system can be divided into valid and invalid triangles based on t he properties of the triangle:

Valid Triangle:

- **Equilateral Triangle (E1):** All sides are equal: A=B=C
- **Isosceles Triangle (E2):** Two sides are equal: A=B, B=C, or C=A
- **Scalene Triangle (E3):** All sides are different: A≠B, B≠C, A≠C
- **Right-angled Triangle (E4):** Follows the Pythagorean theorem A2+B2=C2
  With A≤B≤C

Invalid Triangle:

- **Non-Triangle Case (I1):** Sum of two sides is less than or equal to the third side: A+B≤C or A+C≤B or B+C≤A
- **Non-Positive Inputs (I2):** One or more sides have non-positive values: A≤0, B≤0, C≤ 0.

b) Identify test cases to cover the identified equivalence classes.

Test Case	Input	Expected Outcome	Equivalence Class
TC1	5, 5, 5	Equilateral	E1
TC2	5, 5, 3	Isosceles	E2
TC3	5, 4, 3	Scalene	E3
TC4	3, 4, 5	Right Angled	E4
TC5	1, 2, 3	Invalid	I1
TC6	0, 3, 4	Invalid	I2
TC7	-1, 2, 3	Invalid	I2

TC8	5, 5, 10	Invalid	I1
TC9	5, 6, 10	Scalene	E3
TC10	7, 7, 10	Isosceles	E2

c) Boundary condition A+B > C (Scalene triangle)

Test Case	Input	Expected Outcome	Boundary Cond.
BC1	1, 2 , 3	Invalid	A+B=C
BC2	7, 9, 10	Scalene	A+B>C
BC3	3, 4, 6	Scalene	A+B>C

d) For the boundary condition A = C case (isosceles triangle)

Test Case	Input	Expected Outcome	Boundary Cond.
BC4	1, 2 , 2	Isosceles	B=C
BC5	7, 9, 7	Isosceles	A=C
BC6	2, 2, 3	Isosceles	B=A
BC7	1, 1, 3	Invalid	A+B>C

e) For the boundary condition A = B = C case (equilateral triangle)

Test Case	Input	Expected Outcome	Boundary Cond.
BC8	3, 3, 3	Equilateral	A=B=C
BC9	1, 1, 3	Invalid	A+B>C

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle)

Test Case	Input	Expected Outcome	Boundary Cond.

BC10	3, 4, 5	Right Angled	$A^2+B^2=C^2$
BC11	10, 8, 6	Right Angled	$C^2+B^2=A^2$

g) For the non-triangle case input test points

Test Case	Input	Expected Outcome	Boundary Cond.
BC12	1, 2, 3	Invalid	$A + B = C$
BC13	2, 2, 5	Invalid	$A + B < C$

h) For non-positive input test points

Test Case	Input	Expected Outcome	Boundary Cond.
BC14	0, 3, 4	Invalid	$A=0$
BC15	-1, 0, 2	Invalid	$B=C$ & $A<0$
BC16	1, -1, 0	Invalid	$B<0$ & $C=0$
BC17	1, 12, -1	Invalid	$C<0$