IT314 : Software Engineering

Software Testing
Lab - 8 : Functional Testing (Black-Box)

Hemal Ravrani
202201235

Question 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.

The solution of each problem must be given in the format as follows:

Tester Action and Input Data	**Expected Outcome**
Equivalence Partitioning	
a, b, c	An Error message
a-1, b, c	Yes
Boundary Value Analysis	
a, b, c-1	Yes

Answer:

Equivalence Partitioning Test Cases

Valid partitions:

- **Month:** `1 <= month <= 12`
- **Day:**
  - **For months with 31 days:** `1 <= day <= 31`
  - **For months with 30 days:** `1 <= day <= 30`
  - **For February (non-leap year):** `1 <= day <= 28`
  - **For February (leap year):** `1 <= day <= 29`

- **Year:** `1900 <= year <= 2015`

Invalid partitions:

- **Month:** `month < 1 or month > 12`
- **Day:** `day < 1 or day > max days in month` **(where max days depend on the month and leap year)**
- **Year:** `year < 1900 or year > 2015`

Test Case	Input (Month, Day, Year)	Expected Outcome
TC_001	Valid Input (5,15,2010)	5/14/2010
TC_002	Month less than 1 (0,15,2010)	Invalid Month
TC_003	Month greater than 12 (13,15,2010)	Invalid Month
TC_004	Day less than 1 (4,0,2010)	Invalid Day

TC_00 5	Day greater than 31 (7,32,2010)	Invalid Day
TC_00 6	Year less than 1900 (5,10,1899)	Invalid Year
TC_00 7	Valid leap year input (2,29,2012)	2/28/201 2
TC_008	Invalid day in February (2,30,2011)	Invalid Day
TC_00 9	Year greater than 2015 (6,10,2020)	Invalid Year
TC_01 0	Valid input end of year (12,1,2015)	11/30/201 5

Boundary Value Analysis Test Cases

Test Case	Input (Month, Day, Year)	Expected Outcome
TC_001	Valid Input (1,1,1900)	Invalid (No previous date in range)
TC_002	Valid Input (12,31,2015)	12/30/2015
TC_003	Day just below lower boundary (3,0,2010)	Invalid Day
TC_004	Day on lower boundary (3,1,2010)	2/28/2010
TC_005	Day just above lower boundary (3,2,2010)	3/1/20 10
TC_006	Day on upper boundary (5,31,2010)	5/30/2010
TC_007	Day just above upper boundary (5,32,2010)	Invalid Day
TC_008	Month on lower boundary (1,15,2010)	1/14/2010
TC_009	Month just below lower boundary (0,15,2010)	Invalid Month

TC_010	**Month on upper boundary (12,15,2010)**	**12/14/2010**
TC_011	**Month just above upper boundary (13,15,2010)**	**Invalid Month**
TC_012	**Year on lower boundary (1,1,1900)**	**Invalid (No previous date in range)**
TC_013	**Year just above upper boundary (12,31,2015)**	**12/30/2015**

Code:

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

// Function to check if a year is a leap year
bool isLeapYear(int year)
{
    if (year % 4 == 0)
    {
        if (year % 100 == 0)
        {
            if (year % 400 == 0)
                return true;
            else
                return false;
        }
        else
            return true;
    }
    return false;
}
```

```
// Function to determine 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 validate the date
bool isValidDate(int day, int month, int year)
{
    if (year < 1900 || year > 2015)
        return false;
    if (month < 1 || month > 12)
        return false;
    if (day < 1 || day > daysInMonth(month, year))
```

```
        return false;
    return true;
}

// Function to determine the previous date
void previousDate(int &day, int &month, int &year)
{
    if (day > 1)
    {
        day--;
    }
    else
    {
        if (month == 1)
        {
            month = 12;
            year--;
            day = 31;
        }
        else
        {
            month--;
            day = daysInMonth(month, year);
        }
    }
}

int main()
{
    int day, month, year;

    // Input date
    cout << "Enter day: ";
    cin >> day;
    cout << "Enter month: ";
    cin >> month;
    cout << "Enter year: ";
    cin >> year;

    // Check if the input date is valid
    if (!isValidDate(day, month, year))
    {
        cout << "Invalid date!" << endl;
    }
    else
    {
        // Determine the previous date
        previousDate(day, month, year);

        // Print the previous date
        if (year < 1900)
        {
            cout << "Invalid: No previous date available in
range!" << endl;
        }
        else
        {
            cout << "Previous date: " << day << "/" << month
<< "/" << year << endl;
        }
    }

    return 0;
}
```

Question 2:

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 (EP) Test Cases :

Test Case	Input (Value v, Array a[], Size n)	Expected Outcome
TC_001	v = 5, a[] = {1, 5, 9, 15}, n = 4	1 (index of 5)
TC_002	v = 9, a[] = {2, 4, 6, 9, 12}, n = 5	3 (index of 9)
TC_003	v = 20, a[] = {2, 3, 5, 8}, n = 4	-1 (not found)
TC_004	v = 0, a[] = {1, 3, 7, 10}, n = 4	-1 (not found)
TC_005	v = 15, a[] = {1, 15}, n = 2	1 (index of 15)

TC_006	`v = -5`, `a[] = {-10, -5, 0, 5}`, `n = 4`	`1` (index of `-5`)
TC_007	`v = 5`, `a[] = {5}`, `n = 1`	`0` (index of `5`)
TC_008	`v = 3`, `a[] = {3, 3, 3, 3}`, `n = 4`	`0` (first index)

Boundary Value Analysis (BVA) Test Cases :

Test Case	Input (Value v, Array a[], Size n)	Expected Outcome
TC_001	v = 1, a[] = {1, 2, 3, 4, 5}, n = 5	0 (first index)
TC_002	v = 5, a[] = {1, 2, 3, 4, 5}, n = 5	4 (last index)
TC_003	v = 3, a[] = {1, 2, 3}, n = 3	2 (last index)
TC_004	v = 10, a[] = {10}, n = 1	0 (only element)
TC_005	v = 1, a[] = {2, 4, 6}, n = 3	-1 (not found)
TC_006	v = 2, a[] = {2}, n = 1	0 (first and last)
TC_007	v = 0, a[] = {1, 3, 5}, n = 3	-1 (not found)
TC_008	v = -10, a[] = {-10, -5, 0, 5}, n = 4	0 (first index)

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 (EP) Test Cases :

Input Data	Expected Output
countItem(4, [4, 2, 9, 4, 15], 5)	Returns 2
countItem(8, [1, 2, 3, 4, 5], 5)	Returns 0
countItem(9, [], 0)	Returns 0 (Empty Array)
countItem(1, [1], 1)	Returns 1
countItem(3, [3, 3, 3], 3)	Returns 3
countItem(9876543210, [1, 2, 3, 4, 5], 5)	Returns -1 and prints error: "v exceeds range"
countItem(2, [9876543210, 2, 3, 5], 4)	Returns -1 and prints error: "Array element exceeds range"
countItem(1, largeArray, 500001)	Returns -1 and prints error: "Array size exceeds limit"

Boundary Value Analysis (BVA) Test Cases :

Input Data	Expected Output
countItem(4, [4, 2, 9, 4, 15], 5)	Returns 2

countItem(15, [4, 2, 9, 4, 15], 5)	Returns 1
countItem(9, [4, 2, 9, 4, 15], 5)	Returns 1
countItem(8, [1, 2, 3, 4, 5], 5)	Returns 0
countItem(9, [], 0)	Returns 0
countItem(1, [1], 1)	Returns 1
countItem(5, [2], 1)	Returns 0
countItem(2, largeArray, 500001)	Returns -1 and prints error: "Array size exceeds limit"
countItem(9876543210, [4, 2, 9, 4, 15], 5)	Returns -1 and prints error: "v exceeds range"
countItem(2, [9876543210, 2, 9, 5], 4)	Returns -1 and prints error: "Array element exceeds range"

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 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 (EP) Test Cases :

Input Data	Expected Output
binarySearch(5, [1, 3, 5, 7, 9], 5)	Returns 2
binarySearch(6, [2, 4, 6, 8, 10], 5)	Returns 2
binarySearch(11, [2, 3, 5, 7, 9], 5)	Returns -1 (Value not present in the array)
binarySearch(2, [], 0)	Returns -1 (Empty Array)
binarySearch(7, [1], 1)	Returns -1 (Single element not matching)
binarySearch(4, [1, 3, 4, 4, 5], 5)	Returns 2
binarySearch(2147483648, [1, 2, 3, 4, 5], 5)	Returns -1 and prints error: "v exceeds range"
binarySearch(1, [2147483648, 2, 3, 4, 5], 5)	Returns -1 and prints error: "Array element exceeds range"
binarySearch(2, largeArray, 1000001)	Returns -1 and prints error: "Array size exceeds limit"

Boundary Value Analysis (BVA) Test Cases :

Input Data	Expected Output
binarySearch(5, [1, 3, 5, 7, 9], 5)	Returns 2
binarySearch(9, [1, 3, 5, 7, 9], 5)	Returns 4
binarySearch(1, [1, 3, 5, 7, 9], 5)	Returns 0

binarySearch(0, [1, 2, 3, 4, 5], 5)	Returns -1
binarySearch(7, [7], 1)	Returns 0
binarySearch(3, [3], 1)	Returns 0
binarySearch(5, [1, 3, 5], 3)	Returns 2
binarySearch(2, largeArray, 1000001)	Returns -1 and prints error: "Array size exceeds limit"
binarySearch(2147483648, [1, 3, 5, 7, 9], 5)	Returns -1 and prints error: "v exceeds range"
binarySearch(1, [2147483648, 2, 3, 4, 5], 5)	Returns -1 and prints error: "Array element exceeds range"

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 (EP) Test Cases :

Input Data	Expected Output
triangle(3, 3, 3)	Returns EQUILATERAL (0)
triangle(3, 4, 3)	Returns ISOSCELES (1)
triangle(3, 4, 5)	Returns SCALENE (2)
triangle(1, 2, 3)	Returns INVALID (3) (Not a triangle)
triangle(0, 1, 1)	Returns INVALID (3) (Not a triangle)
triangle(-1, 1, 1)	Returns INVALID (3) (Not a triangle)
triangle(5, 5, 10)	Returns INVALID (3) (Not a triangle)
triangle(1, 1, 1)	Returns EQUILATERAL (0)
triangle(4, 4, 5)	Returns ISOSCELES (1)
triangle(2147483648, 1, 1)	Returns INVALID (3) and prints error: "Input exceeds range"

Boundary Value Analysis (BVA) Test Cases :

Input Data	Expected Output
triangle(3, 3, 3)	Returns EQUILATERAL (0)
triangle(2, 2, 4)	Returns INVALID (3) (Not a triangle)
triangle(3, 4, 5)	Returns SCALENE (2)
triangle(1, 1, 0)	Returns INVALID (3) (Not a triangle)
triangle(3, 3, 2)	Returns ISOSCELES (1)
triangle(2, 2, 2)	Returns EQUILATERAL (0)
triangle(1, 1, 3)	Returns INVALID (3) (Not a triangle)
triangle(2147483648, 1, 1)	Returns INVALID (3) and prints error: "Input exceeds range"
triangle(0, 1, 1)	Returns INVALID (3) (Not a triangle)
triangle(-1, -1, -1)	Returns INVALID (3) (Not a triangle)

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 (EP) Test Cases:

Input Data	Expected Output
prefix("cat", "catalog")	Returns true
prefix("cat", "cat")	Returns true
prefix("cat", "dog")	Returns false
prefix("cat", "caterpillar")	Returns true
prefix("cat", "ca")	Returns false
prefix("cat", "category")	Returns true
prefix("abc", "aBc")	Returns false
prefix("x" * 1000000, "x" * 1000000 + "y")	Returns true

prefix("x" * 1000000 + "y", "x" * 1000000)	Returns false
prefix("x" * Integer.MAX_VALUE, "x" * Integer.MAX_VALUE)	May cause OutOfMemoryError or Overflow
prefix("x" * (Integer.MAX_VALUE - 1), "x" * Integer.MAX_VALUE)	Returns true or may cause OutOfMemoryError

Boundary Value Analysis (BVA) Test Cases:

Input Data	Expected Output
prefix("", "")	Returns true (empty strings are considered prefixes of each other)
prefix("", "xyz")	Returns true (empty string is a prefix of any string)
prefix("x", "")	Returns false (non-empty string can't be a prefix of an empty string)
prefix("x", "x")	Returns true (single character matching)
prefix("x", "y")	Returns false (single character not matching)
prefix("hello", "hello")	Returns true (full string match)
prefix("hello", "hel")	Returns false (prefix is longer than the string)
prefix("abc", "abcd")	Returns true (prefix matches)
prefix("abc", "xyzabc")	Returns false (prefix does not match)
prefix("12345678901234567890", "12345678901234567890")	Returns true (long string match)

prefix("12345678901234567890", "1234567890123456789")	Returns false (prefix longer than string)

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) Equivalence Classes for the System:

1. **Valid Triangles:**
   - **Equilateral Triangle:** A = B = C
   - **Isosceles Triangle:** A = B ≠ C or A = C ≠ B or B = C ≠ A
   - **Scalene Triangle:** A ≠ B ≠ C and A + B > C, A + C > B, B + C > A
   - **Right-Angled Triangle:** Satisfies Pythagorean theorem ($A^2 + B^2 = C^2$ or any permutation).
2. **Invalid Triangles:**
   - **Non-Triangle:** A + B ≤ C, A + C ≤ B, B + C ≤ A
   - **Non-positive Values:** Any side is less than or equal to zero (A ≤ 0, B ≤ 0, C ≤ 0).
   - **Integer Overflow :** Any side exceeds the maximum value for integers (e.g., A > Integer.MAX_VALUE, B > Integer.MAX_VALUE, C > Integer.MAX_VALUE).

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must ensure that the identified set of test cases cover all identified equivalence classes).

Test Case	Expected Outcome	Covered Equivalence Class
(2.5, 2.5, 2.5)	Equilateral	1
(3.0, 3.0, 1.5)	Isosceles	2
(6.0, 4.0, 5.0)	Scalene	3

(5.0, 12.0, 13.0)	Right Angled	4
(1.0, 3.0, 5.0)	Not a Triangle	5
(4.0, 2.0, 3.0)	Not a Triangle	5
(0.0, 3.5, 4.5)	Not a Triangle	6
(-2.0, 3.0, 4.0)	Not a Triangle	6
(Integer.MAX_VALUE + 1, 1.0, 2.0)	Not a Triangle or may cause Overflow	7
(2.0, Integer.MAX_VALUE + 1, 3.0)	Not a Triangle or may cause Overflow	7
(1.0, 1.0, Integer.MAX_VALUE + 1)	Not a Triangle or may cause Overflow	7

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

Test Case	Expected Outcome
(3.0, 2.0, 4.0)	Scalene
(2.0, 2.0, 5.0)	Not a Triangle
(1.5, 2.5, 4.0)	Scalene
(0.0, 3.0, 5.0)	Not a Triangle
(Integer.MAX_VALUE + 1, 1.0, 1.0)	Not a Triangle or may cause Overflow

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

Test Case	Expected Outcome

(5.0, 4.0, 5.0)	Isosceles
(7.0, 7.0, 7.0)	Equilateral
(0.0, 6.0, 0.0)	Not a Triangle
(Integer.MAX_VALUE + 1, 1.0, Integer.MAX_VALUE + 1)	Not a Triangle or may cause Overflow

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

Test Case	**Expected Outcome**
(2.0, 2.0, 2.0)	Equilateral
(1.5, 1.5, 1.5)	Equilateral
(0.0, 0.0, 0.0)	Not a Triangle
(Integer.MAX_VALUE + 1, Integer.MAX_VALUE + 1, Integer.MAX_VALUE + 1)	Not a Triangle or may cause Overflow

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

Test Case	Expected Outcome
(8.0, 6.0, 10.0)	Right Angled
(9.0, 12.0, 15.0)	Right Angled
(0.0, 3.0, 3.0)	Not a Triangle
(Integer.MAX_VALUE, Integer.MAX_VALUE, Integer.MAX_VALUE + 1)	Not a Triangle or may cause Overflow

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

Test Case	Expected Outcome
(4.0, 4.0, 9.0)	Not a Triangle
(0.0, 0.0, 0.0)	Not a Triangle
(Integer.MAX_VALUE, Integer.MAX_VALUE, Integer.MAX_VALUE)	Not a Triangle or may cause Overflow
(3.0, 4.0, 8.0)	Not a Triangle
(1.0, 1.0, 2.5)	Not a Triangle

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

Test Case	Expected Outcome
(0.0, 1.0, 1.0)	Not a Triangle
(0.0, 2.0, 4.0)	Not a Triangle
(-3.0, -1.0, -4.0)	Not a Triangle
(-1.0, 1.0, 3.0)	Not a Triangle
(-2.0, -3.0, 1.0)	Not a Triangle
(1.0, 2.0, -4.0)	Not a Triangle
(0.0, 0.0, 5.0)	Not a Triangle
(2.0, 0.0, 2.0)	Not a Triangle
(3.0, -1.0, 3.0)	Not a Triangle