Dhirubhai Ambani Institute of Information and Communication Technology

IT314 SOFTWARE ENGINEERING
Lab-08

Name:
Sania Patel

ID:
202201053

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.

Valid Equivalence Classes

- **Month:** 1 ≤ month ≤ 12 (valid month range)
- **Day:**
    - Days in February: 1 ≤ day ≤ 28 (non-leap year), 1 ≤ day ≤ 29 (leap year)
    - Days in April, June, September, November: 1 ≤ day ≤ 30
    - Days in January, March, May, July, August, October, December: 1 ≤ day ≤ 31
- **Year:** 1900 ≤ year ≤ 2015

Invalid Equivalence Classes

- **Month:**
    - month < 1 (invalid month)
    - month > 12 (invalid month)
- **Day:**
    - day < 1 (invalid day)
    - day > 31 (invalid day)
    - Invalid days for specific months (e.g., day = 31 for April, June, September, November)
- **Year:**
    - year < 1900 (invalid year)
    - year > 2015 (invalid year)

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
(15, 5, 2010)	Previous date: 14/05/2010	Valid date

(1, 1, 1900)	Previous date: 31/12/1899	Valid date (boundary)
(1, 13, 2000)	Error: Invalid month	Invalid month > 12
(31, 6, 2005)	Error: Invalid day	Invalid day > 30 for June
(0, 12, 2015)	Error: Invalid day	Invalid day < 1
(29, 2, 2013)	Error: Invalid date	February non-leap year
(29, 2, 2012)	Previous date: 28/02/2012	Valid leap year date
(32, 5, 2010)	Error: Invalid day	Invalid day > 31
(15, 0, 2010)	Error: Invalid month	Invalid month < 1
(15, 8, 1800)	Error: Invalid year	Invalid year < 1900

Sample Boundary Value Analysis Test Cases:

Tester Action and Input Data	**Expected Outcome**	**Boundary Value Analysis**
(1, 1, 1900)	Previous date: 31/12/1899	Boundary year = 1900
(31, 12, 2015)	Previous date: 30/12/2015	Boundary year = 2015
(28, 2, 2012)	Previous date: 27/02/2012	Leap year boundary
(1, 3, 2012)	Previous date: 29/02/2012	Leap year boundary
(31, 1, 2010)	Previous date: 30/01/2010	Boundary day = 31

(1, 1, 2000)	Previous date: 31/12/1999	Boundary date and year
(31, 4, 2000)	Error: Invalid day	Invalid boundary (April)

Python code :

source code

```python
import datetime

def previous_date(day, month, year):
    try:
        date = datetime.date(year, month, day)
        previous_day = date - datetime.timedelta(days=1)
        return previous_day.strftime("%d/%m/%Y")
    except ValueError:
        return "Error: Invalid date"

# Test cases
test_cases = [
    (15, 5, 2010),  # valid
    (1, 1, 1900),   # valid
    (1, 13, 2000),  # invalid month
    (31, 6, 2005),  # invalid day for June
    (0, 12, 2015),  # invalid day
    (29, 2, 2013),  # invalid date, non-leap year
    (29, 2, 2012),  # valid leap year date
    (32, 5, 2010),  # invalid day
    (15, 0, 2010),  # invalid month
    (15, 8, 1800),  # invalid year
]

# Execute test cases
for day, month, year in test_cases:
    result = previous_date(day, month, year)
    print(f"Input: ({day}, {month}, {year}) -> Output: {result}")

```

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);
}
```

Answer :

Valid Equivalence Classes:

1. **v is in the array** (the search value exists in the array).
2. **v is not in the array** (the search value does not exist in the array).
3. **Empty array** (an array with no elements).
4. **Array with duplicate values** (the search value appears more than once in the array).

Invalid Equivalence Classes:

1. **Array is null** (no array is passed).
2. **Array contains negative numbers**, but v is positive (and vice versa).

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
linearSearch(3, [1, 2, 3, 4, 5])	Index: 2 (found at index 2)	v exists in the array

linearSearch(6, [1, 2, 3, 4, 5])	-1 (not found)	v does not exist in the array
linearSearch(1, [])	-1 (empty array)	Empty array
linearSearch(5, [1, 2, 5, 5, 7])	Index: 2 (first occurrence)	Array with duplicate values
linearSearch(-3, [-5, -4, -3, -2, -1])	Index: 2 (found at index 2)	v exists and is negative
linearSearch(4, [-1, -2, -3, -4])	-1 (not found)	Positive v, array contains negatives

Boundary Value Analysis :

Tester Action and Input Data	**Expected Outcome**	**Boundary Value Analysis**
linearSearch(1, [1])	Index: 0 (found at index 0)	Array with a single element (min size)
linearSearch(5, [1, 2, 3, 4, 5])	Index: 4 (last element)	v is the last element
linearSearch(1, [1, 2, 3, 4, 5])	Index: 0 (first element)	v is the first element
linearSearch(100, [1, 2, 3, ..., 100])	Index: 99 (found at max index)	Large array boundary (max size)

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);}
```

Valid Equivalence Classes:

1. **`v` exists in the array** (one or more occurrences).
2. **`v` does not exist in the array** (0 occurrences).
3. **Array is empty** (0 occurrences regardless of `v`).
4. **Array with all identical values** (if all elements are `v`, or none are `v`).
5. **Array with both positive and negative integers** (test for mixed values).

Invalid Equivalence Classes:

1. **Array is null** (no array is passed).

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
countItem(3, [1, 2, 3, 3, 5])	2 (two occurrences)	`v` exists (2 occurrences)
countItem(6, [1, 2, 3, 4, 5])	0 (no occurrence)	`v` does not exist
countItem(1, [])	0 (empty array)	Empty array
countItem(5, [5, 5, 5, 5])	4 (four occurrences)	Array with all identical values
countItem(-1, [-1, 0, 1, -1, 2])	2 (two occurrences)	`v` exists (with negative numbers)

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome	Boundary Value Analysis
countItem(1, [1])	1 (found 1 occurrence)	Array with a single element
countItem(5, [5, 5, 5, 5])	4 (found all elements are `v`)	Array with all identical elements
countItem(1, [1, 2, 3, 1, 5])	2 (first and last elements are `v`)	`v` is at the boundaries

countItem(10, [1, 2, 3, ..., 10])	1 (found at max index)	Large array boundary (max size)

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);
}
```

Valid Equivalence Classes:

1. **v exists in the array** (test with different positions: first, middle, and last).
2. **v does not exist in the array**.
3. **Empty array**.
4. **Array with all identical values** (either v matches the identical value or it does not).
5. **Array with v as the smallest or largest value**.

Invalid Equivalence Classes:

1. **Array is not sorted** (assumption is that the array is sorted).

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
binarySearch(3, [1, 2, 3, 4, 5])	Index: 2	v exists in the array (middle)
binarySearch(6, [1, 2, 3, 4, 5])	-1	v does not exist in the array
binarySearch(1, [])	-1	Empty array
binarySearch(5, [5, 5, 5, 5, 5])	Any valid index (0–4)	Array with all identical values
binarySearch(-1, [-3, -2, -1, 0, 1, 2])	Index: 2	v exists in the array (with negatives)

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome	Boundary Value Analysis
binarySearch(1, [1])	Index: 0	Array with a single element
binarySearch(5, [1, 2, 3, 4, 5])	Index: 4	v is the last element
binarySearch(1, [1, 2, 3, 4, 5])	Index: 0	v is the first element
binarySearch(100, [1, 2, 3, ..., 100])	Index: 99	v is the largest element

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);
}
```

Valid Equivalence Classes:

1. **Equilateral triangle**: All sides are equal (`a == b == c`).
2. **Isosceles triangle**: Two sides are equal, one is different (`a == b != c`, etc.).
3. **Scalene triangle**: All sides are different (`a != b != c`).
4. **Valid triangle**: The sum of any two sides must be greater than the third (`a + b > c`, etc.).

Invalid Equivalence Classes:

1. **Invalid triangle**: One or more sides form an impossible triangle (`a >= b + c`, etc.).
2. **Negative or zero lengths**: Triangle sides cannot be zero or negative.

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
triangle(3, 3, 3)	EQUILATERAL (0)	Equilateral triangle
triangle(3, 3, 2)	ISOSCELES (1)	Isosceles triangle
triangle(3, 4, 5)	SCALENE (2)	Scalene triangle

triangle(1, 2, 3)	INVALID (3)	Invalid triangle (sum of two sides equals the third)
triangle(0, 3, 4)	INVALID (3)	Invalid triangle (zero-length side)
triangle(-1, 2, 2)	INVALID (3)	Invalid triangle (negative-length side)

Boundary Value Analysis :

Tester Action and Input Data	**Expected Outcome**	**Boundary Value Analysis**
triangle(1, 1, 1)	EQUILATERAL (0)	Minimum valid equilateral triangle
triangle(2, 2, 3)	ISOSCELES (1)	Isosceles triangle with minimal difference
triangle(1, 2, 2)	ISOSCELES (1)	Isosceles triangle (boundary case)
triangle(1, 1, 2)	INVALID (3)	Invalid triangle (sum equals third side)
triangle(1000, 1, 1)	INVALID (3)	One side much larger than the others
triangle(2, 2, 4)	INVALID (3)	Invalid triangle (sum equals third side)

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;
}
```

Valid Equivalence Classes:

1. **`s1` is a prefix of `s2`** (e.g., `s1 = "he"`, `s2 = "hello"`).
2. **`s1` is not a prefix of `s2`** (e.g., `s1 = "hi"`, `s2 = "hello"`).
3. **`s1` is an empty string** (empty strings are considered valid prefixes).
4. **`s1` equals `s2`** (when both strings are the same, `s1` is trivially a prefix).
5. **`s1` is longer than `s2`** (cannot be a prefix).

Invalid Equivalence Classes:

- None for this function, since `s1` and `s2` are assumed to be non-null based on the problem description.

Sample Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome	Equivalence Partitioning
prefix("he", "hello")	true	`s1` is a prefix of `s2`
prefix("hi", "hello")	false	`s1` is not a prefix of `s2`
prefix("", "hello")	true	`s1` is an empty string (valid prefix)
prefix("hello", "hello")	true	`s1` equals `s2`
prefix("hellothere", "hello")	false	`s1` is longer than `s2`

Boundary Value Analysis :

Tester Action and Input Data	Expected Outcome	Boundary Value Analysis
prefix("", "")	true	Both strings are empty

prefix("h", "hello")	true	`s1` is a single character, `s2` is longer
prefix("hello", "helloo")	true	`s1` is just one character shorter
prefix("helloo", "hello")	false	`s1` is just one character longer
prefix("h", "h")	true	Both strings are single character

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 Equivalence Classes (Form a Triangle):

1. **Equilateral triangle**: A=B=C
2. **Isosceles triangle**: A=B=C, A=C≠B, B=C≠A
3. **Scalene triangle**: A≠B≠C
4. **Right-angled triangle**: A^2+B^2=C^2, A^2 + B^2 = C^2, A^2+B^2=C^2 or its permutations (Pythagorean theorem)
5. **Valid triangle**: A+B>C, A+C>B, B+C>A

Invalid Equivalence Classes (Cannot Form a Triangle):

6. **Non-triangle (sum condition fails)**: A+B≤C, A+C≤B, or B+C≤A
7. **Non-positive input**: A≤0, B≤0, or C≤0

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)

Test Case ID	Input Values (A, B, C)	Expected Output	Equivalence Class Description
TC1	3.0, 3.0, 3.0	Equilateral	*A=B=CA = B = C*A=B=C (equilateral triangle)
TC2	5.0, 5.0, 7.0	Isosceles	*A=B≠CA = B \neq C*A=B=C (isosceles triangle)
TC3	7.0, 5.0, 5.0	Isosceles	*B=C≠AB = C \neq A*B=C=A (isosceles triangle)
TC4	3.0, 4.0, 5.0	Right-angled	*A2+B2=C2A^2 + B^2 = C^2*A2+B2=C2 (right triangle)
TC5	6.0, 8.0, 10.0	Right-angled	*A2+B2=C2A^2 + B^2 = C^2*A2+B2=C2 (right triangle)
TC6	7.0, 5.0, 6.0	Scalene	*A≠B≠CA \neq B \neq C*A=B=C (scalene triangle)
TC7	3.0, 4.0, 10.0	Invalid	Non-triangle (sum condition fails)
TC8	0.0, 4.0, 5.0	Invalid	Non-positive side length
TC9	-1.0, 5.0, 6.0	Invalid	Non-positive side length
TC10	5.0, 5.0, 10.0	Invalid	Non-triangle (sum condition fails)

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
BC1	3.0, 4.0, 7.0	Invalid	*A+B=CA + B = C*A+B=C (fails boundary)
BC2	3.0, 4.0, 6.999	Scalene	*A+B>CA + B > C*A+B>C (just passes)
BC3	5.0, 6.0, 10.999	Scalene	*A+B>CA + B > C*A+B>C (just passes)

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
BC4	5.0, 5.0, 10.0	Invalid	Fails triangle inequality
BC5	5.0, 5.0, 9.999	Isosceles	Isosceles boundary
BC6	5.0, 5.0, 5.001	Isosceles	Near boundary

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
BC7	5.0, 5.0, 5.0	Equilateral	Equilateral triangle
BC8	5.0, 5.0, 5.001	Isosceles	Near-equilateral case

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
BC9	3.0, 4.0, 5.0	Right-angled	Exact right-angled triangle
BC10	3.0, 4.0, 4.999	Scalene	Just fails right-angle test
BC11	6.0, 8.0, 10.0	Right-angled	Right-angled triangle

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
BC12	3.0, 4.0, 7.0	Invalid	*A*+*B*=*CA* + *B* = *C*A+B=C (non-triangle)

BC13	6.0, 8.0, 14.0	Invalid	*A+B=CA + B = C*A+B=C (non-triangle)
BC14	1.0, 2.0, 3.0	Invalid	*A+B=CA + B = C*A+B=C (non-triangle)

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

Test Case ID	Input Values (A, B, C)	Expected Output	Boundary Condition
NP1	0.0, 3.0, 4.0	Invalid	Zero-length side
NP2	-1.0, 3.0, 4.0	Invalid	Negative-length side
NP3	5.0, 0.0, 7.0	Invalid	Zero-length side