IT314 - Software Engineering

Lab - 08

Nisharg modi (202201346)

Questions

1. Equivalence Partitioning (EP):

- **Valid input classes**:
  - Date is valid and exists (e.g., 15th July 2010)
- **Invalid input classes**:
  - Month is out of range (e.g., 0, 13)
  - Day is out of range (e.g., day > 31 or day < 1)
  - Year is out of range (e.g., year < 1900 or year > 2015)

2. Boundary Value Analysis (BVA):

- Focuses on testing the edges of the input range:
  - Boundary values for day (1, 31)
  - Boundary values for month (1, 12)
  - Boundary values for year (1900, 2015)

Test Cases Using Equivalence Partitioning (EP):

1. **Valid date**:
   - **Input:** day = 15, month = 7, year = 2010
   - **Expected Output:** 14 July 2010 (valid previous date)
2. **Invalid month (out of range)**:
   - **Input:** day = 10, month = 13, year = 2005
   - **Expected Output:** "Invalid date" (month out of range)
3. **Invalid day (out of range)**:
   - **Input:** day = 32, month = 5, year = 2012
   - **Expected Output:** "Invalid date" (day out of range)
4. **Invalid year (out of range)**:
   - **Input:** day = 20, month = 5, year = 2017
   - **Expected Output:** "Invalid date" (year out of range)
5. **Valid leap year date (29th February)**:
   - **Input:** day = 29, month = 2, year = 2008
   - **Expected Output:** 28 February 2008 (valid leap year case)

Test Cases Using Boundary Value Analysis (BVA):

1. **Day at minimum boundary**:
    - **Input:** day = 1, month = 5, year = 2005
    - **Expected Output:** 30 April 2005 (previous month's last day)
2. **Day at maximum boundary**:
    - **Input:** day = 31, month = 12, year = 2015
    - **Expected Output:** 30 December 2015
3. **Month at minimum boundary**:
    - **Input:** day = 10, month = 1, year = 2000
    - **Expected Output:** 9 January 2000
4. **Month at maximum boundary**:
    - **Input:** day = 10, month = 12, year = 2014
    - **Expected Output:** 9 December 2014
5. **Year at lower boundary**:
    - **Input:** day = 1, month = 1, year = 1900
    - **Expected Output:** 31 December 1899
6. **Year at upper boundary**:
    - **Input:** day = 1, month = 1, year = 2015
    - **Expected Output:** 31 December 2014

Question 2.1

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);
}
```

Test Case 1: Value present in the middle of the array

- **Input:** v = 5, a = [1, 2, 3, 4, 5, 6, 7]
- **Expected Output:** 4 (since a[4] == 5)

Test Case 2: Value present at the beginning of the array

- **Input:** v = 1, a = [1, 2, 3, 4, 5]
- **Expected Output:** 0 (since a[0] == 1)

Test Case 3: Value present at the end of the array

- **Input:** v = 5, a = [1, 2, 3, 4, 5]
- **Expected Output:** 4 (since a[4] == 5)

Test Case 4: Value not present in the array

- **Input:** v = 8, a = [1, 2, 3, 4, 5]
- **Expected Output:** -1 (value 8 is not in the array)

Test Case 5: Empty array

- **Input:** v = 3, a = []
- **Expected Output:** -1 (array is empty)

Test Case 6: Single-element array where the value is present

- **Input:** v = 3, a = [3]
- **Expected Output:** 0 (since a[0] == 3)

Test Case 7: Single-element array where the value is not present

- **Input:** v = 3, a = [4]
- **Expected Output:** -1 (value 3 is not in the array)

Test Case 8: Multiple occurrences of the value

- **Input:** v = 2, a = [1, 2, 2, 2, 3]
- **Expected Output:** 1 (returns the index of the first occurrence of 2)

Test Case 9: Negative numbers in the array

- **Input:** v = -3, a = [1, -2, -3, 4, 5]
- **Expected Output:** 2 (since a[2] == -3)

Test Case 10: Large array

- **Input:** v = 100000, a = [i for i in range(1, 100001)]
- **Expected Output:** 99999 (last element of the array is 100000, located at index 99999)

Question 2.2

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);  }
```

The function countItem returns how many times a value v appears in an array ‘a’.

Test Cases:

1. **Value appears multiple times**
    - **Input:** v = 3, a = [1, 3, 3, 5, 3]
    - **Expected Output:** 3
2. **Value does not appear in the array**
    - **Input:** v = 4, a = [1, 2, 3, 5, 6]
    - **Expected Output:** 0
3. **Value appears once**
    - **Input:** v = 5, a = [1, 2, 3, 4, 5]
    - **Expected Output:** 1
4. **Empty array**
    - **Input:** v = 3, a = []
    - **Expected Output:** 0
5. **Array contains only the target value**
    - **Input:** v = 3, a = [3, 3, 3, 3]
    - **Expected Output:** 4

Question 2.3

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);
}
```

Test Cases:

1. **Value present in the middle**
    - **Input:** v = 6, a = [1, 2, 3, 6, 8, 10]
    - **Expected Output:** 3
2. **Value present at the beginning**
    - **Input:** v = 1, a = [1, 2, 3, 6, 8, 10]
    - **Expected Output:** 0
3. **Value present at the end**
    - **Input:** v = 10, a = [1, 2, 3, 6, 8, 10]
    - **Expected Output:** 5
4. **Value not present**
    - **Input:** v = 5, a = [1, 2, 3, 6, 8, 10]
    - **Expected Output:** -1
5. **Empty array**
    - **Input:** v = 5, a = []

- **Expected Output:** -1

Question 2.4

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);
}
```

Test Cases:

1. **Equilateral triangle**
    - **Input:** a = 3, b = 3, c = 3
    - **Expected Output:** EQUILATERAL
2. **Isosceles triangle**
    - **Input:** a = 3, b = 3, c = 5
    - **Expected Output:** ISOSCELES
3. **Scalene triangle**
    - **Input:** a = 3, b = 4, c = 5
    - **Expected Output:** SCALENE
4. **Invalid triangle**
    - **Input:** a = 1, b = 2, c = 10
    - **Expected Output:** INVALID

5. **Zero-length sides**
    - **Input:** a = 0, b = 3, c = 4
    - **Expected Output:** INVALID

Question 2.5

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;
}
```

The function prefix checks if string s1 is a prefix of string s2.

Test Cases:

1. **Exact prefix**
    - **Input:** s1 = "pre", s2 = "prefix"
    - **Expected Output:** true
2. **Not a prefix**
    - **Input:** s1 = "sub", s2 = "prefix"
    - **Expected Output:** false
3. **Empty prefix string**
    - **Input:** s1 = "", s2 = "prefix"
    - **Expected Output:** true

4. **Prefix longer than the string**
   - **Input:** s1 = "prefixlong", s2 = "prefix"
   - **Expected Output:** false

Question 2.6

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
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) Equivalence Classes

1. **Valid triangles**:
   - Equilateral: All sides are equal.

    - Isosceles: Two sides are equal.
    - Scalene: All sides are different.
    - Right-angled: Satisfies a2+b2=c2a^2 + b^2 = c^2a2+b2=c2.
2. **Invalid triangles**: One side is greater than or equal to the sum of the other two.
3. **Non-positive inputs**: Sides that are zero or negative.

b) Test Cases

Equivalence Class	**Input Values**	**Expected Output**
Equilateral	A = 3.0, B = 3.0, C = 3.0	Equilateral
Isosceles	A = 3.0, B = 3.0, C = 5.0	Isosceles
Scalene	A = 3.0, B = 4.0, C = 5.0	Scalene
Right-angled	A = 3.0, B = 4.0, C = 5.0	Right-angled
Invalid	A = 1.0, B = 2.0, C = 10.0	Invalid
Non-positive	A = 0.0, B = 3.0, C = 4.0	Invalid

c) Boundary Condition A + B > C

- **Test Case**: A = 1.0, B = 2.0, C = 2.99
- **Expected Output**: Scalene

d) Boundary Condition A = C

- **Test Case**: A = 3.0, B = 5.0, C = 3.0
- **Expected Output**: Isosceles

e) Boundary Condition A = B = C

- **Test Case**: A = 3.0, B = 3.0, C = 3.0
- **Expected Output**: Equilateral

f) Boundary Condition $A^2 + B^2 = C^2$

- **Test Case**: A = 3.0, B = 4.0, C = 5.0
- **Expected Output**: Right-angled

g) Non-triangle Case

- **Test Case**: A = 1.0, B = 2.0, C = 4.0
- **Expected Output**: Invalid

h) Non-positive Input

- **Test Case**: A = -3.0, B = 4.0, C = 5.0
- **Expected Output**: Invalid