Lab-8
Functional Testing (Black-Box)

Ayush Pandita - 202201256

Q1) Consider a program for determining the previous date. It's 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?

Equivalence Classes:

- **E1:** day<1 (invalid day for any month)
- **E2:** 1<=day<=31 (valid days for any month)
- **E3:** day > 31 (invalid day for any month)
- **E4:** month<1 (invalid month for any year)
- **E5:** 1<=month<=12 (valid month for any year)
- **E6:** month>12 (invalid month for any year)
- **E7:** year<1900 (invalid year value)
- **E8:** 1900<= year <= 2015 (valid year value)
- **E9:** year >2015 (invalid year value)

Test Case #	**Day**	**Month**	**Year**	**Equivalence Classes Covered**	**Expected Outcome**	**Remarks**
TC1	15	6	2000	E2, E5, E8	14 June 2000	Valid date
TC2	32	1	2000	E3, E5, E8	Invalid Date	Day exceeds valid range
TC3	0	5	2010	E1, E5, E8	Invalid Date	Day less than 1
TC4	29	2	2000	E2, E5, E8	28 February 2000	Leap year valid case
TC5	30	2	2001	E2, E5, E8	Invalid Date	February only has 28 days
TC6	15	13	2000	E2, E6, E8	Invalid Date	Month exceeds 12
TC7	15	0	2000	E2, E4, E8	Invalid Date	Month less than 1
TC8	15	6	1899	E2, E5, E7	Invalid Date	Year below valid range
TC9	15	6	2016	E2, E5, E9	Invalid Date	Year exceeds valid range

Test Case #	Day	Month	Year	Equivalence Classes Covered	Expected Outcome	Remarks
TC10	1	1	1900	E2, E5, E8	31 December 1899	Valid boundary test
TC11	1	3	2000	E2, E5, E8	29 February 2000	Leap year boundary test
TC12	1	1	2015	E2, E5, E8	31 December 2014	Valid edge case for year

Q2) Write a set of test cases - 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 is correct or not.**

Q2) 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 for linearSearch Function

- **E1:** v is a non-integer value (invalid input).
- **E2:** v is null (invalid input).
- **E3:** v is an integer value (valid input).
- **E4**: Array a contains all integers (valid input).
- **E5**: Array a is null or empty (edge case).
- **E6**: Array a contains one or more non-integer values (invalid input).

- **E7:** v exists in a (valid input, should return the index of v).
- **E8:** v does not exist in a (valid input, should return -1).

Test Case #	v	Array a[]	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	3	[1, 2, 3, 4, 5]	E3, E4, E7	2	v exists in a at index 2
TC2	6	[1, 2, 3, 4, 5]	E3, E4, E8	-1	v does not exist in a
TC3	null	[1, 2, 3, 4, 5]	E2, E4	Error/Exception	v is null
TC4	"5"	[1, 2, 3, 4, 5]	E1, E4	Error/Exception	v is non-integer (string)
TC5	3	[]	E3, E5	-1	a is empty
TC6	3	null	E3, E5	Error/Exception	a is null
TC7	3	[1, 2, "3", 4, 5]	E3, E6	Error/Exception	a contains a non-integer value
TC8	3	[3]	E3, E4, E7	0	v is the only element in a
TC9	2	[3]	E3, E4, E8	-1	v does not exist in single-element a

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 for countItem Function

- **E1:** v is a non-integer value (invalid input).
- **E2:** v is null (invalid input).
- **E3:** v is an integer value (valid input).
- **E4:** Array a contains all integers (valid input).
- **E5:** Array a is null or empty (edge case).
- **E6:** Array a contains one or more non-integer values (invalid input).
- **E7:** v appears at least once in a (valid input, positive count returned).
- **E8:** v does not appear in a (valid input, count of 0 returned).

Test Case #	v	Array a[]	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	3	[1, 2, 3, 4, 3]	E3, E4, E7	2	v appears twice in a

Test Case #	**v**	**Array a[]**	**Equivalence Classes Covered**	**Expected Outcome**	**Remarks**
TC2	6	[1, 2, 3, 4, 5]	E3, E4, E8	0	`v` does not appear in `a`
TC3	null	[1, 2, 3, 4, 5]	E2, E4	Error/Exception	`v` is `null`
TC4	"3"	[1, 2, 3, 4, 3]	E1, E4	Error/Exception	`v` is non-integer (string)
TC5	3	[]	E3, E5	0	`a` is empty
TC6	3	null	E3, E5	Error/Exception	`a` is `null`
TC7	3	[1, 2, "3", 4, 3]	E3, E6	Error/Exception	`a` contains a non-integer value
TC8	2	[2]	E3, E4, E7	1	`v` appears once in single-element `a`
TC9	2	[3]	E3, E4, E8	0	`v` does not appear in single-element `a`

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 Function

- **E1:** v is a non-integer value (invalid input).
- **E2:** v is null (invalid input).
- **E3:** v is an integer value (valid input).
- **E4:** Array a contains only integers (valid input).
- **E5:** Array a is null or empty (edge case).
- **E6:** Array a contains one or more non-integer values (invalid input).
- **E7:** v exists in a (valid input, index returned).
- **E8:** v does not exist in a (valid input, -1 returned).

Test Case #	v	Array a[]	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	4	[1, 2, 3, 4, 5]	E3, E4, E7	3	`v` exists in `a` at index 3
TC2	6	[1, 2, 3, 4, 5]	E3, E4, E8	-1	`v` does not exist in `a`
TC3	null	[1, 2, 3, 4, 5]	E2, E4	Error/Exception	`v` is `null`
TC4	"3"	[1, 2, 3, 4, 5]	E1, E4	Error/Exception	`v` is non-integer (string)
TC5	3	[]	E3, E5	-1	`a` is empty
TC6	3	null	E3, E5	Error/Exception	`a` is `null`
TC7	3	[1, 2, "3", 4, 5]	E3, E6	Error/Exception	`a` contains a non-integer value
TC8	2	[2]	E3, E4, E7	0	`v` is the only element in `a`

Test Case #	**v**	**Array a[]**	**Equivalence Classes Covered**	**Expected Outcome**	**Remarks**
TC9	1	[2]	E3, E4, E8	-1	v does not exist in single-element a
TC10	5	[1, 2, 3, 4, 5]	E3, E4, E7	4	v exists at the last index
TC11	1	[1, 2, 3, 4, 5]	E3, E4, E7	0	v exists at the first index

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 || b==c || c==a)
        return (ISOSCELES);
    return (SCALENE);
}
```

Equivalence Classes for triangle Function

- **E1:** One or more side lengths are non-integer values (invalid input).
- **E2:** One or more side lengths are null (invalid input).
- **E3:** All side lengths are positive integers (valid input).
- **E4:** One or more side lengths are less than or equal to 0 (invalid input).
- **E5:** Triangle inequality is satisfied (valid triangle).
    - $a + b > c$, $b + c > a$, $a + c > b$
- **E6:** Triangle inequality is violated (invalid triangle).
    - One side is greater than or equal to the sum of the other two.
- **E7:** All three sides are equal (Equilateral triangle).

- **E8:** Exactly two sides are equal (Isosceles triangle).
- **E9:** No two sides are equal (Scalene triangle).

Test Case #	a	b	c	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	3	3	3	E3, E5, E7	EQUILATERAL	All sides are equal
TC2	3	3	2	E3, E5, E8	ISOSCELES	Two sides are equal
TC3	3	4	5	E3, E5, E9	SCALENE	All sides are different
TC4	1	2	3	E3, E6	INVALID	Violates triangle inequality
TC5	5	1	1	E3, E6	INVALID	One side too large
TC6	0	3	4	E4	INVALID	One side is zero
TC7	-1	3	4	E4	INVALID	One side is negative

Test Case #	**a**	**b**	**c**	**Equivalence Classes Covered**	**Expected Outcome**	**Remarks**
TC8	3.5	4	5	E1	Error/Exception	One side is non-integer
TC9	null	4	5	E2	Error/Exception	One side is `null`
TC10	1	1	2	E3, E6	INVALID	Exactly equal to other two sides
TC11	1	1	1	E3, E5, E7	EQUILATERAL	Minimum positive equilateral
TC12	3	3	4	E3, E5, E8	ISOSCELES	Valid isosceles case

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 for prefix Function

- **E1:** s1 is an empty string (valid prefix of any string).
- **E2:** s2 is an empty string (valid input, but s1 must also be empty to return true).
- **E3:** s1.length() > s2.length() (invalid prefix).
- **E4:** s1.length() <= s2.length() (valid input).
- **E5:** All characters of s1 match the corresponding characters of s2 (valid prefix).
- **E6:** At least one character of s1 does not match the corresponding character of s2 (not a prefix).
- **E7:** s1 equals s2 (valid prefix).
- **E8:** Both s1 and s2 are non-empty but do not match fully (not a prefix).

Test Case #	s1	s2	Equivalence Classes Covered	Expected Outcome	Remarks
TC1	""	"prefix"	E1, E4, E5	true	An empty string is a prefix of any string
TC2	"pre"	"prefix"	E4, E5	true	s1 is a valid prefix of s2
TC3	"post"	"prefix"	E4, E6	false	s1 is not a prefix of s2
TC4	"prefix"	"prefix"	E4, E5, E7	true	s1 is equal to s2
TC5	"long"	"short"	E3	false	s1 is longer than s2
TC6	""	""	E1, E2, E7	true	Both strings are empty
TC7	"pref"	"pre"	E3	false	s1 is longer than s2
TC8	"pre"	"predator"	E4, E5	true	s1 is a valid prefix of s2

Test Case #	s1	s2	Equivalence Classes Covered	Expected Outcome	Remarks
TC9	"a"	"b"	E4, E6	false	Single-character strings do not match

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, 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.**

- **E1:** One or more side lengths are non-floating-point values (invalid input).
- **E2:** One or more side lengths are non-positive (negative or zero) (invalid input).
- **E3:** All side lengths are valid positive floating-point values (valid input).
- **E4:** Triangle inequality is satisfied: A + B > C, C + B > A, A + C > B
- **E5:** Triangle inequality is violated: One side is greater than or equal to the sum of the other two.
- **E6:** All three sides are equal (Equilateral triangle).
- **E7:** Exactly two sides are equal (Isosceles triangle).
- **E8:** All sides are different (Scalene triangle) and the triangle is not right-angled.
- **E9:** Right-angled triangle: A^2 + B^2 = C^2.

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

Test Case #	A	B	C	Equivalence Classes Covered	Expected Output	Remarks
TC1	"a"	4.0	5.0	E1	Error/Exception	One side is non-floating-point
TC2	-1.0	3.0	4.0	E2	Invalid Triangle	One side is negative
TC3	0.0	4.0	5.0	E2	Invalid Triangle	One side is zero
TC4	3.0	3.0	3.0	E3, E4, E6	Equilateral	All sides are equal
TC5	3.0	3.0	2.0	E3, E4, E7	Isosceles	Two sides are equal
TC6	3.0	4.0	5.0	E3, E4, E8, E9	Right-angled	$A^2 + B^2 = C^2$
TC7	5.0	4.0	3.0	E3, E4, E8, E9	Right-angled	Right-angled (reversed order)

Test Case #	A	B	C	Equivalence Classes Covered	Expected Output	Remarks
TC8	3.0	4.0	8.0	E3, E5	Invalid Triangle	Violates triangle inequality
TC9	3.0	4.0	7.0	E3, E4, E8	Scalene	Just satisfies inequality boundary
TC10	3.0	3.0	5.9	E3, E4, E7	Isosceles	Near boundary for isosceles
TC11	3.0	3.0	6.0	E3, E5	Invalid Triangle	Violates inequality boundary
TC12	5.0	5.0	7.0	E3, E4, E7	Isosceles	Valid isosceles triangle
TC13	1.0	1.0	2.0	E3, E5	Invalid Triangle	Just violates inequality boundary

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

Test Cases for Scalene Triangle Boundary Condition

TC9:

- A = 3.0, B = 4.0, C = 7.0
- Reason: This is a valid scalene triangle because A+B=7.0>C A + B = 7.0 > C A+B=7.0>C.

TC8:

- A = 3.0, B = 4.0, C = 8.0
- Reason: This is an invalid triangle because A+B=7.0<C A + B = 7.0 < C A+B=7.0<C, violating the inequality.

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

Test Cases for Isosceles Triangle Boundary Condition

TC10:

- A = 3.0, B = 3.0, C = 5.9
- Reason: This is near the boundary for an isosceles triangle where two sides are almost equal.

TC11:

- A = 3.0, B = 3.0, C = 6.0
- Reason: This is an invalid triangle because the inequality is violated: A+B=CA + B = CA+B=C.

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

Test Case for Equilateral Triangle Boundary Condition

TC4:

- A = 3.0, B = 3.0, C = 3.0
- Reason: All three sides are equal, forming an equilateral triangle.

f) For the boundary condition $A^2 + B^2 = C^2$ case (right angle triangle), identify test cases to verify the boundary.

Test Cases for Right-Angled Triangle Boundary Condition

TC6:

- A = 3.0, B = 4.0, C = 5.0
- Reason: This is a right-angled triangle since 3^2 + 4^2 = 5^2

TC7:

- A = 5.0, B = 4.0, C = 3.0
- Reason: This is also a valid right-angled triangle, but with sides in a different order.

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

Test Cases for Non-Triangle Boundary Condition

TC8:

- A = 3.0, B = 4.0, C = 8.0
- Reason: This violates the triangle inequality: A+B<CA + B < CA+B<C.

TC13:

- A = 1.0, B = 1.0, C = 2.0
- Reason: This just fails the inequality: A+B=CA + B = CA+B=C, making it invalid.

h) For non positive input, identify test points.

Test Cases for Non-Positive Inputs

TC2:

- A = -1.0, B = 3.0, C = 4.0
- Reason: One side is negative, making it an invalid input.

TC3:

- A = 0.0, B = 4.0, C = 5.0
- Reason: One side is zero, making it an invalid input.