Question 1

Consider a program for determining the previous date. Its input is a 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 the 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

Equivalence Classes:

1. 1 <= Month <= 12
2. Month < 1
3. Month > 12
4. 1 <= Day <= 31
5. Day < 1
6. Day > 31
7. 1900 <= Year <= 2015
8. Year < 1900
9. Year > 2015

Boundary Value Analysis:

Input (mm/dd/yyyy)	Expected Output	Equivalence Classes
10/31/2004	Valid Input (10/30/2004)	E1, E4, E7
0/1/1899	Invalid Input	E2, E4, E8
12/32/2015	Invalid Input	E1, E6, E7
13/0/2016	Invalid Input	E3, E5, E9
12/31/2015	Valid Input (12/30/2015)	E1, E4, E7
6/24/2009	Valid Input (6/23/2009)	E1, E4, E7
5/3/2001	Valid Input (5/2/2001)	E1, E4, E7
1/1/2000	Valid Input (12/31/1999)	E1, E4, E7

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 Classes:

1. The value v occurs once in the array a.
2. The value v occurs multiple times in array a.
3. The value v is not present in the array a.
4. The value v is at the first or last position of the array a.
5. The array a is empty.

Boundary Value Analysis:

Input (v, a[])	Expected Output	Equivalence Classes
1, [3,2,3,1,5]	3	E1
5, []	-1	E5
2, [3,2,2,4,2,5]	1	E2
2, [2,4,5]	0	E4
3, [4,5,6]	-1	E3
4, [1,2,4]	2	E4

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:

1. The value v occurs once in the array a.
2. The value v occurs multiple times in array a.
3. The value v is not present in the array a.
4. The value v is at the first or last position of the array a.
5. The array a is empty.

Boundary Value Analysis:

Input (v, a[])	Expected Output	Equivalence Classes
1, [3,2,3,1,5]	1	E1
5, []	0	E5
2, [3,2,2,4,2,5]	3	E2
2, [2,4,5]	1	E4
3, [4,5,6]	0	E3
4, [1,2,4]	1	E4

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 Classes:

1. The value v occurs once in the array a.
2. The value v occurs multiple times in array a.
3. The value v is smaller than the first element in array a.
4. The value v is larger than the last element in array a.
5. The value v is at the first or last position of array a.
6. The array a is empty.

Boundary Value Analysis:

Input (v, a[])	Expected Output	Equivalence Classes
1, [1,2,3,3,5]	0	E1, E5
5, []	-1	E6
3, [2,3,3,3,4,5]	1	E2
1, [2,4,5]	-1	E3
9, [4,5,6]	-1	E4
4, [1,2,4]	2	E5

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 Classes:

1. Equilateral Triangle: All sides are equal.
2. Isosceles Triangle: Exactly two sides are equal.
3. Scalene Triangle: All sides are different.

4. Non-Triangle: The sum of the lengths of any two sides is not greater than the third side.
5. Negative Values: One or more sides have negative lengths.
6. Zero Values: One or more sides are zero.

Boundary Value Analysis:

Input Data	Expected Output	Expected Outcome
3, 3, 3	0	E1
3, 3, 4	1	E2
4, 5, 6	2	E3
1, 3, 2	3	E4
-1, 6, 3	3	E5
0, 0, 5	3	E6

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:

1. s1 is a non-empty string and is a prefix of s2.
2. s1 is an empty string, which is considered a prefix of any string s2.
3. s1 is a non-empty string, but s2 is empty.

4. s1 is equal to s2.
5. s1 is longer than s2.
6. s1 is not a prefix of s2 (they differ after some characters).

Boundary Value Analysis:

Input (s1, s2)	Expected Output	Equivalence Classes
"soft", "software"	YES	E1
"", "software"	YES	E2
"soft", ""	NO	E3
"soft", "hardware"	NO	E6

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:

- Equilateral Triangle: All sides are equal.
- Isosceles Triangle: Exactly two sides are equal.
- Scalene Triangle: All sides are different.
- Right-Angled Triangle: Satisfies the Pythagorean theorem ($A^2 + B^2 = C^2$).

Invalid Equivalence Classes:
- Non-Triangle: The sum of the lengths of any two sides is not greater than the third side.
- Negative Values: One or more sides have negative lengths.
- Zero Values: One or more sides are zero.

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)

Input Data	Description	Expected Outcome
2.0, 2.0, 2.0	All sides are equal	Equilateral Triangle
3.0, 3.0, 4.0	Two sides are equal	Isosceles Triangle

7.0, 9.0, 10.0	All sides are different	Scalene Triangle
3.0, 4.0, 5.0	Satisfies the Pythagorean theorem	Right-Angled Triangle
1.0, 2.0, 3.0	The sum of the two shorter sides is equal to the longest	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Negative Values
0.0, 1.0, 1.0	One side is zero	Zero Values

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

Input Data	Description	Covered Classes
2.0, 3.0, 4.0	Valid scalene triangle	Scalene Triangle
2.0, 2.5, 5.0	Just below boundary	Non Triangle
1.0, 1.0, 2.0	Exactly on the boundary	Non Triangle

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

Input Data	Description	Covered Classes
5.0, 5.0, 3.0	Two sides equal, valid isosceles	Isosceles Triangle
3.0, 3.0, 6.0	Just below boundary	Non Triangle
2.0, 2.0, 4.0	Exactly on the boundary	Non Triangle

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

Input Data	Description	Covered Classes

3.0, 3.0, 3.0	All sides equal, valid Equilateral	Equilateral Triangle
2.0, 2.0, 2.0	Valid Equilateral	Equilateral Triangle
2.0, 2.0, 3.0	Just isosceles	Isosceles Triangle

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

Input Data	Description	Covered Classes
3.0, 4.0, 5.0	Valid right-angled triangle	Right-angled Triangle
5.0, 12.0, 13.0	Valid right-angled triangle	Right-angled Triangle
2.0, 2.0, 3.0	Not a Right-angled Triangle	Not a Right-angled Triangle

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

Input Data	Description	Covered Classes
1.0, 2.0, 3.0	Sum of two sides equals the third	Non-Triangle
2.0, 4.0, 3.0	Valid triangle	Scalene
10.0, 1.0, 1.0	Impossible lengths	Non-Triangle

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

Input Data	Description	Covered Classes
0.0, 0.0, 0.0	All sides are zero	Non-Triangle
-1.0, 2.0, 3.0	One side is negative	Non-Triangle
2.0, 0.0, 2.0	One side is zero	Non-Triangle