IT 314
Lab 8
Name : Jayesh Padiya
Student ID : 202201358

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?
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 Partition

Input Date (Day-Month-Year)	Expected Outcome
15-6-2005	Previous date
10-0-2000	Invalid (Month out of range)
5-15-2010	Invalid (Month out of range)
31-12-2015	Previous date
30-4-2010	Previous date
28-2-2001	Previous date

0-10-1999	Invalid (Date out of range)
35-1-2010	Invalid (Date out of range)
1-1-1900	Previous date
10-10-1896	Invalid (Year out of range)
15-7-2019	Invalid (Year out of range)

Boundary Value Analysis

Input Date	Expected Outcome
15-1-2000	Previous date
31-12-2015	Previous date
10-0-2000	Invalid (Month out of range)
5-13-2010	Invalid (Month out of range)
1-5-2010	Previous date
31-12-2015	Previous date
30-4-2010	Previous date
29-2-2004	Previous date
28-2-2001	Previous date
0-3-2000	Invalid (Date out of range)
32-1-2010	Invalid (Date out of range)
1-1-1900	Previous date
15-6-2015	Previous date
15-7-2016	Invalid (Year out of range)

Code

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

bool isLeapYear(int year)
 {
   return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

string validateDate(int day, int month, int year)
 {
   if (year < 1900 || year > 2015) {
       return "Invalid (Year out of range)";
   }

   if (month < 1 || month > 12)
  {
       return "Invalid (Month out of range)";
   }

   int daysInMonth[] = {31, (isLeapYear(year) ? 29 : 28), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

   if (day < 1 || day > daysInMonth[month - 1])
  {
       return "Invalid (Date out of range)";
   }

   return "Previous date";
}

int main()
 {
   int testCases[][3] =
 {
       {15, 6, 2005}, {10, 0, 2000}, {5, 15, 2010}, {31, 12, 2015}, {30, 4, 2010},
       {28, 2, 2001}, {0, 10, 1999}, {35, 1, 2010}, {1, 1, 1900}, {10, 10, 1896},
       {15, 7, 2019}, {15, 1, 2000}, {31, 12, 2015}, {10, 0, 2000}, {5, 13, 2010},
       {1, 5, 2010}, {31, 12, 2015}, {30, 4, 2010}, {29, 2, 2004}, {28, 2, 2001},
       {0, 3, 2000}, {32, 1, 2010}, {1, 1, 1900}, {15, 6, 2015}, {15, 7, 2016}
   };
```

```
    int numTestCases = sizeof(testCases) / sizeof(testCases[0]);

    for (int i = 0; i < numTestCases; i++)
    {
        int day = testCases[i][0];
        int month = testCases[i][1];
        int year = testCases[i][2];

        string result = validateDate(day, month, year);

    }

    return 0;
}
```

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);
}
```

Input Data	Description	Expected Outcome
v = 7, a = {10, 20, 7, 30, 40}	v is present in a[]	2 (valid)
v = 15, a = {11, 22, 33, 44, 55}	v is not present in a[]	-1 (valid)
v = 8, a = {}	a[] is empty	-1 (valid)
v = 6, a = {4, 6, 6, 8, 10}	v appears multiple times in a[]	1 (valid)
v = 12, a = nullptr	a[] is null	Error
v = 9, a = {2, 4, 6, 'B', 10}	a[] contains non-integer values	Error
v = 'A', a = {5, 10, 15, 20, 25}	v is not an integer	Error

Input Data	Description	Expected Outcome
x = 7, arr = {7}	Array contains a single element, and x is present	0
x = 2, arr = {7}	Array contains a single element, and x is absent	-1
x = 8, arr = {1, 3, 5, 6, 7}	x is outside the array's range and not present	-1
x = 4, arr = {}	Empty array	-1

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 Partition

Input Data	Description	Expected Outcome
arr = [2, 4, 6, 8, 6], x = 6	Array contains x, which appears more than once	2
arr = [1, 2, 4, 5], x = 3	Array does not contain x	0

arr = [], x = 4	Empty array	0
arr = [2, 4, 6, 6, 8], x = 6	Array contains x multiple times	2

Boundary Value

Input Data	Description	Expected Outcome
arr = [9], x = 9	Array contains one element, and x matches	1
arr = [8], x = 2	Array contains one element, and x does not match	0
arr = [1, 3, 5, 7, 9, 11], x = 5	Array has multiple elements, and x appears once	1
arr = [4, 6, 8, 10], x = 4	x matches the first element of the array	1
arr = [2, 4, 6, 8], x = 8	x matches the last element of the array	1

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 Class

Input Data	**Description**	**Expected Outcome**
v = 4, a = {2, 4, 6, 8, 10}	v is found in the array a[]	1 (valid)
v = 9, a = {3, 6, 7, 11, 13}	v is not present in the array a[]	-1 (valid)
v = 2, a = {}	The array a[] is empty	-1 (valid)
v = 0, a = {1, 2, 3, 5, 7}	v is smaller than the smallest element	0 (valid)
v = 12, a = {2, 4, 6, 8, 10}	v exceeds the largest value in the array	0 (valid)
v = 7, a = nullptr	The array a[] is null	Error
v = 5, a = {1, 'x', 5, 7}	a[] contains non-integer values	Error
v = 'c', a = {2, 4, 6, 8, 10}	v is not an integer	Error

Boundary Value

Input Data	Description	Expected Outcome
v = 7, a = {}	The array is empty	-1
v = 12, a = {2, 4, 6, 8, 10}	v is outside the bounds of the elements in the array	-1

v = 4, a = {8}	Array contains one element, but v is not in it	-1
v = 8, a = {8}	Array contains one element, and v matches the element	0
v = 6, a = {3, 6}	Array has two elements, and v is in the second half	1
v = 3, a = {3, 6}	Array has two elements, and v is found in the first half	0

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);
}
```

Input Data	Description	Expected Outcome
a = 5, b = 5, c = 5	All sides are equal (equilateral triangle)	EQUILATERAL (0)
a = 4, b = 4, c = 6	Two sides are equal (isosceles triangle)	ISOSCELES (1)
a = 3, b = 4, c = 5	No sides are equal (scalene triangle)	SCALENE (2)
a = 1, b = 2, c = 3	Triangle inequality violated (invalid triangle)	INVALID (3)
a = 0, b = 4, c = 5	One side is zero (invalid triangle)	INVALID (3)
a = -1, b = 4, c = 5	One side is negative (invalid triangle)	INVALID (3)

Input Data	Description	Expected Outcome
a = 1, b = 1, c = 1	Minimum valid values where all sides are equal (equilateral)	EQUILATERAL (0)
a = 2, b = 2, c = 3	Two sides are equal and one different (isosceles)	ISOSCELES (1)
a = 2, b = 3, c = 4	All sides are different (scalene)	SCALENE (2)
a = 1, b = 2, c = 3	Triangle inequality violated (invalid)	INVALID (3)
a = 0, b = 2, c = 2	One side is zero, boundary of invalid input	INVALID (3)

a = 1, b = 10, c = 20	Large difference between sides, invalid triangle	INVALID (3)

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;
}
```

Input Data	**Description**	**Expected Outcome**
s1 = "pre", s2 = "prefix"	s1 is a valid prefix of s2	TRUE
s1 = "fix", s2 = "prefix"	s1 is not a prefix of s2	FALSE
s1 = "prefix", s2 = "prefix"	s1 is exactly equal to s2	TRUE
s1 = "longer", s2 = "short"	s1 is longer than s2	FALSE
s1 = "", s2 = "prefix"	s1 is an empty string (empty string is always a valid prefix)	TRUE

Input Data	Description	Expected Outcome
s1 = "a", s2 = "abc"	s1 is one character and is a prefix of s2	TRUE
s1 = "a", s2 = "bcd"	s1 is one character and is not a prefix of s2	FALSE
s1 = "abc", s2 = "abc"	s1 is exactly equal to s2	TRUE
s1 = "abcd", s2 = "abc"	s1 is longer than s2	FALSE
s1 = "", s2 = "abc"	s1 is an empty string (empty string is always a prefix)	TRUE
s1 = "abc", s2 = ""	s1 is non-empty, but s2 is empty	FALSE

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
Equilateral triangle (all sides equal)
Isosceles triangle (two sides equal)
Scalene triangle (all sides unequal)
Right-angled triangle (satisfies Pythagorean theorem)
Invalid triangle (fails triangle inequality)
Non-triangle case (sides are zero or negative)
Non-positive input (any side length <= 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)

A	B	C	Expected Result	Equivalence Class
5	5	5	Equilateral	Equilateral Triangle
6	6	9	Isosceles	Isosceles Triangle
4	5	6	Scalene	Scalene Triangle
3	4	5	Right-Angled	Right-Angled Triangle
1	2	4	Invalid (Non-Triangle)	Non-Triangle
-1	3	4	Invalid (Non-positive)	Non-Positive Input
0	4	5	Invalid (Non-positive)	Non-Positive Input

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

A	B	C	Expected Result	Boundary Condition
3	4	7	Invalid (Non-Triangle)	Boundary (A + B = C)

3	4	6.99	Scalene	Boundary (A + B > C)

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

A	**B**	**C**	**Expected Result**	**Boundary Condition**
5	5	8	Isosceles	Boundary (A = B, A ≠ C)
5	8	5	Isosceles	Boundary (A = C, A ≠ B)

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

A	**B**	**C**	**Expected Result**	**Boundary Condition**
6	6	6	Equilateral	Boundary (A = B = C)

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

A	**B**	**C**	**Expected Result**	**Boundary Condition**
3	4	5	Right-Angled	Boundary (A2 + B2 = C2)

5	12	13	Right-Angled	Boundary (A2+ B2 = C2)
6	8	10	Right-Angled	Boundary (A2 + B2 = C2)

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

A	**B**	**C**	**Expected Result**	**Boundary Condition**
1	2	3	Invalid (Non-Triangle)	Boundary (A + B = C)
2	2	5	Invalid (Non-Triangle)	Boundary (A + B < C)

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

A	B	C	Expected Result	Test Point
-2	4	5	Invalid (Non-positive)	Negative Input
0	3	4	Invalid (Non-positive)	Zero Input
-1	-2	-3	Invalid (Non-positive)	Negative Input