IT-314 SoftWare Engineering
Lab : 8

FUNCTIONAL TESTING

Name : Harmit Khimani

Student ID : 202201231

Lab Group : 3

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.

The solution of each problem must be given in the format as follows:

Tester Action and Input Data	***Expected Outcome***
Equivalence Partitioning	
a, b, c	An Error message
a-1, b, c	Yes
Boundary Value Analysis	
a, b, c-1	Yes

Equivalence Classes:

EC	Range/Value	Description
EC1	1<=month<=12	Valid
EC2	month<1	Invalid
EC3	month>12	Invalid
EC4	1<=day<=31	Valid for month with 31 days
EC5	1<=day<=30	Valid for months with 30 days
EC6	1<=day<=29	Valid for February in leap years
EC7	1<=day<=28	Valid for February in non leap years
EC8	day<1	Invalid
EC9	day>31	Invalid
EC10	day=31 in 30 day month	Invalid

EC11	Day =30 or 31 inFebruary	Invalid
EC12	1900<=year<=2015	Valid
EC13	year<1900	Invalid
EC14	YEAR>2015	Invalid
EC15	year%4==0,year%100!=0or year%400==0	LeapYear

Equivalence Class Test Cases:

Test Case	Day	Month	Year	Description	Expected Outcome	EC
TC1	15	5	2010	Valid	14/5/2010	1,4,12
TC2	31	1	2000	Valid	30/12/19	1,4,12
TC3	29	2	2004	Valid ,Leap Year	28/2/2004	1,6,12,15
TC4	28	2	2005	Valid	27/2/2005	1,7,12
TC5	30	4	2012	Valid	~~29//4/2012~~	1,5,12
TC6	31	9	1998	Invalid	~~Error~~	1,10,12
TC7	32	5	2011	Invalid	~~Error~~	1,9,12
TC8	10	13	2005	Invalid	~~Error~~	3,4,12
TC9	15	5	1899	Invalid	~~Error~~	1,4,13
TC10	15	5	2016	Invalid		1,4,14

Boundary Value Analysis:

Consider all the edge cases which are:-

- Day : 1, 31 (for month with 31 days), 30 (for months with 30 days), 28/29 (February for non leap/leap years)
- Month:1and12
- Year:1900and2015

Test Case	Day	Month	Year	Description	Expected Outcome	EC
TC11	1	1	1900	Boundary: Start of year	31/12/1899	1,4,12
TC12	31	12	2015	Boundary: end of year	30/12/15	1,4,12
TC13	1	2	1900	Boundary: First day of Month	31/1/1900	1,6,12,15
TC14	31	3	2015	Boundary: Lastday of Month	30/3/15	1,7,12
TC15	29	2	2004	Boundary: LeapYear	28/2/2004	1,5,12
TC16	29	2	2005	Invalid	Error	1,10,12
TC17	30	4	2010	Boundary: Lastday of Month	29/4/10	1,9,12
TC18	1	3	2015	Boundary: First day of Month	28/2/15	3,4,12
TC19	15	6	2005	Valid	14/6/2004	1,4,1
TC20	1	1	1899	Invalid	Error	3
						1,4,1

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);
}
```

TestCase	Method	Description	Expected Outcome
TC1	Equivalence Partitioning	linearSearch(3 , [1 2 3 4])	3
TC2	Equivalence Partitioning	linearSearch(3 , [ ])	Error Empty Array
TC3	Equivalence Partitioning	linearSearch(7 , [1 2 3 4])	-1
TC4	Equivalence Partitioning	linearSearch(3 , [1 2 3 3 4])	3
TC5	Equivalence Partitioning	linearSearch(“abcv” , [1 2 3 4])	Error Wrong Input
TC6	Partitioning Equivalence	linearSearch(3.14 , [1 2 3 4])	Error Wrong Input
TC7	Partitioning Equivalence	linearSearch(NULL , [1 2 3 4])	Error Empty Array
TC8	Partitioning	linearSearch(4 , NULL)	Error No Input

B1	Boundary Value Analysis	linearSearch(1 , [1 2 3 4])	1
B2	Boundary Value Analysis	linearSearch(4 , [1 2 3 4])	4
B3	Boundary Value Analysis	linearSearch(0 , [1 2 3 4])	-1
B4	Boundary Value Analysis	linearSearch(4 , [4])	1

MODIFIED CODE:

```
#include <iostream>

#include <vector>

#include <limits>


int linearSearch(int v, const std::vector<int> &a)

{

   if (a.empty())

   {

      std::cout << "Error: Input array is empty." << std::endl;

      return -1;

   }


   for (size_t i = 0; i < a.size(); ++i)

   {
```

```
        if (a[i] == v)
            return static_cast<int>(i);
    }

    return -1;
}

int main()
{
    std::vector<int> array1 = {1, 2, 3, 4};
    std::vector<int> array2 = {};
    std::vector<int> array3 = {-2, -1, 0, 1};
    int value;

    while (true)
    {
        std::cout << "Enter a value to search (or -999 to exit): ";
        std::cin >> value;

        if (std::cin.fail() || std::cin.peek() != '\n')
        {
            std::cout << "Error: Invalid input type." << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```
            continue;
        }

        if (value == -999)
            break;

        std::cout << "Test Case (array1): " << linearSearch(value, array1) << std::endl;
        std::cout << "Test Case (array2): " << linearSearch(value, array2) << std::endl;
        std::cout << "Test Case (array3): " << linearSearch(value, array3) << std::endl;
    }

    return 0;
}
```

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);
```

TestCase	Method	Description	Expected Outcome
TC1	Equivalence Partitioning	linearSearch(3 , [1 2 3 4])	1
TC2	Equivalence Partitioning	linearSearch(3 , [ ])	Error Empty Array
TC3	Equivalence Partitioning	countItem(7 , [1 2 3 4])	0
TC4	Equivalence Partitioning	countItem(3,[12334])	2
TC5	Equivalence Partitioning	countItem(“abcv” , [1 2 3 4])	Error Wrong Input
TC6	Equivalence Partitioning	countItem(3.14 , [1 2 3 4])	Error Wrong Input
TC7	Equivalence Partitioning	countItem(NULL , [1 2 3 4])	Error Empty Array
TC8	Boundary Value Analysis	countItem(4 , NULL)	Error No Input
B1	Boundary Value Analysis	countItem(1 , [1 2 3 4])	1
B2	Boundary Value Analysis	countItem(4 , [1 2 3 4])	1
B3	Boundary Value Analysis	countItem(0 , [1 2 3 4])	0
B4	Value Analysis	countItem(4 , [4])	1

MODIFIED CODE:

```
#include <iostream>
#include <vector>
#include <limits>
#include <type_traits>


int countItem(int v, const std::vector<int>& a) {
    int count = 0;
    for (size_t i = 0; i < a.size(); i++) {
        if (a[i] == v)
            count++;
    }
    return count;
}


int main() {
    std::vector<int> array1 = {1, 2, 3, 4, 3, 3};
    std::vector<int> array2 = {};
    std::vector<int> array3 = {-2, -1, 0, 1};
    int value;
    while (true) {
        std::cout << "Enter a value to count (or -999 to exit): ";
        std::cin >> value;
```

```
        if (std::cin.fail() || std::cin.peek() != '\n') {
            std::cout << "Error: Invalid input type." << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (value == -999)
            break;
        std::cout << "Count in array1: " << countItem(value, array1) << std::endl;
        std::cout << "Count in array2: " << countItem(value, array2) << std::endl;
        std::cout << "Count in array3: " << countItem(value, array3) << std::endl;
    }
    return 0;
}
```

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);
}
```

TestCase	Method	Description	Expected Outcome
TC1	Equivalence Partitioning	BinarySearch(3 , [1 2 3 4])	3
TC2	Equivalence Partitioning	BinarySearch(3 , [ ])	Error Empty Array
TC3	Equivalence Partitioning	BinarySearch(7 , [1 2 3 4])	-1
TC4	Equivalence Partitioning	BinarySearch(3 , [1 2 3 3 4])	3
TC5	Equivalence Partitioning	BinarySearch(“abcv” , [1 2 3 4])	Error Wrong Input
TC6		BinarySearch(3.14 , [1 2 3 4])	Error

	Partitioning		Wrong Input
TC7	Equivalence Partitioning	BinarySearch(NULL , [1 2 3 4])	Error Empty Array
TC8	Equivalence Partitioning	BinarySearch(4 , NULL)	Error No Input
B1	Boundary Value Analysis	BinarySearch(1 , [1 2 3 4])	1
B2	Boundary Value Analysis	BinarySearch(4 , [1 2 3 4])	4
B3	Boundary Value Analysis	BinarySearch(0 , [1 2 3 4])	-1
B4	Boundary Value Analysis	BinarySearch(4 , [4])	1

MODIFIED CODE:

```
#include <iostream>

#include <vector>

#include <limits>

#include <type_traits>


int binarySearch(int v, const std::vector<int>& a) {

   int lo = 0, hi = a.size() - 1;

   while (lo <= hi) {

      int mid = (lo + hi) / 2;
```

```
        if (v == a[mid])
            return mid;
        else if (v < a[mid])
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    return -1;
}
int main() {
    std::vector<int> array1 = {1, 2, 3, 4, 5};
    std::vector<int> array2 = {};
    std::vector<int> array3 = {-3, -2, -1, 0, 1};
    int value;
    while (true) {
        std::cout << "Enter a value to search (or -999 to exit): ";
        std::cin >> value;
        if (std::cin.fail() || std::cin.peek() != '\n') {
            std::cout << "Error: Invalid input type." << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
```

```
        }
        if (value == -999)
            break;
        std::cout << "Index in array1: " << binarySearch(value, array1) << std::endl;
        std::cout << "Index in array2: " << binarySearch(value, array2) << std::endl;
        std::cout << "Index in array3: " << binarySearch(value, array3) << std::endl;
    }
    return 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);
}
```

TestCase	Method	Description	Expected Outcome
TC1	Equivalence Partitioning	triangle(3, 3 , 3)	0
TC2	Equivalence Partitioning	triangle(3, 3 , 2)	1
TC3	Equivalence Partitioning	triangle(3, 2 , 1)	2
TC4	Equivalence Partitioning	triangle(3, 3, 6)	3
TC5	Equivalence Partitioning	triangle(1, 1, 2)	3
TC6	Equivalence Partitioning	triangle(0, 1, 2)	3
TC7	Equivalence Partitioning	triangle(3, 2, -1)	Error Invalid Input
TC8		triangle(1.5, 3, 3)	Error

	Partitioning		Invalid Input
B1	Boundary Value Analysis	triangle(1, 1, 1)	0
B2	Boundary Value Analysis	triangle(3, 3, 4)	1
B3	Boundary Value Analysis	triangle(1, 2, 10)	2
B4	Boundary Value Analysis	triangle(1, 2, 3)	3

MODIFIED CODE:

```
#include <iostream>

#include <limits>

#include <type_traits>

const int EQUILATERAL = 0;

const int ISOSCELES = 1;

const int SCALENE = 2;

const 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;
}
int main() {
    int a, b, c;
    while (true) {
        std::cout << "Enter three sides of a triangle (or -999 to exit): ";
        std::cin >> a >> b >> c;
        if (std::cin.fail() || std::cin.peek() != '\n') {
            std::cout << "Error: Invalid input type." << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        if (a == -999 || b == -999 || c == -999)
            break;
        int result = triangle(a, b, c);
        if (result == INVALID)
            std::cout << "Triangle is invalid." << std::endl;
        else if (result == EQUILATERAL)
            std::cout << "Triangle is equilateral." << std::endl;
        else if (result == ISOSCELES)
            std::cout << "Triangle is isosceles." << std::endl;
        else
            std::cout << "Triangle is scalene." << std::endl;
```

```
    }
    return 0;
}
```

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;
}
```

Test Case	Method	Input	Expected Outcome
E1	Equivalence Partitioning	prefix("pre", "pref")	True
E2	Equivalence Partitioning	prefix("pre", "testinghere")	False

E3	Equivalence Partitioning	prefix("long", "longest")	True
E4	Equivalence Partitioning	prefix("", "anystring")	True
E5	Equivalence Partitioning	prefix("string", "wellll")	False
E6	Equivalence Partitioning	prefix("abc", "abcdefghi")	True
E7	Equivalence Partitioning	prefix("abcd", "abc")	False
E8	Equivalence Partitioning	prefix("0152123", "12345")	False
E9	Equivalence Partitioning	prefix("abc", "1abc")	True
E10	Equivalence Partitioning	prefix("prefix", "pre")	False
E11	Equivalence Partitioning	prefix("abc", null)	Error: Invalid input type.

E12	Equivalence Partitioning	prefix(null, "abc")	Error: Invalid input type.
E13	Equivalence Partitioning	prefix("", null)	Error: Invalid input type.
E14	Equivalence Partitioning	prefix(null, null)	Error: Invalid input type.
B1	Boundary Value Analysis	prefix("abc", "ab")	True
B2	Boundary Value Analysis	prefix("abc", "abcc")	True
B3	Boundary Value Analysis	prefix("abc", "ab")	False
B4	Boundary Value Analysis	prefix("", "")	True
B5	Boundary Value Analysis	prefix("longprefix", "prefix")	False

MODIFIED CODE:

```
#include <iostream>
#include <string>
bool prefix(const std::string& s1, const std::string& s2) {
    if (s1.length() > s2.length())
        return false;
    for (size_t i = 0; i < s1.length(); ++i) {
        if (s1[i] != s2[i])
            return false;
    }
    return true;
}
int main() {
    std::string s1, s2;
    while (true) {
        std::cout << "Enter two strings (or 'exit' to quit): ";
        std::getline(std::cin, s1);
        if (s1 == "exit")
            break;
        std::getline(std::cin, s2);
        if (s2 == "exit")
            break;
        if (s1.empty() || s2.empty()) {
```

```
            std::cout << "Error: Strings cannot be empty." << std::endl;
            continue;
        }
        bool result = prefix(s1, s2);
        if (result) {
            std::cout << "The first string is a prefix of the second string." << std::endl;
        } else {
            std::cout << "The first string is not a prefix of the second string." << std::endl;
        }
    }
    return 0;
}
```

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 $A_2 + B_2 = C_2$ 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) Identify the Equivalence Classes

1. **Valid Inputs:**

- Class1:EquilateralTriangle(A=B=C)
- Class2:IsoscelesTriangle(A=BorA=CorB=C,butnotallequal)
- Class3:ScaleneTriangle(A≠B≠C)
- Class4:Right-AngledTriangle($A^2+B^2=C^2$oranypermutation)

2. **Abstract Inputs:**

- Class5:Non-Triangle(A+B≤Coranypermutation)
- Class 6: invalid input types (A < 0, B < 0, or C < 0,strings,characters etc)

b) Identify Test Cases for Equivalence Classes

TestCase	Tester Input	Expected Outcome	Equivalence Class
E1	triangle(3, 3, 3)	Equilateral	Class 1
E2	triangle(2, 2, 3)	Isosceles	Class 2
E3	triangle(3, 4, 5)	Scalene	Class 3
E4	triangle(3, 4, 6)	Non-Triangle	Class 5

E5	triangle(1, 1, 2)	Non-Triangle	Class 5
E6	triangle(0, 2, 2)	Non-Triangle	Class 5
E7	triangle(-1, 2, 2)	Error: Invalid input type.	Class 6
E8	triangle("abc", 2, 2)	Error: Invalid input type.	Class 6
E9	triangle(3, 4, 5)	Right-Angled	Class 4

c) Boundary Condition A + B > C (Scalene Triangle)

Test Case ID	Tester Action and Input Data	Expected Outcome
B1	triangle(2.0, 3.0, 4.0)	Scalene
B2	triangle(2.0, 3.0, 5.0)	Non-Triangle
B3	triangle(3.0, 3.0, 5.0)	Non-Triangle

d) Boundary Condition A = C (Isosceles Triangle)

Test Case ID	Tester Action and Input Data	Expected Outcome
B4	triangle(3.0, 4.0, 3.0)	Isosceles
B5	triangle(3.0, 2.0, 3.0)	Isosceles

e) Boundary Condition A = B = C (Equilateral Triangle)

Test Case ID	Tester Action and Input Data	Expected Outcome
B6	triangle(3.0, 3.0, 3.0)	Equilateral
B7	triangle(0.0, 0.0, 0.0)	Non-Triangle

f) Boundary Condition $A^2 + B^2 = C^2$ (Right-Angle Triangle)

Test Case ID	Tester Action and Input Data	Expected Outcome
B8	triangle(3.0, 4.0, 5.0)	Right-Angled
B9	triangle(5.0, 12.0, 13.0)	Right-Angled
B10	triangle(1.0, 1.0, 1.414)	Right-Angled

B11	triangle(1.0, 2.0, 2.236)	Non-Triangle

g) Non-Triangle Case Test Cases

Test Case ID	Tester Action and Input Data	Expected Outcome
N1	triangle(1.0, 1.0, 3.0)	Non-Triangle
N2	triangle(5.0, 10.0, 4.0)	Non-Triangle

h) Non-Positive Input Test Cases

Test Case ID	Tester Action and Input Data	Expected Outcome
P1	triangle(-1.0, 2.0, 2.0)	Error: Invalid input type.
P2	triangle(0.0, 0.0, 2.0)	Non-Triangle
P3	triangle(1.0, -2.0, 2.0)	Error: Invalid input type.