IT314 Software engineering

Vivek Patel

202201025

Lab 8: Functional Testing (Black-Box)

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?

Equivalence Classes

- E1: 1 <= month <= 12 (Valid)
- E2: month < 1 (Invalid)
- E3: month > 12 (Invalid)
- E4: Valid Day (1<=day<=31)
- E5: Day < 1 (Invalid)
- E6: Day > 31 (Invalid)
- E7: Valid Year (1900 <= year <= 2015)
- E8: Year < 1900 (Invalid)
- E9: Year > 2015 (Invalid)

Test Cases:

Test Case	Classes Covered	Expected outcome
(1, 1, 1900)	E1, E4, E7	Valid Date
(0, 1, 2010)	E2	Invalid Date
(13, 1, 2010)	E3	Invalid Date
(1, 0, 2010)	E5	Invalid Date
(1, 32, 2010)	E6	Invalid Date
(1, 1, 1899)	E8	Invalid Date
(1, 1, 2016)	E9	Invalid Date

Q2) 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:

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.

Equivalence Partitioning

Tester action and input data	**Expected outcome**
linearSearch(5, [1, 2, 3, 4, 5])	4
linearSearch(6,[1, 2, 3, 4, 5])	-1
linearSearch(1, [])	-1
linearSearch(1, [1, 2, 3])	0
linearSearch(6, [6, 6, 6, 6])	0
linearSearch(1.09, [1, 2, 3, 4, 5])	Error message
linearSearch(10, NULL)	Error message

Boundary Value Analysis

Tester Action and input data	Expected outcome
linearSearch(1, [1, 2, 3])	0
linearSearch(3, [1, 2, 3])	2
linearSearch(0, [1, 2, 3])	-1
linearSearch(4, [1, 2, 3])	-1
linearSearch(-1, [1, 2, 3])	-1

Modified Linear Search Program

```
#include <stdio.h>
int linearSearch(int v, int a[], int length) {
    if (a == NULL) {
        printf("Error: Null array passed.\n");
        return -1;
    }
    for (int i = 0; i < length; i++) {
        if (a[i] == v) {
            return i;
        }
    }
    return -1;
}


int main() {
```

```c
    // Test cases
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {3, 1, 3, 4};
    int arr3[] = {1, 2, 3};

    // Equivalence Partitioning
    printf("Test Case 1: %d\n", linearSearch(5, arr1, 5)); // Expected: 4
    printf("Test Case 2: %d\n", linearSearch(6, arr1, 5)); // Expected: -1
    printf("Test Case 3: %d\n", linearSearch(1, NULL, 0)); // Expected: Error message
    printf("Test Case 4: %d\n", linearSearch(1, arr3, 3)); // Expected: 0
    printf("Test Case 5: %d\n", linearSearch(3, arr2, 4)); // Expected: 0

    // Test Case 6 is not valid in C since we can't pass a float to an int parameter
    // Uncommenting this will lead to a compilation error, so it's not included here.

    // Boundary Value Analysis
    printf("Boundary Test Case 1: %d\n", linearSearch(1, arr3, 3)); // Expected: 0
    printf("Boundary Test Case 2: %d\n", linearSearch(3, arr3, 3)); // Expected: 2
    printf("Boundary Test Case 3: %d\n", linearSearch(0, arr3, 3)); // Expected: -1
    printf("Boundary Test Case 4: %d\n", linearSearch(4, arr3, 3)); // Expected: -1
    printf("Boundary Test Case 5: %d\n", linearSearch(-1, arr3, 3)); // Expected: -1

    return 0;
}
```

P2. The function countItem returns the number of times a value v appears in an array of integers a.

Equivalence Partitioning

Tester action and input data	Expected outcome
countItem(2, [1, 2 ,3 ,2 ,4])	2
countItem(5, [1, 2, 3, 2, 4])	0
countItem(1, [])	0
countItem(3, [3, 3, 3])	3
countItem(4, [1, 2, 3]	0
countItem(2.5, [1, 2, 3, 2, 4]	Error message
countItem(1, NULL)	Error message

Boundary Value Analysis

Tester action and input data	Expected outcome
countItem(1, [1, 2, 3])	1
countItem(2, [1, 2, 3])	1
countItem(3, [1, 2, 3])	1
countItem(0, [1, 2, 3])	0
countItem(-1, [1, 2, 3])	0

Modified countItem Program:

```
#include <stdio.h>

int countItem(int v, int a[], int length) {

   if (a == NULL) {
       printf("Error: Null array passed.\n");
       return -1;
   }
   int count = 0;
   for (int i = 0; i < length; i++) {
       if (a[i] == v) {
           count++;
       }
   }
   return count;
}

int main() {
   // Test cases
   int arr1[] = {1, 2, 3, 2, 4};
   int arr2[] = {3, 3, 3};

   // Equivalence Partitioning
   printf("Test Case 1: %d\n", countItem(2, arr1, 5)); // Expected: 2
   printf("Test Case 2: %d\n", countItem(5, arr1, 5)); // Expected: 0
   printf("Test Case 3: %d\n", countItem(1, NULL, 0)); // Expected: Error message
   printf("Test Case 4: %d\n", countItem(3, arr2, 3)); // Expected: 3
   printf("Test Case 5: %d\n", countItem(4, arr1, 5)); // Expected: 0

   // Boundary Value Analysis
   printf("Boundary Test Case 1: %d\n", countItem(1, arr1, 5)); // Expected: 1
```

```
  printf("Boundary Test Case 2: %d\n", countItem(2, arr1, 5)); // Expected: 2

  printf("Boundary Test Case 3: %d\n", countItem(3, arr1, 5)); // Expected: 1

  printf("Boundary Test Case 4: %d\n", countItem(0, arr1, 5)); // Expected: 0

  printf("Boundary Test Case 5: %d\n", countItem(-1, arr1, 5)); // Expected: 0

  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.

Equivalence Partitioning

Tester action and input data	Expected outcome
binarySearch(3, [1, 2, 3, 4, 5])	2
binarySearch(0, [1, 2, 3, 4, 5])	-1
binarySearch(6, [1, 2, 3, 4, 5])	-1
binarySearch(1, [1, 2, 3, 4, 5])	0
binarySearch(5, [1, 2, 3, 4, 5])	4
binarySearch(3.5, [1, 2, 3, 4, 5])	Error message
binarySearch(1, NULL)	Error message

Boundary Value Analysis

Tester action and input data	Expected outcome
binarySearch(1, [1, 2, 3])	0
binarySearch(3, [1, 2, 3])	2
binarySearch(2, [1, 2, 3])	1
binarySearch(0, [1, 2, 3])	-1
binarySearch(4, [1, 2, 3])	-1

Modified Binary Search Program:-

```
#include <stdio.h>

int binarySearch(int v, int a[], int length) {
    if (a == NULL) {
        printf("Error: Null array passed.\n");
        return -1;
    }

    int lo = 0, hi = length - 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() {
```

```c
    // Test cases
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {1, 2, 3};

    // Equivalence Partitioning
    printf("Test Case 1: %d\n", binarySearch(3, arr1, 5)); // Expected: 2
    printf("Test Case 2: %d\n", binarySearch(0, arr1, 5)); // Expected: -1
    printf("Test Case 3: %d\n", binarySearch(6, arr1, 5)); // Expected: -1
    printf("Test Case 4: %d\n", binarySearch(1, arr1, 5)); // Expected: 0
    printf("Test Case 5: %d\n", binarySearch(5, arr1, 5)); // Expected: 4

    // Test case for non-integer input is not valid in C
    // printf("Test Case 6: %d\n", binarySearch(3.5, arr1, 5)); // Uncommenting will lead to compilation error

    printf("Test Case 7: %d\n", binarySearch(1, NULL, 0)); // Expected: Error message

    // Boundary Value Analysis
    printf("Boundary Test Case 1: %d\n", binarySearch(1, arr2, 3)); // Expected: 0
    printf("Boundary Test Case 2: %d\n", binarySearch(3, arr2, 3)); // Expected: 2
    printf("Boundary Test Case 3: %d\n", binarySearch(2, arr2, 3)); // Expected: 1
    printf("Boundary Test Case 4: %d\n", binarySearch(0, arr2, 3)); // Expected: -1
    printf("Boundary Test Case 5: %d\n", binarySearch(4, arr2, 3)); // Expected: -1

    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 length).

Equivalence Partitioning

Tester action and input data	Expected outcome
triangle(3, 3, 3)	0 (EQUILATERAL)
triangle(3, 4, 3)	1 (ISOSCELES)
triangle(3, 4, 5)	2 (SCALENE)
triangle(1, 1, 2)	3 (INVALID)
triangle(0, 1, 1)	3 (INVALID)
triangle(-1, 2, 3)	3 (INVALID)
triangle(1.5, 1.5, 1.5)	Error message

Boundary Value Analysis

Tester action and input data	Expected outcome
triangle(1, 1, 1)	0 (EQUILATERAL)
triangle(2, 2, 3)	**1 (ISOSCELES)**
triangle(3, 3, 4)	**1 (ISOSCELES)**
triangle(2, 2, 5)	**3 (INVALID)**
triangle(3, 4, 7)	**3 (INVALID)**

Modified triangle Program

```
#include <stdio.h>

#define EQUILATERAL 0
#define ISOSCELES 1
#define SCALENE 2
#define INVALID 3

int triangle(int a, int b, int c) {
  if (a <= 0 || b <= 0 || c <= 0 || 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() {
    // Test cases
    printf("Test Case 1: %d\n", triangle(3, 3, 3)); // Expected: 0 (EQUILATERAL)
    printf("Test Case 2: %d\n", triangle(3, 4, 3)); // Expected: 1 (ISOSCELES)
    printf("Test Case 3: %d\n", triangle(3, 4, 5)); // Expected: 2 (SCALENE)
    printf("Test Case 4: %d\n", triangle(1, 1, 2)); // Expected: 3 (INVALID)
    printf("Test Case 5: %d\n", triangle(0, 1, 1)); // Expected: 3 (INVALID)
    printf("Test Case 6: %d\n", triangle(-1, 2, 3)); // Expected: 3 (INVALID)

    // Non-integer inputs are not valid in C
    // printf("Test Case 7: %d\n", triangle(1.5, 1.5, 1.5)); // Uncommenting will lead to compilation error

    // Boundary Value Analysis
    printf("Boundary Test Case 1: %d\n", triangle(1, 1, 1)); // Expected: 0 (EQUILATERAL)
    printf("Boundary Test Case 2: %d\n", triangle(2, 2, 3)); // Expected: 1 (ISOSCELES)
    printf("Boundary Test Case 3: %d\n", triangle(3, 3, 4)); // Expected: 1 (ISOSCELES)
    printf("Boundary Test Case 4: %d\n", triangle(2, 2, 5)); // Expected: 3 (INVALID)
    printf("Boundary Test Case 5: %d\n", triangle(3, 4, 7)); // Expected: 3 (INVALID)

    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).

Equivalence Partitioning

Tester action and input data	Expected outcome
prefix(“pre”, “prefix”)	true
prefix(“abc”, “abcdef”)	true
prefix(“test”, “testing”)	true
prefix(“longprefix”, “short”)	false
prefix(“wrong”, “right”)	false
prefix(“”, “nonempty”)	true
prefix(“nonempty”, “”)	false

Boundary Value Analysis:

Tester action and input data	Expected outcome
prefix(“”, “”)	true
prefix(“a”, “a”)	true
prefix(“ab”, “a”)	false
prefix(“long”, “long”)	true
prefix(“short”, “longer”)	false

Modified prefix Program

```
public class PrefixTest {
  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;
  }

  public static void main(String[] args) {
    // Test cases
    System.out.println("Test Case 1: " + prefix("pre", "prefix")); // Expected: true
    System.out.println("Test Case 2: " + prefix("abc", "abcdef")); // Expected: true
    System.out.println("Test Case 3: " + prefix("test", "testing")); // Expected: true
    System.out.println("Test Case 4: " + prefix("longprefix", "short")); // Expected: false
    System.out.println("Test Case 5: " + prefix("wrong", "right")); // Expected: false
    System.out.println("Test Case 6: " + prefix("", "nonempty")); // Expected: true
    System.out.println("Test Case 7: " + prefix("nonempty", "")); // Expected: false

    // Boundary Value Analysis
    System.out.println("Boundary Test Case 1: " + prefix("", "")); // Expected: true
    System.out.println("Boundary Test Case 2: " + prefix("a", "a")); // Expected: true
    System.out.println("Boundary Test Case 3: " + prefix("ab", "a")); // Expected: false
    System.out.println("Boundary Test Case 4: " + prefix("long", "long")); // Expected: true
    System.out.println("Boundary Test Case 5: " + prefix("short", "longer")); // Expected: 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

1. E1: Equilateral Triangle
   - All sides are equal (A = B = C).
2. E2: Isosceles Triangle
   - Exactly two sides are equal (A = B ≠ C, A = C ≠ B, B = C ≠ A).
3. E3: Scalene Triangle
   - All sides are different (A ≠ B, B ≠ C, A ≠ C).
4. E4: Right-Angled Triangle

- Satisfies Pythagorean theorem ($A^2 + B^2 = C^2$, with A and B as the shorter sides).

5. E5: Invalid Triangle
    - Does not satisfy the triangle inequality ($A + B \leq C$ or $A + C \leq B$ or $B + C \leq A$).
6. E6: Non-positive Input
    - Any side length is non-positive ($A \leq 0$, $B \leq 0$, $C \leq 0$).

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)

Test Case	**A**	**B**	**C**	**Expected outcome**	**Equivalence class**
1	3	3	3	Equilateral	E1 (Equilateral triangle)
2	3	3	4	Isosceles	E2 (Isosceles triangle)
3	3	4	5	Scalene	E3 (Scalene triangle)
4	3	4	6	Invalid	E5 (Invalid triangle)
5	5	12	13	Right Angled	E4 (Right Angled triangle)
6	0	2	3	Invalid	E6 (Non-positive Input)

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

Test Case	A	B	C	Expected outcome
Boundary test case 1	2	3	4.5	Scalene
Boundary test case 2	2	2	3.9	Scalene

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

Test case	A	B	C	Expected outcome
Boundary Test case 1	3	4	3	Isosceles
Boundary test case 2	5	2	5	Isosceles

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

Test Case	A	B	C	Expected outcome
Boundary Test Case 1	3	3	3	Equilateral
Boundary Test Case 2	5	5	5	Equilateral

f) For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle),

identify test cases to verify the boundary.

Test Case	**A**	**B**	**C**	**Expected outcome**
Boundary Test Case 1	3	4	5	Right Angled
Boundary Test case 2	5	12	13	Right Angled

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

Test Case	**A**	**B**	**C**	**Expected Outcome**
Boundary Test Case 1	2	3	6	Invalid
Boundary Test Case 2	1	1	3	Invalid

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

Test Case	**A**	**B**	**C**	**Expected Outcome**
Test Case 1	0	2	3	Invalid
Test Case 2	-1	2	3	Invalid
Test Case 3	1	0	3	Invalid
Test Case 4	1	2	-3	Invalid