IT314 (Software Engineering)

Software Testing

Lab Session 08 - Functional Testing (Black-Box)

Name: Aastha Alpeshbhai Bhavsar.

ID: 202201259.

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.

Ans.:

Equivalence Partitioning:-

Valid Equivalence Classes:

1. Valid Date Input:
    - Examples:
        - (1, 1, 1900)
        - (29, 2, 2012) (leap year)
        - (28, 2, 2015) (non-leap year)

Invalid Equivalence Classes:

1. Invalid Month:
    - Example:
        - (15, 1, 2000) (month > 12)
2. Invalid Day:
    - Examples:
        - (32, 1, 2000) (day > 31)
        - (1, 13, 2000) (day > days in month)
3. Invalid Year:
    - Examples:
        - (1, 1, 1899) (year < 1900)
        - (1, 1, 2016) (year > 2015)

Tester Action and Input Data	Expected Outcome
(1, 1, 1900)	Previous date: (31, 12, 1899)
(31, 12, 2015)	Previous date: (30, 12, 2015)
(15, 1, 2000)	An error message
(1, 0, 2000)	An error message
(29, 2, 2015)	An error message

Boundary Value Analysis:-

-> Valid Edge Cases:

- (1, 1, 1900) → Previous date: (31, 12, 1899)
- (28, 2, 2015) → Previous date: (27, 2, 2015)

-> Invalid Edge Cases:

- (0, 1, 1900) → Invalid input
- (29, 2, 2015) → Invalid input
- (1, 1, 1899) → Invalid input

Tester Action and Input Data	Expected Outcome
(1, 1, 1900)	Previous date: (31, 12, 1899)
(1, 1, 2015)	Previous date: (31, 12, 2014)
(29, 2, 2012)	Previous date: (28, 2, 2012)
(0, 1, 1900)	An error message
(1, 13, 1900)	An error message
(32, 1, 2015)	An error message
(1, 1, 1899)	An error message

You would modify the program to handle the input, validate it, and determine the previous date based on the given inputs. After implementing the logic, you can run the test suite to verify if the program returns the expected outcomes.

Modified program:

```
def previous_date(day, month, year):
    if year < 1900 or year > 2015:
        return "Invalid date"
    if month < 1 or month > 12:
        return "Invalid date"
    if day < 1 or day > 31:
        return "Invalid date"
```

```
    # Check for valid day according to month
    if month in [4, 6, 9, 11] and day > 30:
        return "Invalid date"
    if month == 2:
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            if day > 29:  # Leap year
                return "Invalid date"
        else:
            if day > 28:  # Non-leap year
                return "Invalid date"

    # Logic for finding the previous date
    if day == 1:
        if month == 1:
            return (31, 12, year - 1)
        else:
            month -= 1
            day = 31  # Assume the previous month has 31 days for simplification
            # Adjust day based on the previous month
            if month in [4, 6, 9, 11]:
                day = 30
            elif month == 2:
                if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
                    day = 29
                else:
                    day = 28
    else:
        day -= 1

    return (day, month, year)

# Example of executing test cases
test_cases = [
    (1, 1, 1900),  # Valid
    (31, 12, 2015),  # Valid
    (15, 1, 2000),  # Invalid
    (0, 1, 1900),   # Invalid
    (29, 2, 2015),  # Invalid
]

for day, month, year in test_cases:
    print(f"Input: ({day}, {month}, {year}) => Output: {previous_date(day, month, year)}")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\CODING\HTML & CSS\Thapa Tutorial\LOGIN> python -u "d:\CODING\HTML & CSS\Thapa Tutorial\LOGIN\se.py"
Input: (1, 1, 1900) => Output: (31, 12, 1899)
Input: (31, 12, 2015) => Output: (30, 12, 2015)
Input: (15, 1, 2000) => Output: (14, 1, 2000)
Input: (0, 1, 1900) => Output: Invalid date
Input: (29, 2, 2015) => Output: Invalid date
PS D:\CODING\HTML & CSS\Thapa Tutorial\LOGIN>
```

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.

Ans: Equivalence Partitioning (EP) Test Cases:

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
EP-01	5	[1, 3, 5, 7]	2	Valid
EP-02	6	[1, 3, 5, 7]	-1	Invalid
EP-03	5	[]	-1	Invalid
EP-04	1	[1, 3, 5, 7]	0	Valid
EP-05	7	[1, 3, 5, 7]	3	Valid

Boundary Value Analysis (BVA) Test Cases:

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
BVA-01	1	[1, 2, 3]	0	Valid
BVA-02	3	[1, 2, 3]	2	Valid
BVA-03	2	[1, 2, 3]	1	Valid
BVA-04	4	[1, 2, 3]	-1	Invalid
BVA-05	5	[]	-1	Invalid

Modified Code:

```
int linearSearch(int v, int a[], int length) {
  if (a == NULL || length == 0) {
    return -1;  // Handle null or empty array
  }
```

```
  int i = 0;
  while (i < length) {
    if (a[i] == v) {
      return i;
    }
    i++;
  }
  return -1;
}
```

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

Ans.: **Equivalence Partitioning (EP) Test Cases:**

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
EP-01	3	[1, 3, 5, 3, 3]	3	Valid
EP-02	1	[1, 2, 3]	1	Valid
EP-03	4	[1, 2, 3]	0	Valid
EP-04	1	[]	0	Valid
EP-05	1	NULL	0	Invalid

Boundary Value Analysis (BVA) Test Cases:

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
BVA-01	1	[1, 2, 3, 1]	2	Valid
BVA-02	4	[1, 2, 3]	0	Valid
BVA-03	1	[1]	1	Valid
BVA-04	2	[1]	0	Valid
BVA-05	2	[2, 2, 2]	3	Valid

Modified Code:

```
int countItem(int v, int a[], int length) {
  if (a == NULL || length == 0) {
    return 0;  // Handle null or empty array
  }

  int count = 0;
  for (int i = 0; i < length; i++) {
    if (a[i] == v) {
```

```
        count++;
      }
    }
    return count;
}
```

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.

Ans.: Equivalence Partitioning (EP) Test Cases:

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
EP-01	5	[1, 3, 5, 7, 9]	2	Valid
EP-02	1	[1, 3, 5, 7, 9]	0	Valid
EP-03	9	[1, 3, 5, 7, 9]	4	Valid
EP-04	6	[1, 3, 5, 7, 9]	-1	Valid
EP-05	5	[]	-1	Invalid
EP-06	3	[5, 3, 1, 7, 9]	-1	Invalid (unsorted)

Boundary Value Analysis (BVA) Test Cases:

Test Case ID	v (Value)	a[] (Array)	Expected Output	Valid/Invalid Input
BVA-01	1	[1, 2, 3]	0	Valid
BVA-02	3	[1, 2, 3]	2	Valid
BVA-03	4	[1, 3, 5]	-1	Valid
BVA-04	1	[1]	0	Valid
BVA-05	2	[1]	-1	Valid

Modified Code:

```
int binarySearch(int v, int a[], int length) {
  if (a == NULL || length == 0) {
    return -1;  // Handle null or empty array
  }

  int lo = 0, hi = length - 1, mid;
  while (lo <= hi) {
```

```
    mid = lo + (hi - lo) / 2;  // To avoid overflow
    if (v == a[mid]) {
      return mid;
    } else if (v < a[mid]) {
      hi = mid - 1;
    } else {
      lo = mid + 1;
    }
  }
  return -1;
}
```

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).

Ans.: Equivalence Partitioning (EP) Test Cases:

Test Case ID	a	b	c	Expected Output	Valid/Invalid Input
EP-01	5	5	5	EQUILATERAL	Valid
EP-02	5	5	3	ISOSCELES	Valid
EP-03	3	4	5	SCALENE	Valid
EP-04	10	3	2	INVALID	Invalid
EP-05	0	4	5	INVALID	Invalid

Boundary Value Analysis (BVA) Test Cases:

Test Case ID	a	b	c	Expected Output	Valid/Invalid Input
BVA-01	10	5	5	INVALID	Invalid
BVA-02	1	1	1	EQUILATERAL	Valid
BVA-03	5	5	9	INVALID	Invalid
BVA-04	3	4	5	SCALENE	Valid
BVA-05	10	5	4	INVALID	Invalid

Modified Code:

```
final int EQUILATERAL = 0;
```

```
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;

int triangle(int a, int b, int c) {
   // Check for non-positive sides
   if (a <= 0 || b <= 0 || c <= 0) {
      return INVALID;  // Sides must be positive
   }

   // Check for triangle inequality
   if (a >= b + c || b >= a + c || c >= a + b) {
      return INVALID;  // Invalid triangle
   }

   // Check for equilateral triangle
   if (a == b && b == c) {
      return EQUILATERAL;
   }

   // Check for isosceles triangle
   if (a == b || a == c || b == c) {
      return ISOSCELES;
   }

   // If none of the above, it must be scalene
   return SCALENE;
}
```

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).

Ans.: Equivalence Partitioning (EP) Test Cases:

Test Case ID	s1	s2	Expected Output	Valid/Invalid Input
EP-01	"pre"	"prefix"	true	Valid
EP-02	"wrong"	"prefix"	false	Valid
EP-03	"longer"	"short"	false	Valid
EP-04	""	"anything"	true	Valid
EP-05	"nonempty"	""	false	Valid

Boundary Value Analysis (BVA) Test Cases:

Test Case ID	s1	s2	Expected Output	Valid/Invalid Input

BVA-01	"same"	"same"	true	Valid
BVA-02	"a"	"apple"	true	Valid
BVA-03	"b"	"apple"	false	Valid
BVA-04	"prefi"	"prefix"	true	Valid
BVA-05	"prefa"	"prefix"	false	Valid

Modified Code:

```
public static boolean prefix(String s1, String s2) {
   // If s1 is longer than s2, it cannot be a prefix
   if (s1.length() > s2.length()) {
      return false;
   }
   // Using the built-in startsWith method for simplicity
   return s2.startsWith(s1);
}
```

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.

Ans.:

1. Equivalence Class for Valid Triangles:
   - Equilateral Triangle: All three sides are equal (A = B = C).
   - Isosceles Triangle: Exactly two sides are equal (A = B ≠ C, A = C ≠ B, B = C ≠ A).
   - Scalene Triangle: All sides are different (A ≠ B, B ≠ C, A ≠ C).
   - Right Triangle: Fulfills the Pythagorean theorem ($A^2 + B^2 = C^2$ or any permutation).

2. Equivalence Class for Invalid Triangles:
   - Non-Triangle: The sides do not satisfy the triangle inequality (A + B ≤ C, A + C ≤ B, B + C ≤ A).
   - Non-Positive Values: One or more sides are less than or equal to zero (A ≤ 0, B ≤ 0, C ≤ 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).

Ans.: We will now identify test cases that correspond to the equivalence classes defined above.

Test Case ID	A	B	C	Expected Output	Equivalence Class
TC-01	5.0	6.0	7.0	Scalene	Valid Scalene Triangle
TC-02	5.0	5.0	8.0	Isosceles	Valid Isosceles Triangle
TC-03	5.0	5.0	5.0	Equilateral	Valid Equilateral Triangle
TC-04	3.0	4.0	5.0	Right-angled	Valid Right-angled Triangle
TC-05	1.0	2.0	3.0	Invalid Triangle	Invalid Triangle (A + B <= C)
TC-06	-1.0	2.0	3.0	Invalid Input	Non-triangle (Non-positive Input)
TC-07	0.0	4.0	5.0	Invalid Input	Non-triangle (Zero Input)

c) For the Boundary Condition A + B > C Case (Scalene Triangle).

Ans.: To verify the boundary condition where A+B>CA + B > CA+B>C for a scalene triangle, test cases should explore values close to this boundary.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
BC-01	1.0	2.0	2.999	Scalene	A + B is slightly greater than C
BC-02	1.0	2.0	3.0	Invalid	A + B is equal to C (invalid triangle)
BC-03	1.0	2.0	3.001	Scalene	A + B is slightly less than C

d) For the Boundary Condition A = C Case (Isosceles Triangle).

Ans.: To verify the boundary condition where two sides are equal, we will test cases where A=CA = CA=C and values around this boundary.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
BC-04	3.0	4.0	3.0	Isosceles	A = C (valid isosceles triangle)
BC-05	3.0	4.0	2.999	Scalene	A is slightly greater than C
BC-06	3.0	4.0	3.001	Scalene	A is slightly less than C

e) For the Boundary Condition A = B = C Case (Equilateral Triangle).

Ans.: For an equilateral triangle, test cases should check when all three sides are equal and small variations around this condition.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
BC-07	3.0	3.0	3.0	Equilateral	A = B = C (valid equilateral triangle)

BC-08	3.0	3.0	2.999	Isosceles	C is slightly less than A and B
BC-09	3.0	3.0	3.001	Isosceles	C is slightly greater than A and B

f) For the Boundary Condition A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 Case (Right-angle Triangle).

Ans.: To verify right-angle triangles, we test for cases where A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 and small deviations around this condition.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
BC-10	3.0	4.0	5.0	Right-angled	A^2 + B^2 = C^2 (valid right triangle)
BC-11	3.0	4.0	5.001	Scalene	A^2 + B^2 < C^2
BC-12	3.0	4.0	4.999	Scalene	A^2 + B^2 > C^2

g) For the Non-triangle Case (A + B <= C), Identify Test Cases to Explore the Boundary.

Ans.: This boundary involves checking when the sum of two sides is less than or equal to the third side.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
BC-13	1.0	2.0	3.0	Invalid	A + B = C (invalid triangle)
BC-14	1.0	2.0	3.001	Invalid	A + B < C (invalid triangle)

h) For Non-positive Input, Identify Test Points.

Ans.: We need to verify that non-positive inputs (including zero) are properly handled.

Test Case ID	A	B	C	Expected Output	Boundary Condition Verified
NP-01	-1.0	4.0	5.0	Invalid	Negative value for A
NP-02	0.0	4.0	5.0	Invalid	Zero value for A
NP-03	3.0	-4.0	5.0	Invalid	Negative value for B
NP-04	3.0	0.0	5.0	Invalid	Zero value for B