Software engineering (IT314)

Lab-8

Name: Gunjan Saroliya

ID: 202201225

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 Partitioning (EP):

- **Valid Classes:**
  - Days: 1 <= day <= 31
  - Months: 1 <= month <= 12
  - Years: 1900 <= year <= 2015
- **Invalid Classes:**
  - Days: day < 1, day > 31
  - Months: month < 1, month > 12
  - Years: year < 1900, year > 2015

Tester Action	Input Data	Expected Outcome
Valid date	day = 5, month = 3, year = 2000	Previous date output
Invalid day (low)	day = 0, month = 5, year = 2010	Error message
Invalid day (high)	day = 32, month = 5, year = 2010	Error message
Invalid month (low)	day = 15, month = 0, year = 2010	Error message
Invalid month (high)	day = 15, month = 13, year = 2010	Error message
Invalid year (low)	day = 15, month = 6, year = 1899	Error message
Invalid year (high)	day = 15, month = 6, year = 2016	Error message

Boundary Value Analysis (BVA):

- **Days:**
  - Lower boundary: day = 1
  - Upper boundary: day = 31
- **Months:**
  - Lower boundary: month = 1
  - Upper boundary: month = 12
- **Years:**
  - Lower boundary: year = 1900

- Upper boundary: year = 2015

Tester Action	**Input Data**	**Expected Outcome**
Lower boundary day	day = 1, month = 6, year = 2000	Previous date output
Upper boundary day	day = 31, month = 5, year = 2000	Previous date output
Lower boundary month	day = 15, month = 1, year = 2000	Previous date output
Upper boundary month	day = 15, month = 12, year = 2000	Previous date output
Lower boundary year	day = 15, month = 6, year = 1900	Previous date output
Upper boundary year	day = 15, month = 6, year = 2015	Previous date output
Invalid boundary day (high)	day = 32, month = 12, year = 2000	Error message
Invalid boundary year (low)	day = 31, month = 12, year = 1899	Error message

Program Execution and Outcome Verification:

```
def is_valid_date(day, month, year):
if not (1900 <= year <= 2015):
```

```
return False, "Invalid year"
if not (1 <= month <= 12):
return False, "Invalid month"
if not (1 <= day <= 31):
return False, "Invalid day"
return True, "Valid date"

def previous_date(day, month, year):
# Implement the logic for calculating the previous date
# This is a placeholder for simplicity
return f"Previous date is {day-1}/{month}/{year}"

# Test cases execution
test_cases = [
(5, 3, 2000),   # Valid
(0, 5, 2010),   # Invalid day (low)
(32, 5, 2010),  # Invalid day (high)
(15, 0, 2010),  # Invalid month (low)
(15, 13, 2010), # Invalid month (high)
(15, 6, 1899),  # Invalid year (low)
(1, 6, 2015),   # Valid boundary test
]

for day, month, year in test_cases:
is_valid, message = is_valid_date(day, month, year)
if is_valid:
```

print(previous_date(day, month, year))

else:

print(message)

Q.2. Programs

P1.

Equivalence Classes:

Test Case #	**Input Array a[]**	**Equivalence Class**	**Expected Outcome**
1	[1, 2, 3, 4, 5]	Class 1	2
2	[1, 2, 3, 4, 5]	Class 2	-1
3	[]	Class 3	-1
4	[-3, -1, 0, 1, 2]	Class 1 (negative)	1
5	[-3, -1, 0, 1, 2]	Class 2 (negative)	-1

Boundary Value Analysis Test Cases:

Test Case #	**Input Array a[]**	**Boundary Condition**	**Expected Outcome**
6	[1, 2, 3, 4, 5]	First element (lower boundary)	0
7	[1, 2, 3, 4, 5]	Last element (upper boundary)	4
8	[1, 2, 3, 4, 5]	Beyond upper boundary	-1
9	[-3, 0, 1, 5]	Lower boundary (negative)	0
10	[-3, 0, 1, 5]	Upper boundary (positive)	3

Program to Test the Input Data:

```
def linear_search(v, a):
    for i in range(len(a)):
        if a[i] == v:
            return i
    return -1

# Test cases
test_cases = [
    ([1, 2, 3, 4, 5], 3),       # Test case 1: Value present (valid)
    ([1, 2, 3, 4, 5], 6),       # Test case 2: Value not present
    ([], 3),                    # Test case 3: Empty array
    ([-3, -1, 0, 1, 2], -1),    # Test case 4: Value present (negative numbers)
    ([-3, -1, 0, 1, 2], -5),    # Test case 5: Value not present (negative numbers)
    ([1, 2, 3, 4, 5], 1),       # Test case 6: First element
    ([1, 2, 3, 4, 5], 5),       # Test case 7: Last element
    ([1, 2, 3, 4, 5], 6),       # Test case 8: Beyond last element
    ([-3, 0, 1, 5], -3),        # Test case 9: First element negative
    ([-3, 0, 1, 5], 5)          # Test case 10: Last element positive
]

for array, value in test_cases:
    result = linear_search(value, array)
    print(f"linear_search({value}, {array}) = {result}")
```

P2.

Equivalence Classes:

Test Case #	Input Array a[]	Equivalence Class	Expected Outcome
1	[1, 2, 3, 4, 5]	Class 1	1
2	[1, 2, 2, 3, 5]	Class 1 (multiple matches)	2
3	[1, 2, 3, 4, 5]	Class 2	0
4	[]	Class 3 (empty array)	0
5	[-3, -1, 0, 1, 2]	Class 1 (negative numbers)	1
6	[-3, -1, 0, 1, 2]	Class 2 (negative numbers)	0

Boundary Value Analysis Test Cases:

Test Case #	Input Array a[]	Boundary Condition	Expected Outcome
7	[1, 2, 3, 4, 5]	First element (lower boundary)	1
8	[1, 2, 3, 4, 5]	Last element (upper boundary)	1
9	[1, 1, 1, 1, 1]	All elements are the same	5
10	[-3, 0, 1, 5]	Positive upper boundary	1
11	[-3, 0, 1, 5]	Negative lower boundary	1
12	[0, 0, 0, 1]	Zero boundary	3

Program to Test the Input Data:

```
def count_item(v, a):
    count = 0
    for i in a:
        if i == v:
            count += 1
    return count
```

```
# Test cases
test_cases = [
    ([1, 2, 3, 4, 5], 3),       # Test case 1: Value appears once
    ([1, 2, 2, 3, 5], 2),       # Test case 2: Value appears twice
    ([1, 2, 3, 4, 5], 6),       # Test case 3: Value not present
    ([], 3),                    # Test case 4: Empty array
    ([-3, -1, 0, 1, 2], -1),    # Test case 5: Value appears once (negative numbers)
    ([-3, -1, 0, 1, 2], -5),    # Test case 6: Value not present (negative numbers)
    ([1, 2, 3, 4, 5], 1),       # Test case 7: First element
    ([1, 2, 3, 4, 5], 5),       # Test case 8: Last element
    ([1, 1, 1, 1, 1], 1),       # Test case 9: All elements the same
    ([-3, 0, 1, 5], 5),         # Test case 10: Positive upper boundary
    ([-3, 0, 1, 5], -3),        # Test case 11: Negative lower boundary
    ([0, 0, 0, 1], 0)           # Test case 12: Zero boundary
]

for array, value in test_cases:
    result = count_item(value, array)
    print(f"count_item({value}, {array}) = {result}")
```

P3.

Equivalence Classes:

Test Case #	Input Array a[]	Equivalence Class	Expected Outcome
1	[1, 2, 3, 4, 5]	Class 1	2

2	[1, 2, 3, 4, 5]	Class 2	-1
3	[]	Class 3	-1
4	[-3, -1, 0, 1, 2]	Class 1 (negative values)	1
5	[-3, -1, 0, 1, 2]	Class 2 (negative values)	-1

Boundary Value Analysis Test Cases:

Test Case #	**Input Array a[]**	**Boundary Condition**	**Expected Outcome**
6	[1, 2, 3, 4, 5]	First element (lower boundary)	0
7	[1, 2, 3, 4, 5]	Last element (upper boundary)	4
8	[1, 2, 3, 4, 5]	Beyond upper boundary	-1
9	[-3, 0, 1, 5]	Lower boundary (negative)	0
10	[-3, 0, 1, 5]	Upper boundary (positive)	3
11	[0]	Single element in the array	0

Program to Test the Input Data:

```
def binary_search(v, a):
    lo, hi = 0, len(a) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if a[mid] == v:
            return mid
        elif a[mid] < v:
            lo = mid + 1
        else:
            hi = mid - 1
```

```
    return -1


# Test cases
test_cases = [
    ([1, 2, 3, 4, 5], 3),       # Test case 1: Value present in the middle
    ([1, 2, 3, 4, 5], 6),       # Test case 2: Value not present
    ([], 3),                    # Test case 3: Empty array
    ([-3, -1, 0, 1, 2], -1),    # Test case 4: Value present (negative)
    ([-3, -1, 0, 1, 2], -5),    # Test case 5: Value not present (negative)
    ([1, 2, 3, 4, 5], 1),       # Test case 6: First element
    ([1, 2, 3, 4, 5], 5),       # Test case 7: Last element
    ([1, 2, 3, 4, 5], 6),       # Test case 8: Beyond upper boundary
    ([-3, 0, 1, 5], -3),        # Test case 9: Lower boundary (negative)
    ([-3, 0, 1, 5], 5),         # Test case 10: Upper boundary (positive)
    ([0], 0)                    # Test case 11: Single element in the array
]


for array, value in test_cases:
    result = binary_search(value, array)
    print(f"binary_search({value}, {array}) = {result}")
```

P4.

Equivalence Classes:

Test Case #	a	b	c	Expected Outcome	Equivalence Class
1	3	3	3	EQUILATERAL	Class 1
2	4	4	5	ISOSCELES	Class 2

3	3	4	5	SCALENE	Class 3
4	3	1	1	INVALID	Class 5
5	0	3	4	INVALID	Class 6 (zero value)
6	-1	3	4	INVALID	Class 6 (negative value)

Boundary Value Analysis Test Cases:

Test Case #	**a**	**b**	**c**	**Expected Outcome**	**Boundary Condition**
7	1	1	1	EQUILATERAL	All sides equal (lower boundary)
8	1	1	2	INVALID	Two sides equal, sum of two equals third
9	2	2	3	ISOSCELES	Two sides equal (just valid)
10	3	4	5	SCALENE	Right-angled triangle
11	1	1	3	INVALID	Triangle inequality boundary
12	-1	-1	-1	INVALID	Negative values

Program to Test the Input Data:

```
# Constants
EQUILATERAL = 0
ISOSCELES = 1
SCALENE = 2
INVALID = 3


def triangle(a, b, c):
    if a <= 0 or b <= 0 or c <= 0:  # Non-positive values
        return INVALID
    if a >= b + c or b >= a + c or c >= a + b:  # Triangle inequality
        return INVALID
```

```
    if a == b == c:  # Equilateral
        return EQUILATERAL
    if a == b or a == c or b == c:  # Isosceles
        return ISOSCELES
    return SCALENE  # Scalene

# Test cases
test_cases = [
    (3, 3, 3),  # Test case 1: Equilateral
    (4, 4, 5),  # Test case 2: Isosceles
    (3, 4, 5),  # Test case 3: Scalene
    (3, 1, 1),  # Test case 4: Invalid (triangle inequality violation)
    (0, 3, 4),  # Test case 5: Invalid (zero value)
    (-1, 3, 4), # Test case 6: Invalid (negative value)
    (1, 1, 1),  # Test case 7: Equilateral (lower boundary)
    (1, 1, 2),  # Test case 8: Invalid (triangle inequality boundary)
    (2, 2, 3),  # Test case 9: Isosceles (just valid)
    (3, 4, 5),  # Test case 10: Scalene (right-angled triangle)
    (1, 1, 3),  # Test case 11: Invalid (triangle inequality)
    (-1, -1, -1) # Test case 12: Invalid (negative values)
]

# Test and print results
for a, b, c in test_cases:
    result = triangle(a, b, c)
    print(f"triangle({a}, {b}, {c}) = {result}")
```

P5.

Equivalence Classes:

Test Case #	s1	s2	Expected Outcome	Equivalence Class
1	"abc"	"abcdef"	TRUE	Class 1 (Exact prefix)
2	"abc"	"abxdef"	FALSE	Class 2 (Non-prefix)
3	"abc"	"ab"	FALSE	Class 3 (s1 longer)
4	""	"abc"	TRUE	Class 4 (Empty string)
5	"abc"	""	FALSE	Class 4 (Empty string)

Boundary Value Analysis Test Cases:

Test Case #	s1	s2	Expected Outcome	Boundary Condition
6	"a"	"ab"	TRUE	Smallest valid prefix
7	"ab"	"a"	FALSE	s1 longer than s2
8	""	""	TRUE	Both empty strings

Program to Test the Input Data:

```
def prefix(s1, s2):
    if len(s1) > len(s2):
        return False
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            return False
    return True


# Test cases
```

```
test_cases = [
    ("abc", "abcdef"),  # Test case 1: Exact prefix
    ("abc", "abxdef"),  # Test case 2: Non-prefix
    ("abc", "ab"),      # Test case 3: s1 longer than s2
    ("", "abc"),        # Test case 4: Empty s1
    ("abc", ""),        # Test case 5: Empty s2
    ("a", "ab"),        # Test case 6: Smallest valid prefix
    ("ab", "a"),        # Test case 7: s1 longer than s2
    ("", "")           # Test case 8: Both empty
]

# Test and print results
for s1, s2 in test_cases:
    result = prefix(s1, s2)
    print(f"prefix('{s1}', '{s2}') = {result}")
```

P6.

a) Identify Equivalence Classes

Equivalence Class	**Description**
Class 1	Equilateral triangle (e.g., A=B=C)
Class 2	Isosceles triangle (e.g., A=B≠C)
Class 3	Scalene triangle (e.g., A≠B≠C)
Class 4	Right-angled triangle (e.g., $A^2 + B^2 = C^2$)
Class 5	Valid triangle (e.g., A+B>C)
Class 6	Invalid triangle (e.g., A+B≤C)
Class 7	Non-positive sides (e.g., A≤0)

b) Test Cases for Equivalence Classes

Test Case #	A	B	C	Expected Outcome	Equivalence Class
1	3	3	3	Equilateral	Class 1 (Equilateral)
2	4	4	5	Isosceles	Class 2 (Isosceles)
3	3	4	5	Scalene	Class 3 (Scalene)
4	3	4	5	Right-angled	Class 4 (Right-angled)
5	3	4	2	Valid triangle	Class 5 (Valid)
6	1	2	3	Invalid triangle	Class 6 (Invalid)
7	-1	2	3	Invalid triangle	Class 7 (Non-positive)
8	0	2	3	Invalid triangle	Class 7 (Non-positive)

c) Test Cases for Boundary Condition: A + B > C (Scalene Triangle)

Test Case #	A	B	C	Expected Outcome	Boundary Condition
9	1.0001	1	1	Scalene	$A+B=C+\epsilon$
10	2	3	5.0001	Scalene	$A+B=C+\epsilon$

d) Test Cases for Boundary Condition: A = C (Isosceles Triangle)

Test Case #	A	B	C	Expected Outcome	Boundary Condition
11	4	5	4	Isosceles	A=C
12	5	5	4.9999	Isosceles	Near isosceles boundary

e) Test Cases for Boundary Condition: A = B = C (Equilateral Triangle)

Test Case #	A	B	C	Expected Outcome	Boundary Condition

13	6	6	6	Equilateral	A=B=C
14	1	1	1.0001	Equilateral	Near equilateral boundary

f) Test Cases for Boundary Condition: $A^2 + B^2 = C^2$ (Right-angled Triangle)

Test Case #	**A**	**B**	**C**	**Expected Outcome**	**Boundary Condition**
15	3	4	5	Right-angled	$A^2 + B^2 = C^2$
16	5	12	13	Right-angled	$A^2 + B^2 = C^2$

g) Test Cases for Non-Triangle Case (Invalid Triangle)

Test Case #	**A**	**B**	**C**	**Expected Outcome**	**Boundary Condition**
17	1	1	2	Invalid triangle	A+B=C (invalid case)
18	2	5	7	Invalid triangle	A+B=C (invalid case)

h) Test Cases for Non-Positive Input

Test Case #	**A**	**B**	**C**	**Expected Outcome**	**Boundary Condition**
19	0	2	3	Invalid triangle	Zero side
20	-1	4	5	Invalid triangle	Negative side

Program to Test the Input Data:

```
def triangle(A, B, C):
    if A <= 0 or B <= 0 or C <= 0:
        return "Invalid triangle"
    if A + B <= C or A + C <= B or B + C <= A:
        return "Invalid triangle"
    if A == B == C:
        return "Equilateral"
```

```
    if A == B or B == C or A == C:
        return "Isosceles"
    if A**2 + B**2 == C**2 or B**2 + C**2 == A**2 or A**2 + C**2 == B**2:
        return "Right-angled"
    return "Scalene"

# Test cases
test_cases = [
    (3.0, 3.0, 3.0),  # Equilateral
    (4.0, 4.0, 5.0),  # Isosceles
    (3.0, 4.0, 5.0),  # Scalene & Right-angled
    (1.0, 2.0, 3.0),  # Invalid triangle
    (-1.0, 2.0, 3.0), # Invalid triangle (negative side)
    (0.0, 2.0, 3.0),  # Invalid triangle (zero side)
]

# Test and print results
for A, B, C in test_cases:
    result = triangle(A, B, C)
    print(f"triangle({A}, {B}, {C}) = {result}")
```