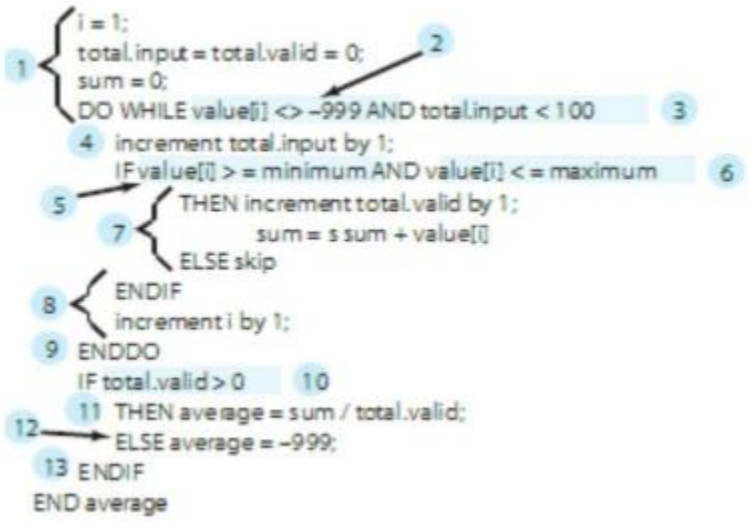
Lab 03: white box Testing

1. Consider the code given below. Calculate cyclomatic complexity and no. of linearly independent paths and Design Testcases for each path.



CYCLOMATIC COMPLEXITY:

V(G) = P + 1, where P is the number of predicate nodes in the flow graph
V(G) = 3 + 1
V(G) = 4

Algorithm:

1. Set i, total.input, total.valid, and sum to 0.
2. Start loop:
3. Check if value[i] is not equal to -999 AND total.input is less than 100.
4. Increment total.input by 1.
5. If value[i] is within the range of minimum and maximum:
6. Increment total.valid by 1.
7. Add value[i] to sum.
8. Increment i by 1.
9. End loop when the condition in step 3 is not met.
10. If total.valid > 0:
11. Calculate average as sum divided by total.valid.
12. Else:
13. Set average to -999.
14. Start loop:
15. If total.valid > 0:
16. Calculate average as sum divided by total.valid.
17. Else:

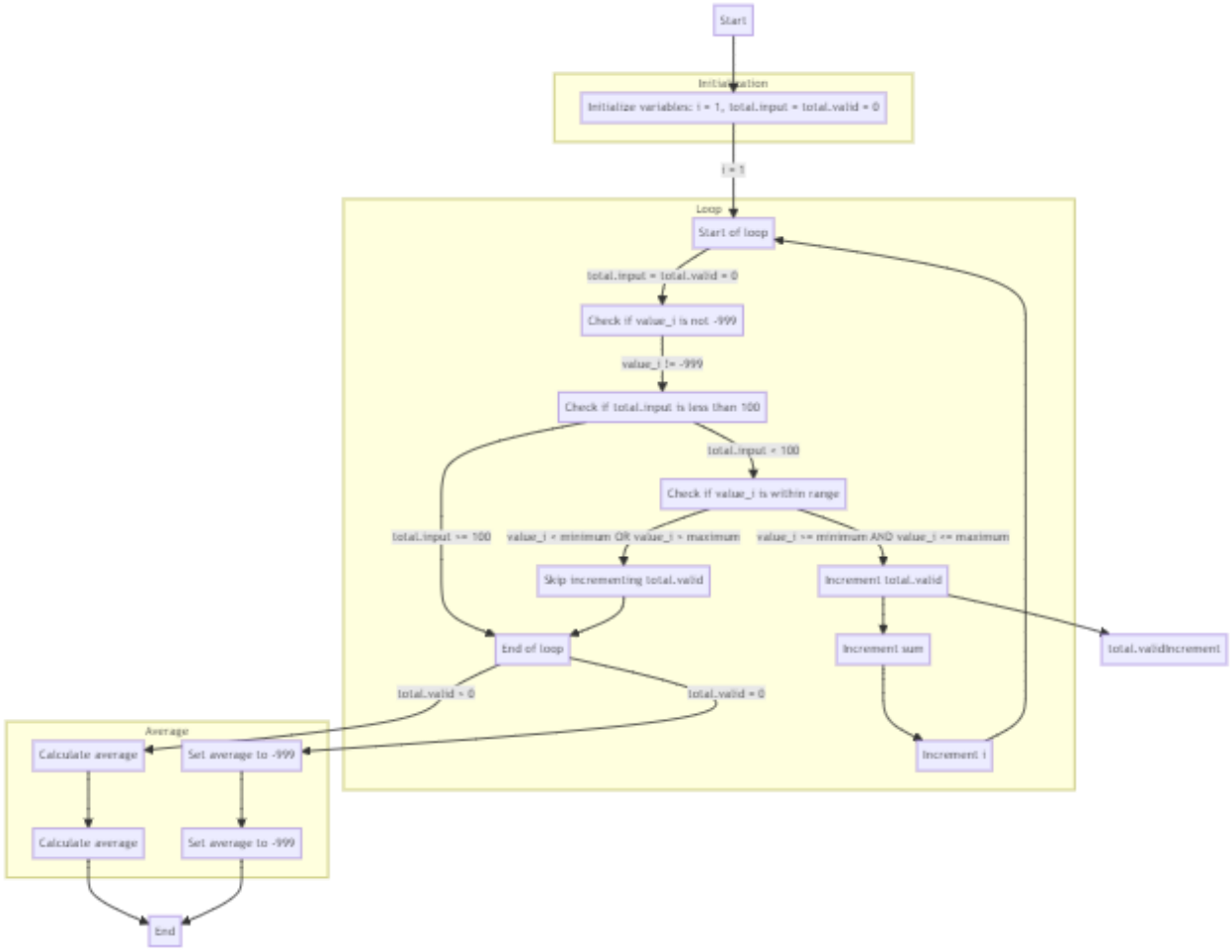
18. Set average to -999.
19. End loop when the condition in step 15 is not met.
20. End algorithm.

Independent Paths:

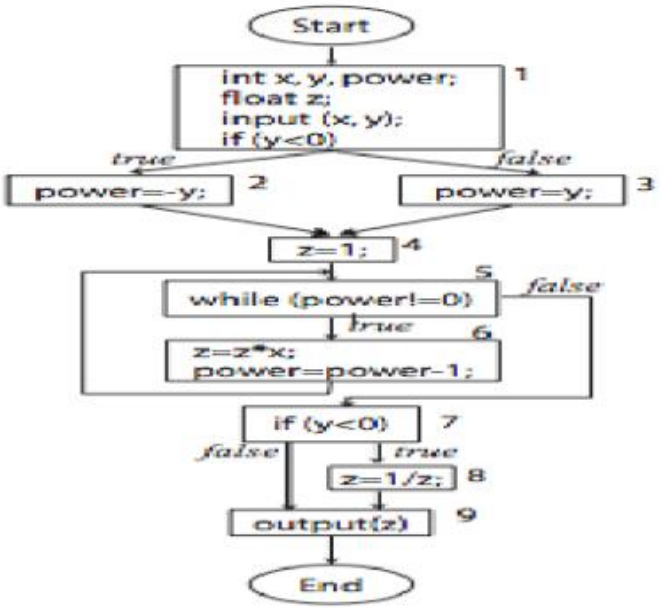
Path 1: Starts at step 2, goes through steps 3, 4, 5, 6, 7, 8, and repeats until step 9. Then, proceeds to step 10, 11, and ends.
Path 2: Starts at step 2, goes through steps 3, 4, 5, 8, 9, 10, 12, and ends.
Path 3: Starts at step 2, goes through steps 3, 4, 5, 6, 8, 9, 10, 13, 14, 15, and ends.
Path 4: Starts at step 2, goes through steps 3, 4, 5, 6, 8, 9, 10, 13, 16, and ends.

S.No.	Test Cases	Test Steps	Test Data	Expected Output	Actual Output	Status
1	Path 1: Valid Input, Loop Executes, Valid Average	Enter values such that loop executes and valid average isfound	Loop Executes	Valid Average	[Average calculated]	Pass
2	Path 1: Invalid Input, Loop Doesn't Execute	Enter values such that loop doesnot execute	Loop Doesn't Execute	-999 (No valid average)	[No valid average]	Pass
3	Path 2: Valid Input, Loop Executes, Valid Average	Enter values such that loop executes and valid average isfound	Loop Executes	Valid Average	[Average calculated]	Pass
4	Path 2: Invalid Input, Loop Doesn't Execute	Enter values such that loop doesnot execute	Loop Doesn't Execute	-999 (No valid average)	[No valid average]	Pass
5	Path 3: Valid Input, Loop Executes, Valid Average	Enter values such that loop executes and valid average isfound	Loop Executes	Valid Average	[Average calculated]	Pass
6	Path 3: Invalid Input, Loop Doesn't Execute	Enter values such that loop doesnot execute	Loop Doesn't Execute	-999 (No valid average)	[No valid average]	Pass
7	Path 4: Valid Input, Loop Executes, Valid Average	Enter values such that loop executes and valid average isfound	Loop Executes	Valid Average	[Average calculated]	Pass

8	Path 4: Invalid Input, Loop Doesn't Execute	Enter values such that loop doesnot execute	Loop Doesn't Execute	-999 (No valid average)	[No valid average]	Pass



2. Consider the flow chart given below and design test cases by using statement coverage method, AND Independent path method also execute test cases using template (provided in lab 2).



1. Start
2. Input x, y
3. Check if y is negative:
4. If y is negative, set power to -y.
5. If y is non-negative, set power to y.
6. Initialize z to 1
7. Loop until power becomes zero:
8. Multiply z by x
9. Decrement power by 1
10. Check if y is negative:
11. If y is negative, divide 1 by z
12. If y is non-negative, output z
13. End program

STATEMENT COVERAGE:

Test Case 1: Positive x (9) and Positive y (2)

This test case covers statements 1, 2, 3 (if condition is false), 5, 6 (if condition is false), and 7-13.
Statement Covered = 12/13 x 100 = 92.3%

Test Case 2: Positive x (9) and Negative y (-2)

This test case covers statements 1, 2, 3 (if condition is true), 4, 6 (if condition is true), and 7-13.
Cumulative Statement Covered = Test 01 + 1/13 x 100 = 100%

INDEPENDENT PATH COVERAGE:

Test Case 1: Positive x (9) and Positive y (2)

Test case 1 covers Path 1, which represents the scenario when the exponent is positive. Statements 1, 2, 3 (if condition is false), 5, 6 (if condition is false), and 7-13.

Test Case 2: Positive x (9) and Negative y (-2)

Test case 2 covers Path 1, which represents the scenario when the exponent is negative. Statements 1, 2, 3 (if condition is true), 4, 6 (if condition is true), and 7-13.

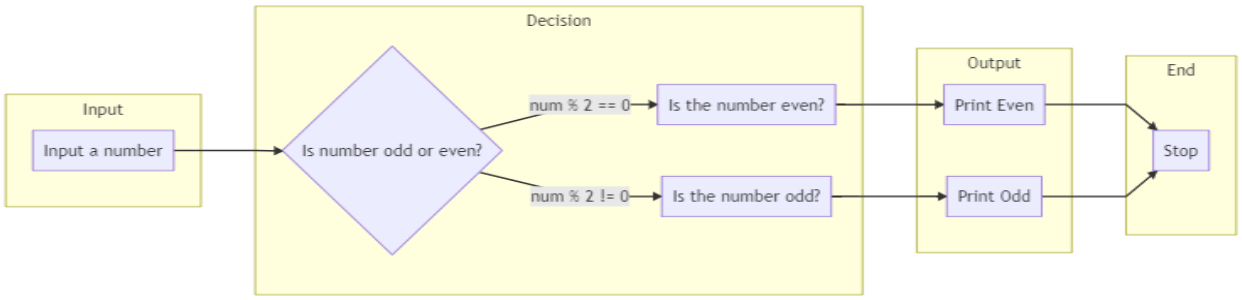
S.No.	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Status
1	Valid Input - PositiveExponent	Enter base (x): Enter exponent (y):	x = 2, y = 3	Result: 8	Result: 8	Pass
2	Valid Input - Negative Exponent	Enter base (x): Enter exponent (y):	x = 2, y = -3	Result: 0.125	Result: 0.125	Pass
3	Valid Input - Zero Exponent	Enter base (x): Enter exponent (y):	x = 2, y = 0	Result: 1	Result: 1	Pass
4	No Input - Press Enter for x	Enter base (x): Enter exponent (y):	Press Enter for x, y = 3	Error: Invalid input forx	Error: Invalid input forx	Fail
5	No Input - Press Enter for y	Enter base (x): Enter exponent (y):	x = 2, Press Enterfor y	Error: Invalid input fory	Error: Invalid input fory	Fail

6	Non-Numeric Input - x	Enter base (x): Enter exponent (y):	x = abc, y = 3	Error: Invalid input forx	Error: Invalid input forx	Fail
7	Non-Numeric Input - y	Enter base (x): Enter exponent (y):	x = 2, y = abc	Error: Invalid input fory	Error: Invalid input fory	Fail
8	Non-Integer Input - y	Enter base (x): Enter exponent (y):	x = 2, y = 3.5	Error: Invalid input fory	Error: Invalid input fory	Fail
9	Large Exponent - Positive	Enter base (x): Enter exponent (y):	x = 2, y = 1000	[Program execution long]	[Program execution long]	Pass
10	Large Exponent - Negative	Enter base (x): Enter exponent (y):	x = 2, y = -1000	[Program execution long]	[Program execution long]	Pass
11	Decimal Base - Positive Exponent	Enter base (x): Enter exponent (y):	x = 2.5, y = 3	Result: 15.625	Result: 15.625	Pass
12	Decimal Base - Negative Exponent	Enter base (x): Enter exponent (y):	x = 2.5, y = -3	Result: 0.064	Result: 0.064	Pass

3. Write a program (Any Language) to find odd or even number. Design and execute test cases using branch coverage method using template /*attach printout here*/

```
def test_odd_even(num):

 if num % 2 == 0:
    return "Even"
 else:
    return "Odd"
```

```
number = int(input("Enter a number to test: "))
result = test_odd_even(number)
print(f"The number {number} is {result}.")
```




```
Enter a number to test: 77
The number 77 is Odd.


...Program finished with exit code 0
Press ENTER to exit console.
```

Branch coverage:

Test case 1:
input number = 9
Path: 1 Input , 2 (num%2==0) Decision , 3 (Print Even) Output , End
Branch coverage = No. of branches covered by test case / total no. of branches x 100
= 50 %

Test case 2:
input number = 9
Path: 1 Input , 2 (num%2!=0) Decision , 3 (Print Odd) Output , End
Branch coverage = No. of branches covered by test case / total no. of branches x 100
= 50 %

S.No.	Test Cases	Test Steps	Test Data	Expected Output	Actual Output	Status
1	Test with even number	1. Enter an even number as input.	Number: 6	Expected: Even	Actual :Even	Pass
2	Test with odd number	1. Enter an odd number as input.	Number: 7	Expected: Odd	Actual: Odd	Pass

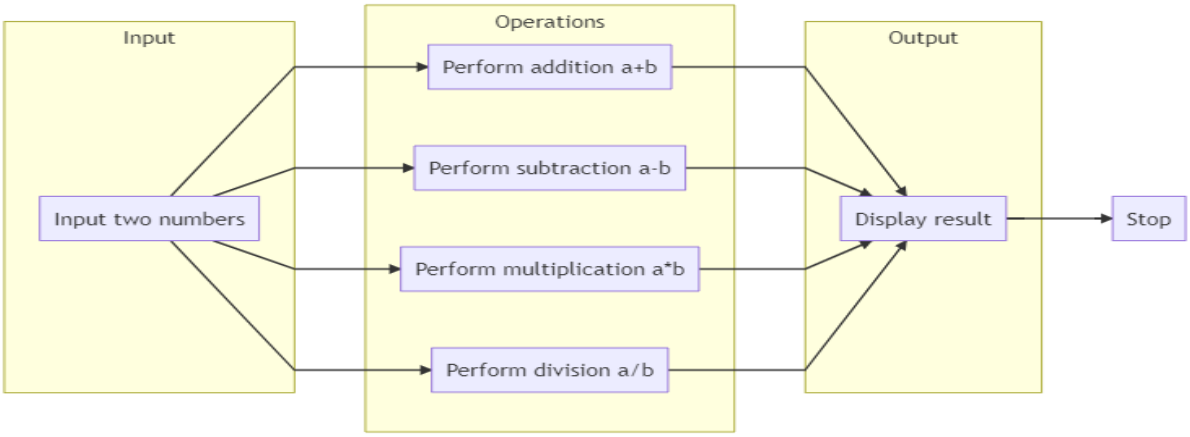
3	Test with zero	1. Enter zero as input.	Number: 0	Expected: Even	Actual :Even	Pass
4	Test with negative even number	1. Enter a negative even number asinput.	Number: -4	Expected: Even	Actual :Even	Pass
5	Test with negative odd number	1. Enter a negative odd number asinput.	Number: -3	Expected: Odd	Actual: Odd	Pass
6	Test with large even number	1. Enter a large even number asinput.	Number: 10000	Expected: Even	Actual :Even	Pass
7	Test with large odd number	1. Enter a large odd number asinput.	Number: 999999	Expected: Odd	Actual: Odd	Pass
8	Test with non-integer input	1. Enter a non-integer (e.g., string,float) as input.	Number: "abc"	Expected: Error	Actual :Error String Given instead of int	Pass
9	Test with floating-point number	1. Enter a floating-point number asinput.	Number: 3.14	Expected: Error	Actual :Error Float Given	Pass
10	Test with very large negativeeven number	1. Enter a very large negative evennumber as input.	Number: -10000000	Expected: Even	Actual :Even	Pass

4. Write a program (Any Language) to perform addition, subtraction, multiplication and division. Observe the output and Test it using any one of white box technique . Attached all printout here.

```
def addition(a, b):

    return a + b
```

```python
def subtraction(a, b):
    return a - b

def multiplication(a, b):
    return a * b

def division(a, b):
    if b == 0:
        return "Error: Division by zero!"
    else:
        return a / b

num1 = int(input("Enter Number One: "))
num2 = int(input("Enter Number Two: "))

print("\n")

print("Addition:", addition(num1, num2))
print("Subtraction:", subtraction(num1, num2))
print("Multiplication:", multiplication(num1, num2))
print("Division:", division(num1, num2))
```

```
Enter Number One: 55
Enter Number Two: 4


Addition: 59
Subtraction: 51
Multiplication: 220
Division: 13.75
```



Project Name:	Basic-Calculator
Test Suite ID:	Calculate_TESEXX(SE-21094)

Test Title:	Test the basic operators in Calculator
Test Priority:	Medium
Module Name:	Do_Calculation
Designed By:	**Muhammad Zubair**
Designed Date:	**3/5/2024**
Executed By:	**Muhammad Zubair**
Executed Date:	**3/5/2024**
Description of Test:	Verify the addition, subtraction, multiplication and division of two numbers(Also test division by zero)
PREREQUISITES:	
Pre-Condition:	Not required
Dependencies:	No dependencies in this test.

- **Test Cases:**

S.No.	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Status
TC-001	Addition with positive numbers	Input two numbers andaddition operator	Num1 = 55, Num2 = 4, Op = +	Answer should be = 59	As expected 59	Pass
TC-002	Subtraction with positive numbers	Input two numbers andsubtraction operator	Num1 = 55, Num2 = 4, Op = -	Answer should be = 51	As expected 51	Pass
TC-003	Multiplication withpositive numbers	Input two numbers andmultiplication operator	Num1 = 5, Num2 = 4, Op = *	Answer should be = 20	As expected 20	Pass
TC-004	Division with positivenumbers	Input two numbers anddivision operator	Num1 = 6, Num2 = 3, Op = /	Answer should be = 2	As expected 2	Pass
TC-005	Addition with negative numbers	Input two numbers andaddition operator	Num1 = -1, Num2 = -1, Op = +	Answer should be = -2	As expected -2	Pass

TC-006	Addition with mixedsigns	Input two numbers andaddition operator	Num1 = -1, Num2 = 11, Op = +	Answer should be = 10	As expected 10	Pass
TC-007	Subtraction with negative numbers	Input two numbers andsubtraction operator	Num1 = -10, Num2 = -3, Op = -	Answer should be = -7	As expected -7	Pass
TC-008	Subtraction withmixed signs	Input two numbers andsubtraction operator	Num1 = 5, Num2 = -9, Op = -	Answer should be = 14	As expected 14	Pass
TC-009	Multiplication with negative numbers	Input two numbers and multiplication operator	Num1 = -5, Num2 = -4, Op = *	Answer should be = 20	As expected 20	Pass
TC-010	Multiplication with mixed signs	Input two numbers and multiplication operator	Num1 = -5, Num2 = 90, Op = *	Answer should be = -450	As expected -450	Pass
TC-011	Division with negative numbers	Input two numbers and division operator	Num1 = -2, Num2 = -2, Op = /	Answer should be = 1	As expected 1	Pass
TC-012	Division with mixed signs	Input two numbers and division operator	Num1 = -88, Num2 = 2, Op = /	Answer should be = -44	As expected -44	Pass
TC-013	Zero Divided by any number	Input zero as num1 and positive integer as num2	Num1 = 0, Num2 = 7, Op = /	Answer should be = 0.0	As expected 0.0	Pass
TC-014	Division by zero	Input positive integer as num1 and zero as num2	Num1 = 5, Num2 = 0, Op = /	Division Error Divided by Zero	Division : Error Divided by Zero	Pass
TC-015	One Invalid input type	Input String as num1 and integer as num2	Num1 = w, Num2 = 9, Op = +	Must terminate the program as soon as string “w” is entered	prints invalid literal for int() with base 10 : 'w'	Pass

TC-016	Both Invalid input type	Input String as num1 and num2	Num1 = "w", Num2 = "x", Op = +	Must terminate the program as soon as w written	prints invalid literal for int() with base 10: 'w'	Pass