Question No: 01

You are testing a photo-enforcement system for traffic control in an intersection. Consider the following scenarios

(Scenario i) A photo will be taken if the light is red (RED) and the front wheels of the car are over the line marking the beginning of the intersection (WHEELS).

a) Which sets of values provides the minimum tests to achieve **100% decision/condition coverage**?

(Scenario ii) A photo should be taken if the signal light is red (RED) or the car is speeding (SPEED) and if the front wheels of the car are over the line marking the beginning of the intersection (WHEELS).

b) Which sets of values provide the minimum tests to achieve 100% **modified condition/decision coverage**?

c) Which sets of values provide the minimum tests to achieve 100% **multiple condition coverage**?

d) Which sets of values provide the minimum tests to achieve 100% **path coverage**?

- Modified condition/decision coverage (MCDC) is a type of white-box testing that ensures that all possible combinations of conditions in a decision are exercised. This is done by creating test cases that test all possible combinations of true and false values for each condition.
- Multiple condition coverage (MCC) is a type of white-box testing that ensures that all possible combinations of conditions in a decision are exercised, regardless of the **order in which the conditions** are evaluated. This is done by creating test cases that test all possible combinations of true and false values for each condition, and **by varying the order in which the conditions are evaluated**.
- Path coverage is a type of white-box testing that ensures that all possible paths through a program are executed. This **is done by creating test cases that exercise all possible paths through the program**.

Ans:

a)

conditions – RED and WHEEL (Condition)

Action – PHOTO TAKEN

	Test case 1	Test case 2	Test case 3	Test case 4
RED	F	T	F	T
WHEEL	F	F	T	T
PHOTO TAKEN	F	F	F	T

Condition should be evaluated to **condition 1 and 4**

b)

(Scenario ii) A **photo should be taken** if the **signal light is red (RED)** or the **car is speeding (SPEED**) and if the front wheels of the car are over the line marking the beginning of the intersection (**WHEELS**).

b) Which sets of values provide the minimum tests to achieve 100% modified condition/decision coverage?

2^n , n is action

2^3 = 8 (2 to power of 3)

Variables – Speed, RED and WHEEL (Condition)

Action – PHOTO TAKEN

	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6	Test case 7	Test case 8
RED	F	F	F	F	T	T	T	T
SPEED	F	F	T	T	F	F	T	T
WHEEL	F	T	F	T	F	T	F	T
PHOTO TAKEN	F	F	F	T	F	T	F	T

Test 2 and 6 is validate RED case

Test 2 and 4 is validate SPEED case

Test 4 and 7 is validate WHEEL case

1,2,4 & 6 minimum test is required

Note: number of test case is n+1, where n is number of condition

c)

	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6	Test case 7	Test case 8
RED	F	F	F	F	T	T	T	T
SPEED	F	F	T	T	F	F	T	T
WHEEL	F	T	F	T	F	T	F	T
PHOTO TAKEN	F	F	F	T	F	T	F	T

d) Which sets of values provide the minimum tests to achieve 100% path coverage?

Path coverage – 1, 2, 3 and 8

	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6	Test case 7	Test case 8
RED	F	F	F	F	T	T	T	T
SPEED	F	F	T	T	F	F	T	T
WHEEL	F	T	F	T	F	T	F	T
PHOTO TAKEN	F	F	F	T	F	T	F	T

Question No. 02

An IOT enabled bulb changes states between red, green and blue on receiving the following input signals via Wi-Fi

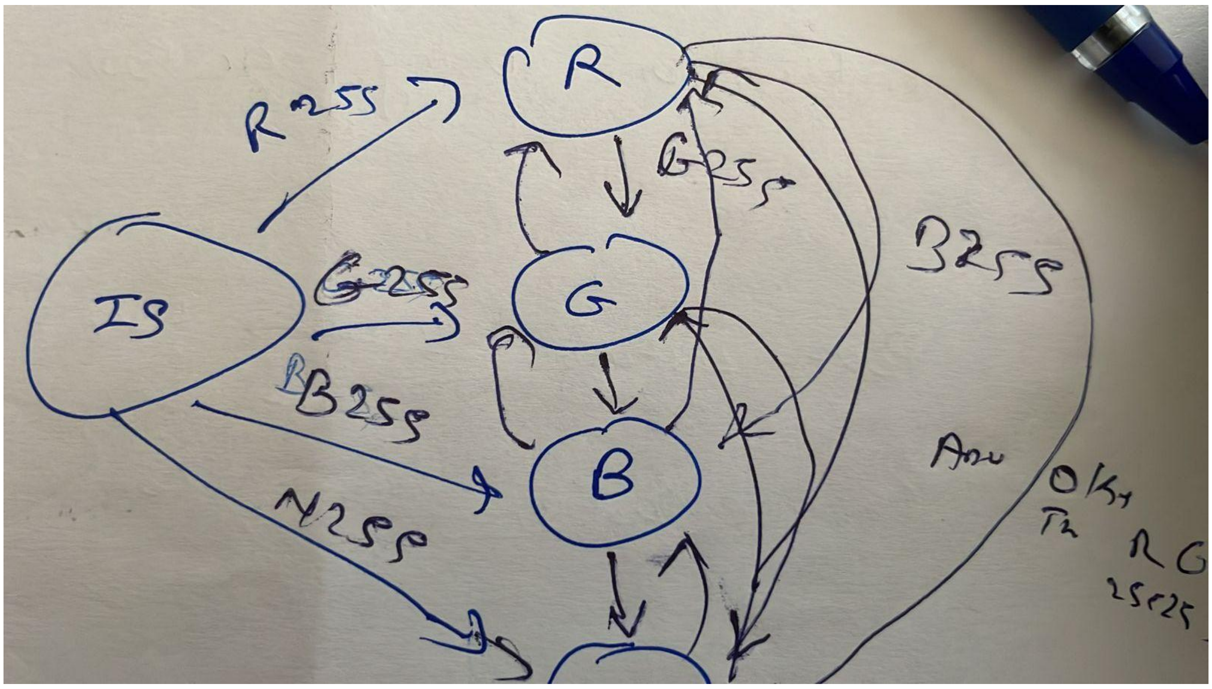
Enable = R255 ------ Red state

Enable = G255 ------ Green state

Enable = B255 ------ Blue state

i) In case of other input being sent at the Enable - the bulb takes a 'Dim state'.

a) Draw the state transition diagram for the bulb.



b) Write test cases to test all the states.

(ii) The bulb needs to interact with a Wi-Fi enabled device in order to receive the 'enable' input. Write system-level test cases to test this system.

1. Test Case: Enable = R255 (Red state)
   - Input: Enable = R255
   - Expected Output: The bulb should change its state to red.
2. Test Case: Enable = G255 (Green state)
   - Input: Enable = G255
   - Expected Output: The bulb should change its state to green.
3. Test Case: Enable = B255 (Blue state)
   - Input: Enable = B255
   - Expected Output: The bulb should change its state to blue.

4. Test Case: Enable = Y255 (Dim state)
    - Input: Enable = Y255 (input other than R, G, or B)
    - Expected Output: The bulb should enter a dim state.
5. Test Case: Enable = "" (Dim state)
    - Input: Enable = "" (empty input)
    - Expected Output: The bulb should enter a dim state.

Question No: 03

Consider the following algorithm that calculates the sum of all even natural numbers between 1 to N. Answer the following questions:

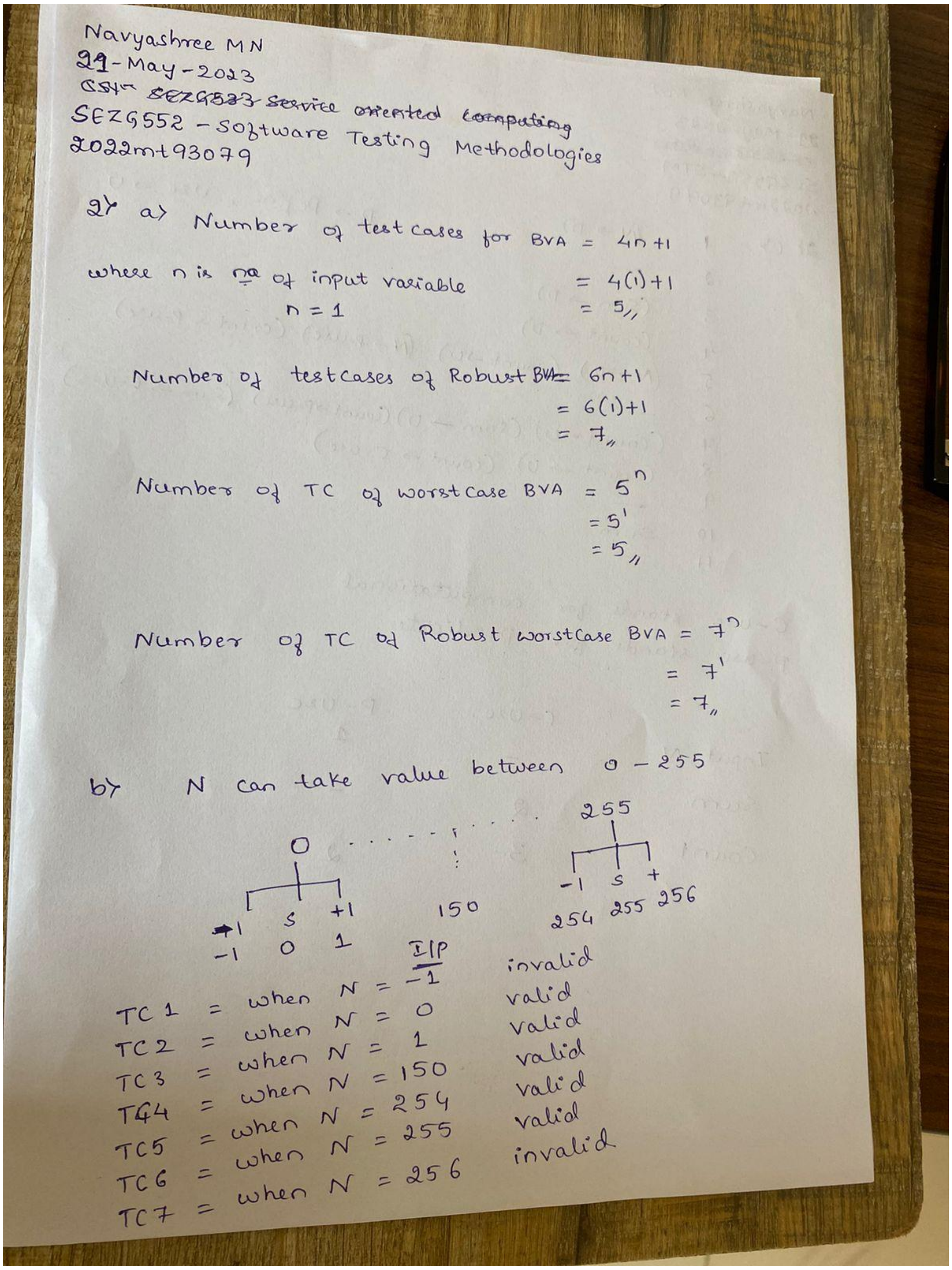
```
int evenNaturalSum(int N)


{
 int sum = 0;


int count = 1;


while(count<=N)
{


if (count%2==0) {sum = sum + count;}


count++;
}


return sum;
}
```

a) Assume that the range of signed int is 2^8=256. Then N can take values only in range 0 to 255 [including end points]. Calculate the number of test cases - Normal BVA, Robust BVA, Worst case BVA and Robust worst case BVA.

b) Write all the test cases for Robust BVA for the above algorithm.

https://www.geeksforgeeks.org/boundary-value-test-cases-robust-cases-and-worst-case-test-cases/

```
min value : 100

close to min : 101

nominal : 300

close to max : 499

max : 500

lesser than min value : 99

larger than max value : 501
```



Navyashree MN
29-May-2023
~~CS4~~ ~~SEZG533-Service oriented computing~~
SEZG552 - Software Testing Methodologies
2022mt93079

2) a) Number of test cases for BVA $= 4n+1$

where n is no of input variable $= 4(1)+1$
$n = 1$ $= 5$

Number of testcases of Robust ~~BVA~~ $= 6n+1$
$= 6(1)+1$
$= 7$

Number of TC of worst case BVA $= 5^n$
$= 5^1$
$= 5$

Number of TC of Robust worstCase BVA $= 7^n$
$= 7^1$
$= 7$

b) N can take value between 0 - 255

0 . . . . . 150 255

-1 S +1   -1 S +
-1 0 1   254 255 256

		I/P	
TC 1	= when	N = -1	invalid
TC 2	= when	N = 0	valid
TC 3	= when	N = 1	valid
TC 4	= when	N = 150	valid
TC5	= when	N = 254	valid
TC 6	= when	N = 255	valid
TC 7	= when	N = 256	invalid

c) Identify all DU pairs for the variables sum, count and N. For all the use nodes, specify whether it is a C-Use or P-Use and why?

In the given algorithm, the variables sum, count, and N are used in different contexts. Let's identify all the Def-Use (DU) pairs for these variables and specify whether each use node is a C-Use (Computation) or P-Use (Predicate) and why.

DU pairs for the variables:

1. Variable: sum
    - Def: sum = sum + count;
    - Use: sum (in the return statement)
2. DU Pair:
    - (Def) sum = sum + count; --> (Use) sum (return statement)

Use Node Type: C-Use (Computation)

3. Explanation: The variable "sum" is used to store the result of the computation (sum = sum + count) and is later used in the return statement. Hence, it is a C-Use.
4. Variable: count
    - Def: count = 1;
    - Use: count <= N, count++
5. DU Pairs:
    - (Def) count = 1; --> (Use) count <= N
    - (Def) count = 1; --> (Use) count++

Use Node Types: P-Use (Predicate), C-Use (Computation)

6. Explanation:
    - The variable "count" is used as a predicate in the condition count <= N, which determines the loop continuation. Hence, it is a P-Use.
    - The variable "count" is also used in the computation count++. It is incremented during each iteration of the loop. Hence, it is a C-Use.
7. Variable: N
    - Def: N (input parameter)
    - Use: count <= N
8. DU Pair:
    - (Def) N (input parameter) --> (Use) count <= N

Use Node Type: P-Use (Predicate)

9. Explanation: The variable "N" is used as a predicate in the condition count <= N, which determines the loop continuation. Hence, it is a P-Use.

To summarize:

Variable: sum

- DU Pair: (Def) sum = sum + count; --> (Use) sum (return statement)
- Use Node Type: C-Use (Computation)

Variable: count

DU Pairs: (Def) count = 1; --> (Use) count <= N

- (Def) count = 1; --> (Use) count++
- Use Node Types: P-Use (Predicate), C-Use (Computation)

Variable: N

- DU Pair: (Def) N (input parameter) --> (Use) count <= N
- Use Node Type: P-Use (Predicate)

Note: In DU analysis, C-Use refers to the use of a variable in a computation (such as arithmetic operations), while P-Use refers to the use of a variable in a predicate (such as conditions or loop controls).

Question 4

Select a structure-based technique according to a given project situation and justify your answer.

You are the Senior Test Analyst working on the testing of Air traffic control software that helps to prevent collisions, organize and expedite the flow of air traffic, and provide information and other support for pilots. A failure analysis has shown that if the software system fails then it may risk the life of thousands of travelers. The government has requested that the level of testing for this software exceeds that normally required by the relevant regulatory standards.

Which is the level of test coverage you would expect to be achieved in of the control software? Also, justify your answer.

A. Multiple Condition coverage

B. Branch coverage + Modified Condition/Decision coverage

C. Branch coverage + Statement coverage

D. Modified Condition/Decision coverage

Question 5

Class Integer Stack provides implementation of Stack data structure, with the implementation of two methods: push() and pop(). Moreover, classes OddIntegerStack and EvenIntegerStack extend the class IntegerStack and overrides the implementation of the push() method. Class Odd Integer Stack is shown below:

```
public class OddIntegerStack extends IntegerStack{

public void push (int element)
{
if (element % 2 == 0)
{
System.out.println("Cannot push even element onto an odd stack");

return;
}
If  (top != maxSize-1){

arrayStack [++top] =element;
}
else{ System.out.println("Overflow error");}
}
}
```

a) Write test cases to test the push()method of the Odd IntegerStack class. Assume maxSize =5

b) Write test cases to test the pop()method of the OddIntegerStack class. Assume maxSize = 5.

c) An object reference of class IntegerStack can be used to store an object of class

Even Integer Stack or OddIntegerStack. Write test cases to test this polymorphic behaviour.

IntegerStack a = new OddIntegerStack();

IntegerStack a = new EvenIntegerStack();

List<IntegerStack> a = new ArrayList<OddIntegerStack>();

List<IntegerStack> a = new ArrayList<EvenIntegerStack>();

d) Discuss, in brief, how you would carry out integration testing of such a system.

To carry out integration testing of the system consisting of the `IntegerStack`, `OddIntegerStack`, and `EvenIntegerStack` classes, you can follow these steps:

1. Identify the integration points: Determine the points where the classes interact with each other. In this case, the integration points are the inheritance relationship between `IntegerStack` and its subclasses (`OddIntegerStack` and `EvenIntegerStack`).
2. Define integration test scenarios: Based on the identified integration points, define test scenarios that cover the interactions between the classes. Some scenarios may include:
   - Testing the inheritance relationship: Verify that the subclasses inherit the methods and attributes correctly from the `IntegerStack` class.
   - Testing method overriding: Ensure that the overridden `push()` method in `OddIntegerStack` and `EvenIntegerStack` behaves as expected and does not interfere with the functionality of the base class's `push()` method.
   - Testing polymorphic behavior: Verify that an `IntegerStack` reference can store objects of both `OddIntegerStack` and `EvenIntegerStack`, and the appropriate methods are called based on the actual object type.
3. Create integration test cases: Based on the defined scenarios, design and implement specific integration test cases. For example:
   - Test that calling the `push()` method on an `OddIntegerStack` object with an odd element correctly adds the element to the stack.

    - Test that calling the `push()` method on an `OddIntegerStack` object with an even element prints the error message and does not modify the stack.
    - Test that calling the `pop()` method on an `IntegerStack` reference assigned to an `EvenIntegerStack` object correctly removes and returns the top element from the stack.
4. Execute integration tests: Run the integration test cases and observe the behavior of the system. Ensure that the interactions between the classes function as expected and that the integration points are properly integrated.
5. Analyze results: Verify whether the integration tests pass or fail. If any issues or failures are encountered, investigate and debug the code to address the problems. Repeat the integration testing process as necessary until all integration issues are resolved.

Integration testing helps ensure that the interactions and collaborations between the classes work correctly, providing a comprehensive evaluation of the system's behavior as a whole. By focusing on the integration points and verifying the correct functioning of inherited methods, overridden methods, and polymorphic behavior, you can gain confidence in the overall functionality and coherence of the system.