..

1. **Why is it important to test boundary values while testing a function?**
   It is conducted **to ensure that the requirements are properly satisfied by the application**. Functional testing verifies that each function of the software application works in conformance with the requirement and specification.

2. **If branch coverage has been achieved on a unit under test, then which coverage is implicitly implied?**
   If branch coverage has been achieved on a unit under test, then it implies that statement coverage has also been achieved.

3. **What do you understand by clean room testing?**
   Cleanroom Testing was pioneered by IBM. This kind of testing depends heavily on walkthroughs, inspection, and formal verification. The programmers don't seem to be allowed to check any of their code by corporal punishment the code apart from doing a little syntax testing employing a compiler. The computer code development philosophy relies on avoiding computer code defects by employing a rigorous examination method. the target of this computer code is that the zero-defect computer code.

   The name 'CLEAN ROOM' was derived from the analogy with semiconductor fabrication units. In these units (clean rooms), defects area unit avoided by producing within the ultra-clean atmosphere. during this reasonable development, inspections to ascertain the consistency of the parts with their specifications has replaced unit-testing.

4. **Give two important types of errors that are checked during code walkthrough.**
   The main objectives of code Walkthrough are to discover the **algorithmic and logical errors** in the code.

5. **For a program containing four binary branches, how many test cases are necessary for path coverage? Justify your answer.**

   To achieve path coverage for a program containing four binary branches, we need to ensure that all possible combinations of branch outcomes have been executed at least once. For a binary branch, there are two possible outcomes: True and False. Therefore, for four binary branches, there are 2^4 = 16 possible combinations of branch outcomes. Each of these combinations represents a unique path through the program. To achieve path coverage, we need to execute all 16 possible combinations of branch outcomes at least once.

6. **In a state table, what do the rows and columns represent?**
   Each row of the table corresponds to a state. Each column corresponds to **an input condition**.

Combination	Description	Anomaly possibilities
dd	Defined the data objects twice	Harmless but suspicious
dk	Defined the data object but killed it without using it.	Bad Programming Practice
du	Defined the data object and using it	NOT an Anomaly
kd	Killed the Data Object and redefined	NOT an Anomaly
kk	Killed the Data Object and killed it again	Bad Programming Practice
ku	Killed the Data Object and then used	Defect
ud	Used the Data Object and redefined	NOT an Anomaly
uk	Used the Data Object and Killed	NOT an Anomaly
uu	Used the Data Object and used it again	NOT an Anomaly

7. **What do you mean by “kk” anomaly?**
8. **What do you mean by “dd” anomaly?**
   Kill-kill - Harmless bug, but not allowed.
   Define-define Redefining a variable without using it . Harmless bug, but not allowed.

9. **What is its effect in data flow testing?**
   Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program.

Reference or define anomalies in the flow of the data are detected at the time of associations between values and variables. These anomalies are:

a. A variable is defined but not used or referenced,
b. A variable is used but never defined,
c. A variable is defined twice before it is used

10. Give two examples of different types of errors that integration testing target to detect.

Integration testing targets to detect different types of errors that can arise due to the interactions between different components or modules of a system. Here are two examples of different types of errors that integration testing can help detect:

1. **Interface errors:** When different components of a system interact with each other, they use interfaces or protocols to communicate and exchange data. Interface errors can occur when there are mismatches or inconsistencies in these interfaces. For example, a change in one component may cause the interface or protocol to change, which can result in compatibility issues with other components that rely on the original interface. Integration testing can help detect such interface errors and ensure that all components can communicate and exchange data effectively.
2. **Performance errors:** When different components of a system interact with each other, they can impact the overall performance of the system. For example, a slow or inefficient component can cause delays or bottlenecks in other components that rely on it. Integration testing can help detect such performance errors and ensure that the overall system performance meets the expected standards. Integration testing can involve testing the system under realistic workloads to identify performance issues and ensure that the system can handle the expected load and concurrency.

11. What do you understand by compatibility testing?

Compatibility testing is a type of software testing that aims to ensure that a software application or system is compatible with different hardware, operating systems, browsers, databases, and other software products that it may interact with or run on. The purpose of compatibility testing is to verify that the software works correctly and as expected in various configurations and environments, and to identify and address any issues that may arise.

12. Briefly explain with one example? What do you mean by non-repudiation?

Non-repudiation is a security concept that refers to the ability to prevent an individual or entity from denying that they have performed a particular action or transaction. In other words, non-repudiation ensures that an action or transaction can be traced back to its originator and that the originator cannot deny having performed it.

A common example of non-repudiation is in electronic transactions, such as online banking or e-commerce. In such transactions, non-repudiation can be achieved through the use of digital signatures or certificates that provide proof of the originator's identity and ensure that the transaction cannot be repudiated later. For example, when a user makes an online purchase using a digital certificate, the certificate can be used to verify the user's identity and confirm that the transaction was authorized by them. This provides strong evidence that the user cannot later deny having made the purchase.

13. **What do you understand by dynamic slice?**
    Static Slice: Statements that may affect value of a variable at a program point for all possible executions.
    Dynamic Slice: Statements that actually affect value of a variable at a program point for that particular execution.

14. Give a positive and a negative test scenario for the “Mobile Number” field of Student Registration Page.

15. A software was tested using the error seeding strategy in which 20 errors were seeded in the code. When the code was tested using the complete test suite, 16 of the seeded errors were detected. The same test suite also detected 200 non-seeded errors. What is the estimated number of latent errors in the software after this testing?

16. Which testing techniques take into account the possible combinations of input conditions, while generating test cases?

17. What is a stub? How is it associated with driver?

18. **What is the difference between verification and validation of software?**
    Verification is a process of determining if the software is designed and developed as per the specified requirements.
    Validation is the process of checking if the software (end product) has met the client's true needs and expectations.

19. **What do you mean by equivalent mutants? Give an example of equivalent mutants. Give an example where it is not possible to determine the state of data variable by just static analysis of the code.**

In the context of software testing and debugging, equivalent mutants are mutations or variations of the original code that result in the same behavior or output as the original code. In other words, if a test case passes on the original code, it should also pass on its equivalent mutants, and if a test case fails on the original code, it should also fail on its equivalent mutants.

An example of an equivalent mutant could be a change in the order of operands in a mathematical expression. For example, if the original code contains the expression "x + y", an equivalent mutant could be "y + x". Both expressions will result in the same output, so if a test case passes on the original code, it should also pass on the mutant, and vice versa.

20. **Suppose a program contains 4 decision inputs, if there are 3 choices at each decision point, how many test cases are necessary for branch testing?**
If a program contains 4 decision inputs and each decision point has 3 choices, we can calculate the number of test cases necessary for branch testing as follows:

For each decision point, there are 3 possible outcomes. Therefore, for 4 decision inputs with 3 choices at each decision point, there are a total of 3^4 = 81 possible combinations of decision outcomes.

Design a black-box test suite for a function named find-rectangle-intersection that accepts four floating point numbers representing two pairs of co-ordinates (x1, y1), (x2, y2), (x3, y3) , (x4, y4). The first two points (x1, y1) and (x2, y2) represent the lower left and upper right points of the first rectangle. The second two points (x3, y3) and (x4, y4) represent the lower left and upper right points of the second rectangle. It is assumed that the length and width of the rectangle are parallel to either the x-axis or y-axis. The program computes the points of intersection of the two rectangles and prints their points of intersection.

To design a black-box test suite for the find-rectangle-intersection function, we need to consider different input combinations that would test the various aspects of the function's functionality, such as the expected output, the accuracy of the floating-point calculations, and the handling of edge cases. Here is a set of test cases that we can use to test the find-rectangle-intersection function:

1. Two non-overlapping rectangles: In this test case, we can create two rectangles that do not overlap. We can use (0,0), (2,2) for the first rectangle and (3,3), (5,5) for the second rectangle. The expected output should be an empty list since there is no intersection between the rectangles.
2. Two rectangles that partially overlap: In this test case, we can create two rectangles that partially overlap. We can use (0,0), (3,3) for the first rectangle

and (2,2), (5,5) for the second rectangle. The expected output should be [(2,2), (3,3)] since the two rectangles overlap at the point (2,2) and (3,3).

3. Two rectangles that fully overlap: In this test case, we can create two rectangles that fully overlap. We can use (0,0), (4,4) for both rectangles. The expected output should be [(0,0), (4,4)] since the two rectangles are identical.
4. Two rectangles that share an edge: In this test case, we can create two rectangles that share an edge. We can use (0,0), (3,3) for the first rectangle and (3,0), (5,2) for the second rectangle. The expected output should be [(3,0), (3,2)] since the two rectangles overlap at the edge (3,0) to (3,2).
5. Two rectangles with one rectangle inside the other: In this test case, we can create two rectangles with one rectangle inside the other. We can use (0,0), (5,5) for the first rectangle and (1,1), (4,4) for the second rectangle. The expected output should be [(1,1), (4,4)] since the second rectangle is fully contained inside the first rectangle.
6. Two rectangles with one rectangle completely outside the other: In this test case, we can create two rectangles with one rectangle completely outside the other. We can use (0,0), (2,2) for the first rectangle and (5,5), (6,6) for the second rectangle. The expected output should be an empty list since there is no intersection between the rectangles.
7. Two rectangles with vertical edges: In this test case, we can create two rectangles with vertical edges. We can use (0,0), (1,5) for the first rectangle and (2,2), (3,7) for the second rectangle. The expected output should be an empty list since the rectangles do not overlap horizontally.
8. Two rectangles with horizontal edges: In this test case, we can create two rectangles with horizontal edges. We can use (0,0), (5,1) for the first rectangle and (2,2), (7,3) for the second rectangle. The expected output should be an empty list since the rectangles do not overlap vertically.

Design the black-box test suite for a program that accepts two strings and checks if the first string is a substring of the second string and displays the number of times the first string occurs in the second string. Assume that each of the two strings has size less than twenty characters.

Test case 1: Valid inputs

- Input: String 1 = "car", String 2 = "racecarcar"
- Expected Output: Number of occurrences of string 1 in string 2 = 2

Test case 2: Invalid input - string 1 is longer than string 2

- Input: String 1 = "elephant", String 2 = "ant"
- Expected Output: Error message indicating that string 1 cannot be longer than string 2

Test case 3: Invalid input - empty string 1

- Input: String 1 = "", String 2 = "hello"
- Expected Output: Error message indicating that string 1 cannot be empty

Test case 4: Invalid input - empty string 2

- Input: String 1 = "dog", String 2 = ""
- Expected Output: Error message indicating that string 2 cannot be empty

Test case 5: Valid inputs - string 1 and string 2 are identical

- Input: String 1 = "hello", String 2 = "hello"
- Expected Output: Number of occurrences of string 1 in string 2 = 1

Test case 6: Valid inputs - string 1 is not a substring of string 2

- Input: String 1 = "cat", String 2 = "dog"
- Expected Output: Number of occurrences of string 1 in string 2 = 0

Test case 7: Invalid input - string 1 and string 2 are NULL

- Input: String 1 = NULL, String 2 = NULL
- Expected Output: Error message indicating that both strings cannot be NULL

Test case 8: Invalid input - special characters in string 1

- Input: String 1 = "h$@lo", String 2 = "hello"
- Expected Output: Error message indicating that string 1 cannot have special characters

Test case 9: Invalid input - special characters in string 2

- Input: String 1 = "hello", String 2 = "h$@lo"
- Expected Output: Error message indicating that string 2 cannot have special characters

Test case 10: Valid inputs - string 1 is a single character and occurs multiple times in string 2

- Input: String 1 = "a", String 2 = "banana"
- Expected Output: Number of occurrences of string 1 in string 2 = 3