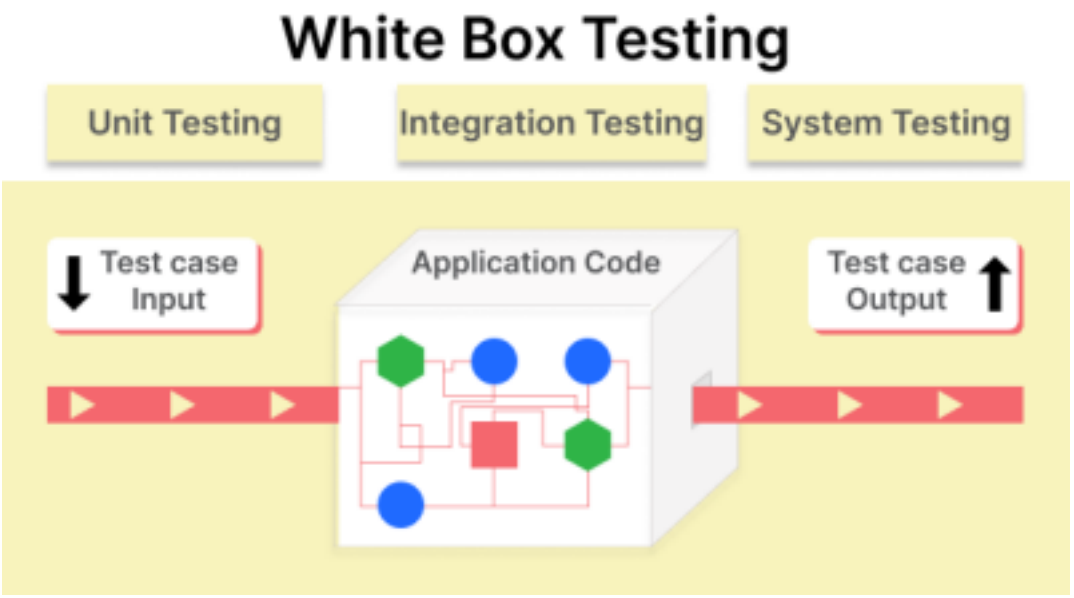
# *WhiteBox Testing*

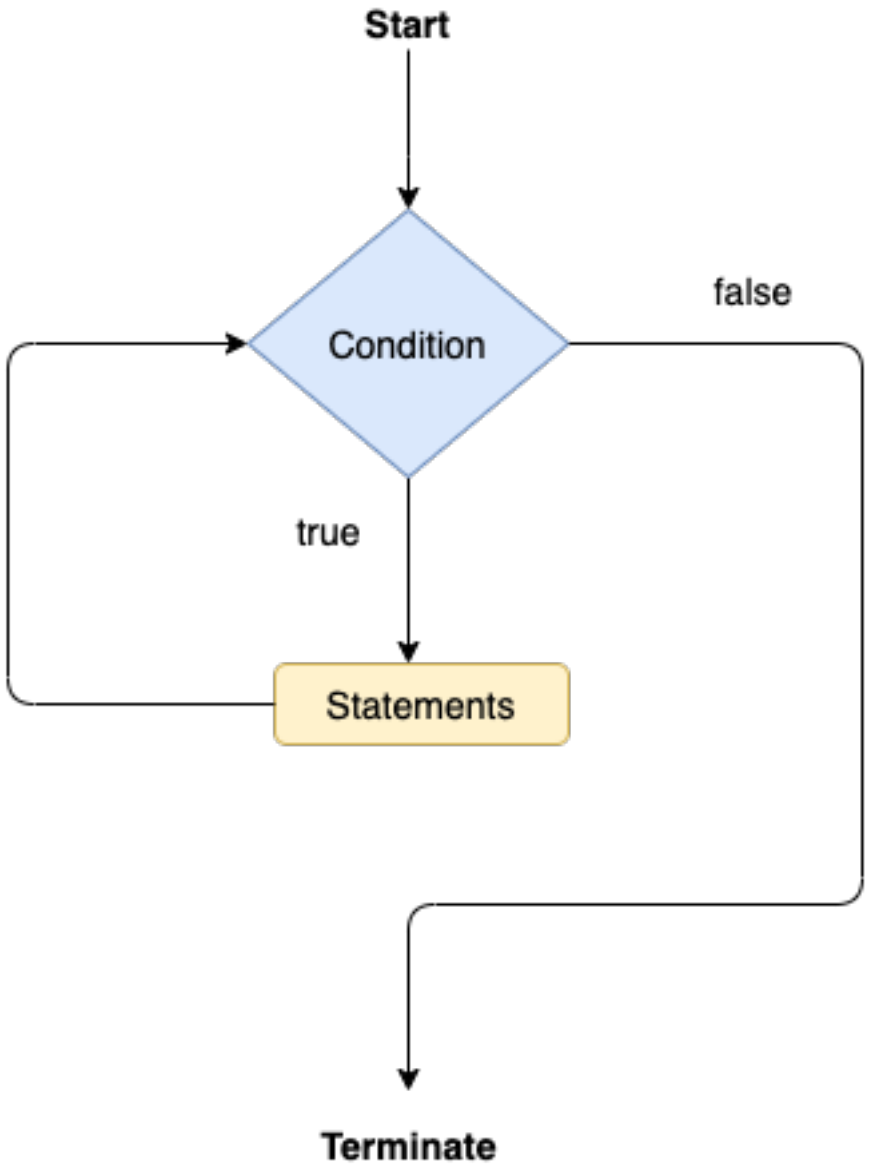
**White box testing**, also known as clear box testing, glass box testing, transparent box testing, or structural testing, is a software testing method that examines the internal structure, design, and coding of an application to verify input-output flow and improve design, usability, and security. It is one of two parts of the box testing approach to software testing, the other being **black box testing.**

White box testing involves testing the internal logic and execution paths of a subroutine by examining the source code.

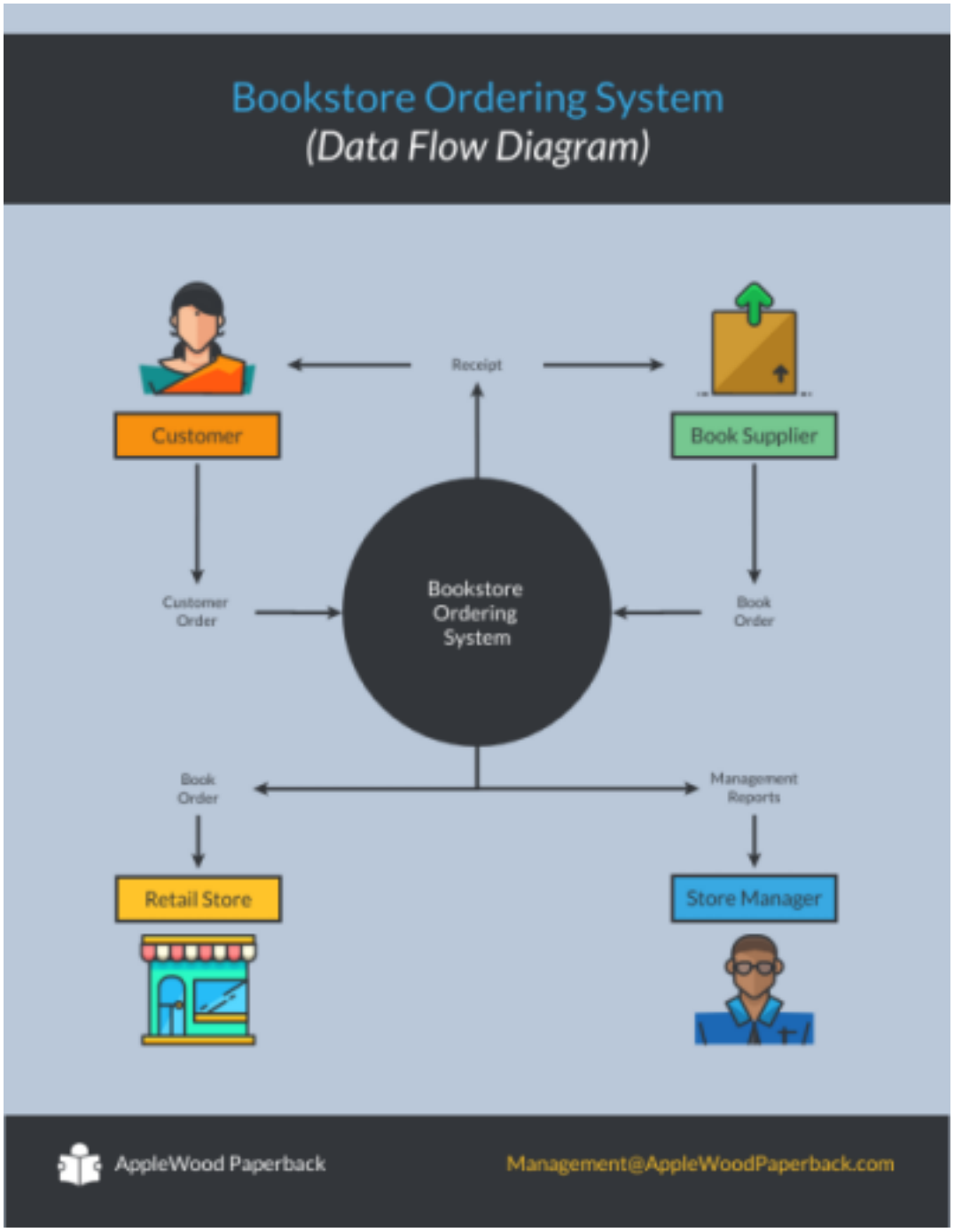



In **white box testing**, testers have access to the source code of the application and use this knowledge to design test cases that can **verify the correctness of the software at the code level.** White box testing is often used to test the following aspects of a software application:

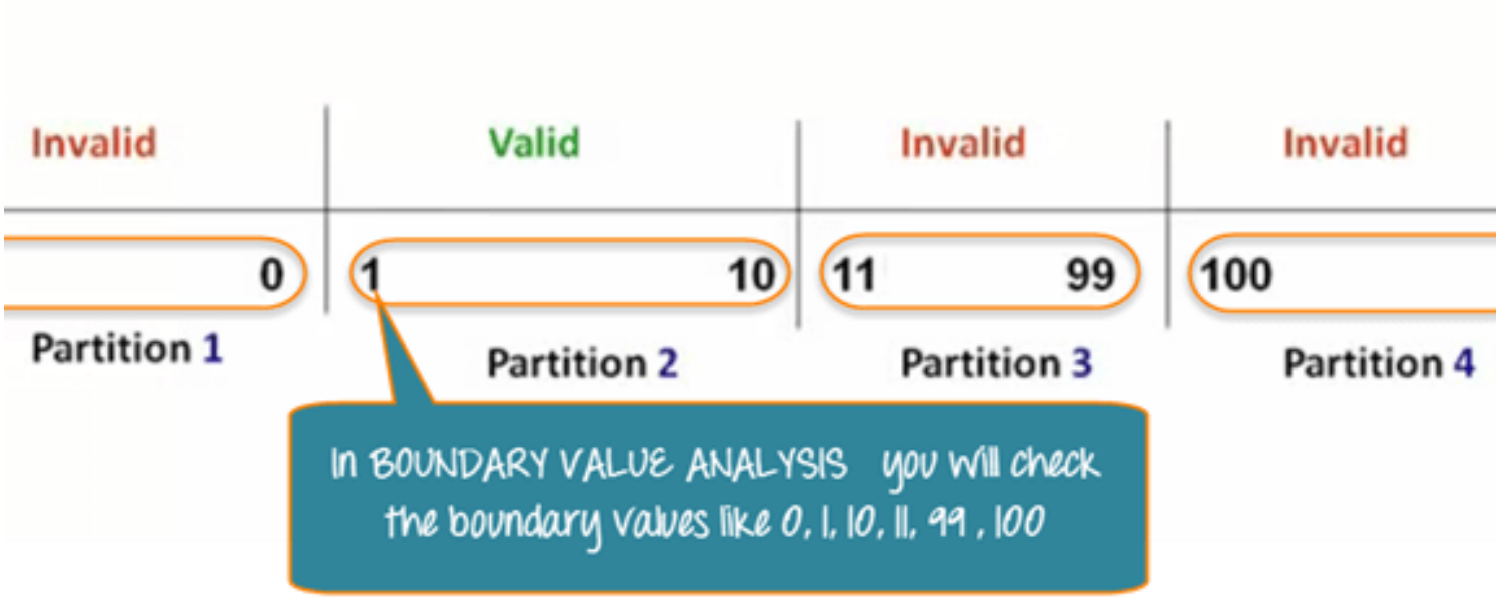

**Control flow:** White box testing can be used to ensure that all of the possible execution paths through an application are tested. This can be done by using techniques such as control flow analysis and path testing.



**Data flow:** White box testing can be used to ensure that all of the possible data flows through an application are tested. This can be done by using techniques such as data flow analysis and taint analysis.




**Boundary values:** White box testing can be used to ensure that the application behaves correctly at the boundaries of its input and output values. This can be done by using techniques such as equivalence partitioning and boundary value analysis.

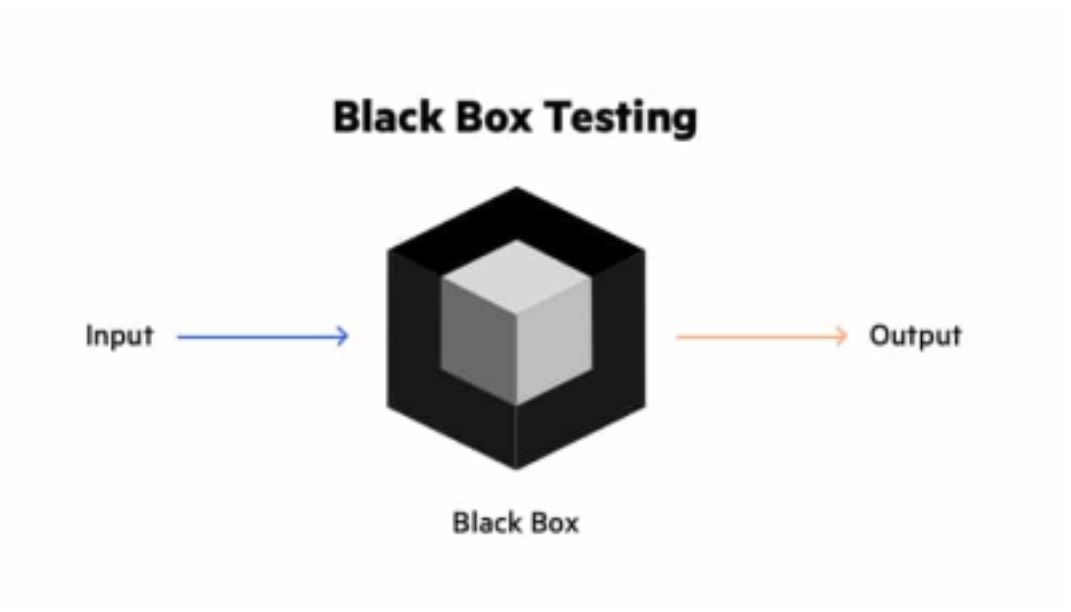



**Error handling:** White box testing can be used to ensure that the application handles errors correctly. This can be done by designing test cases that trigger different types of errors.


White box testing is a powerful tool for ensuring the quality of software applications.

However, it is important to note that white box testing alone cannot guarantee that an application is bug-free.

**Black box testing** is also necessary to test the application from a user's perspective and to ensure that it meets all of its functional requirements.




**Benefits of white box testing:**

• Identify bugs early in the development process.
• Improve code quality by identifying potential design problems and inefficiencies.
• Improve security by identifying potential vulnerabilities

**Drawbacks of white box testing:**

• Time-consuming and expensive.
• Difficult to perform if testers do not have a good understanding of the source code.
• Cannot guarantee that an application is bug-free.

In your provided assembly language code example, you're implementing a nested IF-ELSE statement and conducting white box testing by assigning different values to the variables and tracing the execution paths.

Let's break down the code and the testing results:

```
1:  mov eax, op1      ; Move op1 into eax register
2:  cmp eax, op2      ; Compare op1 with op2
3:  jne L2            ; Jump to L2 if op1 != op2
4:  mov eax, X        ; Move X into eax register
5:  cmp eax, Y        ; Compare X with Y
6:  jg L1             ; Jump to L1 if X > Y
7:  call Routine2     ; Call Routine2
8:  jmp L3            ; Jump to L3 and exit
9:  L1: call Routine1 ; Call Routine1
10: jmp L3            ; Jump to L3 and exit
11: L2: call Routine3 ; Call Routine3
12: L3:               ; Exit point
```

```
387 if op1 == op2
388     if X > Y
389         call Routine1
390     else
391         call Routine2
392     end if
393
394 else
395     call Routine3
396 end if
```

Table 6-6 shows the results of white box testing of the sample code. In the first four columns test values have been assigned to op1, op2, X, and Y. The resulting execution paths are verifiec in columns 5 and 6.

TABLE 6-6 Testing the Nested IF Statement.

| op1 | op2 | X | Y | Line Execution Sequence | Calls |
|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 1, 2, 3, 11, 12 | Routine3 |
| 10 | 20 | 40 | 30 | 1, 2, 3, 11, 12 | Routine3 |
| 10 | 10 | 30 | 40 | 1, 2, 3, 4, 5, 6, 7, 8, 12 | Routine2 |
| 10 | 10 | 40 | 30 | 1, 2, 3, 4, 5, 6, 9, 10, 12 | Routine1 |

The first four columns show the test values assigned to op1, op2, X, and Y. The fifth column shows the execution path that is taken, based on the test values. The sixth column shows the output that is produced, based on the execution path.

For example, in the first test case, the values of op1 and op2 are both 10, and the values of X and Y are 20 and 30, respectively. Since op1 is equal to op2, the execution path will follow the first branch of the IF statement, and Routine1 will be called. The output of Routine1 is unspecified, but it is likely to return a value that indicates that the condition op1 == op2 is true.

In the second test case, the values of op1 and op2 are both 10, and the values of X and Y are 30 and 20, respectively. Again, since op1 is equal to op2, the execution path will follow the first branch of the IF statement. However, this time, the condition X > Y is also true, so the execution path will follow the first branch of the nested IF statement. This will result in Routine1 being called.

In the third test case, the values of op1 and op2 are 10 and 20, respectively, and the values of X and Y are 30 and 20, respectively. Since op1 is not equal to op2, the execution path will follow the second branch of the IF statement. The condition X > Y is false, so the execution path will follow the second branch of the nested IF statement. This will result in Routine2 being called.

In the fourth test case, the values of op1 and op2 are 10 and 20, respectively, and the values of X and Y are 20 and 30, respectively. Since op1 is not equal to op2, the execution path will follow the second branch of the IF statement. The condition X > Y is also false,

so the execution path will fall through to the ELSE clause of the nested IF statement. This will result in Routine3 being called.

White box testing is a valuable tool for ensuring that complex conditional statements are working as expected. By testing all possible combinations of input values, programmers can be confident that their code will handle all possible scenarios correctly.