White-Box Testing

Condition Coverage

- **Condition coverage testing** is a type of white-box testing that tests all the conditional expressions in a program for all possible outcomes of the conditions. It is also called *predicate coverage*.
- Condition coverage testing tests the conditions independently of each other.

❑Condition coverage vs. branch coverage

In *branch coverage*, all conditions must be executed at least once. On the other hand, in *condition coverage*, all possible outcomes of all conditions must be tested at least once.

Example: Consider the code snippet below:

```
1.  int a = 10;
2.  if (a > 0)
3.  {
4.    cout<<"a is positive";
5.  }
```

Condition Coverage

Branch coverage requires that the condition a>0 is executed at least once.

Condition coverage requires that both the outcomes a>0=True and a>0=False for the condition a>0 are executed at least once.

Example 1:

Consider the code snippet below, which will be used to conduct *condition coverage testing*:

```
1.  int num1 = 0;
2.  if(num1>0)
3.  {
4.      cout<<"valid input";
5.  }
6.  Else
7.  {
8.      cout<<"invalid input";
9.  }
```

Condition Coverage

Condition coverage testing

The condition coverage testing of the code above will be as follows:

Test case number	num1>0	Final output
1	True	True
2	False	False

Example 2

Consider the code snippet below, which will be used to conduct *condition coverage testing*:

Multiple Condition Coverage

- Multiple condition coverage (MCC) is achieved, if the test cases make the component conditions of a composite conditional expression to assume all possible combinations of true and false values.
- For example, consider the composite conditional expression [(c1 and c2) or c3]. A test suite would achieve MCC, if all the component conditions c1 , c2 , and c3 are each made to assume all combinations of true and false values. Therefore, at least eight test cases would be required in this case to achieve MCC.
- For a composite conditional expression of n components, 2^n test cases are required for multiple condition coverage.
- For multiple condition coverage, the number of test cases increases exponentially with the number of component conditions.
- Therefore, multiple condition coverage-based testing technique is practical only if n (the number of atomic conditions in the decision expression) is small.

Multiple Condition Coverage

Example: Give an example of a fault that is detected by multiple condition coverage, but not by branch coverage.

Solution: Consider the following C program segment:

```
if(temperature>150 || temperature>50)
setWarningLightOn();
```

The program segment has a bug in the second component condition, it should have been temperature<50.

The test suite {temperature=160, temperature=40} achieves branch coverage. But, it is not able to check that setWarningLightOn(); should not be called for temperature values within 150 and 50.

Determining the set of test cases to achieve MC/DC

- The first step is to draw the truth table involving the basic conditions of the given decision expression.
- Next by inspection of the truth table, the MC/DC test suite can be determined such that each condition independently affects the outcome of the decision.
- Typically, we have extra columns in the truth table, which we fill during analysis of the truth table to keep track of which test cases make a condition to independently affect the outcome of the decision.
- We now illustrate this procedure through the following problems. Though it is harder to prove, for various examples, we can observe that at least n + 1 test cases are necessary to achieve MC/DC for decision expressions with n conditions.

Problem-1: Design MC/DC test suite for the following decision statement:

if (A and B) then

Determining the set of test cases to achieve MC/DC

Solution:

We first draw the truth table:

Test case number	*A*	*B*	*Decision*	*Test case pair for A*	*Test case pair for B*
1	T	T	T	3	2
2	T	F	F		1
3	F	T	F	1	
4	F	F	F		

From the truth table given above, we can observe that the test cases 1, 2 and 3 together achieve MC/DC for the given expression.

Determining the set of test cases to achieve MC/DC

Problem-2: Design a test suite that would achieve MC/DC for the following decision statement:

~~*if((A && B) || C)*~~ if((A && (B|| C))

Solution: We first draw the truth table:

Test case	*ABC*	*Result*	*A*	*B*	*C*
1	TTT	T	5		
2	TTF	T	6	4	
3	TFT	T	7		4
4	TFF	F		2	3
5	FTT	F	1		
6	FTF	F	2		
7	FFT	F	3		
8	FFF	F			

From the table, we can observe that different sets of test cases achieve MC/DC. The sets are {2,3,4,6}, {2,3,4,7} and {1,2,3,4,5}.

Multiple Condition Coverage

Example :

Consider the code snippet below, which will be used to conduct *condition coverage testing*:

```
1.  int num1 = 0;
2.  int num2 = 0;
3.  if((num1>0 || num2<10))
4.  {
5.      cout<<"valid input";
6.  }
7.  Else
8.  {
9.      cout<<"invalid input";
10. }
```

Condition Coverage

Condition coverage testing

The condition coverage testing of the code above will be as follows:

Test case number	num1>0	num2<10	Final output
1	True	Not required	True
2	False	True	True
3	False	False	False

Condition Coverage

Example

Consider the code snippet below, which will be used to conduct *condition coverage testing*:

```
1.  int num1 = 0;
2.  int num2 = 0;
3.  if((num1>0) && (num1+num2<15))
4.  {
5.      cout<<"valid input";
6.  }
7.  Else
8.  {
9.      cout<<"invalid input";
10. }
```

Condition Coverage

Condition coverage testing

The condition coverage testing of the code above will be as follows:

Test case number	num1>0	num1+num2<15	Final output
1	True	True	True
2	True	False	False
3	False	Not required	False

Condition Coverage

Example

Consider the code snippet below, which will be used to conduct *condition coverage testing*:

```
1.  int num1 = 0;
2.  int num2 = 0;
3.  if((num1>0 || num2<10) && (num1+num2<15))
4.  {
5.      cout<<"valid input";
6.  }
7.  Else
8.  {
9.      cout<<"invalid input";
10. }
```

Condition Coverage

Condition coverage testing

The condition coverage testing of the code above will be as follows:

Test case number	num1>0	num2<10	num1+num2<15	Final output
1	True	don't care	True	True
2	True	don't care	False	False
3	False	True	True	True
4	False	True	False	False
5	False	False	Not required	False