Singapore Institute of Management
BSc in Computing & Information Systems
CIS 210 Tutorial 12

---

1. Explain the difference between white box and black box testing, indicate the types of faults that each is more capable of detecting.

Solution:

Black box testing alludes to tests that are conducted at software interface. Black-box tests are used to demonstrate that software functions are operational, that input is properly accepted and output is correctly produced, and that the integrity of external information (e.g. a database) is maintained. A black-box test examines the fundamental aspect of a system with little regard for the internal logical structure of the software.

White-box testing of software involves close examination of procedural detail. Logical paths through the software are tested by providing test cases that exercise specific sets of conditions and loops.

Black box testing does not rely on knowledge of the code whereas white box testing is constructed with knowledge of the code structure.

Black box testing is good at finding missing functionality but poor at insuring the implementation of all structural components whereas white box testing is poor at the former but good at the latter.

2.

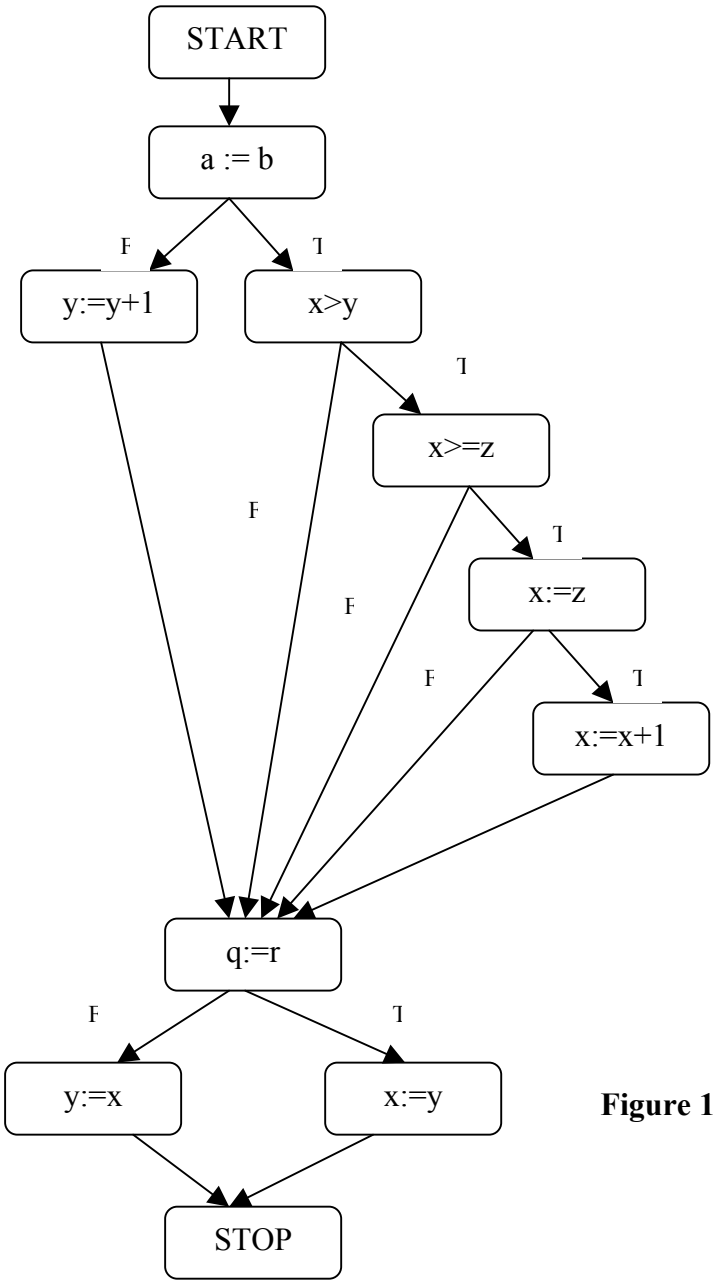


Figure 1

Consider the Control flow graph (CFG) in Figure 1 above. The outgoing edges of predicates are labeled, T and F, to indicate which branch is taken for the *true* and *false* outcomes of the evaluation of associated Boolean expression.

Design a minimal set of test data which achieves 100% statement coverage of this CFG. Give your answer in the form of a table shown in Figure 2 below, where each row corresponds to a single test case and each column gives the initial value of each variable for each test case.

	r	q	a	b	x	y	z
Test input 1							
Test input 2							

Figure 2

Solution:

Only 2 tests are required (minimality is important)

	r	q	a	b	x	y	z
Test input 1	8	5	5	3			
Test input 2	3	3	7	7	5	3	5

3. Based on CFG given in Figure 1, design a test set which is minimal and which achieves 100% branch coverage. Give your answer using the format shown in Figure 2.

Solution:

	r	q	a	b	x	y	z
Test input 1	5	5	7	4			
Test input 2	5	5	5	5	7	5	9
Test input 3	5	5	5	5	7	5	6
Test input 4	4	7	5	5	7	5	7
Test input 5	4	7	5	5	5	7	

4. Examine the following code :

```
Procedure average ;
    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;
    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total input, total.valid, minimum; maximum, sum IS SCALAR;
    TYPE I is INTEGER;

    i=0;
    total.input = total.valid = 0;
    sum = 0;

    DO WHILE value[i] <> -999 and total.input < 100
            Increment total.input by 1;

            IF value[i] >= minimum AND value[i] <= maximum
                    THEN increment total.valid by 1;
                            sum = sum + value[i];
                    ELSE    skip
            ENDIF
            Increment I by 1;
    ENDDO

    IF total.valid > 0
            THEN average = sum/total.valid;
            ELSE average = -999;
    ENDIF
END average
```

Draw a control flow graph for this program. Use it to help choose a test set that gives full path coverage.

Solution:

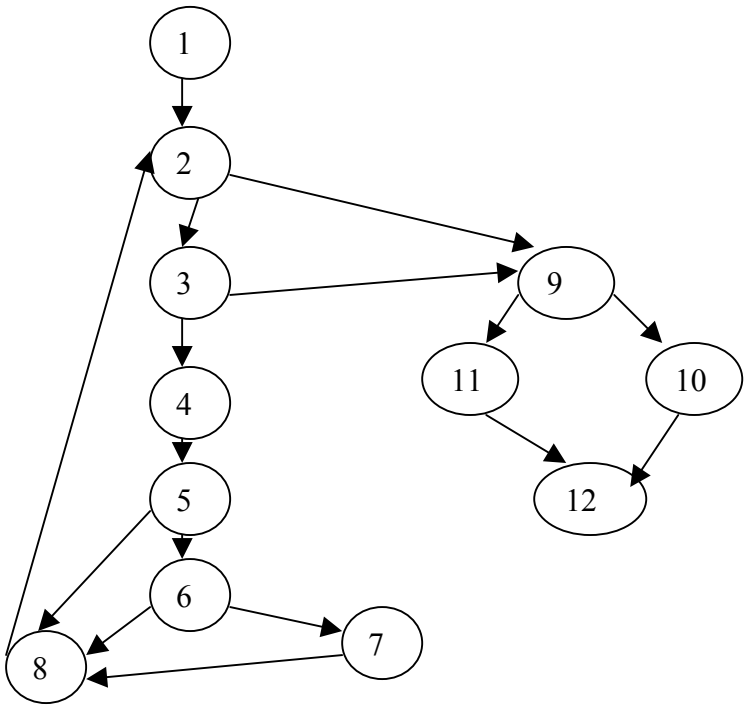
```
Procedure average ;
    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;
    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total input, total.valid, minimum; maximum, sum IS SCALAR;
    TYPE I is INTEGER;

    i=0;
1   total.input = total.valid = 0;
    sum = 0;                                                    2

    DO WHILE value[i] <> -999 and total.input < 100             3
4       Increment total.input by 1;

        IF value[i] >= minimum AND value[i] <= maximum          6
5               THEN increment total.valid by 1;                7
                        sum = sum + value[i];
                ELSE    skip
8       ENDIF
        Increment I by 1;
    ENDDO

    IF total.valid > 0                                          9
            THEN average = sum/total.valid;                     10
            ELSE average = -999;                                11
    ENDIF                                                       12
END average
```



The Linearly Independent paths include :

a) *Path 1 : 1-2-9-10-12*
b) *Path 2 : 1-2-9-12-12*
c) *Path 3 : 1-2-3-9-10-12*
d) *Path 4 : 1-2-3-4-5-8-2-….-12*
e) *Path 5 : 1-2-3-4-5-6-8-2-….-12*
f) *Path 6 : 1-2-3-4-5-6-7-8-2-…-12*

The test cases include

Path No.	***Test Cases***
1	*Value (k) – valid input where k<I defined below* *Value Ii) = -999 where 2<i<100* *Expected results = correct average based on k values and proper totals. Note : cannot be tested alone but must be tested as part of path 4, 5 and 6 tests..*
2	*Value (1) = -999* *Expected results : average = -999; other totals at initial values*
3	*Attempt to process 101 or more values* *First 100 values will be valid*

	Expected results : same as test case 1
4	*Value (i) = valid input where i < 100* *Value k < minimum where k < i* *Expected results : correct average based on k values and proper totals*
5	*Value (i) = valid input where i < 100* *Value (k) > maximum where k*i* *Expected results : correct average based on n values and proper totals*
6	*Value (i) = valid input where I < 100* *Expected results : correct average based on n values and proper totals.*

5. What is mutation testing? Your answer should include a definition of test coverage in terms of mutation testing. Give an example to illustrate a test case which would kill a mutant in a program code.

Solution:

Mutation testing involves deliberate 'corruption' of the program by changing a statement or a predicate in the program code. The output from the test will indicate whether a mutant has been 'killed'. 100% mutant coverage is defined as given a set of mutants m and a set of test cases, t, t achieves 100% mutation coverage with respect to m if all mutants in m are killed by at least one test case in t.

An example,

If p > q then x:=1 else y:=2 z:=x+3 if z = p then z:=z + 1;	If p > q then x:=1 else y:=2 z:=x+3 if z = p then z:=z + 2;

{ (p,4), (q, 3) }

6. Explain what is the equivalent mutant problem. Illustrate your answer with an example.

Solution:

Equivalent mutant is defined as a mutant m of a program p is said to be equivalent to p if it produces the same output when supplied with the same input (for all possible inputs)

Example,

Replacing statement *x:= x + x* with statement *x := 2 * x*