Code-based Testing Exercise answer

Study the Triangle example.

```
1.  Triangle (a, b, c: Integer): String
2.    IsATriangle: Boolean
3.  begin
4.    if (a<=b+c or b<=a+c or c<=b+c)
5.    then IsATriangle := true
6.    else IsATriangle := false  fi;
7.    if (IsATriangle)
8.    then if (a=b and b=c)
9.            then return "Equilateral"
10.           else if (a≠b and a≠c and b≠c)
11.                   then return "Scalene"
12.                   else return "Equilateral" fi; fi;
13.   else return "Not a triangle" fi;
14. end
```

Take as triangle definition: inputs a,b,c must satisfy each of the following requirements.
(1) a>0,
(2) b>0,
(3) c>0,
(4) a+b>=c,
(5) a+c>=b,
(6) b+c>=a.

This allows for triangles with an empty surface. (The stricter version could be obtained e.g. by adding two extra constraints: (7) a>|b-c|, (8) b>|a-c|)

An equilateral is a triangle for which a=b=c.
An isosceles is a triangle which is not equilateral and for which a=b or b=c or a=c.
A scalene is a triangle which is not equilateral and not isosceles.

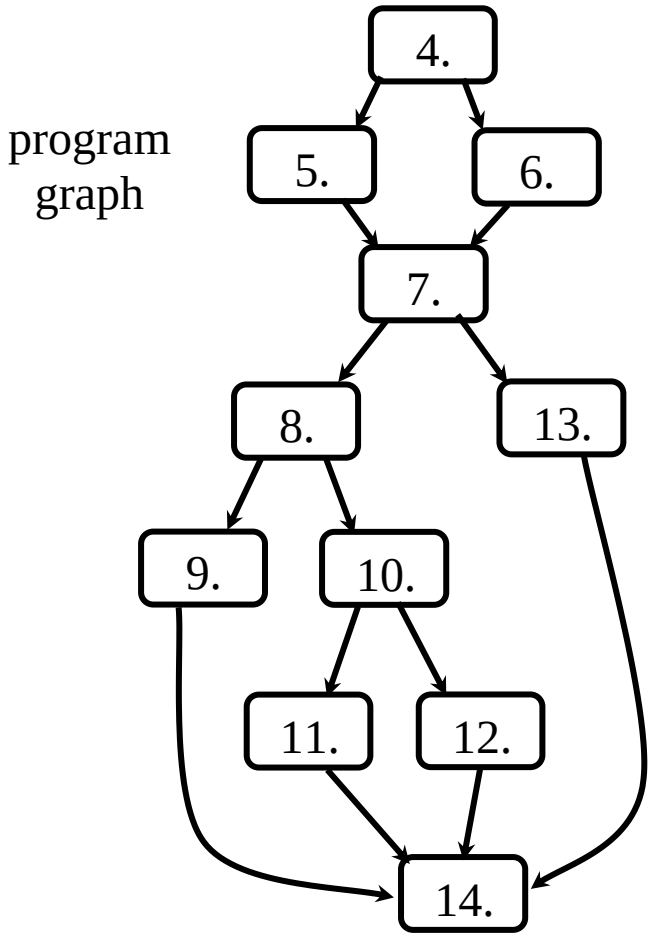
With these requirements, the program contains 4 errors:

- *1 for not checking input domains at all*
- *1 for using 'or' instead of 'and' in the condition in node 4*
- *1 for using 'c' instead of 'a' in the last atomic predicate in node 4*
- *1 for returning 'Equilateral' instead of 'Isosceles' in node 12*

Counting the errors is not completely straightforward.

1. Construct tests for the branch coverage and condition coverage criterion. Select a set of tests as small as possible, make the tests as effective as possible: if you have a choice, take an input combination that yields a failure.

In this case, statement (node) coverage equals branch coverage, as is easily seen from the control flow graph.



Node/branch coverage: *See tests 1-4 in the table below.*
Condition coverage: *Fix names for the atomic predicates as follows:*

- *Node 4: p1 =”a<=b+c”, p2=”b<=a+c”, p3=”c<=b+c”*
- *Node 7: p4=”IsATriangle”*
- *Node 8: p5=”a=b” , p6=”b=c”*
- *Node 10: p7=”a≠b”, p8=”a≠c”, p9=”b≠c”*

The tests are in the table below. The last two columns indicate for each atomic predicate whether it holds in the test execution or not. Note that we can write it down like that because there are no loops.

Test	***a***	***b***	***c***	***Exp. output***	***Actual output***	***Failure?***	***path***	***TRUE***	***FALSE***
1	*-1*	*-1*	*-1*	*Not a triangle*	*Not a triangle*	*FALSE*	*4,6,7,13,14*		*p1,p2,p3,p4*
2	*0*	*0*	*0*	*Not a triangle*	*Equilateral*	*TRUE*	*4,5,7,8,9,14*	*p1,p2,p3,p4,p5,p6*	
3	*1*	*1*	*2*	*Isosceles*	*Equilateral*	*TRUE*	*4,5,7,8,10,12,14*	*p1,p2,p3,p4,p5,p8,p9*	*p6,p7*
4	*-1*	*1*	*2*	*Not a triangle*	*Scalene*	*TRUE*	*4,5,7,8,10,11,14*	*p1,p2,p4,p8,p9*	*p3,p5,p6,p7*
5	*1*	*2*	*1*	*Isosceles*	*Equilateral*	*TRUE*	*4,5,7,8,10,11,14*	*p1,p2,p3,p4,p7,p9*	*p5,p6,p8*
6	*2*	*1*	*1*	*Isosceles*	*Equilateral*	*TRUE*	*4,5,7,8,10,11,14*	*p1,p2,p3,p4,p6,p7,p8*	*p5,p9*
6						*5*			

2. Determine which input combinations detect an error and how many different errors are detected in total.

For node/branch coverage, at least 3 errors are expected to be detected. We can see that the condition in node 4 has two errors: the 'or' operator instead of the 'and' and the typo 'c' for 'a' in the last atomic predicate. But will a debugger based on the code and requirements recognise both of these? If so, all errors are found.