}

DD2459: Software Reliability, sofRel15

Lab 1: White-box Testing

Answer all 4 questions.

Introduction:

The *triangle program* is a famous testing problem that originated in Myers classical 1979 textbook on testing. It has appeared in many books and papers since, as it is often a good benchmark for new ideas about testing. The program requirement is defined as follows:

"*The program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral*" (Myers, page 1)

We need to recall some facts from elementary geometry:

1. A *triangle* is a polygon with three sides.

2. The *vertices* of a triangle must not be in a straight line.

3. An *equilateral triangle* has three sides of equal length.

4. An *isosceles triangle* has two sides of equal length.

5. A *scalene triangle* has three sides of different lengths.

The **Triangle Test algorithm** below (hopefully) implements the requirement defined above.

```
enumeration Kind = { scalene, isosceles, equilateral, notriangle,
badside } // a data type definition

Kind triangleTest( s1, s2, s3 : int ) {
      if s1 <= 0 or s2 <= 0 or s3 <= 0
      then return badside
      else
            if s1+s2 <= s3 or s2+s3 <= s1 or s1+s3 <= s2 according to triangle inequality
            then return notriangle                                       theorem
            else
                  if s1==s2 & s2==s3
                  then
                        return equilateral
                  else
                        if s1==s2 or s2==s3 or s1==s3
                        then
                              return isosceles
                        else
                              return scalene
}
```

Question 1. Draw a condensation graph for the Triangle Test algorithm.

In this exercise, you will write out test requirements as paths through this condensation graph to achieve different levels of control flow coverage.

> **Example: NC TR1:** n_1, n_2, n_3, n_4
>
> is a test requirement that attempts to cover 4 nodes (which four?) in a condensation graph for Algorithm 1, attempting to achieve node coverage.

1.1 (a) Write a set of test requirements that achieve **node coverage** (NC) for the Triangle Test algorithm.
(b) Write out a minimized set of test cases satisfying the requirements of (a).

1.2. (a) Write out a set of test requirements that achieve **edge coverage** (EC) for the Triangle Test algorithm.
(b) Write out a minimized corresponding set of test cases.
(c) Why are node coverage and edge coverage the same in this example?

Question 2. In this exercise, you will write out test requirements as logical constraints on the input variable values s1, s2 and s3 to achieve different levels of logic coverage.

> **Example: PC TR1**: s1 <= 0 | s2 <= 0 | s3 <= 0
>
> is a test requirement that makes a predicate (which?) in a condensation graph for Question 1, *true*, attempting to achieve predicate coverage.
>
> Then you must write out a test case that satisfies each requirement. If you can minimize the set of test cases by eliminating redundant test cases that is a (locally) optimal solution. A test case satisfying requirement PC TR1 might be:
>
> **PC TC1**: s1 = 0, s2 = 0, s3 = 0
>
> which satisfies this test requirement at a *boundary*.

2.1. (a) Write out a set of test requirements that achieve **predicate coverage** (PC) for the Triangle test algorithm 1. (Recall that non-distributive predicate coverage is sufficient here.)

(b) Can you modify the condensation graph in some simple way so that predicate coverage and edge coverage (or node coverage) are not the same?
(c) Write out a corresponding set of test cases.

2.2. (a) Write out a set of test requirements that achieve **clause coverage** (CC) for the Triangle Test Algorithm.
(b) Write out a corresponding set of test cases.

2.3. (a) Write out a set of test requirements that **restricted active clause coverage** (RACC) (also known as MCDC) for the Triangle Test Algorithm.
(b) Write out a corresponding set of test cases.

Question 3. Consider the following piece of code:

```
x = x+1;

while ( x < -100 || x > 100) {

     if (x < -100) then { x = x+1; } else

          if (x > 100) then { x = x-1; }

}

return x;
```

You can assume that `x:int` is the single input variable to the above program.

(a) Draw a condensation graph for this code.
(b) Define a minimal set TR of test requirements on the input variable x that would achieve full (100%) node coverage for this program. Carefully explain why your test requirement set is actually minimal.

(c) Produce a set TC of test cases that satisfy your test requirements for TR in Part 3.(b).

(d) Would predicate coverage yield a better test suite than your answer to 3.(c)? Motivate your answer.

PTO.

<u>Question 4</u>. **Self-Assessment**

For each of the five sets of test cases you have produced in Questions 1 and 2 (i.e. for each of the five coverage models NC, EC, PC, CC, RACC) , answer the following 14 self assessment questions. For each coverage model, score 1 point for a requirement that is satisfied (maximum 14 points). Which coverage model achieves the highest score?

1. Do you have a test case that represents a valid scalene triangle?

2. Do you have a test case that represents a valid equilateral triangle?

3. Do you have a test case that represents a valid isosceles triangle?

4. Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of two equal sides?

5. Do you have a test case in which one side has a zero value?

6. Do you have a test case in which one side has a negative value?

7. Do you have a test case with three integers such that the sum of two is equal to the third?

8. Do you have at least three test cases in category 7 such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides?

9. Do you have a test case with three integers greater than zero such that the sum of two numbers is less than the third?

10. Do you have at least three test cases in category 9 such that you have tried all three permutations

11. Do you have a test case in which all sides are zero?

12. Do you have at least one test case specifying non-integer values? *

13. Do you have at least one test case specifying the wrong number of values (2 or less, four or more) **

14. For each test case, did you specify the expected output from the program in addition to the input values?

Reference: G.J. Myers, *The Art of Software Testing*, John Wiley and Sons, 1979.