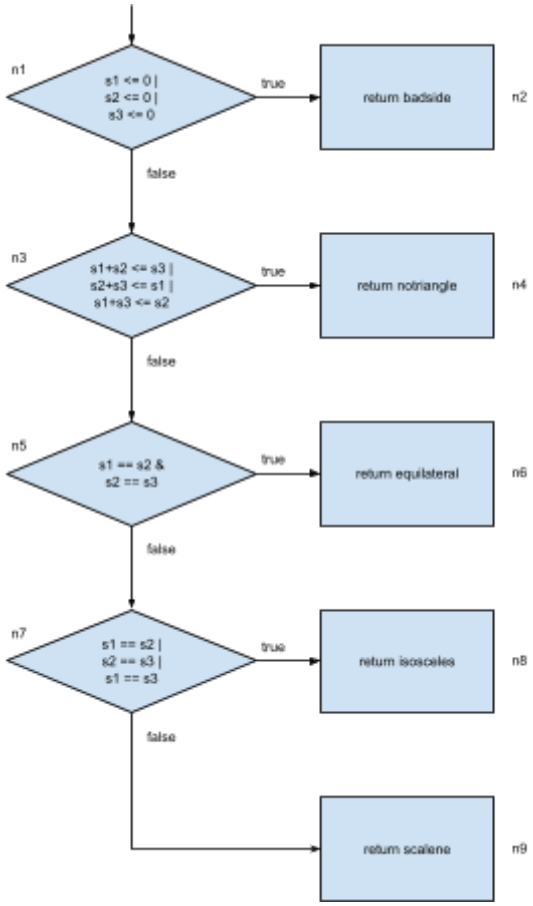
Alex Gunnarsson
alexgu@kth.se

DD2459 - Lab 1

Question 1:

Condensation graph:



1.1. (a)
NC TR1: n1, n2
NC TR2: n1, n3, n4
NC TR3: n1, n3, n5, n6
NC TR4: n1, n3, n5, n7, n8
NC TR5: n1, n3, n5, n7, n9

(b)
NC TC1: s1 = -1, s2 = -1, s3 = -1 (Expected output: badside)
NC TC2: s1 = 1, s2 = 1, s3 = 5 (Expected output: notriangle)
NC TC3: s1 = 1, s2 = 1, s3 = 1 (Expected output: equilateral)
NC TC4: s1 = 2, s2 = 2, s3 = 1 (Expected output: isosceles)
NC TC5: s1 = 3, s2 = 4, s3 = 5 (Expected output: scalene)

1.2. (a)
EC TR1: n1, n2

Alex Gunnarsson
alexgu@kth.se

EC TR2: n1, n3, n4
EC TR3: n1, n3, n5, n6
EC TR4: n1, n3, n5, n7, n8
EC TR5: n1, n3, n5, n7, n9

(b)
EC TC1: s1 = -1, s2= -1, s3 = -1 (Expected output: badside)
EC TC2: s1 = 1, s2= 1, s3 = 5 (Expected output: notriangle)
EC TC3: s1 = 1, s2= 1, s3 = 1 (Expected output: equilateral)
EC TC4: s1 = 2, s2= 2, s3 = 1 (Expected output: isosceles)
EC TC5: s1 = 3, s2= 4, s3 = 5 (Expected output: scalene)

(c)
NC and EC are the same in this example because the condensation graph does not include multiple ways of reaching a node using different edges. The graph has a tree structure, which means that if we cover all nodes we also cover all edges, and vice versa.

Question 2:

2.1. (a)
PC TR1: s1 <= 0 | s2 <= 0 | s3 <= 0
PC TR2: s1 > 0 & s2 > 0 & s3 > 0 (Define this as !P1)
PC TR3: !P1 & (s1+s2 <= s3 | s2+s3 <= s1 | s1+s3 <= s2)
PC TR4: !P1 & (s1+s2 > s3 & s2+s3 > s1 & s1+s3 > s2) (Define this as !P1 & !P2)
PC TR5: !P1 & !P2 & (s1 == s2 & s2 == s3)
PC TR6: !P1 & !P2 & (s1 != s2 | s2 != s3) (Define this as !P1 & !P2 & !P3)
PC TR7: !P1 & !P2 & !P3 & (s1 == s2 | s2 == s3 | s1 == s3)
PC TR8: !P1 & !P2 & !P3 & (s1 != s2 & s2 != s3 & s1 != s3)

PC TC1: s1 = -1, s2 = -1, s3 = -1 (Expected output: badside)
PC TC2: s1 = 1, s2 = 1, s3 = 5 (Expected output: notriangle)
PC TC3: s1 = 1, s2 = 1, s3 = 1 (Expected output: equilateral)
PC TC4: s1 = 2, s2 = 2, s3 = 1 (Expected output: isosceles)
PC TC5: s1 = 3, s2 = 4, s3 = 5 (Expected output: scalene)

(b)
Not always the same. Similar to 1.2.(c), it may be that there are different ways of reaching a node. Therefore, we don't have to satisfy the predicate to be true in one case and false in another, while still reaching that same node and achieving NC.

(c)
By, for example, instead of returning "badside" (n2) if the first condition (n1) is true, we return "scalene" (n9). Therefore, we do not need to make the predicate in n2 true

Alex Gunnarsson
alexgu@kth.se

in order to achieve NC, since it will be reached by making the predicate in n7 false. As such, we achieve NC but not PC.

NC TC1: s1 = -1, s2 = -1, s3 = -1 (Expected output: badside)
NC TC2: s1 = 1, s2 = 1, s3 = 5 (Expected output: notriangle)
NC TC3: s1 = 1, s2 = 1, s3 = 1 (Expected output: equilateral)
NC TC4: s1 = 2, s2 = 2, s3 = 1 (Expected output: isosceles)

PC TC1: s1 = -1, s2 = -1, s3 = -1 (Expected output: badside)
PC TC2: s1 = 1, s2 = 1, s3 = 5 (Expected output: notriangle)
PC TC3: s1 = 1, s2 = 1, s3 = 1 (Expected output: equilateral)
PC TC4: s1 = 2, s2 = 2, s3 = 1 (Expected output: isosceles)
PC TC5: s1 = 3, s2 = 4, s3 = 5 (Expected output: scalene)
^ Last one is needed for PC but not NC

2.2. (a)
CC TR1: s1 <= 0
CC TR2: s2 <= 0
CC TR3: s3 <= 0
CC TR4: s1 > 0
CC TR5: s2 > 0
CC TR6: s3 > 0
CC TR7: !P1 & s1+s2 <= s3
CC TR8: !P1 & s2+s3 <= s1
CC TR9: !P1 & s1+s3 <= s1
CC TR10: !P1 & s1+s2 > s3
CC TR11: !P1 & s2+s3 > s1
CC TR12: !P1 & s1+s3 > s1
CC TR13: !P1 & !P2 & s1 == s2
CC TR14: !P1 & !P2 & s1 != s2
CC TR15: !P1 & !P2 & s2 == s3
CC TR16: !P1 & !P2 & s2 != s3
CC TR17: !P1 & !P2 & !P3 & s1 == s2
CC TR18: !P1 & !P2 & !P3 & s2 == s3
CC TR19: !P1 & !P2 & !P3 & s1 == s3
CC TR20: !P1 & !P2 & !P3 & s1 != s2
CC TR21: !P1 & !P2 & !P3 & s2 != s3
CC TR22: !P1 & !P2 & !P3 & s1 != s3

(b)
CC TC1: s1 = -1, s2 = -1, s3 = -1 (TR1, TR2, TR3)
(Expected output: badside)
CC TC2: s1 = 1, s2 = 1, s3 = 5 (TR4, TR5, TR6) + (TR7, TR11, TR12)
(Expected output: notriangle)

Alex Gunnarsson
alexgu@kth.se

CC TC3: s1 = 1, s2 = 5, s3 = 1 (TR9, TR10)
(Expected output: notriangle)
CC TC4: s1 = 5, s2 = 1, s3 = 1 (TR8)
(Expected output: notriangle)
CC TC5: s1 = 2, s2 = 2, s3 = 3 (TR13, TR16, TR17, TR22)
(Expected output: isosceles)
CC TC6: s1 = 3, s2 = 2, s3 = 2 (TR14, TR15, TR18, TR20)
(Expected output: isosceles)
CC TC7: s1 = 2, s2 = 3, s3 = 2 (TR19, TR21)
(Expected output: isosceles)

2.3. (a)
s1 <= 0 | s2 <= 0 | s3 <= 0:
RACC TR1: s1 <= 0 & s2 > 0 & s3 > 0
RACC TR2: s1 > 0 & s2 <= 0 & s3 > 0
RACC TR3: s1 > 0 & s2 > 0 & s3 <= 0
RACC TR4: s1 > 0 & s2 > 0 & s3 > 0
s1+s2 <= s3 | s2+s3 <= s1 | s1+s3 <= s2:
RACC TR5: !P1 & (s1+s2 <= s3 & s2+s3 > s1 & s1+s3 > s2)
RACC TR6: !P1 & (s1+s2 > s3 & s2+s3 <= s1 & s1+s3 > s2)
RACC TR7: !P1 & (s1+s2 > s3 & s2+s3 > s1 & s1+s3 <= s2)
RACC TR8: !P1 & (s1+s2 > s3 & s2+s3 > s1 & s1+s3 > s2)
s1 == s2 & s2 == s3:
RACC TR9: !P1 & !P2 & (s1 != s2 & s2 == s3)
RACC TR10: !P1 & !P2 & (s1 == s2 & s2 != s3)
RACC TR11: !P1 & !P2 & (s1 == s2 & s2 == s3)
s1 == s2 | s2 == s3 | s1 == s3:
RACC TR12: !P1 & !P2 & !P3 & (s1 == s2 & s2 != s3 & s1 != s3)
RACC TR13: !P1 & !P2 & !P3 & (s1 != s2 & s2 == s3 & s1 != s3)
RACC TR14: !P1 & !P2 & !P3 & (s1 != s2 & s2 != s3 & s1 == s3)
RACC TR15: !P1 & !P2 & !P3 & (s1 != s2 & s2 != s3 & s1 != s3)

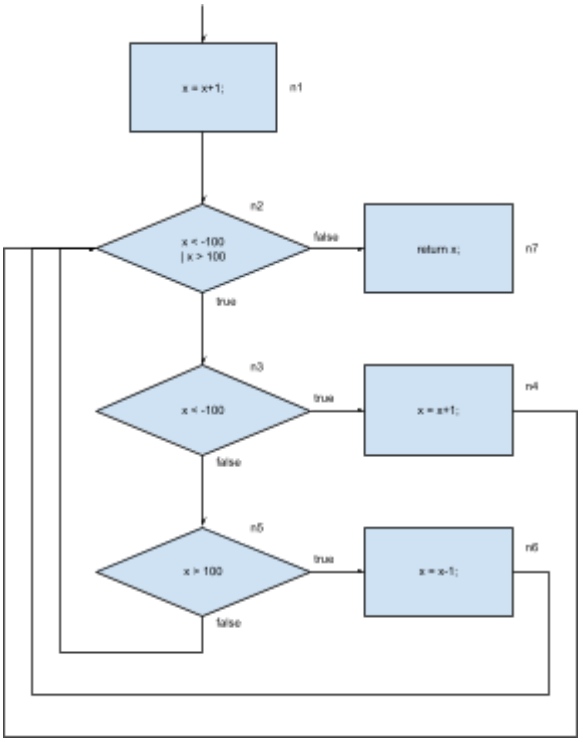
(b)
RACC TC1: s1 = -1, s2 = 1, s3 = 1 (TR1)
(Expected output: badside)
RACC TC2: s1 = 1, s2 = -1, s3 = 1 (TR2)
(Expected output: badside)
RACC TC3: s1 = 1, s2 = 1, s3 = -1 (TR3)
(Expected output: badside)
RACC TC4: s1 = 1, s2 = 1, s3 = 2 (TR4, TR5)
(Expected output: notriangle)
RACC TC5: s1 = 2, s2 = 1, s3 = 1 (TR6)
(Expected output: notriangle)
RACC TC6: s1 = 1, s2 = 2, s3 = 1 (TR7)

Alex Gunnarsson
alexgu@kth.se

(Expected output: notriangle)
RACC TC7: s1 = 1, s2 = 2, s3 = 2 (TR8, TR9, TR13)
(Expected output: isosceles)
RACC TC8: s1 = 2, s2 = 2, s3 = 1 (TR10, TR12)
(Expected output: isosceles)
RACC TC9: s1 = 2, s2 = 2, s3 = 2 (TR11)
(Expected output: equilateral)
RACC TC10: s1 = 2, s2 = 1, s3 = 2 (TR14)
(Expected output: isosceles)
RACC TC11: s1 = 3, s2 = 4, s3 = 5 (TR15)
(Expected output: scalene)

Question 3.

(a)



(b)
NC TR1: x <= -102
NC TR2: x >= 100
Better would be just one TR, but it is impossible for one x-value to lead to visiting both n4 and n6 in the same execution, since it would first have to be (for example) less than -100, so we go through n4 and add 1. But to later visit n6, the value of x has to have jumped above 100, which is impossible since all we do is add 1 per iteration of the loop, and whenever it reaches -100 it will return x and stop the loop. So 2 TRs is the second best, which we can achieve. 1 is added to x, after which it has to be either below -100 or above 100

Alex Gunnarsson
alexgu@kth.se

(c)
NC TC1: x = -102 (Expected output: -100)
NC TC2: x = 100 (Expected output: 100)

(d)
It is impossible to achieve 100% predicate coverage, since the last clause will never evaluate to false based on the code. We enter the loop knowing that x is either less than -100 or greater than 100. If the first condition is false, then we know that x is not less than -100, which means it must always be greater than 100.

Question 4:

1. **Do you have a test case that represents a valid scalene triangle?**
   NC: yes
   EC: yes
   PC: yes
   CC: no
   RACC: yes

2. **Do you have a test case that represents a valid equilateral triangle?**
   NC: yes
   EC: yes
   PC: yes
   CC: no
   RACC: yes

3. **Do you have a test case that represents a valid isosceles triangle?**NC:
   NC: yes
   EC: yes
   PC: yes
   CC: yes
   RACC: yes

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?**
   NC: no
   EC: no
   PC: no
   CC: yes
   RACC: yes

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

Alex Gunnarsson
alexgu@kth.se

NC: no
EC: no
PC: no
CC: no
RACC: no

6. **Do you have a test case in which one side has a negative value?**
   NC: yes
   EC: yes
   PC: yes
   CC: yes
   RACC: yes

7. **Do you have a test case with three integers such that the sum of two is equal to the third?**
   NC: no
   EC: no
   PC: no
   CC: no
   RACC: yes

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?**
   NC: no
   EC: no
   PC: no
   CC: no
   RACC: yes

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?**
   NC: yes
   EC: yes
   PC: yes
   CC: yes
   RACC: yes

10. **Do you have at least three test cases in category 9 such that you have tried all three permutations**
    NC: no
    EC: no
    PC: no
    CC: yes

Alex Gunnarsson
alexgu@kth.se

RACC: yes

11. **Do you have a test case in which all sides are zero?**
    NC: no
    EC: no
    PC: no
    CC: no
    RACC: no

12. **Do you have at least one test case specifying non-integer values or does this not make sense?**
    NC: no
    EC: no
    PC: no
    CC: no
    RACC: no
    I thought about it, but it doesn't make sense if we restrict the type to specifically be int.

13. **Do you have at least one test case specifying the wrong number of values (2 or less, four or more) or does this not make sense?**
    NC: no
    EC: no
    PC: no
    CC: no
    RACC: no
    Doesn't make sense if we have hardcoded the amount of arguments.

14. **For each test case, did you specify the expected output from the program in addition to the input values?**
    NC: yes
    EC: yes
    PC: yes
    CC: yes
    RACC: yes

Total:
NC: 6
EC: 6
PC: 6
CC: 6
RACC: 10

RACC has the highest score.