

Tutorial 4 Flow of control (I) - branching

Please test the correctness of your **programs in Q.1 and Q.2** using PASS.

1. Write a program which reads 3 integer values from the user. The 3 values are interpreted as representing the lengths of the sides of a triangle. The program prints a message saying whether the triangle is *equilateral* (all sides equal), *isosceles* (only 2 sides equal), *scalene* (all sides unequal), or *impossible* (can't form a triangle). A triangle can be formed only if the sum of the length of *any* 2 sides is greater than the length of the 3rd side and the length of all the sides of the triangle are *positive*.

Example 1	Example 2	Example 3	Example 4
<u>3</u> <u>4</u> <u>5</u> <i>scalene</i>	<u>3</u> <u>3</u> <u>3</u> <i>equilateral</i>	<u>5</u> <u>5</u> <u>2</u> <i>isosceles</i>	<u>1</u> <u>2</u> <u>10</u> <i>impossible</i>

Hint1: If you'd like to check for equality, you should not write something like: `if (A==B==C)`, but instead, you should use the `&&` operator: `if (A==B && B==C)`

Hint2: The *order* of checking may affect the complexity of your code (*although it still works*). You may wish to check for impossible cases first, and identify the scalene case last.

Try to think of different test cases to test your program.

For example, what is the output of {10 5 5} ? Is it isosceles?

Question 2 below requires knowledge on loops (Lecture Notes #4), you may wish to delay the question to next week, after you learnt *while-loop* and *for-loop*.

2. Modify the program in Q.1 such that it can handle multiple triangles. The user first enters the number of triangles, `n`, followed by `n` sets of the three sides of a triangle. For each set, output the triangle type as in Q1.

The following example shows the case when there're two input triangles (i.e. `n=2`).

```
2
3 3 3
equilateral
3 4 5
Scalene
```