The test for right triangles will fail on virtually all fractional values, because of truncation errors. For example, the triangle `A=3.0, B=4.0, C=5.0` will be recognized as right-angled on most machines. But if we scale the values down by a factor of ten, the triangle `A=0.3, B=0.4, C=0.5` will often not be "right-angled." The code has to be replaced by some criterion of "near enough." (And this must be relative, not absolute, as we saw in Chapter 1.)

We do not have space to go more deeply into the mysteries of floating point computation; that is the province of numerical analysts. We intend only to emphasize that floating point computations should be used cautiously when controlling an algorithm. They should seldom be used for counting, nor should two computed floating point values be compared only for equality.

Don't compare floating point numbers just for equality.

Let us summarize the main lessons of this chapter. Remember that the errors that we have shown are by no means all that can happen; they represent the common ones.

(1) Initialize variables before using them. Be sure that variables in subroutines and inner loops are properly reset between successive uses. Set constants at compile time and variables at run time. If a debugging compiler is available to check for initialization errors, use it.

(2) Watch for off-by-one errors. Be sure that things are done the right number of times, and that comparison tests branch the right way on equality.

(3) Check that array references do not go out of bounds. Again, if subscript-range checking is available from your compiler, use it.

(4) Avoid multiple exits from a loop. Keep exit tests close together and as near the top as possible.

(5) Test your program at its internal boundaries. This should be done before the program is run, and as a running check. Ask whether each loop might be performed zero times under some circumstances, and if you are writing in Fortran, augment `DO` statements with `IF`'s if they may have to be skipped.

(6) Program defensively. Be aware of the kinds of things that could go wrong, and add code to check for them.

(7) Do not count with floating point numbers. Do not expect fractional floating point values to obey the familiar laws of arithmetic — they do not.