`A*X**2 = 0`, a valid equation with a double root at zero.

`B*X = 0`, which has a single root at zero.

`C = 0`, which is true only when `C` is zero.

Trivial they may be, but all are different and two out of the three are legitimate. Finally, for the case where only `A` is zero, the program prints out two roots, `-C/B` and 0.0, even though the equation has only one root.

A useful way to write a complex program in any language, not just Fortran, is to code it first in a convenient, expressive pseudo-language (typically *not* Fortran or PL/I) and then, when it appears correct, to translate it into the language at hand (Fortran in this case). At the highest level, we might even write something like

```
REPEAT
   read and print coefficients A, B and C
   solve quadratic Ax^2 + Bx + C
```

`REPEAT` implies an indefinitely repeated loop. On each iteration we fetch a new set of coefficients and solve the corresponding quadratic.

The next step is to fill in some details, a process sometimes called "top-down design" or "successive refinement." (We will have more to say about this in Chapter 4.) Solving the quadratic is a multi-way decision, to decide what specific kind of quadratic must be dealt with. Thus the second version is

```
REPEAT
   read and print A, B, C
   IF (A = 0 & B = 0 & C = 0)
      stop
   ELSE IF (A = 0 & B = 0)
      equation is C = 0
   ELSE IF (A = 0)
      only root is -C/B
   ELSE IF (C = 0)
      roots are -B/A and 0
   ELSE
      Realpart = -B/(2A)
      Discrim = B**2 - 4AC
      Imagpart = sqrt(abs(Discrim))/(2A)
      IF (Discrim >= 0)
         roots are Realpart+Imagpart and Realpart-Imagpart
      ELSE
         roots are (Realpart, Imagpart) and (Realpart, -Imagpart)
```

Indeed this is no language in particular, but it is sufficiently precise for our needs, and readily understood. There is no need for more formality. Not only is the pseudo-code readable and precise, but it is sufficiently close to normal programming languages that we can apply principles of programming style to it just as if it were executable. We can even check that it works, in the sense of doing the right things at the right times.

Once we are satisfied, we translate. Since most Fortrans do not allow grouping of statements, let alone a PL/I-like `IF-ELSE`, we must link up the pieces of the `IF-ELSE` structures by `GOTO`'s. At the same time we can fix minor details like output formats and statement labels, and make the variable names more mnemonic.