Use free-form input when possible.

The following READ statement sets five variables:

```
      READ(1,10) KONST1,KONST2,FEED,DIAM,RPM
   10 FORMAT(I6,I6,F5.3,F5.3,F4.0)
```

Suppose you had to use this program periodically (after negotiating to have the input format made more uniform). Without looking, is the third argument the feed rate or the diameter? If you last used the program a week ago, would you remember? When there are many (i.e., more than one or two) arguments or parameters to be provided to a program, let the users specify their parameters by name; that way they have to input only what they want changed from the default, and they don't have to remember any particular order.

Parameters can be read directly by name with the GET DATA statement of PL/I. Input like

```
FEED=27.0    DIAM=3.5    RPM=3600.0
```

lets users give input arguments in an arbitrary order and format, as long as they can remember what the names are. Fortran's NAMELIST feature does much the same thing, although it is nonstandard and its use is clumsy.

If input parameters are supplied by name, you can use default values in a graceful way. If some parameter is normally given a certain value, build that value into the program; then if users do not specify its value, they will get the built-in value "by default." Of course the defaults have to be chosen intelligently, to satisfy some significant fraction of the user population. On output reports, it may be helpful to print the defaulted values as well as the inputs, so the user will know what the program did. (We have also sometimes found it useful to print date and time, to help people identify their output.)

Use self-identifying input. Allow defaults.
Echo both on output.

In our version of the maze program of Chapter 4, you may recall that we wrote a separate function READMAZE to read each new maze, rather than embedding the input code in-line in the main routine. This allowed our main loop to be simply

```
DO WHILE (READMAZE() = YES);
   IF FINDPATH() = YES THEN
      CALL PRINTPATH;
   ELSE
      PUT SKIP(2) LIST ('NO PATH');
END;
```

All of the details of maze input are hidden from the main routine, including coping with end of file and invalid input. If READMAZE returns YES (actually '1'B) it