While we are making uniform formats, consider this excerpt from a program in another text:

```
      READ (5,100) MAX,JLOW,JHIGH
  100 FORMAT (I3,2I4)
       ...
      READ (5,7) N(I)
    7 FORMAT (I4)
       ...
```

Here two (randomly numbered) FORMAT statements are used, because the pointless irregularity in the first means that it cannot be used with the second READ. (Did the programmer use I3 because he "knew" that MAX would never exceed 999? Users now have to learn two formats; when the program grows, they may have to learn a third.) Do it this way:

```
      READ(5,100) MAX, JLOW, JHIGH
  100 FORMAT(3I5)
       ...
      READ(5,100) N(I)
```

The idea is not to save the small space represented by a second FORMAT, but to make life simpler for the user. Even if you prefer to write FORMAT's after each I/O statement (a good practice), make them as similar as possible.

Formats should also be chosen with some thought to the probable device used to create input — usually a keypunch or a terminal. Free-form input is easiest; next best are uniform fields near the left end of the card or line. Imagine typing with this format, even after the missing comma is inserted:

```
GET EDIT(X,ZILCH)(X(10),F(4)X(65),A(1));
```

If possible, numbers should be justified left, not right. (That was the purpose of our floating-to-fixed conversion above.) Spread the data out a little — cramming it into the absolute minimum space makes it too hard for reading by people, who may well have to look at it to find errors.

Make input easy to proofread.

The ideal arrangement for reading numbers, especially for getting programs working quickly, is free-form input, where the data layout is essentially unspecified.

Free-form input is easy in PL/I:

```
GET LIST (A, B, C);
```

reads input until it finds three numbers. These can be on one card or line, or on several, separated by blanks or commas or card/line boundaries.

Not all Fortran implementations allow free-form input, but some, especially for interactive systems, do provide unrestricted input, often as

```
READ A, B, C
```

You must weigh the question of future portability against ease of use right here and now. In this case, our vote goes to ease of use.