comp.lang.c FAQ list • Question 12.20

Q: Why does everyone say not to use `scanf`? What should I use instead?

A: `scanf` has a number of problems--see questions 12.17, 12.18a, and 12.19. Also, its `%s` format has the same problem that `gets()` has (see question 12.23)--it's hard to guarantee that the receiving buffer won't overflow. [footnote]

More generally, `scanf` is designed for relatively structured, formatted input (its name is in fact derived from ``scan formatted''). If you pay attention, it will tell you whether it succeeded or failed, but it can tell you only approximately where it failed, and not at all how or why. You have very little opportunity to do any error recovery.

Yet interactive user input is the least structured input there is. A well-designed user interface will allow for the possibility of the user typing just about anything--not just letters or punctuation when digits were expected, but also more or fewer characters than were expected, or no characters at all (i.e. just the RETURN key), or premature EOF, or anything. It's nearly impossible to deal gracefully with all of these potential problems when using `scanf`; it's far easier to read entire lines (with `fgets` or the like), then interpret them, either using `sscanf` or some other techniques. (Functions like `strtol`, `strtok`, and `atoi` are often useful; see also questions 12.16 and 13.6.) If you do use any `scanf` variant, be sure to check the return value to make sure that the expected number of items were found. Also, if you use `%s`, be sure to guard against buffer overflow.

Note, by the way, that criticisms of `scanf` are not necessarily indictments of `fscanf` and `sscanf`. `scanf` reads from `stdin`, which is usually an interactive keyboard and is therefore the least constrained, leading to the most problems. When a data file has a known format, on the other hand, it may be appropriate to read it with `fscanf`. It's perfectly appropriate to parse strings with `sscanf` (as long as the return value is checked), because it's so easy to regain control, restart the scan, discard the input if it didn't match, etc.

Additional links:

- longer explanation by Chris Torek
- longer explanation by yours truly

References: K&R2 Sec. 7.4 p. 159

Hosted by