Syntax Only a Compiler Could Love

As Kernighan and Ritchie acknowledge, “C is sometimes castigated for the syntax of its declarations” (K&R, 2nd Ed, p. 122). C's declaration syntax is trivial for a compiler (or compiler-writer) to process, but hard for the average programmer. Language designers are only human, and mistakes will be made. For example, the Ada language reference manual gives an ambiguous grammar for Ada in an appendix at the back. Ambiguity is a very undesirable property of a programming language grammar, as it significantly complicates the job of a compiler-writer. But the syntax of C declarations is a truly horrible mess that permeates the *use* of the entire language. It's no exaggeration to say that C is significantly and needlessly complicated because of the awkward manner of combining types.

There are several reasons for C's difficult declaration model. In the late 1960s, when this part of C was designed, "type models" were not a well understood area of programming language theory. The BCPL language (the grandfather of C) was type-poor, having the binary word as its only data type, so C drew on a base that was deficient. And then, there is the C philosophy that the declaration of an object should look like its use. An array of pointers-to-integers is declared by `int *p[3]`; and an integer is referenced or used in an expression by writing `*p[i]`, so the declaration resembles the use. The advantage of this is that the precedence of the various operators in a "declaration" is the same as in a "use". The disadvantage is that operator precedence (with 15 or more levels in the hierarchy, depending on how you count) is another unduly complicated part of C. Programmers have to remember special rules to figure out whether `int *p[3]` is an array of pointers-to-int, or a pointer to an array of ints.

The idea that a declaration should look like a use seems to be original with C, and it hasn't been adopted by any other languages. Then again, it may be that *declaration looks like use* was not quite the splendid idea that it seemed at the time. What's so great about two different things being made to look the same? The folks from Bell Labs acknowledge the criticism, but defend this decision to the death even today. A better idea would have been to declare a pointer as

```
int &p;
```

which at least suggests that p is the address of an integer. This syntax has now been claimed by C++ to indicate a call by reference parameter.

The biggest problem is that you can no longer read a declaration from left to right, as people find most natural. The situation got worse with the introduction of the volatile and const keywords with ANSI C; since these keywords appear only in a declaration (not in a use), there are now fewer cases in which the use of a variable mimics its declaration. Anything that is styled like a declaration but doesn't have an identifier (such as a formal parameter declaration or a cast) looks funny. If you want to cast something to the type of pointer-to-array, you have to express the cast as:

```
char (*j)[20]; /* j is a pointer to an array of 20 char */
j = (char (*)[20]) malloc( 20 );
```

If you leave out the apparently redundant paratheses around the asterisk, it becomes invalid.

The above is from Peter van der Linden’s book: **Expert C Programming,** SunSoft Press, 1994.