

# Re: one-liner for character replacement

---

*Source:* <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2008-06/msg00040.html>

---

- *From:* "James Giles" <jamesgiles@xxxxxxxxxxxxxxxxxxxx>
  - *Date:* Tue, 03 Jun 2008 02:55:20 GMT
- 

glen herrmannsfeldt wrote:

....

PL/I keeps no reserved words, using a free form semicolon terminated statement syntax.

The latter known to be a bad idea (from direct productivity experiments).

PL/I uses () for bracketing delimiters, except for statement groups which use PROC/END, BEGIN/END, or DO/END. It originally only had one case, and has no reserved words (except in the 48 character source character set option).

Again, that kind of statement grouping has long been known to be a bad idea. Well, PL/I gets it half right.

Data types may be the biggest complaint about PL/I. Floating point and fixed point data precision may be specified in bits or digits, as needed. An appropriate type will be supplied. For fixed point, a scaling factor (number of digits after the radix point.) (There is some suggestion that it also specifies the base used in arithmetic. This isn't so obvious as it seems.)

Yeah, I wouldn't recommend anyone immitate PL/I's types or it's syntax, One \*supporter\* of PL/I recommended that you should never use the divide operator (/) on fixed-point types: there were too many "gotcha's". Instead, he recommended that people should use the DIVIDE intrinsic (which has 4 arguments). Sounds real natural to me.

4) Modules or COMMON? [...]

## Re: one–liner for characater replacement

PL/I uses `STATIC EXTERNAL` structures, somewhat similar to the way they work in C.

Too prolific in spreading things around the global namespace. Using either `COMMON` or modules only adds the name of the block or the module to the global namespace – not the name of every object you decide to share.

Internal procedures complicate separate compilation, so I don't believe they should be the only option.

A language intended only for small programs probably doesn't need to worry about separate compilation. In any case, I already excluded internal procedures. Module procedures require dependent (but still separate) compilation (assuming the implementation is a compiler). Either that, or you need the modules to separate their definition part from their implementation part (kind of an unnecessary complication for something that's supposed to be a simple language).

[...] Having `COMPLEX` as a predefined structure data type is an interesting idea. PL/I uses it as an attribute to any numeric data type. That is, fixed point or floating point data is either `REAL` or `COMPLEX`.

Another thing I wouldn't recommend even for a complex language, much less a simple one.

9) Operator precedence? C/C++ have 15 (or more, depends on how you count). Fortran has 10 (or more, depends on how you count). Pascal has 4. I think there should be 6 or 8 depending on whether you allow user defined operators or not. User defined operators? I could be convinced either way.

PL/I seems to have seven:

Yeah, right after I posted the last article, I realized that 7 are needed (or nine – if user defined operators are permitted):

1) user defined prefix operators

Re: one–liner for characater replacement

## Re: one-liner for character replacement

- 2) exponentiation (I prefer to spell it ^)
- 3) multiply, divide (\*, /)
- 4) unary + or -, infix + or -
- 5) concatenate (//)
- 6) relational operators (oh, you know the usual ones)
- 7) logical not ( $\neg$  is fine)
- 8) logical and ( $\wedge$ ), logical or ( $\vee$ ), logical xor ( $\>\<$ )
- 9) user defined infix operators.

- 1) \*\*, prefix +, prefix -, not ( $\neg$ )

This doesn't match normal mathematical usage (where unary signs are lower precedence than exponentiation). Further, all logical operators really should be lower than the relational operators.

- 6) boolean AND (&)
- 7) boolean OR (|)

Dijkstra recommended that those operators have the same precedence and that parenthesis should always be used to emphasize order (that is, the language processor should insist on it). I have come to agree.

The four operators at the highest level seem a little unusual, as languages go. PL/I will convert data where needed, such that some combinations not allowed in other languages are allowed in PL/I.

Probably another bad idea. Too many things that are often (maybe even usually) errors cannot be detected or reported by the compiler.

And of course there's the lexical issues: should user defined operators be delimited by periods or something else? If something else, should period be used as the component selector for derived types? Should enumerated types be permitted? If so, shouldn't LOGICAL be a built-in enumeration? Should enumeration literals be spelled distinctively? Should \*they\* be spelled delimited by periods (like .true. and .false.)? Etc.

....

One that I don't see mentioned is I/O. That may or may not be important for any particular problem.

Re: one-liner for character replacement

And I left out whole-array operations. Are they simple enough?  
As I mentioned before, I probably left out a whole lot of important  
stuff.

—

J. Giles

"I conclude that there are two ways of constructing a software  
design: One way is to make it so simple that there are obviously  
no deficiencies and the other way is to make it so complicated  
that there are no obvious deficiencies." — C. A. R. Hoare

.