

Re: Derived type argument to subroutine

Source: <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2008-04/msg00675.html>

- *From:* Dick Hendrickson <dick.hendrickson@xxxxxxx>
 - *Date:* Thu, 17 Apr 2008 12:38:41 GMT
-

Arno wrote:

1. The standard does not specify that *ANYTHING* is passed by reference. In practice, they often, but not always are.

I also thought that in Fortran the default was passing arguments by reference and that passing by value has to be explicitly stated. I also thought that an argument changed in a subroutine is changed in the caller as well. I make pretty often use of that, so I really hope I can rely on the fact that arguments are passed by reference.

See for instance below:

```
program test

integer :: i
i = 2

call mycalc(i)

write(*,*) i

end program

subroutine mycalc(val)

integer :: val

val = val*2

end subroutine
```

Would the printed value always be 4, or is it undefined as the passing of arguments can be either by reference or value, dependent on compiler, platform etc.!?

Arno

Re: Derived type argument to subroutine

It's always 4. That's a requirement of the language, but it does NOT say anything about how things are "passed". In fact, the standard specifies nothing about the argument passing mechanism. The standard merely says that an association is established between the dummy argument (val) and the actual argument (i). Once val is associated with i, anything that happens to val also happens to i. With an important exception (see below), there is nothing a standard conforming program can do to tell what argument passing method is used.

Two common ways of doing "associating" them generates code that looks like "pass by reference" or "copy-in/copy-out". But it's really neither method. The reason the distinction is important is that Fortran has a set of anti-aliasing rules that basically prevent hidden associations between dummy arguments. This allows a large number of optimizations, such as keeping variables in registers for long times (and doesn't require synchronization points, as in C). It also allows subprograms to be inlined and then fully optimized.

The exception to the above involving arguments with the pointer or target attribute. The exceptions are "natural" in the sense that if you are using pointers, you are, obviously, doing something with addresses and there needs to be some rules about how the compiler passes things. But, even for these, the rules don't specify the exact mechanism. They merely impose so many restrictions that "pass by reference" is the only practical option.

Dick Hendrickson

.