

## Re: qsort and arbitrary types

**Source:** <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2004-06/0410.html>

---

**From:** Dick Hendrickson (*dick.hendrickson\_at\_att.net*)

**Date:** 06/10/04

Date: Thu, 10 Jun 2004 16:30:33 GMT

Janne Blomqvist wrote:

> *In article <m1smd41m52.fsf@macfortran.local>, Richard Maine wrote:*

>

> *[pass by reference...]*

>

>

>> *With newer features and compilers, the exceptions are becoming less  
>>and less exceptional. Some new features are pretty much impossible to  
>>implement with pass by reference. Even for old features, things like  
>>copy-in/copy-out are sometimes used by modern optimizers.*

>

>

> *You are of course correct. Although I perhaps didn't express myself  
> very clearly, what I meant was that semantically, argument passing  
> behaves like pass by reference in the sense that modifying an argument  
> in a procedure also modifies it in the caller. Obviously, if an  
> argument is marked as INTENT(IN) or VALUE that isn't true, but I was  
> talking about the "general" case.*

>

> *Whether the argument passing is actually implemented as pass by  
> reference, pass by value-result or in some other way is implementation  
> dependent, as you say.*

>

>

>> *I recommend against programming based on the assumption that arguments  
>>will be passed by reference. It certainly doesn't seem like an  
>>assumption that will leave you well prepared for the future... or  
>>even some parts of the present.*

>

>

> *Hmm., are there actually legal programs which work differently  
> depending on if pass by reference is used or not? I can think of a few  
> examples where pass by reference would differ from pass by  
> value-result, but those examples would require aliasing which isn't  
> allowed anyway.*

No, there aren't! The standard attempts to guarantee that

a legal program can't tell how argument passing really works.

Your last statement is somewhat misleading. Aliasing is allowed, but any variables which are aliased can't be redefined in the called routines. Your statement implies that something like

```
call sub (x,x)
is illegal. It's only illegal if sub defines either of
it's arguments.
```

Also, the idea of "pass by reference" is a misleading concept for Fortran if you mean that an address is passed into the routine. If you take a macro look at code for something like

```
subroutine add(x,s,n)
  real x(n,n)
  do i = 2,n
    x(i,n) = x(i, n-1) + s
  enddo
end
```

you'd "conclude" that s and n were passed in "by value", using a pretty inefficient way to get the values into a register for the useful life of the subroutine. Once the compiler generates a few indirect loads, s and n are treated "just like" traditional pass by value arguments.

The Fortran rules were designed to allow the compiler to keep variables in registers, whereas, traditional pass by value generally requires the compiler to do stores and loads whenever an argument is changed.

Dick Hendrickson

>