

Re: Q: Checking the size of a non-allocated array?

Source: <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2007-08/msg00315.html>

- *From:* nospam@xxxxxxxxxxxxxx (Richard Maine)
 - *Date:* Fri, 10 Aug 2007 09:42:38 -0700
-

James Tursa <aklassyguy@xxxxxxxxxxxxxx> wrote:

On Thu, 9 Aug 2007 21:56:21 -0700, nospam@xxxxxxxxxxxxxx (Richard Maine) wrote:

passing an unallocated allocatable as
an actual argument is already invalid

The subroutine in question is intended to be part of a library that other users will call. i.e., I don't have control over what they may write so I am trying to be as nice as possible and catch their errors

Oh. Then you are pretty much out of luck in terms of anything standard or portable. I've been there. Not quite this case, but other ones where I needed to try to catch errors of other users. In one case, some users could write subroutines that were used with my program. Yet other users would use the combined package. Try as I might, I just could not get some people to code with adequate robustness. They would do things like take square roots of values that might be negative. Sure the physical quantity modeled couldn't be negative, but imperfect instrumentation could and did sometimes end up with negative values.

I ended up writing an exception handler to catch this stuff (and using nonstandard C interop tricks to install it). I couldn't really count on being able to recover and continue. But at least I could get out a message saying what kind of error had occurred and where. Without that, all the final user saw was that he had gotten disconnected from the server. This was a client-server application (using other C interop tricks), with the user not even having an account on the server machine. My exception handler and message at least directed the users to the right person. Before I put that in, the questions all came to me.

Re: Q: Checking the size of a non-allocated array?

The worst outcome of this construct would seem to be some type of run-time error on the call to `size()`, but they would have gotten a run-time error anyway downstream...

or quite likely upstream. Recall that it is the call in the first place that is invalid. There is at least a chance of getting a run-time error on the call.

As a side question, is there some philosophical reason why `size(x,1)` for unallocated `x` shouldn't be allowed to simply return 0?

First note that you don't have an unallocated array in the subroutine. It is the actual argument that is unallocated. As mentioned above and before, the problem is in the subroutine call in the first place – not in the use of the `size` intrinsic. Yes, there are huge philosophical reasons why the standard doesn't define what happens afterwards when you have already done something nonstandard beforehand. But I assume that's not what you are really asking. If so, that's too big and basic a matter for me to cover in finite space.

But supposing you did have an unallocated allocatable, insisting that `size` of it return 0 is much like insisting that any other undefined expression return 0. If you expect the `size` to return zero, then how about just `x(42)`. Why not also ask that to return zero?

Also, it is comparable to disassociated or undefined pointers. In that case, there would be serious implementation problems (and likely performance penalties) from insisting that it always be valid to ask about the size of any such pointer array. That would pretty much amount to getting rid of the concept of undefined association status and forcing the compiler to keep track of all cases where pointers currently go undefined. Otherwise, you end up with pointers that "look" valid, but point to space subsequently reused for other things. While it is probably possible in principle, that is a lot bigger implementation mess than intended.

2nd side question related to the validity of putting an unallocated array in a call to a routine that does not have the allocatable attribute for the associated dummy argument. Obviously the compiler has to allow it since the compiler doesn't know at compile time whether or not the array will be allocated at run-time. Is there some philosophical reason for not allowing this in the standard (beyond the fact that `size()` is not defined)?

Again, you keep bringing up `size`. This has nothing to do with the `size`

Re: Q: Checking the size of a non-allocated array?

Re: Q: Checking the size of a non-allocated array?

intrinsic. It is the call that is invalid. And I wouldn't say that the compiler has to "allow it". More a matter that the compiler might not detect the error. As to the "philosophy" of it, I would say that you are concentrating more on the implementation details than on the abstractions. The abstraction is that if the variable is unallocated, there is no array to pass. That model can translate into implementation matters. Imagine copy-in/copy-out schemes.

Note, by the way, that there are applications that make useful distinction between an array being unallocated versus being allocated with size zero. In one case, you don't (yet?) know how big it is (will be?). In the other case, you have established the size and know that it is zero.

--

Richard Maine | Good judgement comes from experience;
email: last name at domain . net | experience comes from bad judgement.
domain: summertriangle | -- Mark Twain

.