Doctor Fortran in "I've Come Here For An Argument"

Submitted by Steve Lionel (Intel) (https://software.intel.com/en-us/user/512685) on March 31, 2009

Last updated on January 1, 2015

One of the most fundamental aspects of Fortran programming is passing arguments to procedures. It is also one of the most misunderstood aspects. In this space today I'll try to make things clearer.

First, some terminology. In Fortran, there are "actual arguments" and "dummy arguments". An actual argument is what you put inside the parentheses in a call to a procedure. This can be a variable, a named (PARAMETER) constant, a literal (such as 3), a procedure or an expression. "variable" here means more than just a variable name. It could be an array reference, a derived type component reference, a substring reference or a combination of any or all of these. If you can put it on the left side of an assignment, it's a variable.

A dummy argument is the corresponding element in the argument list of the called procedure. Some languages call these "formal arguments". When you call a procedure, each dummy argument becomes "associated" with its corresponding actual argument, if any. (OPTIONAL dummy arguments do not get associated with omitted actual arguments.) The rules for argument association are complex and have many special cases, and that's what I'll spend most of this post going over.

Argument Association

On a fundamental level, argument association is pretty simple. Although most people would tell you that Fortran uses "pass by reference", meaning that the address of the actual argument is used for the dummy argument, that's not quite true. The standard says that argument association is "usually similar to call by reference" and then adds "The actual mechanism by which this happens is determined by the processor." [Fortran 2003 Note 12.22]

There are some cases where what actually gets passed is not the original argument. The most common case for this is when a non-contiguous array pointer or a array slice is passed to an argument for which the compiler does not see an explicit interface specifying that the corresponding dummy argument is an assumed-shape array. Because the called routine is expecting a contiguous array, the compiler needs to make a copy of the actual argument, pass the copy, and then copy back any changes. The compiler can warn you about this at run-time if you use the option /check:arg_temp_created (Windows) or -check arg_temp_created (Linux/Mac OS). The compiler will generate a run-time test to see if the argument is actually contiguous and skip the copy if it is.

This would probably be a good time to mention INTENT. If the compiler sees that the dummy argument is INTENT(OUT), it doesn't need to make the copy going in, and if it is INTENT(IN), it can skip the "copy out". Explicit interfaces and explicit INTENT are always a good idea.

Another case where a copy is made is if the argument has the VALUE attribute. By this I mean the Fortran 2003 VALUE attribute, not the one on a !DEC$ ATTRIBUTES directive. In this case, the copy is made by the called routine on entry into a temporary, and any changes discarded on exit.

Sequence Association

A lot of Fortran programs, especially older ones, rely on sequence association. What is it? An actual argument that is an array element, an array expression or a character scalar (of default or C_CHAR kind) represents a sequence of elements. This sequence starts with the first element or character passed and continues to the end of the array or last character. The corresponding dummy argument, which need not have the same number of dimensions (rank) as the actual argument, or the same character length, is associated with this sequence.

Let's take a simple example common to older code.

```
real a(10)
call sub(a(3))
...
subroutine sub (b)
real b(8)
```

When sub is called, b(1) will be associated with a(3), b(2) with a(4) and so on up through b(8) being associated with a(10). Usually, the programmer would declare b as assumed-size with a dimension of (*), in which case the implied extent of b is 8 because that's the number of elements remaining in array a.

What happens if you declare b to be fewer or more than 8 elements? Fewer is fine, but more is not. The compiler may or may not detect this error.

There is a significant restriction regarding sequence association which more and more people are running into, especially as they "update" old code to include modern Fortran features such as pointers. You don't get sequence association if the actual argument is assumed-shape or is an array with a "vector subscript". Why? Because the argument may not be contiguous and thus there can't be an association with a sequence of elements.

Here's where this gets most people into trouble... In general, a Fortran rule says that "if the actual argument is scalar, the corresponding dummy argument shall be scalar, unless the actual argument is of type default character, of type character with the C character kind, or is an element or substring of an element of an array that is not an assumed-shape or pointer array." [Fortran 2003 12.4.1.2]

If in the example above, array a was an assumed-shape array (might be a POINTER or ALLOCATABLE or a dummy argument itself) and you passed a(3), the compiler would give you error #6585 quoting the text from the standard shown in the previous paragraph. The workaround for an assumed-shape array would be to pass an array slice, but you can't then depend on sequence association and the dummy argument has to be no larger than the slice you passed.

You'll also get this error (as error #8299) if you are passing a scalar that isn't an array element, such as a constant 1 or a scalar variable, to an array dummy argument. In some cases you can use an array constructor to turn the scalar into an array, such as [1], but you will lose sequence association benefits.

Now, what's the deal with that exception for character types when the dummy is an array of default character kind or C kind? This lets you pass a normal character value or variable to a dummy argument declared as an array of single characters, as this is the official "interoperable" mapping for C arguments declared as "char*". Note that you'll have to supply any trailing NUL yourself - Fortran won't do it for you.

Rules, Rules, Rules

Here are just a few of the other language rules that must be followed when passing arguments. You can read them all in section 12.4.1.2 of the Fortran 2003 standard (http://j3-fortran.org/doc/2003_Committee_Draft/04-007.pdf).

- The types of the actual and dummy arguments must be "type compatible", Typically this means the same type, though things start to get fuzzy when the Fortran 2003 feature of "polymorphism" comes into play (coming soon to Intel Fortran.)
- If the actual and dummy are character scalars, the length parameter of the actual must be greater than or equal to the length parameter of the dummy, if it is not passed-length.
- If the dummy is a pointer, the actual must be too. Same goes for allocatable.

The compiler can help a lot with making sure the rules are followed if you are using explicit interfaces (modules, contained procedures or INTERFACE blocks.) Otherwise, use the /warn:interface and /gen-interface (Windows), -warn interface and -gen-interface (Linux, Mac OS) options to have the compiler do interface checking for all your Fortran routines. On Windows, these options are enabled by default in new Visual Studio projects.

That's all for this round. If you have comments on this post, please add them below. If you are looking for tech support, please post in our user forums (/en-us/forums/).

For more complete information about compiler optimizations, see our Optimization Notice (/en-us/articles/optimization-notice#opt-en).

Comments (14)

Top

Anonymous said on Mon, 03/12/2012 - 13:12

Hi, Steve!

I´m trying to convert what I think was an F66 code to run using the Intel Visual Fortran compiler.

in the f66 code there is no declaration of array "A". in some parts of code, "A" is used to pass character data. in other parts (subroutines) is used to pass "single precision" numerical data.

that treatment should work in IVF? or I'll have to re-make the code to fix it?

many thanks in advance!

Frank

Anonymous said on Tue, 03/09/2010 - 14:36

How does one pass a 2 dimensional array to the older LAPACK routines that use 1 dimensional dummy arguments in the subroutine & assume the 2D arrays are stored column wise?

Steve Lionel (Intel) said on Fri, 10/23/2009 - 09:52

That's not the declaration - those are uses. But it does suggest the solution. Replace INT(0,4) with [INT(0,4)] or, if you want to be consistent with the others, (/INT(0,4)/) which means the same thing - make an array out of the value.

carl_caves said on Fri, 10/23/2009 - 09:04

Steve, the declaration of writeone_ik4 is as follows:

cal writeone_ik4_ (ds_Xtrack_idx,ipix, (/gome2_curr_ixtrack /))

cal writeone_ik4_ (ds_iscan ipix, (/curr_iscan /))

cal writeone_ik4_ (ds_Xtrack_dim, iscan, (/nxtrack_per_scan(gome_curscan /))
cal writeone_ik4_ (ds_utc_year, iscan, (/ gome2_utc_year /))
cal writeone_ik4_ (ds_utc_doy, iscan, (/ jday /))

Any suggestion?

Thank you.

Steve Lionel (Intel) said on Fri, 10/23/2009 - 06:15

I'd need to see the declaration of writeone_ik4 to give you specific advice. There is no compiler option to remove the error, since it is clear that there is an explicit interface visible for writeone_ik4.

This error message is telling you that the VAL dummy argument to writeone_ik4 is an array, but that you have passed a scalar to it, which is not legal. The solution might be as simple as enclosing the actual argument in [] brackets to make it an array, but this works only if the routine does not want to write to the argument.

If you need more help, please ask in our user forum (http://software.intel.com/en-us/forums/intel-fortran-compiler-for-linux-and-mac-os-x/)

carl_caves said on Fri, 10/23/2009 - 04:28

I am trying to compile a code and I get this error:

error #8299: A scalar actual argument must be passed to a scalar dummy argument unless the actual argument is of type character or is an element of an array that is neither assumed shape nor pointer. [VAL]
CALL writeone_ik4(ds_f_saa, ipix, INT(0, 4))
----------^

I haven't written this code, so it is very difficult to modify it. Is there any compilation option in order to solve this problem.

The compilation option that I am using is:
ifort -c -O3 -fPIC -static

The Intel fortran compiler version is: Intel(R) Fortran Compiler Professional for applications running on IA-32, Version 11.1 Build 20090630 Package ID: l_cprof_p_11.1.046

Does anyone have any suggestion to solve this problem?

Thank you in advanced.

Steve Lionel (Intel) said on Mon, 09/14/2009 - 11:11

krodberg, your code snippet has two errors which the compiler, more or less, correctly notes. The first one is that you pass TEXT(3) to an argument declared as CHARACTER*16. With sequence association, TEXT(3) is only 8 characters, comprised of TEXT(3) and TEXT(4), but you pass it to an argument declared CHARACTER*16 and this is not allowed. That said, I know that the compiler doesn't check this sort of thing properly and even if TEXT had more elements to satisfy the standard, you might still get the error. This will be fixed in a future update.

The second error is very much what is described in this article - POINTER arrays do not participate in sequence association, so it's not allowed to pass a pointer array element to an argument that is an array.

That said, you can likely eliminate both of these error messages by turning off generated interface checking, which is the default on Windows in the Visual Studio environment. The property Fortran > Diagnostics > Check Routine Interfaces should be set to No. This won't "fix" your incorrect program, but it will prevent the compiler from noticing the errors. You may also need to turn off string and array bounds checking. Of course, this also means that you may get wrong results if the code improperly accesses the arguments.

If you need more help, please ask in our support forum at http://software.intel.com/en-us/forums/intel-visual-fortran-compiler-for-windows/

krodberg said on Mon, 09/14/2009 - 10:59

I just started using the Intel Visual Fortran 11 compiler and ran into what I think is a similar problem to what is defined above. I am trying to compile some code from the USGS which I have compiled with CVF yet I get an error with IVF. I need IVF to compile Modflow as a DLL for an OpenMI project.

I've clipped out a small snip-it and set it up to compile in CVF and show the error I get in IVF. Is there a way to compile it in Intel without revising the USGS code?

```
module a
REAL, SAVE, DIMENSION(:,:,:), POINTER ::SUB
```

Rate Us ☆☆☆

Look for us on: English

```
program test
use a, only: sub
character*4 text(4)
integer iswocf(3)

KQ = 1
IF(ISWOCF(3).GE.0) CALL ULAPRW(SUB(1,1,KQ),TEXT(3),KSTP,KPER,NCOL,NROW,K,ISWOCF(3),IOUT)

end

SUBROUTINE ULAPRW(BUF,TEXT,KSTP,KPER,NCOL,NROW,ILAY,IPRN,IOUT)
CHARACTER*16 TEXT
DIMENSION BUF(NCOL,NROW)
return
end
```

Error 1 error #7938: Character length argument mismatch. [TEXT]
Error 2 error #7836: If the actual argument is scalar, the corresponding dummy argument shall be scalar unless the actual argument is an element of an array that is not an assumed-shape or pointer array, or a substring of such an element. [BUF]
Error 3 Compilation Aborted (code 1)

Steve Lionel (Intel) said on Wed, 05/20/2009 - 04:47

ifort.exe is one of the files of the Intel Visual Fortran for Windows product - it's the executable used to invoke the compiler. For other software to invoke the compiler, a number of environment variables have to be set properly. This is best done by starting a command prompt session using the "Fortran build environment" shortcut provided in the Start menu for Intel Visual Fortran. It can be done from another script by calling the "ifortvars.bat" file, whose location is given by "%IFORT_COMPILER11%bin". It takes an argument of "IA32", "INTEL64" or "IA64" to specify the target platform. Once this is done, the "ifort" command will call the compiler (and linker if required.)

For Linux, the Intel Fortran Compiler for Linux also provides an "ifort" for a similar purpose. For more information, read the compiler documentation on use from the command line.

shafaatihawaii.edu said on Tue, 05/19/2009 - 16:32

is ifort a file for windows or linux?if i just install intel visual fortran is it enough for other softwares using fortran link properly?where can i find ifort.exe

Add a Comment

^Top

(For technical discussions visit our developer forums. For site or software product issues contact support.)

Please sign in to add a comment. Not a member? Join today >

Terms of Use *Trademarks Privacy Cookies Publications >

Rate Us ☆☆☆

Look for us on: English >