Proper use of modules in Fortran

Asked 12 years, 4 months ago Modified 2 years ago Viewed 6k times

6

I work with FORTRAN a lot, but I never had formal instruction in the proper way to write source code. I currently use modules to store global variables, but I understand you could also use them to store subroutines and functions. The codes I work with have many subroutines, as they are very large and complex. Should all functions and subroutines be in modules? If so, why?

function module fortran subroutine

Share Follow

asked Aug 14, 2012 at 13:14
rks171
1,020 ● 13 ● 27

Very good discussion and helpful. I'll keep these things in mind as I go forward in structuring my programs. – rks171 Aug 15, 2012 at 0:30

1 agreed, although your question is broad, it serves as a great catalyst for the well dictated responses below. +1 – JonnyB Sep 11, 2014 at 17:24

4 Answers

Sorted by: Highest score (default)

23

In general the answer to your first question is *Yes* and I'll come to an answer to your second question in a moment. Note first that this is a general answer to a general question and the bright sparks who hang around SO Fortran questions may well come up with special circumstances in which modules are inapplicable. I retort, in advance, that this answer is aimed at a newcomer to modules. Once you are no longer a newcomer you can formulate your own answer to your questions.

Modules are most useful to the programmer as an aid to organising and structuring a program or suite of programs. They provide a mechanism for encapsulating the definitions of user-defined types and functions/subroutines which operate on those types. In Fortran 90 and 95 this encapsulation was somewhat ad-hoc in the sense that it relied on the programmer's ideas about how to decompose a program into parts. With the introduction of the object-oriented facilities in Fortran 2003 there are now even clearer 'rules' for identifying what elements belong in each module.

You could, for example, conceive of a module for types and procedures for rational arithmetic. By keeping all the code which implements your great ideas in one module you can hide the implementation from other parts of your program (which do not need to know the details) and expose only those parts you wish to expose (look at the `PRIVATE` and `PUBLIC` keywords). You can, right away, see another advantage to organising your code into modules; it's much easier to `USE` your rational arithmetic module in a new program than it is to cut and past the code from your mega-source file into another mega-source file. When you want to work on your rational arithmetic, you work on code in one module, not in code spread all around your files.

Modules also allow you to manage name clashes. For example, your rational arithmetic module might define an operation called `add`, and you might also have a multiple-precision integer arithmetic module which defines an operation called `add`. If you attempt to `USE` both these modules in a program (or another module) then compiler will warn (possibly raise an error) that the same name is defined twice within the scope that uses the modules. You can use renaming when you use associate module entities. You can also use an `ONLY` clause to import only those module entities that the user needs.

Note that module `USE` is transitive, if A uses B and B uses C you don't have to also declare that A uses C (though if you've renamed entities or specified `ONLY` clauses you'll have to make sure what is transitive in a particular case).

In a nutshell, modules are the principal Fortran mechanism for dealing with complexity in programs by breaking them into manageable chunks. Fortran 2008, when the feature is implemented by compilers, introduces `SUBMODULE`s too, which promise even better support for dealing with complexity in this way.

Modules are also useful in that the language standards require that compilers generate explicit interfaces to procedures defined in modules for type-checking against the arguments at compile time. Note that these interfaces (which you never really see) are called explicit to contrast with implicit interfaces which is what procedures which are not defined inside modules (or `CONTAIN`ed within a program unit which uses them) have. You can, of course, write explicit interfaces for such procedures, but it's almost always easier in the short and long runs to let the compiler do it for you.

As @Telgin has already noted, modules are an aid to incremental compilation too.

Share Follow

answered Aug 14, 2012 at 13:37

High Performance Mark
78.3k ● 7 ● 109 ● 168

1 I don't see how modules help with incremental compilation. In fact, (as far as I'm concerned) compilation headaches are the best argument against using modules. (I'm not advocating that we get rid of modules just because they're harder to compile -- I just don't see how they help at all). – mgilson Aug 14, 2012 at 13:54

Generally if I have already compiled a module and the `.o` and `.mod` files are present when sought for linking and have the right timestamps then my Makefiles don't recompile them. I'm aware, mind you, of the *Fortran compilation cascade* which ensues when a module is modified though its interface remains unchanged. – High Performance Mark Aug 14, 2012 at 14:09

1 But if you already have a compiled `.o` present with the right timestamp, that works too (independent of modules). The problem if modules is that they impose an order of compilation that isn't there otherwise, and that compiling a module generates 2 files which is a situation that (sadly) `make` seems to have entirely forgotten. But again, I think this is a small price to pay for the power and safety you gain by using modules. – mgilson Aug 14, 2012 at 14:12

1 Ahh, I see what you mean. I can't say I see the ordering of compilation as a problem, Fortran requires this in other contexts too (eg not using a type before it's defined within a module). But perhaps I've been working with Fortran too long. I'm well aware that a lot of 'features' that modern software engineers regard as 'issues' are just part of the landscape to me. – High Performance Mark Aug 14, 2012 at 14:16

6

One of the major benefits of using modules is that your compiler will automatically perform interface checking on any functions or subroutines you `use` from a module to ensure that your are calling the routine with the appropriate parameter types. A good article on this topic is Doctor Fortran Gets Explicit - Again!. From this article:

> There are several ways to provide an explicit interface. The simplest and best way is to put the procedure in a module, or make it a CONTAINed procedure of the calling program or procedure. This has the advantage of not requiring you to write the information twice and thus increasing the chances of getting it wrong in one of the

> places. When you have a module procedure, or a contained procedure, its interface is automatically visible to everything else in the module or in the parent scope. Assuming the name hasn't been declared PRIVATE, the interface is also available to places where you USE a module containing a module procedure.

I would recommend putting all *related* routines in the same module. Modules are, to me, the equivalent of classes in other languages. Modules are a way to group related data and routines (which may operate on that data). So modules offer a way of making your code easier to navigate, if your routines are grouped logically into modules, and add type checking to your function and subroutine calls.

Share Follow

edited Dec 18, 2022 at 1:23
Zade Johnston
103 ● 1 ● 5

answered Aug 14, 2012 at 13:37
Chris
46.2k ● 16 ● 139 ● 160

3

Modules help the Fortran program by automatically providing the explicit interfaces, as already described in some of the other answers. This allows the compiler to check for consistency between arguments in procedure calls and the procedure declarations, which catches a lot of mistakes -- here are some Stackoverflow answer that show this benefit of modules: Computing the cross product of two vectors in Fortran 90, FORTRAN functions and Fortran segmentation fault in pointer array

Share Follow

edited May 23, 2017 at 12:32
Community Bot
1 ● 1

answered Aug 14, 2012 at 15:13
M. S. B.
29.4k ● 2 ● 51 ● 75

2

Moving related functions, subroutines and variables to their own modules improves the maintainability of a program. Later on when you have to update part of it, you don't have to go digging through a single source file with tens (or hundreds) of thousands of lines of code and can just open up the relevant files.

Of course, you're probably thinking that you can just use your editor's search function to find the relevant subroutine, but after a few months of letting the code sit alone, you'll probably quickly find that you can't quite recall the name of subroutines or how they fit together. Having them grouped nicely helps a lot with that.

Moving subroutines and functions into a module also improves compilation speed if the compiler only has to rebuild a single module instead of the entire program.

Share Follow

edited Aug 14, 2012 at 13:48

answered Aug 14, 2012 at 13:19

Telgin

1,604 ● 10 ● 10