Subject: Modules need initialization parts
From: Van Snyder
Reference: 97-114r2, section 24 (pages 35-36).

Number

TBD

Title

Modules need initialization parts.

Submitted By

J3

Status

For consideration.

Basic Functionality

Provide for an *initialization-part* that consists of an *execution-part* and perhaps some more syntax, somewhere in a module, that is specified to be executed exactly once before any procedure within the module is executed, or before any part (including an initialization part) of a program unit that accesses it by use association is executed.

Rationale

There are three reasons to do this: convenience, clarity and safety. Convenient because the initialization gets done without user code needing to invoke it, and without the initialization part needing to have an explicit "first time flag" to prevent executing it twice. Clear because it puts initialization in a consistent place, specified by the standard. Safe because it guarantees the initialization gets done without needing to depend on scoping units that access the module to invoke the initialization.

Estimated Impact

Minor.

Detailed Specification

Provide for an *initialization-part* that consists of an *execution-part* and perhaps some more syntax, somewhere in a module, that is specified to be executed exactly once before any procedure within the module is executed, or before any part (including an initialization part) of a program unit that accesses it by use association is executed.

One syntax to do this is to add [*execution-part*] in R1104, giving

R1104 *module* **is** *module-stmt*
[*specification-part*]
[*execution-part*]
[*module-subprogram-part*]
end-module-stmt

This is the way that Ada and Modula-2 work, and the way a Fortran main program works (with *module-subprogram-part* replaced by *internal-subprogram-part*, which has identical syntax).

No matter what syntax is used, it will be necessary to add a requirement that the initialization part shall be executed no more than once before any procedure within the module is executed, or before any

part (including an initialization part) of a program unit that accesses it by use association is executed. Thus if A uses B the initialization part for B is executed before the one for A, which is executed before (perhaps long before) any procedure in A. It can be processor dependent whether an initialization part is not executed if no *execution-part* in a scoping unit that accesses the module is executed.

"Exactly once" is preferable to executing it again if the module goes "out of scope" and comes back, or to leaving this up to the processor. It's easier to describe, probably easier to implement, and consistent with SAVE.

Ada and Modula-2 both have initialization parts for their equivalents of Fortran's modules. Since they are both widely implemented, it's clear it's possible to do this. Surely Fortran processor developers are at least as clever as Modula-2 and Ada processor developers!

Here is a possible implementation. The main program, each external procedure, and each initialization part have, in effect (but maybe not in the details of implementation):

```
logical, save :: FIRST = .TRUE.
if ( first ) then
  first = .false.
  call initializer_for_first_accessed_module
  call initializer_for_second_accessed_module
  ....
  ! In a module, execute the initialization part's execution part.
  ....
end if
```

Each interoperable module procedure with a binding label has:

```
logical, save :: FIRST = .TRUE.
if ( first ) then
  first = .false.
  call initializer_for_the_module
end if
```

In some cases this could be done more efficiently by putting a GOTO instruction to the initialization part into the "data bank" of each module, which instruction is changed to a RETURN instruction by the initialization part, and similarly in each external procedure and interoperable module procedure that has a binding label. This isn't as efficient as the ETH-Zürich method described below, but it's not terribly inefficient, either.

The ETH-Zürich Modula-2 processor determines an order to execute the initialization parts by doing a depth-first traversal of the dependency DAG. It inserts a CALL to the first initialization part before the first executable statement of the main program. At the end of each initialization part but the last one it inserts a GOTO the next one in the list. At the end of the last one, it inserts a RETURN. There are no other calls or "first time" flags. This method may need cooperation from the linker or an auxiliary processor.

Different implementations could do it different ways; the standard should not specify how it's done, but an example in Annex C may be useful.

No matter how initialization is done, the standard should specify the name of a C function that the processor provides, and that can be invoked to do the initialization in the event the main program is not a Fortran main program unit. This is necessary for the ETH-Zürich method, and could be empty for the other two methods described above.

History