Subject: Module initialization
From: Van Snyder

1 Introduction

Richard Maine has described an application that can be realized most elegantly by automatic initialization of modules and submodules. It can be realized without automatic initialization, but this increases both maintenance and development costs.

To outline one small part of the problem that illustrates the utility of automatic initialization, consider a program that will be augmented by many users to support their needs. Their augmentation will consist of adding new input routines, among other things; they should not need to change the majority of the program.

Without automatic initialization, they would need to modify some part of the existing program in order to add something to a data structure to provide access to the new procedures.

If a facility of automatic initialization is provided, the user can provide an additional module, extend a type, and use the initializer to link an object of the extended type into a list. The main part of the program can then access the new functionality, without any modifications having been made to the main part of the program.

The augmentation is a little bit cleaner with submodule initializers (because the "add this to your list" routine need not be public), but submodule initializers allow subversion of the privacy of data in an ancestor module. More severe restrictions than those on pure procedures would be required to prevent this.

2 Proposition

Allow executable statements after the *specification-part* of a module and before the *module-subprogram-part*. These statements are called the **module initializer**. This is analogous to where executable statements are allowed in the main program or a subprogram.

The initializers of all modules upon which a program unit depends by use association shall be executed before any *executable-construct* in the *execution-part* of that program unit is executed, or any specification expression in that program unit is evaluated. An initializer shall not be executed more than once.

3 Edits

Edits refer to 02-007r3. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[*execution-part*]	9:32+
[Editor: "contains" ⇒ "may contain"; "accesses" ⇒ "access".]	13:14
[Editor: "units. These" ⇒ "units, and may contain an *execution-part*. The"]	13:15
The *execution-part* of a module is a **module initializer**.	13:16+

[Editor: In the "executable statement" and "FORMAT statements" rows of the MODULE column of table 2.2, change "no" to "yes" twice.] 14

[Editor: After "program." insert "Executing an *end-module-stmt* causes normal termination of execution of a module initializer."] 15:1

The *end-block-data-stmt* statement is nonexecutable. 15:3

Execution of a main program, a subprogram, or a module initializer involves execution of the *executable-construct*s within its scoping unit. With the following exceptions, the effect of execution is as if the *executable-construct*s are executed in the order they appear until a STOP, RETURN or END statement is executed. The exceptions are the following: 15:5-11

The initializers of all modules upon which a program unit depends by use association or host association, either directly or indirectly, shall be executed before any *execution-construct* in the *execution-part* of that program unit is executed, or any specification expression in that program unit that is not an initialization expression is evaluated. 15:18+ New ¶

It is processor dependent whether the initializers of modules that are not accessed by use association are executed.

An initializer shall not be executed more than once.

If a program contains a Fortran main program, execution of the program begins by executing any necessary initializers, followed by execution of the main program.

Execution of the main program or an initializer begins with the first *executable-construct* of its *execution-part*. When a procedure is invoked, execution begins with the first *executable-construct* following the invoked entry point.

[Editor: Add a sentence in the same paragraph: "The execution sequence of a module initializer excludes module procedures within the module."] 15:20

[Editor: Insert "' and may contain an *execution-part*" after "units".] 246:3

[*execution-part*] 246:9+

[Editor: In the first line of Note 11.4, replace the first comma by "and" and delete ", and FORMAT statements".] 246:25+4

[Editor: Insert a new subclause before C.1:] 431:3+

C.1 Section 2 notes

A sequence of execution of module initializers that satisfies the requirements in 2.3.4 may be determined by a processor that is applied after every program unit is translated and before the program units are linked together into a program, or by the processor that links the program units into a program. The processor may cause all initializers to be executed before the main program is executed. Processors for languages other than Fortran that provide for the equivalent of module initializers, including Ada and Modula-2, have used these and other strategies.

Alternatively, the need to execute an initializer may be determined dynamically as the program executes. In this case, it would be necessary for each initializer to determine whether it had been executed, so as not to be executed more than once. If an initializer has been executed, every initializer upon which it depends by use or host association has also been executed, so it is not necessary to proceed from one initializer to initializers for modules upon which it depends if it has been executed.

4 For the record, for submodule initializers. . .

If submodule initializers are ever done, they should be restricted so as to inhibit their ability to subvert the ancestor module's private data. Sufficient restrictions are (1) a variable that is accessed by use association, directly or indirectly, shall not appear in a variable definition context, and (2) a procedure invoked by an initializer or initializer procedure shall be an initializer procedure; it shall not have a dummy procedure, and shall observe these restrictions.