Subject: The PROTECTED attribute.
From: Malcolm Cohen
To: J3
Date: 20th August 2001
References: 00-169, 00-192, 00-193, 00-301, 00-307, 00-348, WG5/N1458.

1. Introduction

The PROTECTED attribute described by J3/00-307 and other past papers is a good idea. Various other languages (object-oriented and otherwise) have this useful facility. Unfortunately, inadequate exposition, terminological confusions and overly ambitious goals in previous papers has not allowed the true simplicity of the idea to come forth.

This paper provides a rigorous specification of the facility proposed, with proposed syntax and edits to the draft standard to implement the facility.

2. Brief Discussion

It is desirable to allow variables from modules to be made public without allowing them to be modified from outside of the module — e.g. when the module provider needs to control modification of the variables but there is no problem allowing the user to read the values. In Fortran 90/95, this is not possible - so to achieve the necessary safety one must make the variables private and provide access functions to read them.

This causes unnecessary code bloat, both for the module provider and for the module user. Furthermore, unless the access functions are expanded inline there can be a significant performance penalty.

The basic idea is an attribute which prevents modification of a variable from outside the module. Such a variable would be subject to the same limitations as for "INTENT(IN)" when it is accessed by use association.

3. Specification

The PROTECTED attribute can be applied to variables and procedure pointers in modules. This attribute prevents modification of the entity from outside the module.

Notes:
- For pointers, this means alteration of the pointer association status, i.e. the PROTECTED attribute functions like the INTENT(IN) attribute, but for use-associated variables not dummy variables.
- The PROTECTED attribute is only particularly useful for PUBLIC entities, but there is no other connection between the PROTECTED attribute and accessibility.

The PROTECTED attribute cannot be specified for entities imported from another module - there would be no point since it is either protected or not in the original module. The PROTECTED attribute cannot be specified for objects in common; this would be confusing and error-prone without adding any functionality. A PROTECTED object can only be EQUIVALENCEd to objects that are also PROTECTED.

4. Syntax

The PROTECTED attribute shall be spelled "PROTECTED", and may be given to entities either in type declaration statements, e.g.

```
REAL,PROTECTED :: X
```

or in a PROTECTED statement, e.g.

```
PROTECTED X
```

5. Edits to 01-007r2

[10:48+] Insert "**or** *protected-stmt*".
{Add statement form of the attribute.}

[64:4+] Insert "**or** PROTECTED".

{Add attribute form.}

[65:24+] Insert constraints:
"C532a: (R504) The PROTECTED attribute is permitted only in the specification part of a module.
C532b: (R501) The PROTECTED attribute is permitted only for a procedure pointer or named variable that is not in a common block.
C532c: (R501) If the PROTECTED attribute is specified, the EXTERNAL, INTRINSIC, or PARAMETER attribute shall not be specified.
C532e: A nonpointer object with the PROTECTED attribute that is accessed by use association shall not appear in a variable definition context (16.8.7) or as the *target* in a *pointer-assignment-stmt*.
C532f: A pointer with the PROTECTED attribute that is accessed by use association shall not appear as
(1) A *pointer-object* in a *pointer-assignment-stmt* or *nullify-stmt*,
(2) An *allocate-object* in an *allocate-stmt* or *deallocate-stmt*, or
(3) An actual argument in a reference to a procedure if the associated dummy argument is a pointer with the INTENT(OUT) or INTENT(INOUT) attribute."
{Obvious constraints.
Note that C532e-C532f appear here rather than in the "PROTECTED attribute" subclause because there is no syntax in that subclause.}

[76:19+] Insert new subclause
"5.1.2.11a PROTECTED attribute
The PROTECTED attribute imposes limitations on the usage of module entities.

Other than within the module in which an entity is given the PROTECTED attribute,
- if it is a nonpointer object it is not definable, and
- if it is a pointer it shall not appear in a context that causes its association status to change.

If an object has the PROTECTED attribute, all of its subobjects have the PROTECTED attribute.

NOTE 5.18a
An example of the PROTECTED attribute:

```
MODULE temperature
  REAL,PROTECTED :: temp_c, temp_f
CONTAINS
  SUBROUTINE set_temperature_c(c)
    REAL,INTENT(IN) :: c
    temp_c = c
    temp_f = temp_c*(9.0/5.0) + 32
  END SUBROUTINE
END MODULE
```

The PROTECTED attribute ensures that the variables TEMP_C and TEMP_F cannot be modified other than via the SET_TEMPERATURE_C procedure, thus keeping them consistent with each other."

[82:29+] Insert new subclause
"5.2.10a PROTECTED statement

R546a *protected-stmt* is PROTECTED [::] *entity-name-list*

This statement specifies the PROTECTED attribute (5.1.2.11a) for a list of entities."

[88:4+] Insert new constraint
"C587a (R559) If an *equivalence-object* has the PROTECTED attribute, all of the objects in the equivalence set shall have the PROTECTED attribute."
{Only allow protected objects to be equivalenced to other protected objects.}

[233:17] Before "their" insert "the PROTECTED statement (5.2.10a),"
[233:18] After "5.1.2.1" insert ", 5.1.2.11a".
{The PROTECTED statement and attributes affect accessibility.}

[386:26] Insert ", 5.1.2.11a" after "5.1.2.7".