Miscellaneous Items

Carlos Cruz

NASA GSFC Code 606 (ASTG)
Greenbelt, Maryland 20771

carlos.a.cruz@nasa.gov

October 25, 2018

Agenda

Miscellaneous Items

- Computing Environment
- New constructs
- Module Enhancements
    - IMPORT
    - New Attributes
    - Renaming Operatos
- Changes to Intrinsic Functions
- Complex Constants

Accessing the Computing Environment

- For the following assume we have launched the executable with the command line:
  $ foo.x apple 5 z
- COMMAND_ARGUMENT_COUNT()
  - Returns integer number of command arguments
  - Example command returns 3
- GET_COMMAND([COMMAND,LENGTH,STATUS])
  - All INTENT(OUT) and OPTIONAL
  - LENGTH - integer # of characters in command
  - STATUS - integer (success/failure)
  - Results for example command:
    - COMMAND = "foo.x apple 5 z"
    - LENGTH=15

Computing Environment

- GET_COMMAND_ARGUMENT(NUMBER[,VALUE,LENGTH,STATUS])
  - NUMBER - selects argument
  - VALUE - character, intent(out) value of argument
  - LENGTH - number of characters in argument
  - STATUS - integer (success/failure)
  - Example command yields:
    - GET_COMMAND_ARGUMENT(0,VALUE,LENGTH) yields VALUE="foo.x", LENGTH=5
    - GET_COMMAND_ARGUMENT(2,VALUE,LENGTH) yields VALUE="5", LENGTH=1

ISO_FORTRAN_ENV

A new intrinsic module is ISO_FORTRAN_ENV. It contains the following constants

- INPUT_UNIT, OUTPUT_UNIT, and ERROR_UNIT
  - are default integer scalars holding the unit identified by an asterisk in a READ statement, an asterisk in a WRITE statement, and used for the purpose of error reporting, respectively.
- IOSTAT_END and IOSTAT_EOR
  - are default integer scalars holding the values that are assigned to the IOSTAT= variable if an end-of-file or end-of-record condition occurs, respectively.
- NUMERIC_STORAGE_SIZE, CHARACTER_STORAGE_SIZE, and FILE_STORAGE_SIZE
  - are default integer scalars holding the sizes in bits of a numeric, character, and file storage unit, respectively.

Array Constructor

- Can now use "[" and "]" rather than "(/", "/)" to construct arrays:

  x(1:5) = [0.,1.,2.,3.,4.]
- Can also specify type **inside** constructor
  - VALUE - character, intent(out) value of argument
  - LENGTH - number of characters in argument
  - STATUS - integer (success/failure)
  - Example command yields:
    - GET_COMMAND_ARGUMENT(0,VALUE,LENGTH) yields
      VALUE="foo.x", LENGTH=5
    - GET_COMMAND_ARGUMENT(2,VALUE,LENGTH) yields
      VALUE="5", LENGTH=1

ASSOCIATE construct

ASSOCIATE construct associates named entities with expressions or variables during the execution of its block.

```fortran
use constants, only: gas_constant
ASSOCIATE ( R=>gas_constant, T => temp, P=>press, V=>vol)
    P = n*R*T/V
END ASSOCIATE

ASSOCIATE ( Z => EXP(-(X**2+Y**2)) * COS(THETA) )
    Y = A*Z
END ASSOCIATE
```

ALLOCATE statement

The allocatable attribute is no longer restricted to arrays

```
type (matrix(m=10,n=20)) :: a
type (matrix(m=:,n=:)), allocatable :: b, c
ALLOCATE(b, source=a)
ALLOCATE(c, source=a)
```

allocates the scalar objects b and c to be 10 by 20 matrices with the value of a.

Transferring an allocation

The intrinsic subroutine MOVE_ALLOC(FROM,TO) has been introduced to move an allocation from one allocatable object to another.

```
REAL,ALLOCATABLE :: GRID(:),TEMPGRID(:)
...
ALLOCATE(GRID(-N:N) ! initial allocation of GRID
...
ALLOCATE(TEMPGRID(-2*N:2*N)) ! allocate bigger grid
TEMPGRID(::2) = GRID ! distribute values to new locations
CALL MOVE_ALLOC(TO=GRID,FROM=TEMPGRID)
```

MOVE_ALLOC provides a reallocation facility that avoids the problem that has beset all previous attempts: deciding how to spread the old data into the new object.

SELECT TYPE construct

The SELECT TYPE construct selects for execution at most one of its constituent blocks, depending on the dynamic type of a variable or an expression, known as the 'selector'.

```
CLASS matrix :: mat
...
SELECT TYPE (A => mat)
    TYPE IS (matrix)
       <code here>
    TYPE IS (sparse_matrix)
       <code here>
END SELECT
```

- The first block is executed if the dynamic type of *mat* is *matrix* and the second block is executed if it is *sparse_matrix*.
- The association of the selector *mat* with its associate name *A* is exactly as in an ASSOCIATE construct
- In the second block, we may use *A* to access the extensions thus: *A*%sparse

IMPORT statement

A common pitfall when using F90/F95 is the declaration of an interface block that needs to "use" a derived type defined in the same module:

```
module foo
   type bar
      integer :: I,J
   end type bar
   interface
      subroutine externFunc(B)
         use foo, only: bar ! Not allowed?
         type (bar) :: B
      end subroutine externFunc
  end interface
...
```

IMPORT statement

IMPORT is a new statement to address this issue.

- Very similar to USE statement.
- Specifies all entities in host scoping unit that are accessible
- *Only* allowed in an interface body within a module

Example:

```
...
interface
     subroutine externFunc(B)
        import foo, only: bar
        type (bar) :: B
     end subroutine externFunc
  end interface
```

PROTECTED attribute

F2003 introduces the new attribute PROTECTED which provides a safety mechanism analogous to INTENT(IN)

- Specifies that the variable (or pointer status) may be altered only within the host module.
- Property is recursive. I.e. if a variable of derived type is PROTECTED, all of its sub-objects also have the attribute
- For pointers, only the association status is protected. The target may be modified elsewhere.

Example:

```
module foo
private ! Good default
real, public :: pi
protected :: pi ! Allow value to be read
...
```

Renaming operators

- F2003 extends the rename capability on USE statements to include renaming operators that are not intrinsic operators:

```
USE a_mod, OPERATOR(.MyAdd.) => OPERATOR(.ADD.)
```

- This allows .MyAdd. to denote the operator .ADD. accessed from the module.

Changes to Intrinsic Functions

- Argument COUNT_RATE for SYSTEM_CLOCK() can now be of type real.
  - Previously had to convert integer to compute reciprocal to determine elapsed time
- MAX, MAXLOC, MAXVAL, MIN, MINLOC, MINVAL have all been extend to apply to type CHARACTER
- ATAN2, LOG, and SQRT have minor changes to take into account positive/negative zero for vendors that support the distinction.

Lengths of Names/Constants

- Variables may be declared with names of up to 63 characters
- Statements of up to 256 lines are permitted.
- Primarily aimed at supporting automatic code generation

Complex Constants

Named constants may be used to specify real or imaginary parts of a complex constant:

```fortran
REAL, PARAMETER :: pi = 3.14159265358979323 84
COMPLEX :: C = (1.0, pi)
```