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
- Array Constructor Syntax
- Module Enhancements
  - IMPORT
  - New Attributes
  - Renaming Operatos
- Changes to Intrinsic Functions
- Complex Constants

Computing Environment

- From the intrinsic module ISO_FORTRAN_ENV
- 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)andOPTIONAL
  - 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

Environment examples

Getting command arguments:

```
1  use ISO_FORTRAN_ENV
2  character(len=MAXLEN_ARG) :: arg1, arg2
3  call get_command_argument(1, VALUE=arg1)
4  call get_command_argument(2, VALUE=arg2)
5  read(arg1,'(i)') nx
6  read(arg2,'(i)') ny
```

Getting an environment variable::

```
1  use ISO_FORTRAN_ENV
2  character(len=100) :: myShell
3  call get_environment_variable('SHELL', myShell)
```

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

IMPORT statement

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

```
1   module foo
2      type bar
3         integer :: I,J
4      end type bar
5      interface
6         subroutine externFunc(B)
7            use foo, only: bar ! Not allowed?
8            type (bar) :: B
9         end subroutine externFunc
10     end interface
11  ...
```

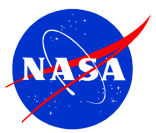
IMPORT statement

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

Exmaple:

```
1  ...
2  interface
3       subroutine externFunc(B)
4          import foo, only: bar
5          type (bar) :: B
6       end subroutine externFunc
7    end interface
```



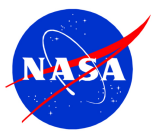
PROTECTED attribute

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:

```
1  module foo
2  private ! Good default
3  real, public :: pi
4  protected :: pi ! Allow value to be read
5  ...
```



VOLATILE attribute

- Introduced for a data object to indicate that its value might be modified by means external to the program.
  - Non standard extensions (e.g. threads)
  - Card connected to external lab instrument
- Effect is that the compiler is required to not rely on values in cache or other temporary memory.
- If an object has the VOLATILE attribute, so do all of its subobjects.
- For pointers, attribute refers only to the association status, *not* the target.

Renaming operators

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

```
1  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:

```
1  REAL, PARAMETER :: pi = 3.14159265358979323384
2  COMPLEX :: C = (0.0,pi)
```