

Coding Standard

The HEFSS Team
NASA Langley Research Center
Hampton, Virginia, USA

June 17, 2002

Note: parenthetical numbers refer to line numbers in the sample program which follows.

Style

- Free format with no character past column 80
- Indentation: begin in first column and recursively indent all subsequent blocks by two spaces
- Start all comments within body of code in first column [15]
- Use all lowercase characters, however mixed-case may be used in comments and strings
- Align continuation ampersands within code blocks [49]
- No tab characters
- Name `end`'s [57]

Comments

- For cryptic variable names, state description using a comment line immediately preceding declaration or on end of the declaration line [35]
- For subroutines, functions, and modules, insert a contiguous comment block immediately preceding declaration containing a brief overview followed by an optional detailed description [15]

Variable Declarations

- Do not use FORTRAN intrinsic function names
- Avoid multi-line variable declarations
- Declare `intent` on all dummy arguments [36]
- Declare the kind for all reals, including literal constants, using a kind definition module
- Declare `dimension` attribute for all non-scalars [36]
- Line up attributes within variable declaration blocks
- Any scalars used to define extent must be declared prior to use [33]
- Declare a variable name only once in a scope. This includes `use module` statements.

Module Headers

- Declare `implicit none` [8]
- Include a public character parameter containing the CVS `Id` tag [10]
- Include a `private` statement and explicitly declare public attributes

Subroutines and Functions

- The first executable line should be `continue` [41]
- Use the `only` attribute on all `use` statements [31]
- Keep `use` statements local, i.e., not in the module header
- Group all dummy argument declarations first, followed by local variable declarations
- All subroutines and functions must be contained within a module
- Any pointer passed to a subroutine or function must be allocated by at least size 1 to avoid null or undefined pointers

Control Constructs

- Name control constructs (e.g., `do`, `if`, `case`) which span a significant number of lines or form nested code blocks
- No numbered do-loops
- Name loops that contain `cycle` or `exit` statements
- Use `cycle` or `exit` rather than `goto`
- Use case statements with case defaults rather than if-constructs wherever possible
- Use F90-style relational symbols, e.g., `>=` rather than `.ge.` [45]

Miscellaneous

- In the interest of efficient execution, consider avoiding:
 - Assumed-shape arrays
 - Derived types in low-level computationally intensive numerics
 - `use` modules for large segments of data
- Remove unused variables
- Do not use common blocks or includes

Illustrative Example

```
1  ! A collection of transformations which includes
2  ! stretches, rotations, and shearing. This comment
3  ! block will be associated with the module declaration
4  ! immediately following.
5
6  module transformations
7
8      implicit none
9
10     character (len=*), parameter :: transformations_module_cvs_id = &
11         '$Id: cs_example.f90,v 1.4 2002/04/12 14:41:22 cvs6mp Exp $'
12
13     contains
14
15     ! Computes a stretching transformation.
16     !
17     ! This stretching is accomplished by moving
```

```

18 ! things around and going into a lot of other details
19 ! which would be described here and possibly even
20 ! another "paragraph" following this.
21 !
22 ! This contiguous comment block will be associated with the
23 ! subroutine or function declaration immediately following.
24 ! It is intended to contain an initial section which gives
25 ! a one or two sentence overview followed by one or more
26 ! "paragraphs" which give a more detailed description.
27
28 subroutine stretch ( points, x, y, z )
29
30     use kind_defs
31     use someothermodule, only: some_variables
32
33     integer,                               intent(in)  :: points
34
35     ! component to be transformed
36     real(fp), dimension(points), intent(in)  :: x, y
37     real(fp), dimension(points), intent(out) :: z ! transformation result
38
39     integer :: i
40
41     continue
42
43     i = 0
44
45     if ( x(1) > 0.0_fp ) then
46         call positive ( points, x, y, z )
47     else
48         do i = 1, points
49             z(i) = x(i)*x(i) + 1.5_fp * ( i + x(i) )**i &
50                 + ( y(i) * i ) * ( x(i)**i + 2.0_fp ) &
51                 + 2.5_fp * i + 148.2_fp
52         enddo
53     endif
54
55     end subroutine stretch
56
57 end module transformations
58
59 module kind_defs
60
61     implicit none
62
63     character (len=*), parameter :: kind_defs_cvs_id = &

```

```
64      '$Id: cs_example.f90,v 1.4 2002/04/12 14:41:22 cvs6mp Exp $'  
65  
66      integer, parameter :: hp=4      ! Half precision  
67      integer, parameter :: fp=8     ! Full precision  
68  
69      end module kind_defs
```