Introduction to Fortran 90, Part III

CS 471/Math 471

Brian T. Smith, UNM

Andrew C. Pineda, UNM

Data Objects

- Variables or constants
  - a variable
    - may be of any type and kind
    - is an identifier that is declared in a type statement
    - is given a value initially or during execution
    - may change its value during execution
    - may be a scalar or an array

Example -- Variables

```
real  X
real :: A = -huge(X)     ! Given a value initially
read  *, A

X = abs(A)
X = 0.5*(X + A/X)
```

- Declared X, declared A with initial value
- X and A are changed during execution

Data Object Cont'd

- A constant
  - may be of any type and kind
  - may be a scalar or an array
  - may be a literal constant -- e.g. a digit string
  - may be a named constant (called a parameter)
    - an identifier that is declared, has the PARAMETER attribute, is given a value in a specification statement, and *never* changes

      real, parameter :: FUDGE = 2.0

Parameterized Constants

- A kind parameter can be appended to literal constants, and is either
  - a literal integer constant, or
  - a named integer constant
    - allows for parametric change of the kinds of all constants using the named constant -- WP example
- The kind parameter is separated from value by an underscore and is
  - after the literal constant, for integer, real, and logical
    - 3_8, 1.1e1_WP
  - before the literal constant, for character -- GREEK_'α_β'

Data Objects Cont'd

- Either a scalar or an array (var. or constant)
  - a scalar
    - has only one value at a time (may have components)
    - of any type or kind (including derived type)
    - may be a constant or a variable
  - an array
    - is made up of a rectilinear-shaped collection of values, and is either a constant or variable
    - may be of zero-size (no values)

An Array

- Has a rank -- the number of dimensions
  - from 1 to 7
- Has a size in each dimension
  - may be zero-sized in any dimension, giving a zero-sized array
- Has a shape, that is, a vector whose size is the number of dimensions where each element is the size in that dimension

An Array Cont'd

real A(10,20), B(-2:2,-1:1,-3:-2)

- A of type real, default real kind, rank 2 of shape (10,20) and size 200
- B is of type real, default kind, rank 3, shape (5,3,2) and size 30

- Size is the product of the sizes in each dim.
- May be a variable or constant
  - real, dimension(3), parameter :: E = (/ 1.0, 2.0, 0.0 /)

Other Language Entities

- Expressions (scalar, array, constant, type, kind)
- Lists (arguments, I/O, names, exprs, ...)
- Common block names
- Namelist group names
- Construct names
- Generic names

Specification Statements

- Establish the environment in which the executable statements operate and specify:
  - variables, name constants
  - attributes such as type, kind, arrayness, pointer, argument intent, argument optionality, genericity, targetness, static, accessibility (private, public),...
  - implicit typing
  - common blocks and their names
  - storage sequences
  - interfaces
  - derived-types, operators
  - USE statements

Executable Statements

- Specify the action for the program and are:
  - assignment statements (regular, masked, pointer)
  - input/output statements (READ, WRITE, PRINT,...)
  - control statements (GOTO, IF, STOP, RETURN)
  - control constructs (IF, DO, SELECT CASE)
  - CALL statements
  - ALLOCATE, DEALLOCATE, NULLIFY stmts

Include Statement

- Neither an executable nor specification stmt
- A compiler directive to include all text from a named file in place of the include statement
- Example:

```
include 'file_name'
include "file_name"
```

Sorts of Program Units

- Main program unit
  - optionally starts with a PROGRAM statement
  - is the place where the execution begins
  - controls or directs all computation
- Module program unit
  - specifies data objects (vars. and cons.), derived types, interfaces, procedures, ... to be *shared* (by *name*) with other *specified* program units

Program Units Cont'd

- Subprogram unit -- subroutine or function
  - starts with SUBROUTINE or FUNCTION stmt
  - invoked by a CALL statement or in an expr.
  - passes information primarily via arguments
  - when internal, a computation shared with *only* the host (and its internals)
  - when external, a computation shared with *all* other program units

Function Subprogram Unit

- Invoked by the appearance of its name followed by a list of actual arguments in an expression
- Returns a value via its name
- Accesses values via its arguments
  - may define or redefine its arguments (but then it is *not* a PURE function)

Block Data Subprogram Unit

- Contains *only* data objects that are variables (scalar or array)
- Shared with all program units
- Sharing is accomplished by storage association of the COMMON block, *not* by name of the data object

Procedures Within A Host

- Procedures within a host are those procedures at the end of a host program unit after a CONTAINS statement.
  - module procedures (first-level procedures within a module)
  - internal procedures (procedures within the main program unit, external subprogram unit, or module procedure)

Scope of Names for "Sub-Procedures"

- All names declared in the sub-procedure are *local* to the sub-procedure
- All names of host are known to the "sub-procedure", *except* those names declared in the sub-procedure that are the *same* as those in the host

Structure of Program Units

- All have essentially the same structure, except:
  - a module has *no* executable part
  - a block data subprogram has *no* executable part, NO CONTAINS statement, and internal part

Program Unit Structure Cont'd

- Main program, subroutine, function, block data program units look like:

```
<header>  (PROGRAM, SUBROUTINE,
           FUNCTION, BLOCK DATA)
   <specification-part>
   <execution-part>
 CONTAINS
   <internal-subprograms>    ! May be more than 1
 <end-statement> (PROGRAM, ...)
```


Program Unit Structure Cont'd

- Structure of a module subprogram unit

```
<header>     (MODULE)
   <specification-part>
   CONTAINS
     <module-subprograms>  ! May be more than 1
<end-statement>   (MODULE)
```

- Module subprograms have the same form as subroutine and function subprograms

An Internal Function Example

```
program  MY_SQRT_PROGRAM

  ! This program computes an approximation to the
  ! square root of a real number read from standard
  ! input and prints the result to standard output.  The
  ! program is written in Fortran 90 following the free
  ! source form specified by the Fortran 90 standard.
  ! Make sure there are no identifiers declared
  ! implicitly.

  implicit  none
```


Program Continued

```
! Declare the names used as intrinsic procedures (not
! necessary to do this).

intrinsic  selected_real_kind, sqrt

! Determine the kind number WP for a real type that
! has at least 6 decimal digits of accuracy.

integer, parameter :: WP = selected_real_kind( 6 )
real(WP)  A
```


Program Cont'd

```
! Read in the value of A.

read *, A

! Compute the approximate square root and compare
! with the intrinsic square root.  The approximate
! square root is implemented as an internal function.

print *, 'My approximate square root of ', A, ' is ', &
         MY_SQRT( A )
print *, "The vendor's approx. square root is ",  &
          sqrt( A )
```

Program Cont'd

```
! A main program stops when a CONTAINS
! statement is encountered.
! (A subroutine or function returns when a
!  CONTAINS or END statement is encountered.)

CONTAINS
```


Program Cont'd

```
real(WP) function MY_SQRT( A )

! This functions returns an approximate square of
! the absolute value of its argument A

   implicit  none
   real(WP), intent(in) :: A

   ! Declare various data objects:
   !    A         the value read for which the approx.
   !              square root is to be computed
```


Program Cont'd

```
   !    PREV_X    the previous iterate
   !    FUDGE     a fudge factor to make sure that
   !              iteration terminates.

   real(WP)  LOC_A, PREV_X, X
   real(WP), parameter :: FUDGE = 2.0_WP
   intrinsic  abs, epsilon

   ! A may be negative; make it positive.  Initialize
   ! the iteration process.   ...
```


Program Cont'd

```
   LOC_A = abs(A)
   X = LOC_A
   PREV_X = 0.5_WP*X

   if( A /= 0.0_WP )  then
     ! Approximate the square root.  Perform the
     ! Newton iteration; stop when two consecutive
     ! iterates are close enough.
```


Program Cont'd

```
      do while( abs( X - PREV_X ) >  &
              FUDGE*abs(X)*epsilon(X) )
        PREV_X = X
        X = 0.5_WP*(X + LOC_A/X)
      end do
   else
      ! The square root of zero is zero.
      X = 0.0_WP
   endif
```


Program Cont'd

```
   ! The value is returned through the function name.

   MY_SQRT = X

   ! Execution of the END statement of a function or
   ! Subroutine is the same as the execution of the
   ! RETURN statement.

end function  MY_SQRT

end program MY_SQRT_PROGRAM
```

Arrays In Fortran 90

- A declaration gives an array its shape
    - logical A(2,3,4,5,-1:1)
        - have shape (2,3,4,5,3)
- An array can be used in many of the same places as scalars
  - expressions
    - .not. A
  - assignments
    - A = .true.

Arrays Continued

- All intrinsic operations including assignment and elemental intrinsic functions are element-by-element
    - like a do-loop BUT it must behave as if:
        - *no* order of the loops and indices is assumed, and
        - *all* parts of the right side of an assignment are performed before any assignment occurs
        - *all* operands must be have the same shape or are scalars
        - *all* scalars are broadcast to the appropriate size

Arrays Continued

- In assignment, the left-side size must be an array if any part of the right-side is an array (except if it is the argument of a transformational function)
- In broadcasting a scalar, the appropriate size is the size of the other operand if the operand is binary and is an array, or to the left side of an assignment

Array Examples

Assuming

real A(3,4), B(3,4), C(3,4), D

A+B
- elemental sum

A*B
- elemental product

D*A
- elemental product, D broadcast to the size of A

Array Examples Cont'd

D*SQRT(A+B**C)
- elemental with scalar broadcast of D to shape of A

A == B
- a logical array of the size of A with true values where A(i,j) equals B(i,j)

A = B+D
- A is assigned values of the expression B+D, element-by element

Array Examples Cont'd

C = D
- all elements of C are assigned the value of D

A = A/A(1,1)
- every element of A is divided by the first element of the initial A

Array Sections

- Instead of a subscript, a range of subscript values may be specified
  - called a section subscript
  - creates a section of an array
  - rank of the array is the number of section subscripts
  - form of section subscript (all integers)
    - expr1:expr2 or expr1:expr2:expr3

Array Sections Cont'd

- For the declaration
  real A(10,20,-30:30)
  - A(1:5,3,7)
    - an array of rank 1 of shape (5)
  - A(2:7,6,4:20)
    - an array of rank 2 of shape (6,17)
  - A(:,:,3)
    - default extents from the declaration of A
    - an array of rank 2 of shape (10,20)

Array Section Examples

- A missing third integer is the stride and defaults to 1
- The stride must not be zero
- The stride may be negative
  - the elements are taken in reverse order
    A(10:1:-2,3,4)
  - an array of elements A(10,3,4), A(8,3,4),…, A(2,3,4) -- rank 1 and size 5

Declarations for LU

Real A(N,N), M(N)

- LU factorization j-th step is
  - $A = A - m^* p^T$
- Vector m at the j-th step is:
  - A(J+1:N,J) / A(J,J)
- Vector p at the j-th step is:
  - A(J,J+1:N)

LU Factorization

```
do J = 1, N-1
   …
   M(J+1:N) = A(J+1:N,J) / A(J,J)
   do K = J+1, N
      A(J+1:N,K) = A(J+1:N,K) -  &
                       M(J+1:N)*A(J,K)
   enddo
enddo
```


Masked Array Assignment

- Mask all assignments and all elemental intrinsic operations in a block of assignment statements
- The mask is specified by a "mask_expression" expression

```
WHERE( <mask_expression> )
   <block_of_assignment_statements>
ENDWHERE
```

Masked Assignment Cont'd

- Rules:
  - the arrays assigned to in the WHERE block must be conformable with each other and with the mask expr
  - the mask expression must be of type logical
  - the arrays assigned to may be of any type
  - *all* statements are executed in order for true elms
- Operation (or meaning):
  - *only* where the mask is true will the operations take place (assignment and operations in the exprs)

Masked Assignment With ELSEWHERE

- The WHERE block can have an ELSEWHERE
- Assignment statements in the ELSEWHERE block are executed *in order* for the false values of the mask *only*

```
WHERE( <mask_expression> )
    <block_of_assignment_statements>
ELSEWHERE
    <block_of_assignment_statements>
ENDWHERE
```


The SQRT Loop Using Arrays

```
MASK = abs(X-PREV_X) > FUDGE*abs(X)
do while( any(MASK) )
   where( MASK )
      PREV_X = X
      X = 0.5_WP*(X + LOC_A/X)
   endwhere
   MASK = abs(X-PREV_X) > FUDGE*abs(X)
enddo
```