Major Developments in Fortran Since 1977

Van Snyder
van.snyder@jpl.nasa.gov

Jet Propulsion Laboratory
California Institute of Technology

4 July 2011

Copyright © 2011 California Institute of Technology. Government sponsorship acknowledged.
Summary

The major developments in Fortran since 1977 are

- Derived types – a.k.a. structures or records
- Parameterized types
- Array processing
- Dynamic memory – pointers, allocatable, automatic
- Modules and submodules
- Object-oriented programming
- Coarrays
Derived types

Derived types in Fortran are types consisting of components derived from other types.

```fortran
  type :: T
    integer :: A
    real :: B(10)
    type(u) :: C
  end type T

  type(t) :: S(5)
```
Derived types (cont.)

Components of derived types are selected using %, because “dot” was already in use to spell several operators.

\[ s^a = 10 \] ! An array operation
\[ ! \text{described later} \]
print *, s^a
Parameterized types

Types in Fortran can have “kind” parameters, which are constants, and “length” parameters, which can vary during execution.

Fortran 77 provided a length parameter for character variables.

Fortran 90 provided kind parameters for real, complex, and integer variables. Fortran 2003 extended parameterization to derived types.

```
integer, parameter :: RK = kind(0.0d0)
real(rk) :: X(100)
integer, parameter :: R12 = selected_real_kind(12)
real(r12) :: Y(100)
```
Array processing

Arrays have always been important in Fortran. Major new array features after 1977 are

- Assumed-shape dummy arguments
- Array expressions and assignment
- Intrinsic functions to operate on arrays
- Elemental procedures
Assumed-shape dummy arguments

The extent in each dimension is assumed from the associated actual argument.

Unless otherwise specified, the lower bound in each dimension is 1, regardless of the bounds of the actual argument.

The bounds are declared using a colon, or a lower bound expression followed by a colon.

All of the bounds must be assumed.

```
real :: A(:,−1:)
```
Array expressions and assignment

Arrays can be elements of expressions.
If an array name appears on its own, it designates the whole array.
A “section” of any dimension, or several dimensions, can be specified by a triplet, consisting of the start, end, and stride, separated by colons.
If the start is absent, the low bound is assumed. If the end is absent, the high bound is assumed. If the strid is absent, 1 is assumed.

    print *, a(:,2,:)
Array expressions and assignment (cont.)

 Scalars can be combined with arrays. The effect is as if the scalar were an array of the same shape, with the same value in every element.

 \[
 \text{print } *, 0.5 * \text{a}(:,2,:) 
 \]

 One dimensional arrays can be constructed in expressions.

 \[
 \text{print } *, \left[ ( i, i = 1, 10 ) \right] 
 \]

 prints 1 2 3 4 5 6 7 8 9 10
Intrinsic functions to operate on arrays

Most numeric intrinsic functions are elemental, meaning they are applied to every element of an array

\[
\text{print } *, \cos(0.5 * a(:,2,:))
\]

Many array reduction functions are provided

\[
\text{print } *, \text{maxval}(0.5 * a(:,2,:))
\]
Elemental procedures

In addition to the elemental intrinsic functions provided by Fortran 90, Fortran 95 allows programs to define elemental procedures. They have scalar dummy arguments. If invoked with array actual arguments they are applied to every element of their arguments.

```fortran
  elemental subroutine Sub ( X, Y )
    real(rk), intent(in) :: X
    real(rk), intent(out) :: Y
    y = 4.0_rk * cos(x) + sin(x/2)**2 - 1
  end subroutine Sub
```
Dynamic memory

Fortran 90 provided three kinds of dynamic memory

- Pointers – type, kind, and rank safe, but otherwise like C pointers (except arithmetic on them isn’t allowed)
- Allocatable – No aliasing, so the optimizer does a better job, and no memory leakage
- Automatic – created when a procedure is entered

Pointers, allocatable variables, and automatic variables, can be arrays. Pointers are “fat” – they carry their bounds with them.
Dynamic memory (cont.)

ALLOCATE and DEALLOCATE statements create and destroy dynamic objects – either pointer or allocatable.

The NULLIFY statement nullifies pointers.

```plaintext
real, pointer ::= P(:)
type(t), allocatable ::= Q(:,:)
nullify ( p )
allocate ( p(6), q(-1:1,12) )
```
Dynamic memory (cont.)

Pointer assignment copies pointer association status, or causes a pointer to be associated with a non-pointer variable. The variable needs the TARGET attribute.

\[
\text{target} :: \text{A} ! \text{ or real, target} :: \text{A}(:,,-1:)
\]
\[
p \to a(3,:) ! \text{P is the third row of A}
\]
\[
! \text{The bounds of P are } -1: \text{ubound(A,2)}
\]

Pointers are automatically dereferenced

\[
\text{print } *, \ p ! \text{ Print the third row of A}
\]
Modules and submodules

Types, named constants (parameters), variables, procedures, and a few other arcane things can be put into modules.

```plaintext
module M
    integer, parameter :: RK = kind(0.0d0)
end module M
```
Modules and submodules (cont.)

Things in modules can be used in other places.

module X
  use M, only: RK
contains
  subroutine X_Sub ( A, B )
    real(rk), intent(in) :: A
    real(rk), intent(inout) :: B(:)
  end subroutine X_Sub
end module X
Modules and submodules (cont.)

If a procedure is used from a module, it has explicit interface, which allows checking that the type, kind, rank, and number of actual arguments in a reference matches the definition.

```fortran
use X, only: X_Sub
call X_Sub([1, 2], 43.0)
```

The call statement is rejected because the first actual argument has to be a real scalar of kind RK, not an integer array, and the second actual argument has to be a real array variable of kind RK, not a scalar constant of default kind (which is single precision, not double precision).
Modules and submodules (cont.)

Submodules allow big modules to be broken into pieces, and allow the interfaces of procedures to be in the module, while the bodies are in submodules. Separating a procedure body from its interface limits compilation cascades, and allows the interface to be published as definitive documentation without publishing trade secrets in the procedure body. Submodules are very much like Ada “private child units.”
Object-oriented programming

Object-oriented programming in Fortran is modeled on Simula. It provides

- Type extension
- Type-bound procedures
- Type-bound procedure overriding
- Polymorphism
- Dynamic dispatch
- Finalization
Object-oriented programming – Type extension

```plaintext
type(rk) :: Point
    real(rk) :: X, Y
end type Point

type, extends(point) :: Color_Point
    integer :: Color
end type Color_Point
```

Objects of type Color_Point have X, Y, and Color components.
Object-oriented programming – Type-bound procedures

```
type(rk) :: Point
    real(rk) :: X, Y
contains
    procedure :: Draw => Monochrome_Draw
end type Point

type(point(kind(1.0_rk))) :: Pixel
call pixel%draw ! calls Monochrome_Draw ( pixel )
```
Object-oriented programming – Overriding

type, extends(point) :: Color_Point
  integer :: Color
contains
  procedure :: Draw => Color_Draw
end type Color_Point

type(color_point(kind(1.0_rk))) :: C_Pixel
call c_pixel%draw ! calls Color_Draw ( c_pixel )
Object-oriented programming
Polymorphism and Dynamic dispatch

class(point), pointer :: Pix
pix => pixel
call pix%draw ! calls Monochrome_Draw(pix)
pix => c_pixel
call pix%draw ! calls Color_Draw(pix)

Pix is polymorphic. It can be associated with an object of type Point or any extension of that type. Polymorphic objects have to be pointers, allocatable, or dummy arguments.
Polymorphic pointers and allocatable variables can be allocated with a specified type that is the same as the declared type of the object, or any extension of it.

allocate ( type(color_point(rk)) :: Pix )
Object-oriented programming – Finalization

type :: Finalizable
    type(t), pointer :: Component => NULL()
contains
    final :: Destroy_It
end type Finalizable

type(finalizable) :: F

When F ceases to exist (deallocated, a local variable of a returning procedure, ...), Destroy_It (F) is called. One good reason to want this is to deallocate F%Component.
Coarrays

... are the subject of another presentation ...