Fortran 95 for Fortran 77 Programmers

Compiled by Bil.Kleb@NASA.gov, Jan-Feb 2007

Corrections thanks to Beliavsky, Maine, Herrmannsfeldt, Shepard, Moses, and Hendrickson in this `comp.lang.fortran` thread, which also includes dissenting opinions.

Table of Contents

    - Numeric Intrinsics
    - Array Intrinsics
    - Dynamic Array Allocation
    - Tensor Functions
    - Derived Types
    - Argument Intent
    - Optional and Keyword Arguments
    - Modules
    - Data Hiding
    - Operator Overloading
    - Namelists
    - Construct Names

Sources

- Fortran 95 for Fortran 77 Users
- `comp.lang.fortran`
- A Brief History of Fortran
- http://en.wikipedia.org/wiki/Fortran
- http://en.wikipedia.org/wiki/Fortran_language_features
- Fortran Standards website
- *Fortran 95/2003 Explained*
- Fortran 90 Programming, Class Materials
- Introduction to FORTRAN 90

Brief History

Fortran is 50 years old in 2007.
Created by IBM team lead by John Backus.

- 1957: FORTRAN I
- 1958: FORTRAN II (added linker)
- 1958: FORTRAN III (never publicly released)
- 1961: FORTRAN IV (clean up of II)
- 1966: FORTRAN 66 (first standard)
- 1977: FORTRAN 77 (many new things added)
- 1992: Fortran 90 (superset of 77, many additions)
- 1998: Fortran 95 (deleted a few 77 features; minor tweaks)
- 2004: Fortran 2003 (object-oriented, c-interop, and i/o)
- 200?: Fortran 2008 (co-arrays; minor tweaks)

Fortran 90 Additions

- Free format source code form (`f90`)
- Modern control structures (`CASE`, `DO WHILE`, and `ENDDO`)

- User-defined data types, (`TYPE`)
- Array notation
- Dynamic memory allocation
- Operator overloading
- Keyword and optional arguments
- Argument `INTENT`
- Numeric precision specification
- Modules

Fortran 95 Additions

- `CPU_TIME`
- Null pointer declaration
- Comments in namelists
- `FOR ALL` and nested `WHERE` constructs
- `PURE` and `ELEMENTAL` routines

Fortran 95 Deletions

- Non-integer `DO` indices
- `ASSIGN`
- Branching to `END IF`
- `PAUSE`
- `H` edit descriptor

Constructs Considered Harmful

- `BLOCK DATA`
- `INCLUDE`
- `COMMON`
- `DIMENSION`
- `DOUBLE PRECISION`
- `DSQRT`, `CSQRT`, `DABS`, etc.
- `ENTRY`
- `EQUIVALENCE`
- `PARAMETER`
- Arithmetic `IF`

Constructs Considered Obsolete

- `PRINT` (Kleb opinion)
- Fixed source form
- Computed `GOTO`
- `CHARACTER*`
- `DATA` statements in executables
- Statement functions

- Share do loop termination

F95 Compilers

- Absoft
- Cray
- G95
- Gfortran
- HP
- IBM
- Intel
- Lahey-Fujitsu
- Numerical Analysis Group (NAG)
- PathScale
- Portland Group Inc (PGI)
- Salford
- SGI
- Sun

Some comparisons are available from http://www.polyhedron.com.

From F77 to F95

Fixed to Free Format

F77, `file.f`:
```
12345&statmt .. column * 72
```
F95, `file.f90`:
```
statements .. column * 132
```

Capitals-Only to Insensitive

F77:
```
      STRC77 = 'Upper case except strings'
```
F95:
```
Strict_F95 = 'Insensitive to case except strings'
```

Variable Names

F77:
```
      maxln6
      noundr
      reynln
```
F95:
```
maximum_length_is_31_characters
underscores_allowed
reynolds_number
```

Expression Continuations

F77:

```
       result = many * terms
      &       - final
```

F95:

```
result = many * terms &
       - final
```

String Continuations

F77:

```
       phrase = 'long [cut off at col 72]
      &sentence' [pick up right after &]
```

F95:

```
phrase = 'long &
         &sentence'
```

Comments

F77:

```
c comment, beginning of line only
```

F95:

```
! comment line

b = a * x ! also at end of line
```

Kleb opinion: detailed comments are typically a signal that you failed to communicate with the code itself.

Relational Operators

F77:

```
.lt.  .le.  .eq.  .ne.  .gt.  .ge.
```

F95:

```
 <     <=    ==    /=    >     >=
```

Implicit to Strict Typing

F77:

```
      npoint = ni * nj + 1
```

F95:

```
implicit none

integer :: ni, nj, n_points

n_points = ni * nj + 1
```

Variable Declarations

F77:

```
      integer      countr
      real         float
      character*40 filnam
```

F95:

```
integer           :: counter
real              :: float
character(len=40) :: filename
```

Variable Initialization

F77:

```
     integer fencep
     data fencep /21/
```

F95:

```
integer, save :: fence_posts = 21
```

Constant Parameters

F77:

```
      parameter( ni = 435)
```

F95:

```
integer, parameter :: ni = 435
```

Precision

F77:

```
  real*8 dblpre
```

F95:

```
real, parameter :: dp = selected_real_kind(15,307)

real(dp) :: double_precision = 3.2_dp
```

Array Declarations

F77:

```
real a
dimension a(10,20)
```

F95:

```
real, dimension(10,20) :: a
```

Passing Arrays

F77:

```
      permtr = totlgt( sides, npoint )

      real function totlgt( allsds, numpts )
        integer   numpts
        real      allsds(*)
        totlgt = 0.0
```

```
        do 1 i = 1, numpts
          totlgt = totlgt + allsds(i)
1       continue
      end
```

F95:

```
perimeter = total_length( face_lengths )

real function total_length( sides )
  real, dimension(:) :: sides
  total_length = 0.0
  do i = 1, size(sides)
    total_length = total_length + sides(i)
  enddo
end
```

Or the last section with the `SUM` intrinsic,

```
  total_length = sum(sides)
```

Strings

F77:

```
      character*12 fname
      fname = 'solution.dat'
```

F95:

```
character(len=12) :: filename = 'solution.dat'
```

Plus new instrinsics: `SCAN`, `VERIFY`, `ACHAR`, `IACHAR`, `ADJUSTL`, `TRIM`, `LEN_TRIM`, and `REPEAT`

Array Expressions

F77:

```
      do 10 i = 1, n
        a(i) = 0.0
        b(i) = b(i) + 1.0
        c(i) = a(i)/b(i)
10    continue
```

F95:

```
a = 0.0
b = b + 1.0
c = a/b
```

Array Conditionals

F77:

```
      same = .true.
      do i = 1, n
        if ( a(i) .ne. b(i) ) same = .false.
1     continue
```

F95:

```
same = a == b
```

Array Sections

F77:
```
      do i = 1, 2
        a(i) = b(i+2)
1     continue
```
F95:
```
a(1:2) = b(3:4)
```

`COMMON` or `BLOCK DATA` becomes `MODULE`

F77:
```
      common /shared/ vars, go, here
```
F95:
```
module shared
 real :: variables, go, here
end module shared
```
then include in current scope with
```
use shared
```

Kleb opinion: Merely replacing common blocks with modules is potentially missing an opportunity to curtail programming by side effect through global variables and to refactor toward better data encapsulation. Modules provide many benefits, consider exploring their full potential.

`INCLUDE` becomes `USE`

F77 (extension?):
```
      include 'reuse.i'
```
where `reuse.i` contains FORTRAN like
```
  real vars
  [..]
```
F95:
```
use reuse
```
where a `reuse MODULE` is defined like,
```
module reuse
 real :: vars
contains
 [..]
end module reuse
```

Arithmetic `IF` and Computed `GOTO` become `CASE`

F77:
```
goto (10,20,30) [expression]
if (expression) 10,20,30
```
where `[expression]` is `INTEGER`, `REAL`, or `COMPLEX`.
F95:
```
select case ([expression])
case (:-1)
   slope = -1
case (0)
   slope = 0
case (1:)
```

```
   slope = 1
end select
```
where `[expression]` is `CHARACTER`, `LOGICAL`, `INTEGER`.

Non-Advancing I/O

F77 (extension?):
```
      write (*,'(a,$)') 'Choose: '
      read (*,*) answer
```
F95:
```
write (*,'(a)',advance='no') "Choose: "
read (*,*) answer
```

Entirely New in F90/F95

`PURE` Routines

No side effects.
```
pure function nearest_neighbor(x, field)
```

`ELEMENTAL` Routines

Routine for scalars and arrays.
```
elemental function viscosity( temperature )
```

Timing Intrinsic

```
real :: start_time, end_time

call cpu_time( start_time )
[..]
call cpu_time( end_time )
```

Loop `CYCLE` and `EXIT`

```
do i = 1, 100
  [..]
  if i < 10 cycle
  [..]
  if i*j == 5 exit
  [..]
end do
```

Numeric Intrinsics

```
integer :: an_integer
real    :: a_real

smallest_positive   = tiny( a_real )
```

```
largest_possible    = huge( an_integer )
smallest_difference = spacing( a_real )
```

Array Intrinsics

```
real, dimension(10) :: side_lengths

real    :: perimeter, longest_length
integer :: shortest_side

     perimeter =    sum(side_lengths)
longest_length = maxval(side_lengths)
 shortest_side = minloc(side_lengths)
```

Dynamic Array Allocation

```
integer, dimension(:), allocatable :: stones

integer :: n_stones

read *, n_stones

allocate( stones(n_stones) )
[..]
deallocate( stones )
```

Tensor Functions

```
product = dotproduct( vector_1, vector_2 )

matrix_transpose = transpose( matrix )

b = matmul( a, x )
```

Derived Types

```
type point
  real :: x, y
end type point

type(point) :: origin

origin%x = 0.0
origin%z = 0.0
```

Argument Intent

```
subroutine move( a_point, a_displacement)

  type(point), intent(inout) :: a_point
  type(point), intent(in)    :: a_displacement
```

```
  a_point = a_point + a_displacement

end subroutine move
```

Optional and Keyword Arguments

```
call plot_xy( x, y, domain, range )

subroutine plot_xy( x, y, domain, range )

  real           :: x, y
  real, optional :: domain, range

  if ( present( domain ) ) then
    x_domain = domain
  else
    x_domain = 1.0
  end
```

Missing argument order held by commas, or use optional arguments out of order via keywords,

```
call plot_xy( x, y, range=5.0 )
```

Modules

```
module points
  use point
contains
  subroutine move( a_point, a_displacement )
    [..]
  end subroutine move
end module points
```

Data Hiding

```
type body
  PRIVATE
  integer     :: id
  real        :: mass
  type(point) :: cg
end type body

type(body) :: nose_cone

nose_cone%mass = 10.0  ! NOT allowed
```

Operator Overloading

```
module points

  use point

  interface operator(+)
    module procedure add
  end interface
```

```
contains

  subroutine add( a_point, other_point, new_point )

    type(point) :: a_point, other_point
    type(point) :: new_point

    new_point%x = a_point%x + other_point%x
    new_point%y = a_point%y + other_point%y

  end subroutine add

end module point
```

Namelists

```
real :: x=1.0, y=1.0, xy=1.0, x2=2.0, y2=2.0

namelist /inertia/ x, y, xy, x2, y2

read (*, nml=inertia)
```
with standard input containing
```
&inertia y = 0.5, xy = 1.5, y2 = 2.25 /
```

Construct Names

```
limit: if (gradient > max_value)
  [..]
  gradient = 4.0*epsilon + 2.0
  [..]
end if limit
cells: do cell = 1, n_cells
  [..]
  resid(cell) = resid(cell) + flux(U_l,U_r)
  [..]
end do cells
```

The Soks Wiki