from Fortran-90
to Fortran-95

Uwe Kuester
kuester@hlrs.de

allocatable components in derived type (F2000)

```fortran
type stack
  integer :: index
  integer, allocatable :: contents(:)
end type
```

not possible in F95 but in Fortran 2000
PURE INTEGER FUNCTION MANDELBROT(X)
COMPLEX, INTENT(IN) :: X
COMPLEX :: XTMP
INTEGER :: K
!
Assume SHARED_DEFS includes the declaration
! INTEGER ITOL
USE SHARED_DEFS

K = 0
XTMP = -X
DO WHILE (ABS(XTMP) < 2.0 .AND. K < ITOL)
   XTMP = XTMP**2 - X
   K = K + 1
END DO
MANDELBROT = K
END FUNCTION MANDELBROT

COMPLEX :: xx
INTEGER :: I,J,M,N

......
FORALL (I = 1:N, J = 1:M)
   xx = COMPLEX((I-1)*1.0/(N-1), (J-1)*1.0/(M-1))
   A(I,J) = MANDELBROT(xx)
END FORALL
F95: pure procedures

- procedures must be pure if they are referenced in a forall construct
- referenced in a specification statement
- passed as an actual argument to a pure procedure
- referenced in the body of a pure procedure (including those referenced by a defined operator or defined assignment)

F95: elemental procedures

- elemental procedures are pure procedures with only scalar arguments
- they have the prefix `elemental`
- must not contain IO statements or `stop`
- variables cannot be
  - `save` or in a `data` statement
- global variables must not be changed
- dummy arguments
  - must have an `intent`
  - must be scalar
  - are not allowed to have the pointer attribute
  - cannot appear in a specification statement
  - cannot be dummy procedures
- elemental procedures may be called with conforming arrays
f95: elemental procedures, example function

- example

```fortran
elemental function vip_calc(x, y)
  real :: vip_calc
  real, intent(in) :: x, y
  vip_calc = 3*x + y
end function vip_calc

! A call to vip_calc with scalar arguments
x = vip_calc(1.1, 2.2)

! The result of this call is an array
! conformable with a(1:n) and b(1:n)
```

f95: elemental procedures, example subroutine

- example subroutine

```fortran
elemental pure subroutine source (a, b, c)
  real, intent(in) :: a, b
  real, intent(out) :: c
  . . .
end subroutine source

real, dimension (2,2,3) :: s, t
real q
call source (q, s, t)
```
F95: forall

- forall (v1=l1,u1:s1, ..., vn=ln:un:sn, mask)
  a(e1, ..., em) = right_hand_side
  ! calls of pure functions
  end forall
- loop indices are evaluated before calculation
- loop indices may be evaluated in any order
- each statement is completed before the next will be executed
- an array may appear only once on the left side
- all indices v1, v2, ..., vn must appear in the array assignments on the left side
- pointer assignments are allowed

F95: forall .. end forall is a parallel do .. enddo

where ( A /= 0.0 )     B = 1.0 / A

where ( A /= 0.0 )
B = 1.0 / A
endwhere

forall( i = 1:n, j = 1:n, a(i, j).ne. 0.0 ) b(i, j) = 1.0 / a(i, j)
forall ( i = 1:n, j = 1:n )
  where( a(i, j) .ne. 0.0 )   b(i, j) = 1.0 / a(i, j)
end forall
F95: forall generalizes array syntax

- ! more general than array syntax:
  
  ```fortran
  forall( i = 1:n, j = 1:n ) h(i, j) = 1.0 / real(i + j - 1)
  ```

- this loop cannot be formulated in array syntax

F95: forall ... end forall different from do ... enddo

! no recursion!
! Jakobi, not Gauss-Seidel

```fortran
forall( i = 2:n-1, j = 2:n-1 )
  d(i, j) = 0.25*(c(i, j + 1) + c(i, j - 1) + c(i + 1, j) + c(i - 1, j)) - c(i, j)
  c(i, j) = c(i, j) + eps*d(i, j)
end forall
```

first statement is evaluated completely before the second
F95: masked elsewhere

- where (condition_1)
  ...
  elsewhere(condition_2)
  ...
  elsewhere
  ...
  end where

F95: minloc and maxloc with dim

- the two-dimensional array
  
  0  -5  8  -3
  A = 3  4  -1  2
  1  5  6  -4

- gives the following values

  \begin{align*}
  \text{maxloc}(A) & \quad \text{is} \quad (/1, 3/) \\
  \text{maxloc}(A, \text{dim}=1) & \quad \text{is} \quad (/2, 3, 1, 2/) \\
  \text{maxloc}(A, \text{dim}=2) & \quad \text{is} \quad (/3, 2, 3/) \\
  \end{align*}
F95: initialization of pointers

- pointer can be initialized in specification and in derived types

    ```fortran
    real, pointer, dimension(:) :: vector => null()
    type my_type
        real, pointer, dimension(:) :: vector => null()
    end type my_type
    ```

F95: default initialization of derived type components

- 

    ```fortran
    type my_type
        real :: value = -1.
        integer :: col
    end type my_type
    ```
F95: IEEE arithmetic: plus zero and minus zero

- plus zero and minus zero are identical in
  - relational operations
  - in all intrinsics except sign
  - in the scalar expression in the arithmetic if
- \( \text{sign}(x,y) \) is
  - \(+\text{abs}(x)\) if \( y > 0 \)
  - \(+\text{abs}(x)\) if \( y \) is plus zero
  - \(-\text{abs}(x)\) if \( y \) is minus zero
  - \(-\text{abs}(x)\) if \( y < 0 \).

F95: intrinsic `cpu_time`

- the new intrinsic `cpu_time` delivers the cpu time in seconds

```fortran
real :: time_in_seconds
call cpu_time(time_in_seconds)
```
F95: automatic deallocation of allocatable arrays

- allocatable are deallocated at the end of their scoping unit

```fortran
subroutine allo(nmax, value)
    real, allocatable, dimension(:) :: array
    allocate(array(nmax))
    ...
    value = ...
end subroutine allo
```

F95: zero field edit descriptor

- An edit descriptor with zero field width indicates that as few columns as possible should be used to produce the result.

- On input, w must not be zero. On output, if it is zero, the processor selects the field width to be used; otherwise, it specifies the field width.

```fortran
price = 44.23
print "(a, f0.2, a), "The price is $", price, "."
```

- produces the output:

  The price is $44.23.
used Fortran links

- http://www.nsc.liu.se/~boein/f77to90/f95.html

end