Subroutines

Except functions Fortran allows for another subprogram structure called `subroutine`. Contrary to a function, subroutine does not return a variable which is a result of its action. Instead of that, subroutines may modify variables passed to them as arguments or simply work as standalone subprogram.

A general structure of a subroutine

```
subroutine ExampleSubprogram(arg1, arg2, arg3, ...)

    type, intent :: arg1
    type, intent :: arg2
    type, intent :: arg3
    ...

    [ body ]

end subroutine ExampleSubprogram
```

subroutines are used in the program via calls, i.e.

```
call ExampleSubprogram(arg1, arg2, arg3, ...)
```

Arguments of the subroutine can take several types of `intent` parameter. Depending whether they are to be input, output or input-output variables the `intent` may be set as `intent(in)`, `intent(out)` or `intent(inout)`.

Example

```
program subroutine_example

  implicit none

  real, allocatable, dimension(:,:) :: A
  integer :: n

  n = 3

  call PrintInfo()

  allocate( A(n,n) )
  call UnitMatrix(A,n)
  call PrintMatrix(A,n)

  deallocate(A)

end program subroutine_example

! Print info
subroutine PrintInfo

  print *,' Example subroutine which only offloads some part of code '

end subroutine PrintInfo

! Sets matrix A to a unit matrix
subroutine UnitMatrix(matrix,n)

  implicit none
  integer, intent(in) :: n
  real, dimension(n,n), intent(out) :: matrix
  integer :: i

  matrix = 0.0

  forall( i=1:n ) matrix(i,i) = 1.0

end subroutine UnitMatrix

! Print matrix in human readable format
subroutine PrintMatrix(matrix,n)

  implicit none
  integer :: n
  real, dimension(n,n), intent(in) :: matrix
  integer :: i, j

  do i=1,n
    print *,( matrix(i,j), j=1,n )
  end do

end subroutine PrintMatrix
```

In the program above we have defined three subroutines. 1) `PrintInfo` is a simplest subroutine which takes no argument and only prints information about the program. 2) Subroutine `UnitMatrix` takes matrix as an input and modifies is to make a unit matrix. 3)

PrintMatrix which only prints the passed variable.

The above program will produce the following output

```
  Example subroutine which only offloads some part of code
   1.00000000       0.00000000       0.00000000
   0.00000000       1.00000000       0.00000000
   0.00000000       0.00000000       1.00000000
```

Recursive subroutines

Similar to functions, subroutines can be defined with `recursive` parameter. For example

```
program recursive_subroutine

  implicit none
  integer :: num

  num = 3

  call Test(num)

end program recursive_subroutine

recursive subroutine Test(n)

  implicit none
  integer, intent(inout) :: n

  if( n > 0 ) then

    print *,n

    n = n - 1
    call Test(n)

  end if

end subroutine Test
```

which gives

```
           3
           2
           1
```

Passing subroutine as argument

Subroutine in analogy to function can be also passed as an argument to another subroutine. Similar to functions the subroutine that takes such argument needs to contain an `interface` for the subroutine that is used as a subroutine

```
interface
    subroutine ArgumentSubroutineName(arg1, arg2, ...)
        type, intent :: arg1
        type, intent :: arg2
        ...

        [body]

    end subroutine ArgumentSubroutineName
end interface
```

Lets consider an example program

```
program subroutine_argument

  implicit none

  integer :: n
  real, allocatable, dimension(:,:) :: matrix

  external :: zero_matrix, unit_matrix

  n = 3
  allocate( matrix(n,n) )

  print *, 'Zero matrix'
  call initialize_matrix(matrix,n,zero_matrix)

  print *, 'Unit matrix'
  call initialize_matrix(matrix,n,unit_matrix)

  deallocate( matrix )
```

```
end program subroutine_argument

subroutine initialize_matrix(A,n,matrix_init)

  implicit none
  integer, intent(in) :: n
  real, dimension(n,n), intent(inout) :: A

  integer :: i,j

  interface
    subroutine matrix_init(matrix,ndim)
      implicit none
      integer, intent(in) :: ndim
      real, dimension(ndim,ndim), intent(inout) :: matrix
    end subroutine matrix_init
  end interface

  call matrix_init(A,n)

  do i=1, n
    print *,( A(i,j), j=1,n )
  end do

end subroutine initialize_matrix

subroutine zero_matrix(A,n)

  implicit none
  integer, intent(in) :: n
  real, dimension(n,n), intent(inout) :: A

  A = 0.0

end subroutine zero_matrix

subroutine unit_matrix(A,n)

  implicit none
  integer, intent(in) :: n
  real, dimension(n,n), intent(inout) :: A
  integer :: i

  A = 0.0
  forall( i=1:n ) A(i,i) = 1.0
```

```
end subroutine unit_matrix
```

In the example above we have defined a subroutine `initialize_matrix(A,n,matrix_init)` which takes an argument `matrix_init`. This subroutine also defines an interface to `matrix_init` subroutine and then uses it to initlize elements of the matrix. In the example above we show how to use with two different subroutines `zero_matrix` and `unit_matrix`. Both of them are defined out of the main program and main program defines only their `external` type. The above program outputs

```
 Zero matrix
   0.00000000       0.00000000       0.00000000
   0.00000000       0.00000000       0.00000000
   0.00000000       0.00000000       0.00000000
 Unit matrix
   1.00000000       0.00000000       0.00000000
   0.00000000       1.00000000       0.00000000
   0.00000000       0.00000000       1.00000000
```