Overloading functions and subroutines

Interface allows overloading functions and creating one "virtual" function with varying argument list. A general `interface` construct is

```
interface NewFunction

    type function NewFunction_ver1(arg1, arg2, ...) result(val)
        type, intent(in) :: arg1, arg2
        ...
    end function NewFunction_ver1

    type function NewFunction_ver2(arg1, arg2, ...) result(val)
         type, intent(in) :: arg1, arg2
        ...
    end function NewFunction_ver2

    ...

end interface NewFunction
```

both functions `NewFunction_ver1` and `NewFunction_ver2` need to be defined in the program as regular functions. However, after the interface is defined they both are available via new name `NewFunction` with their original argument lists.

Example program may look like this

```
program overload_function

  implicit none

  interface add
    real function add_real(x,y) result(val)
      real, intent(in) :: x,y
    end function add_real

    integer function add_integer(x,y) result(val)
      integer, intent(in) :: x,y
    end function add_integer
  end interface add

  integer :: a = 1, b = 2
  real :: c = 1.0, d = 2.0

  print *,'integer : ', add(a,b)
  print *,'real : ', add(c,d)

end program overload_function

integer function add_integer(x,y) result(val)

  implicit none
  integer, intent(in) :: x,y

  val = x + y

end function add_integer

real function add_real(x,y) result(val)

  implicit none
  real, intent(in) :: x,y

  val = x + y

end function add_real
```

where the output is

```
 integer :            3
 real :    3.00000000
```

In this case we have defined functions `add_integer` and `add_real` with common interface `add` . Both of functions have arguments of different type. The function `add` is now available for both `integer` and `real` arguments.

The overload interface can be also defined for a subroutine

```
interface CommonSubroutineName

    subroutine SubroutineName_ver1(arg1, arg2, ...)
        type, intent(in) :: arg1, arg2
        ...
    end subroutine SubroutineName_ver1

    subroutine SubroutineName_ver2(arg1, arg2, ...)
         type, intent(in) :: arg1, arg2
        ...
    end subroutine NewFunction_ver2

    ...

end interface CommonSubroutineName
```

For example

```
program overload_subroutine

  implicit none

  interface print_matrix
    subroutine print_matrix_real(A,n)
      integer, intent(in) :: n
      real, dimension(n,n), intent(in) :: A
    end subroutine print_matrix_real

    subroutine print_matrix_integer(A,n)
      integer, intent(in) :: n
      integer, dimension(n,n), intent(in) :: A
    end subroutine print_matrix_integer
  end interface print_matrix

  integer :: n = 4
  real, allocatable, dimension(:,:) :: A
  integer, allocatable, dimension(:,:) :: B
  integer :: i
```

```
  allocate( A(n,n), B(n,n) )

  A = 0.0
  B = 0

  forall( i=1:n )
    A(i,i) = 1.0
    B(i,i) = 1
  end forall

  print *,'real matrix'
  call print_matrix(A,n)
  print *,'integer matrix'
  call print_matrix(B,n)

  deallocate( A, B )

end program overload_subroutine

subroutine print_matrix_real(A,n)
  implicit none
  integer, intent(in) :: n
  real, dimension(n,n), intent(in) :: A
  integer :: i,j

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

subroutine print_matrix_integer(A,n)
  implicit none
  integer, intent(in) :: n
  integer, dimension(n,n), intent(in) :: A
  integer :: i,j

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

which gives

```
real matrix
  1.00000000       0.00000000       0.00000000       0.00000000
  0.00000000       1.00000000       0.00000000       0.00000000
  0.00000000       0.00000000       1.00000000       0.00000000
  0.00000000       0.00000000       0.00000000       1.00000000
integer matrix
           1           0           0           0
           0           1           0           0
           0           0           1           0
           0           0           0           1
```