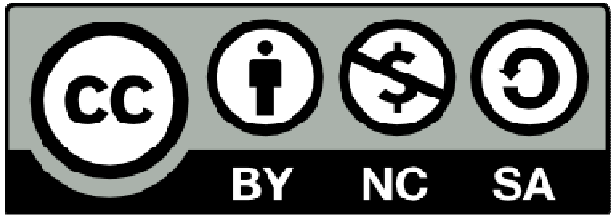
Further Features

Reusing this material



archer

epcc

THE UNIVERSITY OF EDINBURGH

Submodules

- Allows interfaces to be defined in one module and implementation done in another
- Enables splitting and sharing of data and functionality
  - Submodule of a module have access to the all entities and components of that module and any ancestor submodules
- Allows definition of interfaces for others to implement
  - Lets library developers provide some code and let others implement the rest
- Can reduce compilation issues/time

Submodule examples

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

  interface
     module function point_dist(a, b) result(distance)
       type(point), intent(in) :: a, b
       real :: distance
     end function point_dist
  end interface
end module points

submodule (points) points_a
contains
  module procedure point_dist
    distance = sqrt((a%x - b%x)**2 + (a%y - b%y)**2)
  end procedure point_dist
end submodule points_a
```

Parameterised derived types

- Can pass array lengths and kinds parameters when creating an instance of a derived type, i.e.:

```
type :: point(kind)
    integer, kind :: kind
    real(kind) :: x, y
 end type point
 type(point(kind(0.0))) :: a
 type :: 1darray(kind,n)
    integer, kind :: kind=kind(0.0d0) !default value
     integer, len :: n
    real(kind) :: array(n)
 end type 1darray
 type(1darray(10)) :: vectorx
```

Allocation copy

- New intrinsic function to move data and allocation from one variable to another:

```
real, allocatable :: vec(:),tempvec(:)
allocate(tempvec(N))
…
move_alloc(to=vec,from=tempvec)
```

- Types must be compatible
- From target become deallocated after call

Protected attribute

- Can define module variables as `protected`
- Module variable can be viewed outside the module but not changed
  - Protects variable from external change

Interoperability with C

- New module `ISO_C_BINDING`
  - Has the kind types for C intrinsic variables
- Defined types and structures can be inter-operable:

```
TYPE, BIND(C) :: matrix
    ....
END TYPE matrix
```

  - Some restrictions on what can be in the types or structures
- Same can be done for procedures with defined interfaces

Summary

- F2003 and F2008 add further stuff too
  - Co-arrays, etc…
- Lots of nice features
- Language continuing to evolve
- Can still get by with mostly F90 for a lot of program