Programming Techniques 2024-2025

Lecture 5: Custom Operators

Hannu Parviainen

Universidad de la Laguna

October 9, 2024

Custom Operators

- Fortran allows you to define complex derived types.
- But using them directly for calculations leads to cumbersome code (and bugs, lots and lots of bugs)

```fortran
module geometry
  implicit none
  type :: vector2d
     real :: x, y
  end type vector2d
end module geometry
```

```fortran
program test
  use geometry
  implicit none
  type(vector2d) :: a, b, c
  real :: s = 2.3
  a = vector2d(0.0, 1.0)
  b = vector2d(1.0, 1.0)
  c = vector2d(s*a%x+b%x, s*a%y+b%y)
end program test
```

Custom Operators

- Code can be simplified (and made more robust) by using functions.

```fortran
module geometry
  implicit none
  type :: vector2d
     real :: x, y
  end type vector2d

contains
  pure type(vector2d) function sumvv(a, b)
    type(vector2d), intent(in) :: a, b
    sumvv = vector2d(a%x + b%x, a%y + b%y)
  end function sumvv

  pure type(vector2d) function mulrv(a, b)
    real, intent(in) :: a
    type(vector2d), intent(in) :: b
    mulrv = vector2d(a*b%x, a*b%y)
  end function mulrv
end module geometry
```

```fortran
program test
  use geometry
  implicit none
  type(vector2d) :: a, b, c
  real :: s = 2.3
  a = vector2d(0.0, 1.0)
  b = vector2d(1.0, 1.0)
  c = sumvv(mulrv(s, a), b)
end program test
```

Custom Operators

- And defining custom operators can make your code even cleaner.

```
module geometry
  implicit none
  type :: vector2d
     real :: x, y
  end type vector2d

  interface operator(+)
     module procedure sumvv
  end interface

  interface operator(*)
     module procedure mulrv
  end interface

contains
  pure type(vector2d) function sumvv(a, b)
    type(vector2d), intent(in) :: a, b
    sumvv = vector2d(a%x + b%x, a%y + b%y)
  end function sumvv

  pure type(vector2d) function mulrv(a, b)
    real, intent(in) :: a
```

```
program test
  use geometry
  implicit none
  type(vector2d) :: a, b, c
  real :: s = 2.3
  a = vector2d(0.0, 1.0)
  b = vector2d(1.0, 1.0)
  c = s*a + b
end program test
```

How to Define a Custom Operator

Custom operators are defined inside a module using the `operator` keyword.

- Standard operators (+, -, *, /)

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

- A new operator named `.op.`

```
interface operator(.op.)
   module procedure newop
end interface
```

The functions implementing the operators are defined later in the module after the `contains` keyword.