Modules

22.901 Introduction to Computer Programming for Nuclear Engineers

January 30, 2012

Massachusetts Institute of Technology

Outline

- Modules

What are Modules for?

- Modules fulfill multiple purposes
- Used for shared declarations (similar to headers)
- Used for defining global data
- Used for defining procedure interfaces
- Think of them as a high-level interface in your code

And more...

Structure of a Module

```
module <name>
 use definitions
 implicit none
 static data definitions, global to the module
contains
 procedure definitions and interfaces
end module <name>
```

- Keep 1 module to 1 file

How do modules interact?

- Modules can **use** other modules
- Modules may not depend on themselves
- Two modules cannot depend on each other (circular dependency)
- Anything else will most likely work

For example:

```
module bicycle
 implicit none
  type ::  wheel
  integer ::  spokes
  real ::  diameter
  character(len=15) ::  material
 end type wheel
end module bicycle
program bikeshop
  use bicycle
  implicit none
```

Global Data

- Variables in modules define global data
- You need to specify the **save** attribute
- If the whole routine is global data then can just specify one single save at the top under implicit none
- Otherwise use a save attribute for each variable explicitly
- Thus, if a variable is set in one procedure and used in another the save attribute is a must

Procedures in Modules

- Simplest to include procedures in modules
- These procedures go after the **contains**
- Try not to use the same name for local variables in modules and the variables that are listed in the global scope in the file
- There isn't much more to it, just try and organize!

Data and Procedure Accessibility

- It is good practice to limit the scope of data and procedures in modules
- This is a very simple idea
- **private** names accessible only inside the module
- **public** names are accessible by the **use** command
- We will always do the following when creating a module:

```
module somemodule
  any global use commands inside module
  implicit none
  private
  public ::  comma-separate list of public procedures
  module global data (use public command as attribute)
contains
  procedures
end module somemodule
```

Massachusetts Institute of Technology

Partial Inclusion

- When you state **use**, all public data and procedures can be accessed
- To partially include data or procedures use **only**

For example:

```
use bigmodule, only:  errors, invert
```

- Makes only errors and invert visible
- This is very good practice
- Can easily find which procedures and data are used where (i.e. with grep)
- You can only use - only **public** procedures from another module
- This is why partial inclusion and accessiblilty are good ideas

Module Circle I

```
program circle_test

  use class_Circle, only: Circle,circle_print

  implicit none

  type(Circle) :: acircle  ! a circle instance

  ! ask user for radius
  write(*,*) 'Enter circle radius:'
  read(*,*) acircle%radius

  ! print out results
  call circle_print(acircle)

  ! terminate the program
  stop
```

Module Circle II

```
end program circle_test
```

```
module class_Circle

  implicit none
  private
  public :: circle_print

  real :: pi = 3.1415926535897931D0 ! module-wide private
      constant

  type, public :: Circle

    real :: radius ! the radius of a circle
    real :: area   ! the area of a circle
    real :: circum ! the circumference of a circle

  end type Circle
```

Massachusetts Institute of Technology

Module Circle III

```
contains

!
    ================================================================

! CIRCLE_AREA calculates the area of a circle
!
    ================================================================


  function circle_area(this) result(area)

    type(Circle), intent(in)  :: this ! circle instance
    real                      :: area ! the area of the
        circle

    ! calculate the area
    area = pi * this%radius**2
```

Module Circle IV

```
  end function circle_area

!
!-----------------------------------------------------------------------

! CIRCLE_CIRCUM calculate the circumference of a circle
!
!-----------------------------------------------------------------------

  function circle_circum(this) result(circum)

    type(Circle), intent(in) :: this   ! circle instance
    real                     :: circum ! the circumference

    ! calculate circumference
    circum = 2*pi*this%radius

  end function circle_circum
```

Module Circle V

```
!
!=============================================================================

! CIRCLE_PRINT prints information about the circle
!
!=============================================================================


  subroutine circle_print(this)

    type(Circle), intent(inout) :: this ! circle instance

    ! calculate area
    this%area = circle_area(this)

    ! calculate circumference
    this%circum = circle_circum(this)
```

Module Circle VI

```
    ! print results
    write(*,'("The area is:",T30,F0.4,/,"The circumference
        is:",T30,F0.4)')     &
   &              this%area,this%circum

  end subroutine circle_print

end module class_Circle
```