Derived Data Types in Fortran

Bruno Abreu Calfa

Last Update: September 29, 2012

Outline

Outline

Definitions

- In addition to *intrinsic data types* (integer, real, complex, logical, character), Fortran 90/95 and newer allow the programmer to create *user-defined data types*

Definitions

- In addition to *intrinsic data types* (integer, real, complex, logical, character), Fortran 90/95 and newer allow the programmer to create *user-defined data types*
- Motivation: group together all the information about a particular item (similar to `struct` in C/C++, but also allows for object-oriented programming in Fortran 2003 and newer)

Definitions

- In addition to *intrinsic data types* (integer, real, complex, logical, character), Fortran 90/95 and newer allow the programmer to create *user-defined data types*
- Motivation: group together all the information about a particular item (similar to `struct` in C/C++, but also allows for object-oriented programming in Fortran 2003 and newer)
- Use the keyword `TYPE` to create a derived data type following the form below:

```
TYPE [::] type_name
   component definitions
   ...
END TYPE [type_name]
```

Example: Linear System

- Let us define a type to represent a linear system $Ax = b$, where A is an *allocatable* matrix, and x and b are *allocatable* vectors

```
TYPE :: LinearSystem
    INTEGER :: n ! Problem size
    REAL, ALLOCATABLE, DIMENSION(:,:) :: A ! Matrix of
        coefficients
    REAL, ALLOCATABLE, DIMENSION(:) :: x ! Solution
        vector
    REAL, ALLOCATABLE, DIMENSION(:) :: b ! Vector with
        RHS elements
END TYPE LinearSystem
```

- To create variables of the type `LinearSystem`, just do:

```
TYPE (LinearSystem) :: ls1, ls2 ! Two variables of type
    LinearSystem
TYPE (LinearSystem), DIMENSION(10) :: lss ! Array of 10
    LinearSystems
```

Working with Derived Data Types

- Each component (field) of a derived data type is accessed with the *component selector* (`%`)
- Suppose the variable `ls1` is of type `LinearSystem`, then we can set its component `n` to the value `5` as follows:

```
ls1%n = 5
```

- Suppose the variable `lss` is and array of 10 `LinearSystem`s, then we can set the component `b` of its 3rd element to a predefined vector as follows:

```
REAL(KIND = REAL64), DIMENSION(9) :: b

b = (/ 1, 2, 3, 4, 5, 4, 3, 2, 1 /)
ALLOCATE(lss(3)%b(9))
lss(3)%b = b
WRITE(*,*) lss(3)%b
DEALLOCATE(lss(3)%b)
```

Outline

Restricting Access I

- You can restrict access to the contents of a `MODULE` and the components of a `TYPE` by using the keywords `PRIVATE` and `PUBLIC`. Private data are not accessible outside the current program unit (main programs, subroutines, and functions)
- Examples:

```
TYPE :: vector
    PRIVATE
    REAL :: x, y
END TYPE vector
```

The type `vector` as a whole is still available outside the program unit, but its components `x` and `y` *cannot* be accessed separately

Restricting Access II

- An entire derived data type can be declared to be private:

```
TYPE, PRIVATE :: vector
    REAL :: x, y
END TYPE vector
```

The type `vector` is not accessible by any program units that use the module containing it

- Declaring private variables:

```
TYPE :: vector
    REAL :: x, y
END TYPE vector
TYPE (vector), PRIVATE :: vec
```

The type `vector` is public, but the variable `vec` may be used only within the module containing it

Object-Oriented Programming (OOP)

- Starting in Fortran 2003, derived data types can contain procedures and inherit components from other derived data types

Object-Oriented Programming (OOP)

- Starting in Fortran 2003, derived data types can contain procedures and inherit components from other derived data types
- Procedures can be overridden

Object-Oriented Programming (OOP)

- Starting in Fortran 2003, derived data types can contain procedures and inherit components from other derived data types
- Procedures can be overridden
- Polymorphism (objects of various types share a common interface of operations) is allowed

Object-Oriented Programming (OOP)

- Starting in Fortran 2003, derived data types can contain procedures and inherit components from other derived data types
- Procedures can be overridden
- Polymorphism (objects of various types share a common interface of operations) is allowed
- User-defined operators and assignment (also included in Fortran 90/95)