Overloading, abstract classes, and inheritance
Reusing this material

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation may contain images owned by others. Please seek their permission before reusing these images.
Overloading

- Recall that generic interfaces can enable procedure overloading:

```fortran
module maths_functions
  interface my_sum
    module procedure real_sum
    module procedure int_sum
  end interface
contains
function real_sum (a, b)
  implicit none
  real, intent(in) :: a,b
  real_sum = a + b
end function real_sum

function int_sum (a, b)
  implicit none
  integer, intent(in) :: a,b
  int_sum = a + b
end function int_sum
end module
```
Overloading in F2003

- `generic` keyword specifies polymorphism for type-bound procedure
  - polymorphism without interface block
  - Without this, type-bound procedures only resolve to a single method

```f90
GENERIC [, access-spec ] :: generic-spec => binding-name1 [, binding-name2]...

type maths_functions contains
    procedure real_sum
    procedure int_sum
    generic :: sum => real_sum, int_sum
end type
```
Overloading

\textit{generic-spec}

- Interface statement:
  - generic-name, must not be same as other type-binding
  - operator (op)
  - assignment (=)

- Allows for overloading of operators

  \begin{verbatim}
  type maths_functions contains
    procedure real_sum
    procedure int_sum
    generic :: operator(+) => real_sum, int_sum
  end type
  \end{verbatim}
Inheritance

- Can extend types in F2003
  
  \[
  \text{type, extends}(\text{parent\_type\_name}) :: \text{child\_type\_name}
  \]
  
  - Inheritance specified via type extension
  - Parent type is extended by child type
  - Parent type may be a base type
  - Child type has access to all component in base type (and ancestors)

- Child type can add new components
  
  - New variables or procedures

- Includes implicit variable from parent class(es)
Inheritance example

type person
  private
    character(MAXLEN) :: name
    integer :: officeNumber
  contains
    private
      procedure, public :: getName
      procedure, public :: setName
      procedure, public :: getOfficeNumber
      procedure, public :: setOfficeNumber
  end type person

type, extends(person) :: manager
  contains
    private
      procedure, public :: addPerson
      procedure, public :: removePerson
      procedure, public :: movePerson
  end type
Inheritance example

type(manager) :: bob
type(person) :: fred

write(*,*) bob%getName()
write(*,*) bob%person%getName()

write(*,*) fred%getName()
call bob%movePerson(fred,35)  ✓
call fred%movePerson(bob,46)  ✗
Abstract classes

• Can define **abstract classes and deferred procedures**
  • Define data
  • Define procedures and interfaces
  • Define implement procedures
  • Define procedures to be implement by further classes

• Abstract class cannot be instantiated or allocated
  • Can be used for class declaration in methods
    • Important for type hierarchies
Abstract class example

type, abstract :: individual
  private
    character(MAXLEN) :: name
    integer :: officeNumber
  contains
    private
    procedure, non_overridable, public :: getName
    procedure, non_overridable, public :: setName
    procedure, non_overridable, public :: getOfficeNumber
    procedure, non_overridable, public :: setOfficeNumber
    procedure(printStuff), deferred :: print
end type individual
abstract interface
  subroutine printStuff(self)
  import :: individual
  class(individual), intent(in) :: self
  end subroutine printStuff
end interface
Abstract class example

type, extends(individual):: person
contains
   private
   procedure :: print => printPerson
end type person

type, extends(person) :: manager
contains
   private
   procedure :: movePerson()
...
end type manager
Summary

• F2003 allows derived types to extend other derived types
  • Enables OO inheritance

• Abstract classes can be defined
  • Enables interface/specification of code without requiring implementation

• Operators and procedures can be overloaded
  • Same name used to call different procedures/operations based on the arguments passed
Exercise

• Extend your previous examples with operator overloading and class hierarchies (see the exercise sheet).
• Do the same for the percolate example.