Title: Scientific Computation (PHYS 608) - Lecture 2

Date: Oct 27, 2009 10:30 AM

URL: http://pirsa.org/09100178

Abstract:

Internal Functions

Internal Functions

Program

Internal Functions

Program Test
real :: y

Internal Functions

```
Program Test
   real :: y
   integer :: n
```

Internal Functions

Program Test
real :: y
integer :: n

contains

end Program Test

Internal Functions

```
Program Test
    real :: y
    integer :: n



contains
    function fun(x) result(value)
        :
        :
    end function fun
End Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n

    y = fun(1.0)

co

   ...on fun(x) result(value)

                          <- inlined

   ...n
```

Internal Functions

```
Program Test
    real :: y
    integer :: n
    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        :
    end function fun          <- inlined
end Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n

    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        value = x                    ← inlined
    end function fun
End Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n

    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        value = x **          ← inlined
    end function fun
End Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n

    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        value = x**n
    end function fun
End Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n
    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        value = x**n          <- inlined
    end function fun
End Program Test
```

Internal Functions

```
Program Test
    real :: y
    integer :: n
    y = fun(1.0)

contains
    function fun(x) result(value)
        :
        value = x**n
    end function fun
End Program Test
```

← inlined

Subroutines : functions

Subroutines : functions with no return value

subroutine (x, y, z)

Subroutines : functions with no return value

subroutine sub(x,y,z)

Subroutines : functions with no return value

```
subroutine sub(x,y,z)
    real, intent(IN) :: x
    real, intent(OUT) :: y
    real, intent(INOUT) :: z
    :
    :
end subroutine
```

Subroutines : functions with no return value

```
subroutine sub(x,y,z)
    real, intent(IN) :: x
    real, intent(OUT) :: y
    real, intent(INOUT) :: z
        :
end subroutine sub
```

Subroutines : functions with no return value

```
subroutine sub(x,y,z)
    real, intent(IN) :: x
    real, intent(OUT) :: y
    real, intent(INOUT) :: z
    ...
end subroutine sub

Program Test
    ...
    call sub(a,b,c)
end Program Test
```

Program Test
real :: y
integer :: n
y = fun(1.0)

contains
functi... result(value)

← inlined

en...

Call by value / Call by Reference

Subroutines : functions with no return value

```
subroutine sub(x,y,z)
    real, intent(IN) :: x
    real, intent(OUT) :: y
    real, intent(INOUT) :: z
    integer :: i

end subroutine sub
```

```
Program Test
    ⋮
    call sub(a,b,c)

end Program Test
```

Call by value / Call by Reference

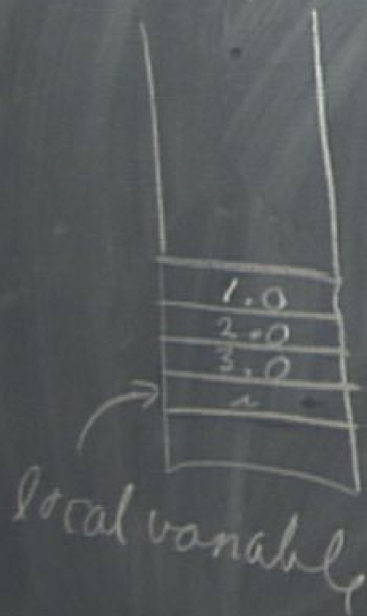
Subroutines : functions with no return value

```
subroutine sub(x,y,z)
   real, intent(IN) :: x
   real, intent(OUT) :: y
   real, intent(INOUT) :: z
   integer :: i

end subroutine sub
```

```
Program Test
   :
   call sub(1.0,2.0,3.0)

end Program Test
```

Call by value / Call by Reference

Call by value / Call by Reference

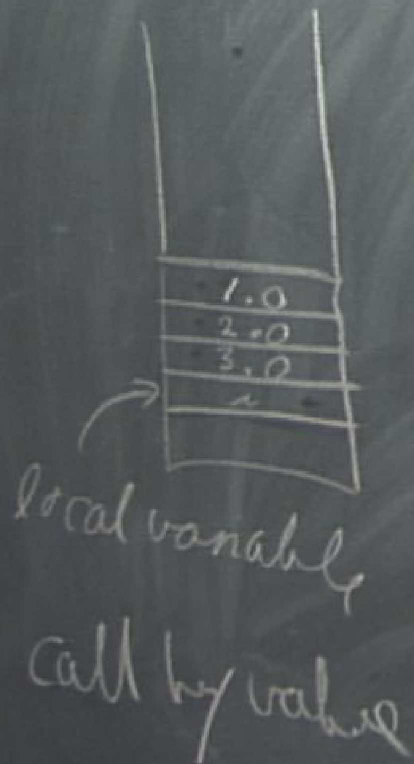


ubroutines : functions with no return value

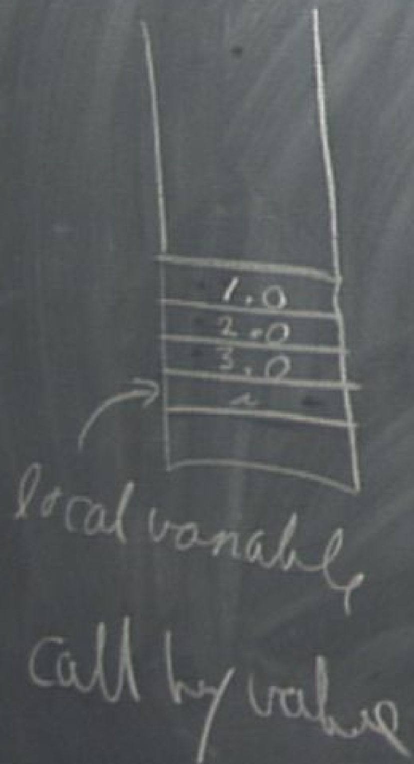
```
subroutine sub(x,y,z)
  real, intent(IN) :: x
  real, intent(OUT) :: y
  real, intent(INOUT) :: z
  integer :: i

end subroutine sub
```

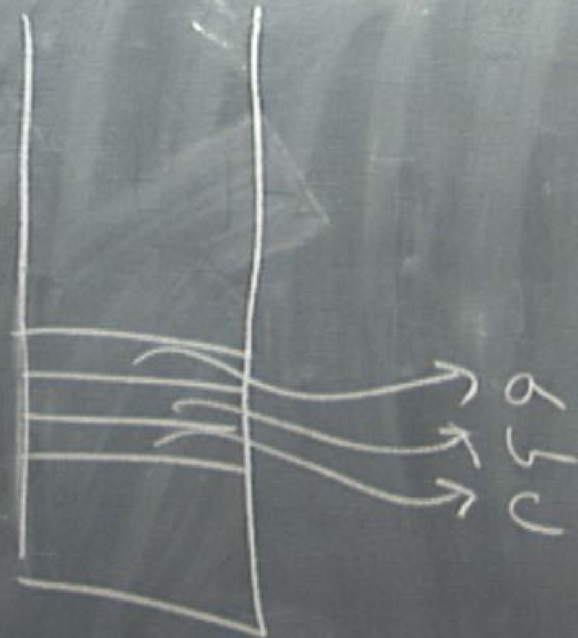
```
Program Test

a=1.0
b=2.0
c=3.0
call sub(a,b,c)

end Program Test
```

Call by value / Call by Reference

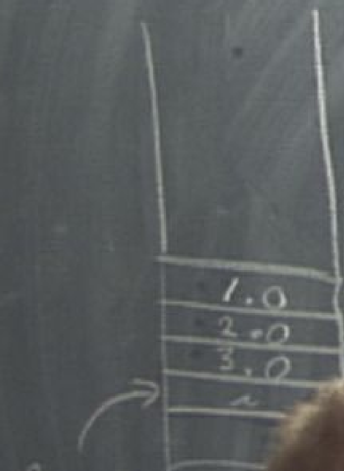


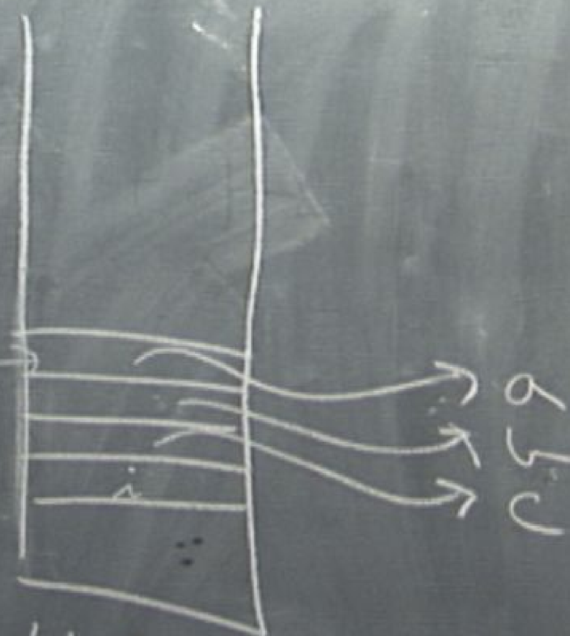
Call by value / Call by Reference



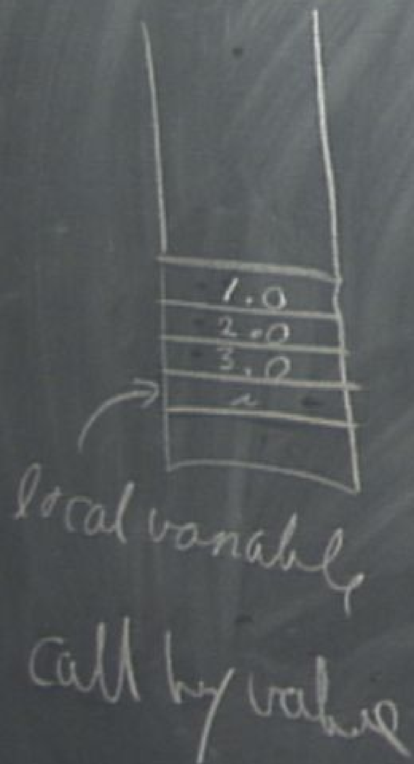


Call by value / Call by Reference

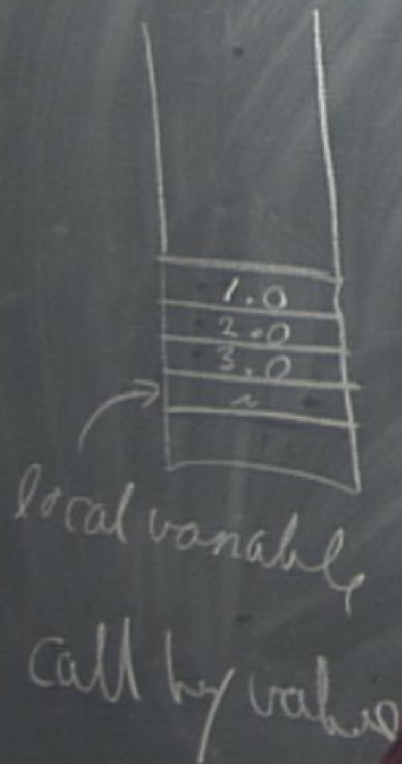


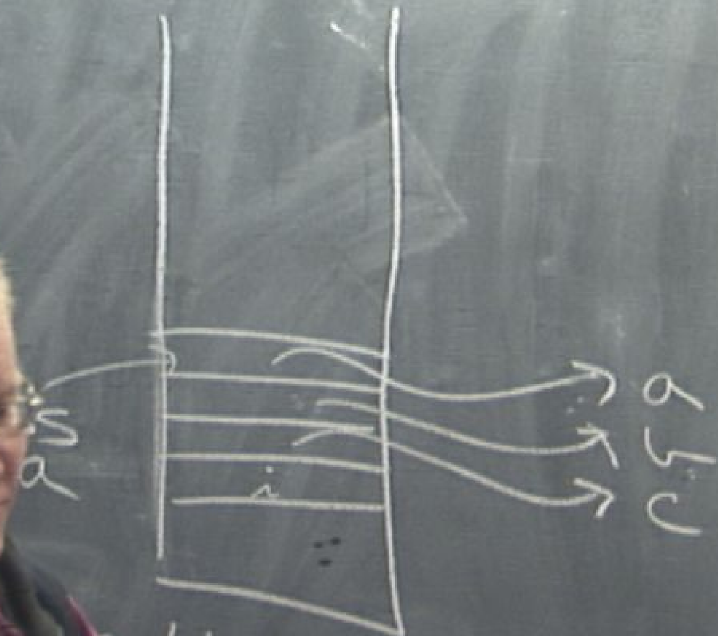


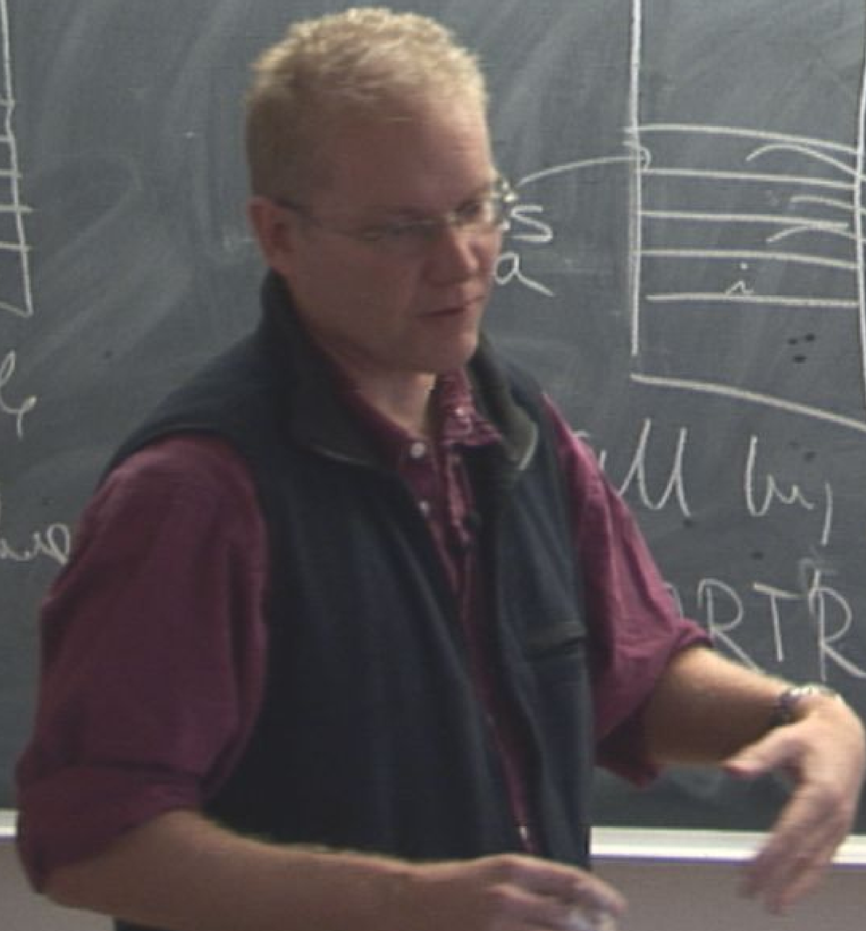
Call by value / Call by Reference



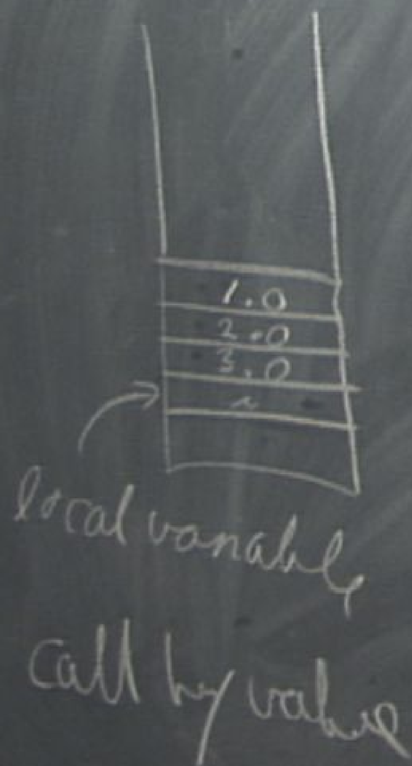
Address of a


1.0
2.0
3.0
local variable
call by value

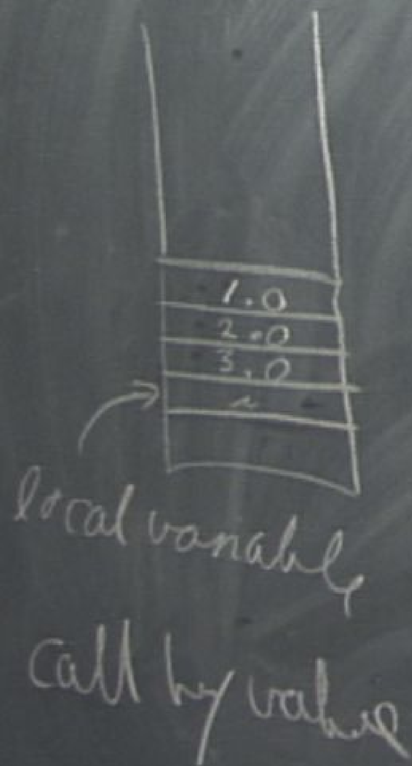

a
b
c

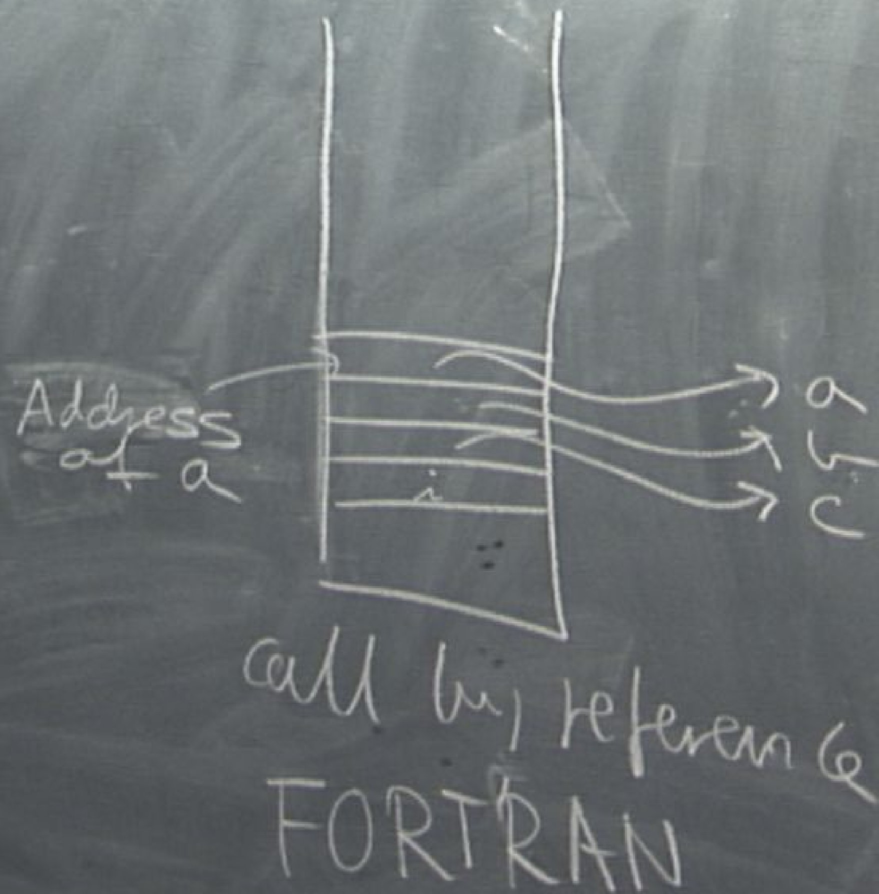


Call by value / Call by Reference



Call by value / Call by Reference





Subroutines : functions with no return value

```
subroutine  sub(x,y,z)
    real, intent(IN)    :: x
    real, intent(OUT)   :: y
    real, intent(INOUT) :: z
    integer             :: i

end subroutine sub
```

```
Program Test

  a=1.0
  b=2.0
  c=3.0
call sub(a,b,c)

end Program Test
```

What happens to i when sub finishes

What happens to i when sub finishes

- i goes "out of scope"

What happens to i when sub finishes

- i goes "out of scope" formally undefin

What happens to i when sub finishes

- i goes "out of scope" formally undefined

What happens to i when sub finishes

- i goes "out of scope" formally undefined

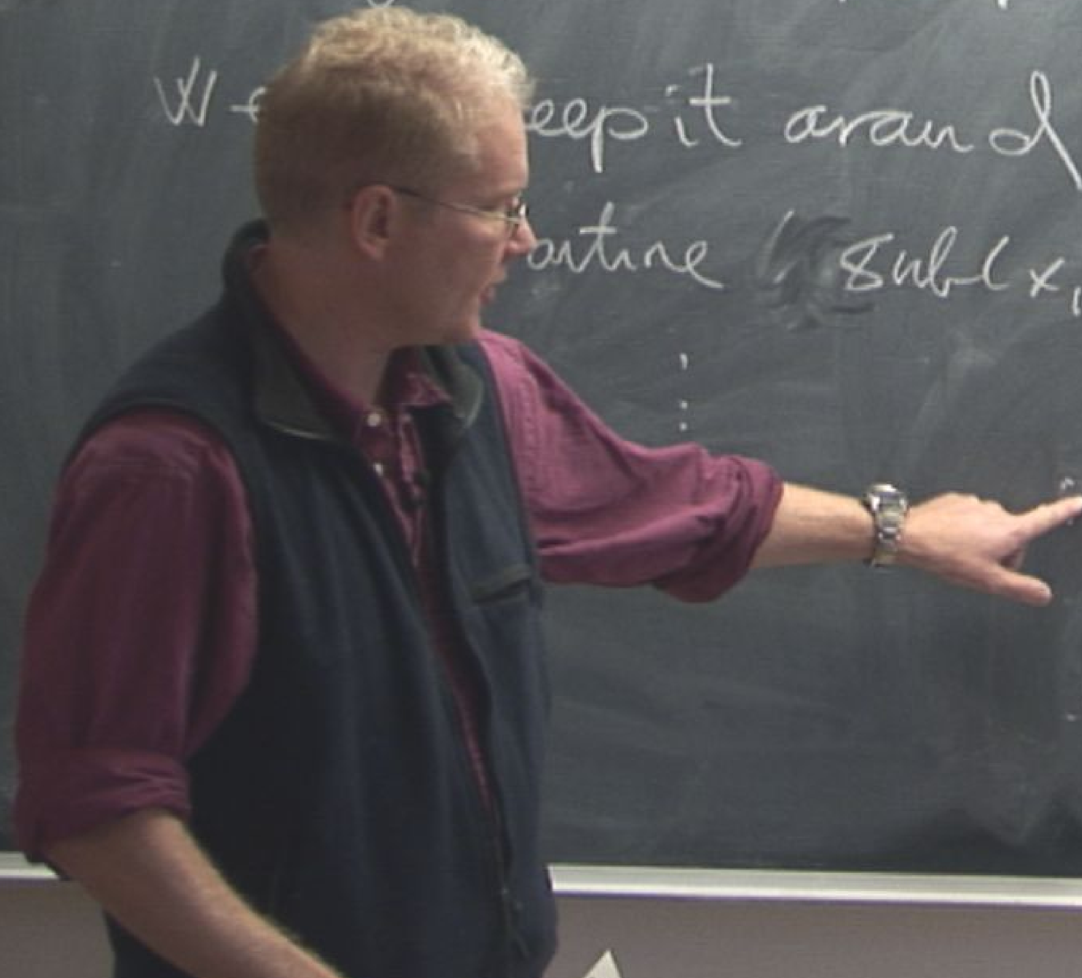
We can keep it around

What happens to i when sub finishes

- i goes "out of scope" formally undefined

W... keep it around

...outine sub(x,y,z)



What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

subroutine sub(x, y, z)

⋮

integer, :: i

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

```
subroutine sub(x,y,z)
   :
   :
integer, save :: i=0
```

What happens to i when sub finishes

- i goes "out of scope" formally undefined
- can keep it around

subroutine sub(x,y,z)

:

integer, save :: i=0

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

subroutine sub(x, y, z)

⋮

integer, save :: i = 0

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

subroutine sub1(x, y, z)

⋮

integer, save :: i=0 ← useful for counters

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

subroutine sub(x,y,z)

:

integer, save :: i=0 ← useful for counters

Sub

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

```
subroutine sub1(x,y,z)
  :
  :
integer, save :: i=0   ← useful for counters
```

Subr

What happens to i when sub finishes

- i goes "out of scope" formally undefined

We can keep it around

subroutine sub(x, y, z)

⋮

integer, save :: i = 0 ← useful for counters

Sub

Scope rules

Program Test
real :: x, a, b

Subr

Scope rules

Program Test
real :: a, b

b = 7.
a = fun

end P

function fun(x) result(value)

Subr

Scope rules

Program Test
real :: x, a, b

b = 7.0
a = fun(x)

end Program

function fun(x) result(value)
real :: x
integer :: i

Subr

Scope rules

```
Program Test
  real :: x, a, b

  b = 7.0
  a = fun(x)

end Program
```

```
function fun(x) result(value)
  real :: x
  integer :: i

  x = sin(b)

end function fun
```

Sub

Scope rules

```
Program Test
   real    x, a, b

   b
   a         (x)

en
          n
```

```
function fun(x) result(value)
   real :: x
   integer :: i

   x = sin(b)

end function fun
```

(The line `x = sin(b)` is crossed out on the board.)

Sub

Scope rules

```
Program Test
  real :: x, a, b

  b = 7.0
  a = fun(x)

end Program
```

```
function fun(x) result(value)
  real :: x
  integer :: i

  x = sin(b)    [crossed out]

end function fun
```

→ b is out of scope

Sub[...]

Scope rules

```
Program Test
  real :: x, a, b

  b = 7.0
  a = fun(x)

end Program
```

Flat scope rules

```
function fun(x) result(value)
  real :: x
  ...er :: i
```

... scope

Sub...

Scope rules

```
Program Test
  real :: x, a, b

  b = 7.0
  a = fun(x)

end Program
```

```
function fun(x) result(value)
  real :: x
  integer :: i

  x = sin(b)   (crossed out)

end function fun
```

b is out of scope

Sub

Flat scope rule : variables can only be accessed

Scope rules

```
Program test
   real :: x, a, b

   b = 7.0
   a = ...(x)

end ...
```

```
function fun(x) result(value)
   real :: x
   integer :: i

   x = sin(b)

end function fun
```

b is out of scope

Scope rule : variables can only be accessed from within the defining prog unit

Subroutines : functions with no return

```
subroutine sub(x,y,z)
   real, intent(IN) :: x
   real, intent(OUT) :: y
   real, intent(INOUT) :: z
   integer :: i

end subroutine sub
```

```
Program ...
   a = 1.0
   b = 2.0
   c = 3.0
   call sub(...

end Prog...
```

e)

```
subroutine sub(x,y,z)
    real, intent(IN) :: x
    real, intent(OUT) :: y
    real, intent(INOUT) :: z
    integer :: i

end subroutine sub
```

```
Program Test
  a=1.0
  b=2.0
  c=3.0
call sub(a,b,c)

end Program Test
```

(Exception: internal function (subroutine)

ssall from within the defining prog unit

Typing / Type Checking

real :: x

integer :: i

⋮

x = i

Typing / Type Checking

real :: x

integer :: i

x = i

Typing / Type Checking

real :: x

integer :: i

x = i ← No problem

Typing / Type Checking

real :: x

integer :: i

x = i ← No problem

Permissive type check

Typing / Type Checking

real :: x

integer :: i

x = i ← No problem

Permissive type checking

Typing / Type Checking

real :: x

integer :: i

⋮

Permissive type checking

$x = i$ } ← No problem

$i = x$ } → implicit type conversion

Always do explicit type conversions

Always do explicit type conversions

$$x = real(i)$$

Typing / Type Checking

real :: x

integer :: i

Permissive type checking

$x = i$ ← No problem

$i = x$ → implicit type conversion

Typing / Type Checking

real :: x
integer :: i

Permissive type checking

⋮

$x = 2.01$
$i = x$

← No problem
→ implicit type conversion

Typing / Type Checking

real :: x

integer :: i

⋮

x = 2.01

i = x

Permissive type checking

← No problem

→ implicit type conversion

Typing / Type Checking

real :: x
integer :: i
⋮
x = 2.01
i = x

} ← No problem
} → implicit type conversion

Permissive type checking

Typing / Type Checking

real :: x

integer :: i

Permissive type checking

$x = 2.01$

$i = x$

← No problem

→ implicit type conversion

Typing / Type Checking

real :: x

integer :: i

⋮

x = 2.01 } ← No problem

i = x } → implicit type conversion

Permissive type checking

Modules

Module My_Mod

Modules

```
Module My_Mod
   implicit none
   type measurement
        :
        :
     end type
```

Modules

```
Module My_Mod
    implicit none
    type measurement
        :
    end type
contains
    function fun(x) result(value)
        :
    end function
end Module My_Mod
```

Program te

Modules

```
Module My_Mod
  implicit none
  type measurement
     :
  end type
contains
  function fun(x) result(value)
     :
  end function
end Module My_Mod
```

```
Program test
  USE MY_MOD
```

Modules

```
Module My_Mod
    implicit none
    type measurement
        :
    end type
contains
    function fun(x) result(value)
        :
    end function
end Module My_Mod
```

```
Program test
    USE MY_MOD

    a = fun(x)

end Program
```

Modules

```
Module  My_Mod
    implicit none
    type  measurement
        :
    end type
    real :: global_a
  contains
    function  fun(x)  result(value)
       :
    end function
  end Module  My_Mod
```

Implicit interface declaration

```
Program test
  USE  MY_MOD

  a = fun(x)

end Program
```

Modules

```
Module My_Mod
    SAVE
    implicit none

    type measurement
        :
    end type
    real :: global_a
  contains
    function fun(x) result(value)
        :
    end function
end Module My_Mod
```

Implicit interface declaration

global variable

```
Program test
  USE MY_MOD

  a = fun(x)

end Program
```

Modules

Module My_Mod
SAVE
implicit none

type measurement
:
end type
real :: global_a
contains
Implicit interface declaration { function fun(x) result(value)
:
end function
end Module My_Mod

global variable

Program test
USE MY_MOD

a = fun(x)

end Program

Modules

Module My_Mod
SAVE
implicit none

private

contains
Implicit interface declaration { function fun(x) result(value)
end function
end Module My_Mod

Program test
USE MY_MOD

...un(x)

en...

Modules

```
Module My_Mod
    SAVE
    implicit none
    public :: global_i
    private :: fun

    integer :: global_i

  contains
Implicit    function fun(x) result(value)
interface       :
declaration end function
  end Module My_Mod
```

```
Program test
  USE MY_MOD

  a = fun(x)

end Program
```

Modules

Module My_Mod
SAVE
implicit none
public :: global_i
private :: fun

integer :: global_i

contains
Implicit interface declaration
function fun(x) result(value)
:
end function
end Module My_Mod

Program test
USE MY_MOD

a = fun(x)

end Program

Modules

```
Module My_Mod
    SAVE
    implicit none
Default => public :: global_i
           private :: fun

    integer :: global_i

  contains
Implicit  { function fun(x) result(value)
interface {    :
declaration { end function
  end Module My_Mod
```

```
Program test
  USE MY_MOD

  a = fun(x)

end Program
```

Modules

```
Module   My_Mod
    SAVE
    implicit none
Default => public :  global_i
    private  :  fun

    integer :: global_i

  contains
Implicit   { function fun(x) result(value)
interface  {          :
declaration{ end function
  end Module My_Mod
```

```
Program test
   USE MY_MOD

   a = fun(x)

end Program
```

Generic Procedures

Generic Procedures

Generic Procedures

interface

Generic Procedures

interface swap

Generic Procedures

```
interface swap

end interface
```

Generic Procedures

interface sort

end interface

Generic Procedures

Module SORTMOD

interface sort

end interface

Generic Procedures

```
Module SORTMOD
  interface sort


  end interface
contains
    subroutine int_sort(  )
```

Generic Procedures

```
Module SORTMOD
  interface sort



  end interface
contains
    subroutine int_sort(   )
      :
    subroutine real_sort(   )
      :
end module SORTMOD
```

Generic Procedures

```
Module SORTMOD
  interface sort
      module procedure int_sort
      module procedure real_sort
  end interface
contains
    subroutine int_sort( )
        :
    subroutine real_sort( )
        :
end module SORTMOD
```

Generic Procedures

```
Module SORTMOD
  interface sort
      module procedure int_sort
      module procedure real_sort
  end interface
contains
     subroutine int_sort(   )
       :
     subroutine real_sort(  )
       :
end module SORTMOD
```

Generic Procedures

```
Module SORTMOD
  interface sort
    module procedure int_sort
    module procedure real_sort
  end interface
contains
  subroutine int_sort( )
    :
  subroutine real_sort( )
    :
end module SORTMOD
```

Program Test
USE SORTMOD

Program Test
USE SORTMOD
real :: A(10)
[illegible]r :: K(10)

```
Program Test
  USE SORTMOD
  real    :: A(10)
  integer :: K(10)

  call Sort(A)
```

Generic Procedures

```
Module SORTMOD
  interface sort
     module procedure int_sort
     module procedure real_sort
  end interface
contains
     subroutine int_sort( )
        :
     subroutine real_sort( )
        :
end module SORTMOD
```

Generic Procedures

```
Module SORTMOD
  interface sort
      module procedure int_sort
      module procedure real_sort
  end interface
contains
     subroutine int_sort(   )
       :
     subroutine real_sort(  )
       :
end module SORTMOD
```

Generic Procedures

```
Module SORTMOD
  interface sort
      module procedure int_sort
      module procedure [illegible]
  end interface
contains
    subroutine int_sort(   )
      :
    subroutine real_sort(  )
      :
end module SORTMOD
```

```
Program Test
    USE SORTMOD
    real :: A(10)
    integer :: K(10)

    call Sort(A)
```

- Any two procedures must be distinguishable by reference to their non-optional arguments

```
Program Test
  USE SORTMOD
  real    :: A(10)
  integer :: K(10)

  call Sort(A)
```

- Any two procedures must be distinguishable by reference to their non-optional arguments

```
Program Test
    USE SORTMOD
    real    :: A(10)
    integer :: K(10)

    call Sort(A)
```

- Any two procedures must be distinguishable by reference to their non-optional arguments; at least one must be of different type or position

interface operator (+)

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

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

↖ operator overloading

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

↖ operator overloading

interface operator (+)
    module procedure add
end interface

↖ operator overloading

interface Assignment (=)

end interface