Functions in Fortran

Victor Eijkhout and Carrie Arnold and Charlie Dey

Fall 2017

Subprogram basics

Subprograms in contains clause

```
Program foo
  < declarations>
  < executable statements >
  Contains
    < subprogram definitions >
End Program foo
```

Subroutines

```
subroutine foo()
  implicit none
  print *,"foo"
  if (something) return
  print *,"bar"
end subroutine foo
```

- Looks much like a main program
- Ends at the end, or when `return` is reached
- Activated with

```
call foo()
```

Function definition and usage

- Return type, keyword `function`, name, parameters
- Function body has statements
- Result is returned by assigning to the function name
- Use: `y = f(x)`

Function example

Code:

```
program plussing
  implicit none
  integer :: i
  i = plusone(5)
  print *,i
contains
  integer function plusone(invalue)
    implicit none
    integer,intent(in) :: invalue
    plusone = invalue+1
  end function plusone
end program plussing
```

Output:

```
6
```

Why a 'contains' clause?

```
Program ContainsScope
  implicit none
  call DoWhat()
end Program ContainsScope

subroutine DoWhat(i)
  implicit none
  integer :: i
  i = 5
end subroutine DoWhat
```

Warning only, crashes.

```
Program ContainsScope
  implicit none
  call DoWhat()
contains
  subroutine DoWhat(i)
    implicit none
    integer :: i
    i = 5
  end subroutine DoWhat
end Program ContainsScope
```

Error, does not compile

Why a 'contains' clause, take 2

Code:

```
Program ContainsScope
  implicit none
  integer :: i=5
  call DoWhat(i)
end Program ContainsScope

subroutine DoWhat(x)
  implicit none
  real :: x
  print *,x
end subroutine DoWhat
```

Output:

```
7.00649232E-45
```

At best compiler warning if all in the same file
For future reference: if you see very small floating point numbers, maybe you have made this error.

Exercise 1

Write a program that asks the user for a positive number; negative input should be rejected. Fill in the missing lines in this code fragment:

Code:

```
program readpos
  implicit none
  real(4) :: userinput
  print *,"Type a positive number:"
  userinput = read_positive()
  print *,"Thank you for",userinput
contains
  real(4) function read_positive()
    implicit none
  /* ... */
  end function read_positive
end program readpos
```

Output:

```
Type a positive number:
No, not  -5.00000000
No, not   0.00000000
No, not  -3.14000010
Thank you for   2.48000002
```

Subprogram arguments

Arguments are defined in subprogram body:

```
subroutine f(x,y,i)
  implicit none
  integer,intent(in) :: i
  real(4),intent(out) :: x
  real(8),intent(inout) :: y
  x = 5; y = y+6
end subroutine f
! and in the main program
call f(x,y,5)
```

Parameter passing

- Everything is passed by reference.
- Use `in`, `out`, `inout` qualifiers to clarify semantics to compiler.
- Terminology: Fortran talks about 'dummy' and 'actual' arguments. Dummy: in the definition; actual: in the calling program.

Intent checking

Compiler checks your intent against your implementation. This code is not legal:

```
subroutine ArgIn(x)
  implicit none
  real,intent(in) :: x
  x = 5 ! compiler complains
end subroutine ArgIn
```

Why intent checking?

Allow compiler optimizations:

```
x = f()
call ArgOut(x)
print *,x
```

Call to `f` removed

```
do i=1,1000
  x = ! something
  y1 = .... x ....
  call ArgIn(x)
  y2 = ! same expression as y1
```

`y2` is same as `y1` because `x` not changed

Exercise 2

Take your prime number testing function `is_prime`, and use it to write program that prints multiple primes:

- Read an integer `how_many` from the input, indicating how many (successive) prime numbers should be printed.
- Print that many successive primes, each on a separate line.
- (Hint: keep a variable `number_of_primes_found` that is increased whenever a new prime is found.)

Modules

Module definition

```
Module FunctionsAndValues
  implicit none

  real(8),parameter :: pi = 3.14

contains
  subroutine SayHi()
    print *,"Hi!"
  end subroutine SayHi

End Module FunctionsAndValues
```

Module use

```
Program ModProgram
  use FunctionsAndValues
  implicit none

  print *,"Pi is:",pi
  call SayHi()

End Program ModProgram
```

Exercise 3

Write a module `PointMod` that defines a type `Point` and a function `distance` to make this code work:

```
use pointmod
implicit none
type(Point) :: p1,p2
real(8) :: p1x,p1y,p2x,p2y
read *,p1x,p1y,p2x,p2y
p1 = point(p1x,p1y)
p2 = point(p2x,p2y)
print *,"Distance:",distance(p1,p2)
```

Put the program and module in two separate files and compile thusly:

```
ifort -g -c pointmod.F90
ifort -g -c pointmain.F90
ifort -g  -o pointmain pointmod.o pointmain.o
```