

Fortran (part 2)

1. Control constructs:
 1. go to statement
 2. if clause
 3. do loop
 4. case construct
2. functions
3. subroutines

A few comments on constructs

- the general form is a block construct
- key words at beginning and end of each block
- block constructs might be nested
 - example: if statement in do loop
 - example: do loop in if statement
 - example: do loop in do loop, etc.

goto statement

- Statement: `go to label`
- Purpose: execution of program jumps to statement label `label`
- Example:

```
x = y + 3.0
goto 2
x = x + 2.0
2 write(*,*) x
```

- Note: A `goto` statement must never branch into a block construct, but it can be used to exit from a block construct.

if statement and construct

- **if statement:** tests logical expression and executes single statement
- **Form:** `if(scalar logical expression) action-stmt`
 - **Example:** `if (a .gt. b) x=a`
- **if construct:** allows the execution more than one statement, or the execution of alternative statements depending on alternative conditions
- **Form:** `if(scalar logical expression) then`
`block`
`end if`
- **Example:** `if (a .gt. b) then !exchange a and b`
`cp = a`
`a = b`
`b=cp`
`endif`

if construct (cont'd)

- **if then – else – endif**
- Form:

```
if( scalar logical expr ) then
    block1
else
    block2
endif
```
- **if then – elseif then – [elseif then] – [else] – endif**
- Form:

```
if( scalar logical expr ) then
    block1
else if( scalar logical expr ) then
    block2
else
    block3
endif
```
- **Example:**

```
if (x*x + y*y /= 0.0) then      !explicit construction of atan2
    if (y == 0.0) then
        phi=sign(y) * pi/2.0
    else
        phi=atan(x/y)
    endif
else
    phi=0.0      !my convention
endif
```

case construct

- **Purpose:** select between various options by not using if construct
- **Form:**

```
select case (expr)
  case selector
    block 1
  case selector
    block 2 ...
end select
```
- **Example:**

```
select case (num)           ! num is of type integer
  case(:-1)                 ! all values below 0
    n_sign = -1
  case(0)
    n_sign = 0
  case(1:)                   ! all values above 0
    n_sign = 1
end select
```
- **Note:** all case selectors have to be non overlapping

```
case(3:11)
case(1, 2, 12:17, 21)
case default
```

do construct

- **Purpose:** do statement gives the ability to iterate
- **Form:** `do i = starti, endi [,increment]`
`block`
`enddo` !old form uses continue
- **Example:** `sum = 0.0`
`do i = 1, n`
`sum = sum + a(i)`
`enddo`
- **Use `exit` statement to exit loop:**
`sum = 0.0`
`do i = 1, n`
`sum = sum + a(i)`
`if(sum.gt.100.0) exit`
`enddo`
- **Use `cycle` statement to skip execution of rest of present iteration:**
`sum = 0.0`
`do i = 1, n`
`do j = 1, m`
`if(i == j) cycle !don't need element j`
`sum = sum + a(j,i)`
`enddo ! j loop`
`enddo !i loop`
- **Note:** Do loops can be nested!

functions

Purpose: functions are passed some arguments, and they return a data type. They do not change their arguments.

```
Example: program main
    implicit none
    real(4):: scapro1, scapro2, x(10), y(10), a, b

    a = scapro(x,y,n)
    b = scapro(x,y)
end program main
!- - - - -
function scapro1(u, v, m)
    implicit none
    integer:: m
    real(4):: u(m), v(m), scapro1

    integer:: j
    scapro1=0.0
    do j =1, m
        scapro1=scapro1 + u(j)*v(j)
    enddo
end function scapro1
!- - - - -
function scapro2(u,v)
    implicit none
    real(4):: scapro2, u(*), v(*)

    ! same instructions as above

end function scapro2
```

subroutines

Purpose: Subroutines are program units that receive arguments, and change them.

Example:

```
program main
  implicit none
  real(4):: x(10), y(10), sum(10)

  call vec_sum(x,y,sum)
end program main
!-----
subroutine vec_sum(x,y,sum)int
  implicit none
  real(4), intent(in):: x(*), y(*)
  real(4), intent(out)::sum(*)

  integer:: j
  do j=1, size(x)
    sum(j) = x(j) + y(j)
  enddo
end subroutine vec_sum
```

Alternative: array valued function

```
function vec_sum(x,y)
  implicit none
  real(4), dimension(:) :: x, y
  real(4), dimension(size(x)):: vec_sum
  integer:: n
  n=size(x)
  vec_sum(1:n) = x(1:n) + y(1:n)
end function
```