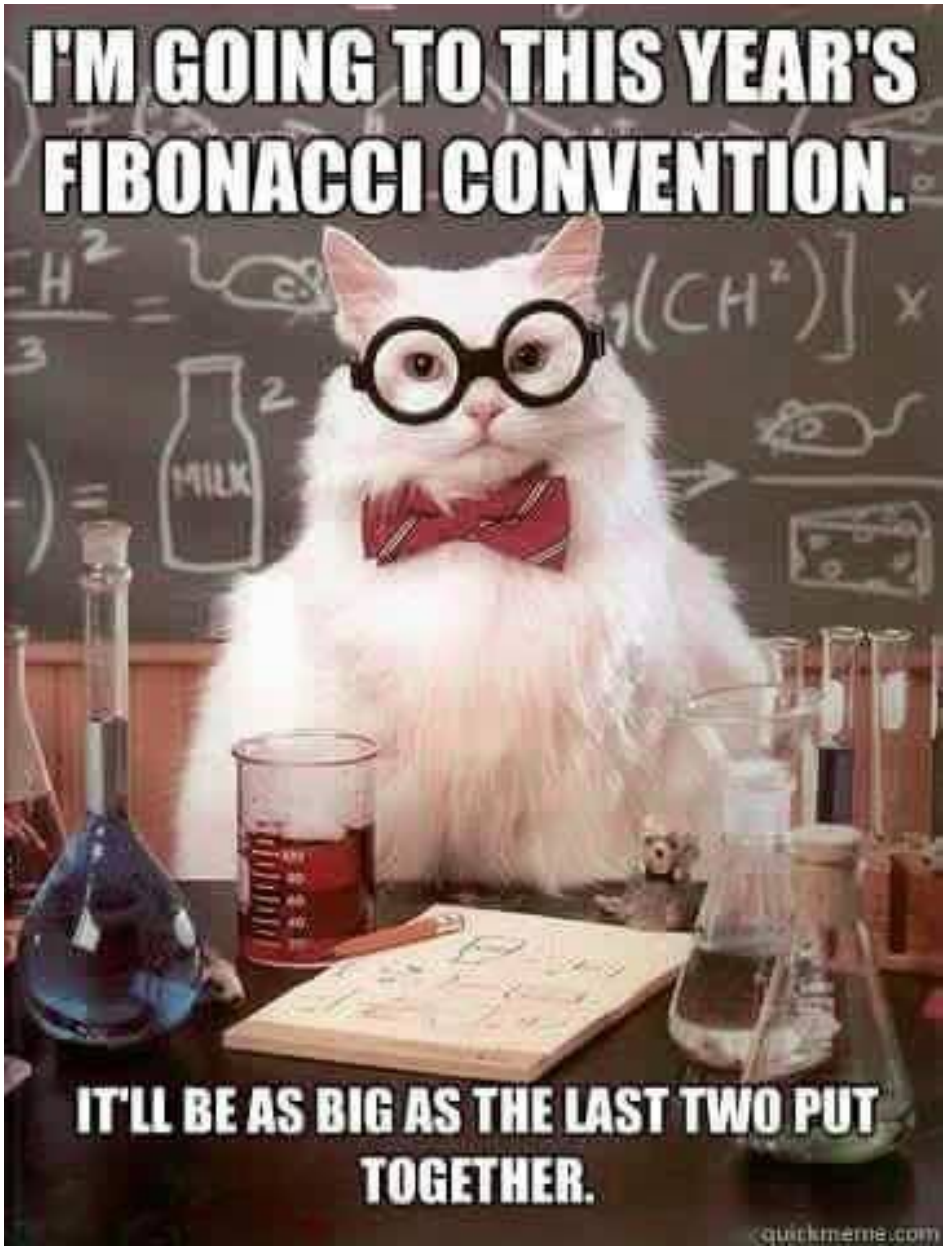
Programming and Modelling (week 38)

C. Thieulot

Institute of Earth Sciences

September 2017


I'M GOING TO THIS YEAR'S FIBONACCI CONVENTION.
MILK
IT'LL BE AS BIG AS THE LAST TWO PUT TOGETHER.
quickmeme.com

Solve carefully...

230 – 220 x 0.5

You probably won't believe it, but the answer is 5!

feedback (1)

- a variable cannot have the same name as the program:

```
program factorial
implicit none

integer factorial
```

→ forbidden

- comments are placed before the instructions
- think about indentation

feedback (1)

- a variable cannot have the same name as the program:

```
program factorial
implicit none

integer factorial
```

→ forbidden

- comments are placed before the instructions
- think about indentation
- when using `cos`, `sin`, etc ... do not declare them as real

feedback (1)

- a variable cannot have the same name as the program:

```
program factorial
implicit none

integer factorial
```

→ forbidden

- comments are placed before the instructions
- think about indentation
- when using `cos`, `sin`, etc ... do not declare them as real
- when declaring an array, its length must be a well defined constant, i.e. a NUMBER

feedback (2)

- equally share the typing/coding

feedback (2)

- equally share the typing/coding
- arrays in do-loops, do not forget `array(i)`

feedback (2)

- equally share the typing/coding
- arrays in do-loops, do not forget `array(i)`
- do not write the whole array inside a do-loop

feedback (2)

- equally share the typing/coding
- arrays in do-loops, do not forget `array(i)`
- do not write the whole array inside a do-loop
- you must know by heart the exact syntax of an `if` statement

feedback (2)

- equally share the typing/coding
- arrays in do-loops, do not forget `array(i)`
- do not write the whole array inside a do-loop
- you must know by heart the exact syntax of an `if` statement
- when dealing with real numbers, do not forget `1.`, `-7.`

feedback (2)

- equally share the typing/coding
- arrays in do-loops, do not forget `array(i)`
- do not write the whole array inside a do-loop
- you must know by heart the exact syntax of an `if` statement
- when dealing with real numbers, do not forget `1.`, `-7.`
- use keyboard more (shortcuts), use mouse less

feedback (3)

Methodology

- more preparation at home:

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"
  "use formula to compute height of bullet"

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"
  "use formula to compute height of bullet"
  $\Rightarrow$ @home : physics & math ; @univ: computer science

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"
  "use formula to compute height of bullet"
  $\Rightarrow$ @home : physics & math ; @univ: computer science
- think, then code

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"
  "use formula to compute height of bullet"
  $\Rightarrow$ @home : physics & math ; @univ: computer science
- think, then code

, then code

feedback (3)

Methodology

- more preparation at home:
  "use trigonometric functions to compute center of circle"
  "use formula to compute height of bullet"
  $\Rightarrow$ @home : physics & math ; @univ: computer science
- think, then code
  

  , then code
- write code progressively, compile and debug often !

function

- We have already seen intrinsic functions:
  cos, exp, log10, sin, ...
- Users can also define their own functions, such as for instance:
  - `convert_in_celsius(temp)`: takes a real temperature and returns its equivalent in Celsius degrees
  - `factorial(n)`: takes an integer number *n* and returns *n*
  - `compute_average(n,array)`: takes a real array of size n and returns its average

The factorial function (1)

Previously:

```
program factorial
implicit none

integer :: fact
integer :: i,n

write(6,*) 'enter a number'
read(5,*) n

if (n>0 .and. n<13) then

   fact=1
   do i=1,n
      fact=fact*i
      write(6,*) i,'! =',fact
   end do

else

   write(6,*) 'the input value of n'
   write(6,*) 'is not correct. Aborting.'

end if

end program
```

The factorial function (2)

```
program example
implicit none
integer, external :: factorial

print *,'fact  3',factorial(3)
print *,'fact  5',factorial(5)
print *,'fact  7',factorial(7)
print *,'fact 11',factorial(11)
print *,'fact 17',factorial(17)

end program

!=======================

function factorial(n)
implicit none
integer :: factorial,n

factorial=1
do i=1,n
   factorial=factorial*i
end do

end function

!=======================
```

```
thebeast:progmod geogarfield$ ./a.out
 fact  3           6
 fact  5         120
 fact  7        5040
 fact 11    39916800
 fact 17  -288522240
```

the compute_average function

```
program example
implicit none
integer, parameter :: n=100
real, dimension(n) :: tab
real, external     :: average

call random_number(tab)

write(*,*) 'avrg of tab is',average(n,tab)

call random_number(tab)

write(*,*) 'avrg of tab is',average(n,tab)

end program

!================================

function average(ntab,tab)
implicit none
integer :: ntab
real :: average,
real, dimension(ntab) ::tab

average=sum(tab)/real(ntab)

end function

!================================
```

```
thebeast:progmod geogarfield$ ./a.out
 avrg of tab is  0.53604215
 avrg of tab is  0.51591498
```

Using functions

The following uses function `factorial(n)` to compute the combinatorial coefficient

$$C(m,n) = \frac{m!}{n!(m-n)!}$$

where m and n are actual arguments:

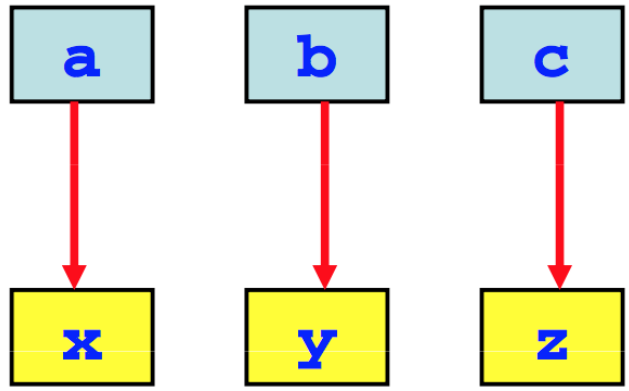
```
...
...
Cmn = factorial(m)/(factorial(n)*factorial(m-n))
...
```

Argument Association (1)

```
WRITE(*,*) Sum(a,b,c)



INTEGER FUNCTION Sum(x,y,z)
    IMPLICIT NONE
    INTEGER,INTENT(IN)::x,y,z
        ……..
END FUNCTION   Sum
```

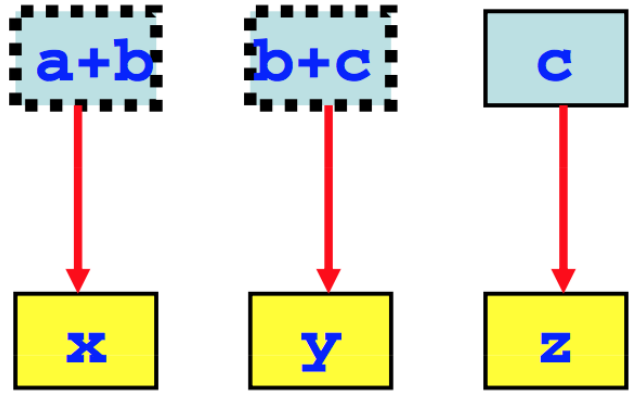


Argument Association (2)

```
WRITE(*,*) Sum(a+b,b+c,c)



INTEGER FUNCTION Sum(x,y,z)
    IMPLICIT NONE
    INTEGER,INTENT(IN)::x,y,z
        ……..
END FUNCTION   Sum
```

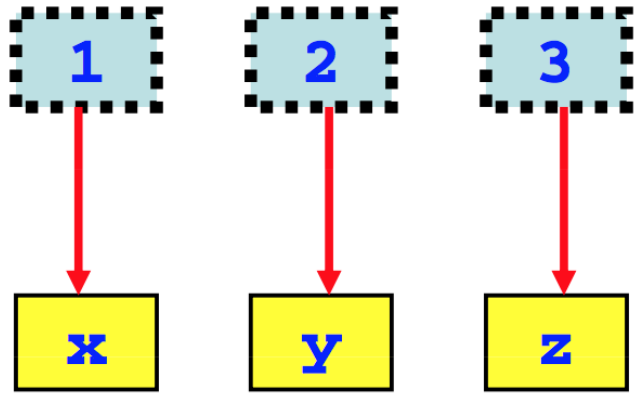


Argument Association (3)

```
WRITE(*,*) Sum(1, 2, 3)



INTEGER FUNCTION Sum(x,y,z)
    IMPLICIT NONE
    INTEGER,INTENT(IN)::x,y,z
        ……..
END FUNCTION   Sum
```



function vs subroutine

A function

- returns a value (or an array of values)
- has a type (integer, real, ...)
- is usually rather simple/short
- does not modify its arguments
- does not contain write statements

function vs subroutine

A function

- returns a value (or an array of values)
- has a type (integer, real, ...)
- is usually rather simple/short
- does not modify its arguments
- does not contain write statements

A **subroutine**

- performs one or many tasks
- does not have a type
- is invoked with `call`
- has arguments (or not) and can return them modified

A very simple subroutine

```
program example
implicit none

call say_hello()

end program

!====================

subroutine say_hello()
implicit none

write(*,*) 'hello world !'

end subroutine

!====================
```

```
> ./a.out
hello world !
```

Example (2)

```
program example
implicit none

real :: vx,vy,vz,vel

call random_number(vx)
call random_number(vy)
call random_number(vz)

call compute_velnorm(vx,vy,vz,vel)

write(*,*) 'vect is    ',vx,vy,vz
write(*,*) 'its norm is',vel

end program

!====================

subroutine compute_velnorm(vect_x,vect_y,vect_z,vectnorm)
implicit none
real :: vect_x,vect_y,vect_z
real :: vectnorm

vectnorm=sqrt(vect_x**2+vect_y**2+vect_z**2)

end subroutine

!====================
```

```
> ./a.out
vect is 0.99755955 0.56682467 0.96591532
its norm is 1.4998026
```

Example (3)

```
program subdem
implicit none
real :: a,b,c,summ,sumsq

call INPUT(a,b,c)
call CALC(a,b,c,summ,sumsq)
call OUTPUT(summ,sumsq)

end program

!==================================

subroutine INPUT(x,y,z)
implicit none
real :: x,y,z
write(*,*) 'ENTER THREE NUMBERS: '
read *,x,y,z
end subroutine

!==================================

subroutine CALC(a,b,c,summ,sumsq)
implicit none
real :: a,b,c,summ,sumsq
summ  = a+b+c
sumsq = summ **2
end subroutine

!==================================

subroutine OUTPUT(summ,sumsq)
implicit none
real :: summ, sumsq
write(*,*) 'The sum of the numbers you entered are: ',summ
write(*,*) 'And the square of the sum is:',sumsq
end subroutine

!==================================
```

2D geometry (1)

- define three random points in $[0, 1] \times [0, 1]$
- compute the coordinates of the barycenter
- compute the area of the triangle they form with

$$A = \frac{1}{4}\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}$$

- compute the shortest side length of the triangle
- compute the angle values with

$$\cos\theta_A = \frac{\boldsymbol{AB} \cdot \boldsymbol{AC}}{|\boldsymbol{AB}|\,|\boldsymbol{AC}|}$$

2D geometry (2)

```
program example
implicit none
real, dimension(2) :: ptA,ptB,ptC,bary
real :: area
real :: Aangle,Bangle,Cangle
character(len=2) :: shortest

call random_number(ptA)
call random_number(ptB)
call random_number(ptC)

call barycenter(ptA,ptB,ptC,bary)

call area_triangle(ptA,ptB,ptC,area)

call find_shortest(ptA,ptB,ptC,shortest)

call compute_angles(ptA,ptB,ptC,Aangle,Bangle,Cangle)

write(*,*) '-----------------------------------'
write(*,*) 'pt A:',ptA
write(*,*) 'pt B:',ptB
write(*,*) 'pt C:',ptC
write(*,*) 'barycenter coordinates are ',bary
write(*,*) 'area of the triangle ', area
write(*,*) 'shortest side is ', shortest
write(*,*) 'angles        ', Aangle,Bangle,Cangle
write(*,*) 'sum of angles ', Aangle+Bangle+Cangle
write(*,*) '-----------------------------------'

end program
```

2D geometry (2)

```
program example
implicit none
real, dimension(2) :: ptA,ptB,ptC,bary
real :: area
real :: Aangle,Bangle,Cangle
character(len=2) :: shortest

call random_number(ptA)
call random_number(ptB)
call random_number(ptC)

call barycenter(ptA,ptB,ptC,bary)

call area_triangle(ptA,ptB,ptC,area)

call find_shortest(ptA,ptB,ptC,shortest)

call compute_angles(ptA,ptB,ptC,Aangle,Bangle,Cangle)

write(*,*) '-----------------------------------'
write(*,*) 'pt A:',ptA
write(*,*) 'pt B:',ptB
write(*,*) 'pt C:',ptC
write(*,*) 'barycenter coordinates are ',bary
write(*,*) 'area of the triangle ', area
write(*,*) 'shortest side is ', shortest
write(*,*) 'angles        ', Aangle,Bangle,Cangle
write(*,*) 'sum of angles ', Aangle+Bangle+Cangle
write(*,*) '-----------------------------------'

end program
```

```
subroutine barycenter(ptA,ptB,ptC,bary)
implicit none
real, dimension(2) :: ptA,ptB,ptC,bary

bary(1)=(ptA(1)+ptB(1)+ptC(1))/3.
bary(2)=(ptA(2)+ptB(2)+ptC(2))/3.

end subroutine

!==========================

subroutine area_triangle(ptA,ptB,ptC,area)
implicit none
real, dimension(2) :: ptA,ptB,ptC
real :: AB,BC,AC,area

AB=sqrt( (ptB(1)-ptA(1))**2 + (ptB(2)-ptA(2))**2 )
BC=sqrt( (ptB(1)-ptC(1))**2 + (ptB(2)-ptC(2))**2 )
AC=sqrt( (ptA(1)-ptC(1))**2 + (ptA(2)-ptC(2))**2 )

area=0.25*sqrt((AB+BC+AC)*(AC+AB-BC)*(AB+BC-AC)*(BC+AC-AB))

end subroutine

!==========================

subroutine find_shortest(ptA,ptB,ptC,shortest)
implicit none
real, dimension(2) :: ptA,ptB,ptC
real :: AB,BC,AC
character(len=2) :: shortest

AB=sqrt( (ptB(1)-ptA(1))**2 + (ptB(2)-ptA(2))**2 )
BC=sqrt( (ptB(1)-ptC(1))**2 + (ptB(2)-ptC(2))**2 )
AC=sqrt( (ptA(1)-ptC(1))**2 + (ptA(2)-ptC(2))**2 )

if (AB<AC .and. AB<BC) shortest='AB'
if (AC<AB .and. AC<BC) shortest='AC'
if (BC<AB .and. BC<AC) shortest='BC'

end subroutine
```

2D geometry (3)

```
program example
implicit none
real, dimension(2) :: ptA,ptB,ptC,bary
real :: area
real :: Aangle,Bangle,Cangle
character(len=2) :: shortest

call random_number(ptA)
call random_number(ptB)
call random_number(ptC)

call barycenter(ptA,ptB,ptC,bary)

call area_triangle(ptA,ptB,ptC,area)

call find_shortest(ptA,ptB,ptC,shortest)

call compute_angles(ptA,ptB,ptC,Aangle,Bangle,Cangle)

write(*,*) '-----------------------------------'
write(*,*) 'pt A:',ptA
write(*,*) 'pt B:',ptB
write(*,*) 'pt C:',ptC
write(*,*) 'barycenter coordinates are ',bary
write(*,*) 'area of the triangle ', area
write(*,*) 'shortest side is ', shortest
write(*,*) 'angles        ', Aangle,Bangle,Cangle
write(*,*) 'sum of angles ', Aangle+Bangle+Cangle
write(*,*) '-----------------------------------'

end program
```

$$\cos\theta_A = \frac{\boldsymbol{AB}\cdot\boldsymbol{AC}}{|\boldsymbol{AB}|\,|\boldsymbol{AC}|}$$

```
subroutine compute_angles(ptA,ptB,ptC,Aangle,Bangle,Cangle)
implicit none
real, dimension(2) :: ptA,ptB,ptC
real :: Aangle,Bangle,Cangle
real :: AB,BC,AC,dot_pr,pi

pi=4.*atan(1.)

AB=sqrt( (ptB(1)-ptA(1))**2 + (ptB(2)-ptA(2))**2 )
BC=sqrt( (ptB(1)-ptC(1))**2 + (ptB(2)-ptC(2))**2 )
AC=sqrt( (ptA(1)-ptC(1))**2 + (ptA(2)-ptC(2))**2 )

dot_pr=(ptB(1)-ptA(1))*(ptC(1)-ptA(1)) &
      +(ptB(2)-ptA(2))*(ptC(2)-ptA(2))

Aangle=acos(dot_pr/AB/AC)/pi*180.

dot_pr=(ptA(1)-ptB(1))*(ptC(1)-ptB(1)) &
      +(ptA(2)-ptB(2))*(ptC(2)-ptB(2))

Bangle=acos(dot_pr/AB/BC)/pi*180.

dot_pr=(ptA(1)-ptC(1))*(ptB(1)-ptC(1)) &
      +(ptA(2)-ptC(2))*(ptB(2)-ptC(2))

Cangle=acos(dot_pr/AC/BC)/pi*180.

end subroutine
```

2D geometry (4)

```
program example
implicit none
real, dimension(2) :: ptA,ptB,ptC,bary
real :: area
real :: Aangle,Bangle,Cangle
character(len=2) :: shortest

call random_number(ptA)
call random_number(ptB)
call random_number(ptC)

call barycenter(ptA,ptB,ptC,bary)

call area_triangle(ptA,ptB,ptC,area)

call find_shortest(ptA,ptB,ptC,shortest)

call compute_angles(ptA,ptB,ptC,Aangle,Bangle,Cangle)

write(*,*) '-----------------------------------'
write(*,*) 'pt A:',ptA
write(*,*) 'pt B:',ptB
write(*,*) 'pt C:',ptC
write(*,*) 'barycenter coordinates are ',bary
write(*,*) 'area of the triangle ', area
write(*,*) 'shortest side is ', shortest
write(*,*) 'angles        ', Aangle,Bangle,Cangle
write(*,*) 'sum of angles ', Aangle+Bangle+Cangle
write(*,*) '-----------------------------------'

end program
```

```
thebeast:progmod geogarfield$ ./a.out
 -----------------------------------
 pt A:  0.99755955      0.56682467
 pt B:  0.96591532      0.74792767
 pt C:  0.36739087      0.48063689
 barycenter coordinates are   0.77695531      0.59846306
 area of the triangle   5.84263988E-02
 shortest side is AB
 angles           87.876724       75.846550       16.276726
 sum of angles    180.00000
 -----------------------------------
```

```
program elefant

use structures

implicit none

include 'mpif.h'

CALL mpi_init(ierr)
call mpi_comm_size (mpi_comm_world,nproc,ierr)
call mpi_comm_rank (mpi_comm_world,iproc,ierr)

!==================================!
                                   !
call header                        !
call clean                         !
call open_files                    !
call read_n_compute_parameters     !
call write_parameters              !
call cailloux_setup                !
call read_materials                !
call grid_setup                    !
call cloud_setup                   !
call timevisu_setup                !
call tracers_setup                 !
call vgrid_setup                   !
call stretch                       !
call landscape_setup               !
call lsurface_setup                !
call paint_stripes                 !
call material_layout               !
call strain_history                !
call nu_layout                     !
call temperature_layout            !
call initialise_mumps_V            !
call initialise_mumps_T            !
call initialise_mumps_P            !
call matrix_setup                  !
call compute_tot_volume_and_mass   !
call compute_elsize                !
call compute_element_centers       !
call compute_hydr_pressure         !
call compute_qcoords               !
                                   !
do istep=1,nstep !---------------------------- timestepping
                                             !
   iter=0                                    !
   conv=.false.                              !
                                             !
   do while (.not.conv .and. iter<abs(niter))!-----
                                             !
      iter=iter+1                            !
                                             !
```