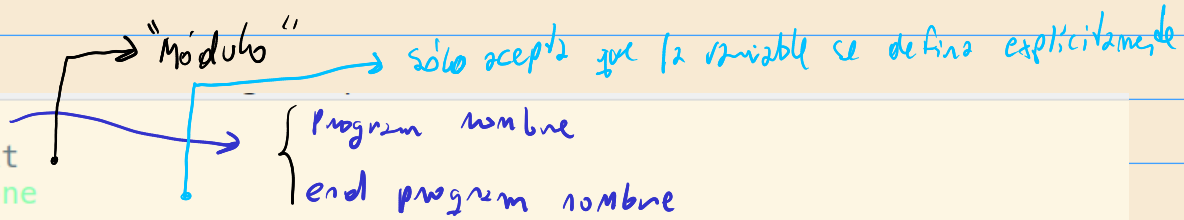
Introducción a FORTRAN

Clase 6, 2025




```
program simple
    use ziggurat
    implicit none
    logical :: es
    integer :: seed,i ,j,k
    real (kind=8) :: x(10),a(10,10),b(10,10)
    real (kind=8), allocatable  :: y(:),c(:,:)
!
!![NO TOCAR] Inicializa generador de número random

    inquire(file='seed.dat',exist=es)
    if(es) then
        open(unit=10,file='seed.dat',status='old')
        read(10,*) seed
        close(10)
        print *,"  * Leyendo semilla de archivo seed.dat"
    else
        seed = 24583490
    end if

    call zigset(seed)
!![FIN NO TOCAR]

! Ej: Número random en [0,1]: uni()


        do i = 1, 500
            print *,i,uni()
        end do

!!
!! EDITAR AQUI
!!

    a=0
    b(:,:) = 1.
```

! → Comentario

! variables adicionales

```
program simple
    use ziggurat
    implicit none
    logical :: es
    integer :: seed,i ,j,k
    real (kind=8) :: x(10),a(10,10),b(10,10)
    real (kind=8), allocatable  :: y(:),c(:,:)


![NO TOCAR] Inicializa generador de número random

    inquire(file='seed.dat',exist=es)
    if(es) then
        open(unit=10,file='seed.dat',status='old')
        read(10,*) seed
        close(10)
        print *,"  * Leyendo semilla de archivo seed.dat"
    else
        seed = 24583490
    end if

    call zigset(seed)
![FIN NO TOCAR]

! Ej: Número random en [0,1]: uni()


        do i = 1, 500
            print *,i,uni()
        end do

!!
!! EDITAR AQUI
!!

    a=0
    b(:,:) = 1.
```

* tipo variable, propiedades :: var

* variables reales (kind=8 → doble precisión)

* Malla de 10x10

* Asignación dinámica de memoria

matriz de 10x10

```
a=0
b(:,:) = 1.
! Alocar variables
allocate(c(10,10),y(10))
do i=1,10
   do j=1,10
      c(i,j)=a(i,j)+b(i,j)
   end do
end do
```

! Pone en 0 todos los elementos de la matriz

! Pone en 1 todos los elementos de la matriz

Usualmente leemos en un archivo datos, parámetros para alocar

```
read n
allocate (c(n,n), y(n))
```

loop, bucle

x(10) → x[n]

```
c = a + b
```

"Indentar"

primera estructura 1 tab

2da estructura 2 tab

3ra " 3 tab

```
c = a + b
c(:,:) = a(:,:) + b(:,:)
```

Expresiones condicionales

```
if(i>5)  then
   a(1,1) =1.
   b(2,2)= 0.
end if
```

* if de una línea

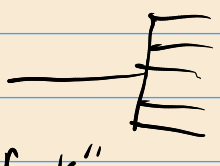
```
if (i>5) print *, "vamos bien"
```

* if, then, else

```
if (i>5) then
a(1,1) = 0.
else
a(1,1) = 1.
end if
```

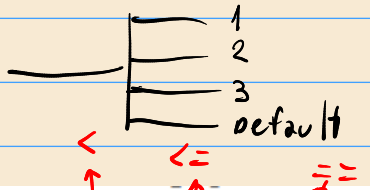
* if, else if, end if

```
if (i>5) then
a(1,1) = 0.
else if (i>10) then
a(1,1) = 1.0
end if
```



"fork"

```
select case (n)
case(1)
   call routine1( )
case(2)
   call routine2( )
case(3)
   call say_good_bye( )
case default
   call this_is_the_end( )
end select
```



1.4 Operators

```
.lt. .le. .eq. .ne. .gt. .ge.
.not. .and. .or. .eqv. .neqv.
x**(-y)
'AB'//'CD'
```

Conectores lógicos : .and. .or. .not.

&& ||

```
if (i<5 .and. j>3) then
   call do_something()
end if
```

Input/output : read, open, close, write, print

unidad — Nombre archivo en el sistema operativo

```
open(unit=10,file='seed.dat',status='old')
read(10,*) seed
close(10)
```

de donde leo (en este caso del archivo seed.dat)

old → el archivo ya existe (debe existir)
new → el archivo no existe previamente (No debe existir).
unknown → puede existir o no.

formato. * = formato libre

* write

```
open (unit=235, file="salida.dat", status="unknown")
write (235,*) seed
close(235)
```

Variantes: escribo una columna con componentes de un vector x(n)

```
Do i=1,n
   write(235,*)  x(i)
end do
```

(*) → formato libre

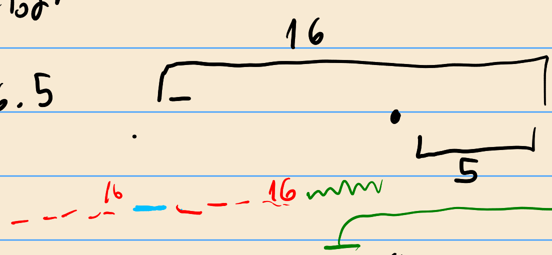
entero de hasta 7 decimales / 16 ; real ; 5 decimales

```
Do i=1,n
write(235, fmt="(i7, 2f16.5)")  i, x(i), hola(i)
end do
```

formato: f 16.5 ("float")

g

1,005 E12




```
write (235, fmt="(f16.5, x, f16.5, a)"  x(i), y(i), "energía molécula"
```

x → 1 espacio longitud.

a → caracteres sin especificación de longitud

Estructura de un programa

Un programa típico tiene:

* main → código principal
* rutinas → bloques de código que cumplen una determinada función
* módulos → librerías de rutinas (Ej: Zigguart valores generadores de números al azar)
  → Grupos de variables globales.

Main

main.f90

```
program hago_algo
  integer :: i
  call algo(v)
end program hago_algo
```

Rutinas

"intención": in, out, inout

algo.f90

```
subroutine algo (a)
 use fft
  real (kind=8), intent(in) :: a
  real (kind=8) :: x(10), v(10)
  integer :: i, j, k
  ...
  Código
  ...
end subroutine algo
```

* Por defecto las variables son locales

simple.f90
- main
- rutina1
- rutina 2
- rutina 3

* La rutina puede estar en el mismo archivo que el main

Se compila como gfortran -O3 simple.f90 -o simple.exe

* La rutina puede estar en un archivo distinto al del main

main.f90
routine1.f90
routine2.f90

→ Se compila con:
gfortran main.f90 routine1.f90 routine2.f90 -o simple.exe
ó mucho mejor:
make (~> Makefile)

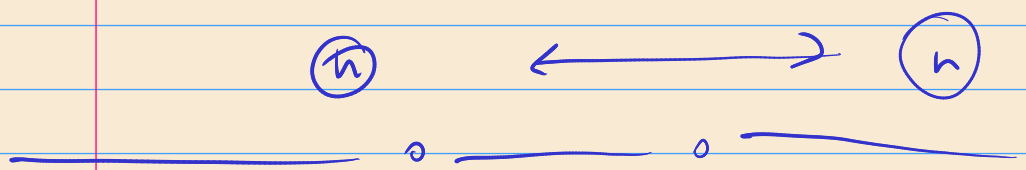
Módulos

```
module globals
real(kind=8), allocatable :: v(:,:), u(:,:), a(:,:), f(:,:)
real(kind=8)              :: temp, press

end module globals
```

Main
use globals

Subroutine
use globals



Modulo
Grupo de rutinas

module routines → routines.f90

Contains
subroutine x1()
⋮
end subroutine x1
subroutine x2()
⋮
end subroutine x2
end module routines

Main:
use routines

call x1()
⋮

routine algo
use routines
call x2()