

Introducción a FORTRAN

```
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(:, :)
```

"main"
"rutinas"

Program nombre

end program nombre

→ "Módulos"

No permite declaraciones
implícitas de variables

![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.
```

```
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.
```

Variable lógica
→ enteros

tipo variable, propiedades :: van

variable, reales
(kind=8 → doble precisión)

vector de dimensión 10

Matriz de 10x10

Allocación dinámica
de memoria

matriz de 10x10

$a=0$ → Pone en 0 todos los elementos de la matriz
 $b(:, :) = 1.$ → Pone en 1 todos los elementos de la matriz

! Alocar variables

`allocate(c(10,10),y(10))`

Usualmente lee en un archivo datos, parámetros para almacenar

read N

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

loop, bucle

```
do i=1,10
  do j=1,10
    c(i,j)=a(i,j)+b(i,j)
  end do
end do
```

$x(10) \rightarrow x(n)$

$c = a + b$

"Indentación"

$c = a + b$

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

primera estructura	1 tab
2da estructura	2 tab
3ra "	3 tab

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`

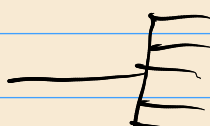
`a(1,1) = 0.`

`else if (i>10) then`

`a(1,1) = 1.0`

`end if`

"fork"



select case (n)

case(1)

call routine1()

case(2)

call routine2()

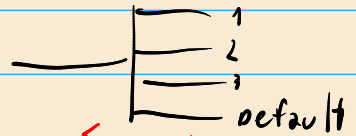
case(3)

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)
```

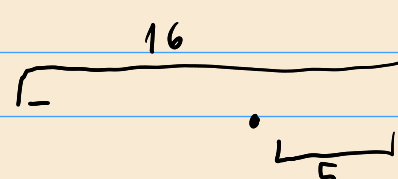
variantes: escribo una columna con componentes de un vector $x(n)$

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

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

entero de hasta 7 decimales / 16
↑
real 16 decimales

formato: $f\ 16.5$ g 16 5



```
- write (235, fmt="(f16.5, x, f16.5, @)") x(i), y(i), "energía molecular"
```

1 espacio
longitud.

caracteres sin especificar su

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: Ziggurat random generators de números al azar)
→ Grupos de variables globales.

main

```
program hago_algo  
:  
  call algo(v)  
:
```

```
end program hago_algo
```

rutinas

```
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
```

"intención"

in
out
inout

* Por defecto las variables son locales

simple.f90

main

rutina1

rutina2

rutina3

* 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 :: r(:, :), v(:, :), a(:, :), f(:, :)

real(kind=8) :: temp, press

end module globals

main

use globals

subroutine

use globals

(n)

←→

(n)

Módulo

Grupo de rutinas

module routines

routine.f90

contains

subroutine x1()

⋮

end subroutine x1

subroutine x2()

⋮

end subroutine x2

main

use routines

call x1()

⋮

routine also

use routines

call x2()