

# INFORMATICA



# Evaluación

	<b>Teoría</b>	<b>Presentaciones 1</b>	<b>Presentaciones 2</b>
6 Septiembre	Introducción		
13 Septiembre	Datos y Operadores		
20 Septiembre	Control de flujo I		
27 Septiembre	Control de flujo II		
4 Octubre	Arrays I		
11 Octubre	Arrays II	CPU	RAM
18 Octubre	EXAMEN PARCIAL		
25 Octubre	Intrínsecas	DISCO DURO	GPU
8 Noviembre	Formato lectura escritura	FPGA	XEON PHI
15 Noviembre	Programación modular I	CLUSTER	RASPBERRY PI
22 Noviembre	Programación modular II	HISTORIA LENG	FUTURO LENG
29 Noviembre	Tipos derivados		
13 Diciembre	EXAMEN PARCIAL		
20 Diciembre	Programación avanzada	SISTEMAS OPERATIVOS	LEY DE MOORE



FIN DE ELECCIÓN DE PRESENTACIÓN 07/10/18

- Tipos de datos: Intrínsecos
  - Numéricos: (`integer`, `real`, `complex`)
  - Lógicos: (`logical`)
  - Carácter: (`character`)

- Tipos de datos: Derivados

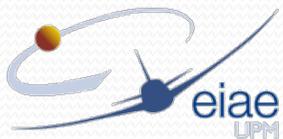
- Arrays

- Vectores

- Matrices

Clase de hoy

- Estructuras



Antes:

- Dudas práctica jueves pasado:
- Ejercicio números primos: ¿?

- Sumatorios: ¿Dudas?

$$S_1 = \sum_{i=1}^n i^2$$

$$S_4 = \sum_{i=1}^n \sum_{j=i}^m (i * j + i - j)$$

$$S_2 = \sum_{i=1}^n (i + 1)^2$$

$$S_5 = \sum_{i=1}^n \sum_{j=i}^{n+m} (i - j)$$

$$S_3 = \sum_{i=1}^n \sum_{j=1}^m \frac{i + j}{i}$$

$$S_6 = \sum_{i=1}^n \sum_{j=0}^i \frac{(i - j)^2}{3}$$

# Antes:

- Estilo de programación:

```
program main
  implicit none
  integer, parameter:: n = 100
  integer:: s
  integer:: i, j

  !---Fin declaracion-----
```

```
    s = 0
  do i=1,n
    do j=1,n
      if (i <= j) then
        s = s + i + j
      endif
    enddo
  enddo
end program main
```

# Antes:

- Uso de las condicionales: I.- **if... else... end if**

```
if ((n-(n/2)*2) == 0) then
    ...
else
    ...
endif
```

**Bien!!**

```
if((n-(n/2)*2) == 0) then
    ...
elseif((n-(n/2)*2) /= 0) then
    ...
end if
```

```
if((n-(n/2)*2) == 0) then
    ...
endif
if((n-(n/2)*2) /= 0) then
    ...
end if
```

**Regular → Mal**

# Antes:

- Uso de las condicionales: II.- **if... elseif... else... endif**

```
IF (x > 0) THEN  
    WRITE (*, *) '+'  
ELSEIF (x == 0) THEN  
    WRITE (*, *) '0'  
ELSE  
    WRITE (*, *) '-'  
ENDIF
```

# Cálculo número de divisores primos

```
integer:: n,i
```

```
n = 7
```

```
i = 2
```

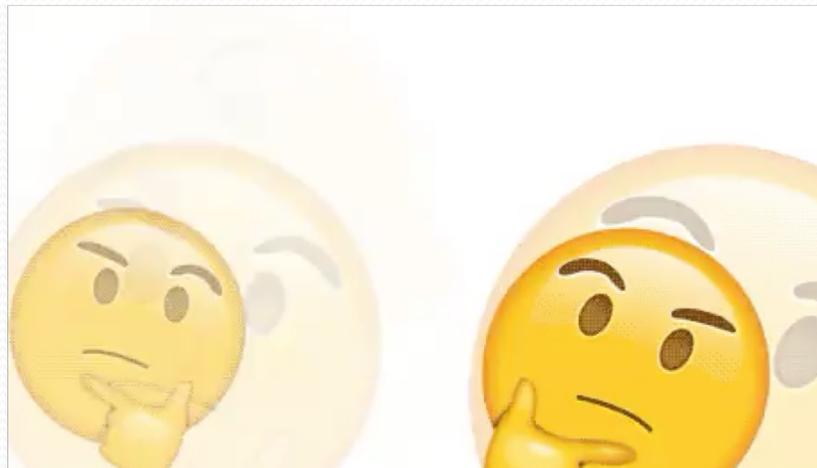
```
write(*,*) n - ( (n/i) * i )           !7-INT(7/2)*2=1
```

```
n = 6
```

```
i = 2
```

```
write(*,*) n - ( (n/i) * i )           !6-INT(6/2)*2=0
```

!Y si probara a dividir el número por todos los que son menores que él...



# Arrays: Vectores y Matrices

- Un *array* es una colección de datos, denominados *elementos*, todos ellos del mismo tipo y *kind*, situados en posiciones contiguas de memoria.
- Los *arrays* se clasifican en
  - *unidimensionales* (vectores)
  - *bidimensionales* (matrices)
  - *multidimensionales* (tablas)

# Arrays: Vectores y Matrices: Declaración

```
program main
```

```
integer :: U(5)           ! Vector  
real    :: V(10)         ! Vector  
real    :: A(2,3)        ! Matriz 2x3  
real    :: Z(3,2)        ! Matriz 3x2  
real    :: T(0:2,-1:0)   ! Matriz 3x2  
integer :: Ta(2,3,4)     ! Tabla  
                        ! 2x3x4
```

```
end program main
```

```
program main
```

```
integer, dimension(5)    :: U  
real, dimension(10)     :: V  
real, dimension(2,3)    :: A  
real, dimension(3,2)    :: Z  
real, dimension(0:2,-1:0) :: T  
integer, dimension(2,3,4) :: Ta
```

```
end program main
```



U: 

U(1)	U(2)	U(3)	U(4)	U(5)
------	------	------	------	------

A: 

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)

T: 

T(0,-1)	T(0,0)
T(1,-1)	T(1,0)
T(2,-1)	T(2,0)

# Arrays: Vectores y Matrices: Declaración

- Lenguaje matemático:

$$A = \begin{pmatrix} 1.1 & 3.8 & 4.3 \\ 1.2 & 5.5 & 7 \end{pmatrix} \quad T = \begin{pmatrix} 1.1 & 5.5 \\ 1.2 & 4.3 \\ 3.8 & 7 \end{pmatrix}$$

- Lenguaje de programación:

$$A(1, 1) = 1.1 \quad A(1, 2) = 3.8 \quad A(1, 3) = 4.3$$

$$A(2, 1) = 1.2 \quad A(2, 2) = 5.5 \quad A(2, 3) = 7.0$$

$$T(0, -1) = 1.1 \quad T(0, 0) = 5.5 \quad T(1, -1) = 1.2$$

$$T(1, 0) = 4.3 \quad T(2, -1) = 3.8 \quad T(2, 0) = 7.0$$

# Vectores y Matrices: Almacenamiento en memoria

`integer` `::` `U(5)`

↓  
u(1)  
u(2)  
u(3)  
u(4)  
u(5)

`integer` `::` `B(5,3)`

b(1,1)    b(1,2)    b(1,3)  
b(2,1)    b(2,2)    b(2,3)  
b(3,1)    b(3,2)    b(3,3)  
b(4,1)    b(4,2)    b(4,3)  
b(5,1)    b(5,2)    b(5,3)

- Velocidad de cálculo en grandes códigos

# Arrays: Vectores y Matrices: Asignación

```
program asignacion
```

```
integer :: U(4)
```

```
U = 10
```

```
end program asignacion
```

U: 

10	10	10	10
----	----	----	----

---

```
program asignacion
```

```
integer :: U(4)
```

```
U = (/0,2,-1,3/)
```

```
end program asignacion
```

U: 

0	2	-1	3
---	---	----	---



# Arrays: Vectores y Matrices: Asignación

```
program asignacion
```

```
integer :: U(4)
```

```
U(2) = 0
```

```
U(3) = -1
```

```
U(1) = 3
```

```
U(4) = 2
```

```
end program asignacion
```

U:

3	0	-1	2
---	---	----	---

# Arrays: Vectores y Matrices: Asignación

```
program asignacion
```

```
integer :: A(2,3)
```

```
A = 10
```

```
end program asignacion
```

A:

10	10	10
10	10	10

```
program asignacion
```

```
integer :: A(2,3)
```

```
A = reshape((/1, 0, -2, 3, 5, 8/), (/2,3/))
```

```
end program asignacion
```

A:

1	-2	5
0	3	8

**OJO!!**



# Arrays: Vectores y Matrices: Asignación

```
program asignacion
```

```
integer :: A(2,3)
```

```
A(1,3) = 2
```

```
A(1,1) = 10
```

```
A(2,1) = -1
```

```
A(2,2) = 14
```

```
A(1,2) = 3
```

```
A(2,3) = -5
```

```
end program asignacion
```

A:

10	3	2
-1	14	-5

# Arrays: Vectores y Matrices: Declaración II

- Parámetro

```
program declaracion

integer, parameter :: n = 2
integer, parameter :: m = 3
integer, parameter :: Linf = 0
integer, parameter :: Lsup = 10

integer :: U(n)
real :: V(n*m)
real :: A(n,m)
real :: T(Linf:Lsup, Linf:Lsup)

end program declaracion
```

- Dinámica

```
program declaracion

integer :: n
Integer :: m

integer, allocatable :: U(:)
real, allocatable :: A(:, :)

end program declaracion
```



# Arrays: Vectores y Matrices: Declaración II

- Parámetro

```
program declaracion

integer, parameter :: n = 2
integer, parameter :: m = 3
integer, parameter :: Linf = 0
integer, parameter :: Lsup = 10

integer :: U(n)
real :: V(n*m)
real :: A(n,m)
real :: T(Linf:Lsup, Linf:Lsup)

end program declaracion
```



- Dinámica

```
program declaracion

integer :: n
Integer :: m

integer, allocatable :: U(:)
real, allocatable :: A(:, :)

end program declaracion
```



$U(:)$  es un vector y  $A(:, :)$  una matriz de tamaño desconocidos al principio del programa

# Arrays: Vectores y Matrices: Asignación II

- Parámetro

```
program declaracion
```

```
integer, parameter :: n = 2  
integer, parameter :: m = 3
```

```
integer :: U(n)  
real :: A(n,m)
```

```
A = 0.d0  
U = 0.d0
```

- Igual que antes

```
end program declaracion
```



- Dinámica

```
program declaracion
```

```
integer :: n  
Integer :: m
```

```
integer, allocatable :: U(:)  
real, allocatable :: A(:, :)
```

```
! Cuerpo de programa
```

```
n = 2  
m = 3
```

```
allocate(U(n))  
allocate(A(n,m))
```

```
A = 0.d0  
U = 0.d0
```

```
end program declaracion
```

# Arrays: Vectores y Matrices: Asignación II

- Parámetro

```
program declaracion
```

```
integer, parameter :: n = 2  
integer, parameter :: m = 3
```

```
integer :: U(n)  
real :: A(n,m)
```

```
A = 0.d0  
U = 0.d0
```

- Igual que antes

```
end program declaracion
```



- Dinámica

```
program declaracion
```

```
integer :: n  
Integer :: m
```

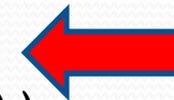
```
integer, allocatable :: U(:)  
real, allocatable :: A(:, :)
```

```
! Cuerpo de programa
```

```
n = 2
```

```
m = 3
```

```
allocate(U(n))  
allocate(A(n,m))
```



Reserva dinámica  
de memoria para  
U(:) y A(:, :)

```
A = 0.d0
```

```
U = 0.d0
```

```
end program declaracion
```

# Arrays: Vectores y Matrices: Asignación III

- Asignación por bucle

```
program asignacion

integer, parameter :: n = 10
integer :: U(n)
integer :: i

!--- Fin declaracion -----

do i=1,n
    U(i) = i*i
end do

write(*,*) U

do i=1,n
    write(*,*) U(i)
end do

end program asignacion
```



# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz identidad con  $n = 5$

$$Id = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz identidad con  $n = 5$

$$Id = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
program main

    integer, parameter :: n = 5
    integer :: Id(n,n)
    integer :: i, j

    !--- Fin declaracion -----

    Id = 0
    do i=1,n
        do j=1,n
            if (i == j) Id(i,j) = 1
        end do
    end do

end program main
```

# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz  $A_{n \times n}$  con  $n = 100$

$$a_{ij} = \begin{cases} i + j & \text{Si } i \leq j \\ 0 & \text{Si } i > j \end{cases}$$

$$A = \begin{pmatrix} 2 & 3 & 4 & \cdots & 101 \\ 0 & 4 & 5 & \cdots & 102 \\ & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 200 \end{pmatrix}$$

# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz  $A_{n \times n}$  con  $n = 100$

$$a_{ij} = \begin{cases} i + j & \text{Si } i \leq j \\ 0 & \text{Si } i > j \end{cases} \quad A = \begin{pmatrix} 2 & 3 & 4 & \cdots & 101 \\ 0 & 4 & 5 & \cdots & 102 \\ & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 200 \end{pmatrix}$$

```
program main
integer, parameter :: n = 100
integer :: A(n,n)
integer :: i, j
!--- Fin declaracion -----

do i=1,n
    do j=1,n
        if (i <= j) then
            A(i,j) = i + j
        else
            A(i,j) = 0
        end if
    end do
end do
end program main
```



# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz  $A_{n \times n}$  con  $n = 100$

$$a_{ij} = \begin{cases} i + j & \text{Si } i \leq j \\ 0 & \text{Si } i > j \end{cases} \quad A = \begin{pmatrix} 2 & 3 & 4 & \cdots & 101 \\ 0 & 4 & 5 & \cdots & 102 \\ & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 200 \end{pmatrix}$$

```
program main
integer, parameter :: n = 100
integer :: A(n,n)
integer :: i, j
!--- Fin declaracion -----

A = 0
do i=1,n
    do j=1,n
        if (i <= j) A(i,j) = i + j
    end do
end do
end program main
```

# Arrays: Vectores y Matrices: Asignación III

Escribir un programa que defina la matriz  $A_{n \times n}$  con  $n = 100$

$$a_{ij} = \begin{cases} i + j & \text{Si } i \leq j \\ 0 & \text{Si } i > j \end{cases} \quad A = \begin{pmatrix} 2 & 3 & 4 & \cdots & 101 \\ 0 & 4 & 5 & \cdots & 102 \\ & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 200 \end{pmatrix}$$

```
program main
integer, parameter :: n = 100
integer :: A(n,n)
integer :: i, j
!--- Fin declaracion -----

A = 0
do i=1,n
    do j=i,n
        A(i,j) = i + j
    end do
end do
end program main
```

# Asignación Dinámica

```
program declaracion
```

```
integer :: n
```

```
Integer :: m
```

```
integer, allocatable :: U(:)
```

```
real, allocatable :: A(:, :)
```

```
! Cuerpo de programa
```

```
allocate(U(n))
```

```
allocate(A(n,m))
```

```
A = 0.d0
```

```
U = 0.d0
```

```
end program declaracion
```

- Inconvenientes de la declaración de un *array* con tamaño fijo:
  - Si el tamaño prefijado es mayor que el número de valores que se van a almacenar, estamos *malgastando* memoria.
  - Si el tamaño prefijado es menor que el número de valores que se van a utilizar, el programa dará un error de ejecución.
- Solución en Fortran: ***arrays dinámicos***.



# Asignación Dinámica

```
program declaracion
```

```
integer :: n,info  
integer, allocatable :: U(:)  
! Cuerpo de programa
```



Declaración de U como array dinámico

```
n = 5
```

```
allocate(U(n), stat=info)
```



Reserva de memoria, con control de error  
Si la asignación es correcta stat = 0.  
En caso contrario stat > 0

```
if (info > 0) stop '** No hay memoria &  
                & suficiente para U**'
```

```
U = 0.d0
```

```
deallocate(U, stat=info)
```



Libera la memoria previamente reservada,  
con control de error

```
if (info > 0) stop '** U no tenía &  
                & memoria reservada**'
```

```
end program declaracion
```

# Arrays: Vectores y Matrices: Asignación III

- **Trabajo para casa:** (subid a Moodle antes del 13 de Noviembre)
  - El **objetivo** de esta práctica es escribir un **programa** que **calcule números primos**.
  - Se proponen tres estructuras de programa: **versión básica, media y avanzada**. Dependiendo de la opción elegida, la pareja optará a tres notas distintas: **aprobado, notable o sobresaliente**.
  - En todo caso, se valorará la **limpieza** del programa (e.g., indentación correcta) así como el uso de **comentarios** explicativos.
  - La **entrega** se realizará a través de **Moodle**.
  - Solo uno de los miembros de la pareja realizará la entrega.
  - El nombre del fichero será DNI1\_DNI2.f90.
  - Un comentario en la parte superior del código especificará si se entrega la versión básica, media o avanzada.

# Arrays: Vectores y Matrices: Asignación III

- Trabajo para casa: (subid a Moodle antes del 13 de Noviembre)

## TRABAJO NÚMEROS PRIMOS: **VERSIÓN BÁSICA** **APROBADO**

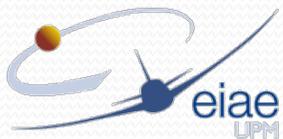
- 1.- El programa pide al usuario que introduzca un número entero mayor que cero,  $n$ .
- 2.- El programa comprueba que el valor introducido es correcto (es mayor que cero). En caso contrario, el programa finaliza.
- 3.- El programa creará un vector de tamaño  $n$ .
- 3.- El programa calcula los números primos menores o iguales a  $n$  y los almacena en el vector creado.
- 4.- El programa escribirá por pantalla el contenido del vector.
- 5.- El programa liberará la memoria del vector.

# Arrays: Vectores y Matrices: Asignación III

- Trabajo para casa: (subid a Moodle antes del 13 de Noviembre)

## TRABAJO NÚMEROS PRIMOS: **VERSIÓN MEDIA** **NOTABLE**

- 1.- El programa pide al usuario que introduzca un número entero mayor que cero,  $n$ .
- 2.- El programa comprueba que el valor introducido es correcto (es mayor que cero). En caso contrario, le pide al usuario que introduzca de nuevo otro número.
- 3.- El programa calcula los números primos menores o iguales a  $n$ .
- 4.- El programa creará un vector de tamaño el número de números primos menores o iguales a  $n$ .
- 5.- El programa almacenará los números primos menores o iguales a  $n$  en el vector creado.
- 6.- El programa escribirá por pantalla el contenido del vector.
- 7.- El programa liberará la memoria del vector.



# Arrays: Vectores y Matrices: Asignación III

- Trabajo para casa: (subid a Moodle antes del 13 de Noviembre)

## TRABAJO NÚMEROS PRIMOS: **VERSIÓN AVANZADA** **SOBRESALIENTE**

- La **versión avanzada es una versión optimizada de la versión media**. Se esperan optimizaciones que aumenten el rendimiento del código (disminución en el tiempo de ejecución). El aumento de rendimiento será especialmente apreciable para valores de  $n$  grandes. Se proponen las siguientes **técnicas de optimización**:
  - Minimización del número de operaciones.
  - Minimización del número de iteraciones de los bucles presentes.
  - Uso de técnicas eficientes para el cálculo de números primos.
  - En todo caso, se añadirá un comentario en la parte superior del código explicando las técnicas de optimización implementadas.

