



# Informática

## Tipos de Datos

### Operaciones Básicas

Dpto. Matemática Aplicada a la Ingeniería Aeroespacial

curso 2017-2018



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter



## Algoritmo

### Problema

Resolver la ecuación:

$$ax^2 + bx + c = 0$$

- Si  $b^2 - 4ac > 0 \Rightarrow$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Si  $b^2 - 4ac = 0 \Rightarrow$

$$x_1 = x_2 = -\frac{b}{2a}$$

- Si  $b^2 - 4ac < 0 \Rightarrow$

$$z = \frac{-b \pm i\sqrt{-(b^2 - 4ac)}}{2a}$$



## Programa

### Problema

Resolver la ecuación:

$$ax^2 + bx + c = 0$$

```

PROGRAM raices
  IMPLICIT NONE
  REAL      :: a, b, c
  REAL      :: x1, x2
  COMPLEX   :: z1, z2
  REAL      :: delta

  WRITE(*,*) 'Introducir a, b, c'
  READ(*,*) a,b,c
  |
  IF (ABS(a) < 1.E-10) THEN
    WRITE(*,*) 'a debe ser distinto de cero'
  ELSE

    delta = b*b - 4.0*a*c

    IF (delta >= 0.0) THEN

      x1 = 0.5*(-b + SQRT(delta))/a
      x2 = 0.5*(-b - SQRT(delta))/a
      WRITE(*,*) 'La solucion es ', x1,x2

    ELSE

      z1 = 0.5*cmplx(-b,SQRT(-delta))/a
      z2 = 0.5*cmplx(-b,-SQRT(-delta))/a
      WRITE(*,*) 'La solucion es ', z1,z2

    END IF

  END IF

END PROGRAM raices
  
```



# Estructura general de un programa



```

PROGRAM raices
  IMPLICIT NONE
  REAL      :: a, b, c
  REAL      :: x1, x2
  COMPLEX   :: z1, z2
  REAL      :: delta

  WRITE(*,*) 'Introducir a, b, c'
  READ(*,*) a,b,c
  |
  IF (ABS(a) < 1.E-10) THEN
    WRITE(*,*) 'a debe ser distinto de cero'
  ELSE

    delta = b*b - 4.0*a*c

    IF (delta >= 0.0) THEN

      x1 = 0.5*(-b + SQRT(delta))/a
      x2 = 0.5*(-b - SQRT(delta))/a
      WRITE(*,*) 'La solución es ', x1,x2

    ELSE

      z1 = 0.5*cmplx(-b,SQRT(-delta))/a
      z2 = 0.5*cmplx(-b,-SQRT(-delta))/a
      WRITE(*,*) 'La solución es ', z1,z2

    END IF

  END IF

END PROGRAM raices

```



**Cuerpo de  
Declaración**



# Estructura general de un programa



```

PROGRAM raices
  IMPLICIT NONE
  REAL      :: a, b, c
  REAL      :: x1, x2
  COMPLEX   :: z1, z2
  REAL      :: delta

  WRITE(*,*) 'Introducir a, b, c'
  READ(*,*) a,b,c
  |
  IF (ABS(a) < 1.E-10) THEN
    WRITE(*,*) 'a debe ser distinto de cero'
  ELSE

    delta = b*b - 4.0*a*c

    IF (delta >= 0.0) THEN

      x1 = 0.5*(-b + SQRT(delta))/a
      x2 = 0.5*(-b - SQRT(delta))/a
      WRITE(*,*) 'La solucion es ', x1,x2

    ELSE

      z1 = 0.5*cmplx(-b,SQRT(-delta))/a
      z2 = 0.5*cmplx(-b,-SQRT(-delta))/a
      WRITE(*,*) 'La solucion es ', z1,z2

    END IF

  END IF

END PROGRAM raices

```

**Sentencias de ejecución**





- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter



# Tipos de Datos



```
program main
  integer :: i
  integer, parameter :: n = 10

  i = 5
  write(*,*) i

  i = i + n
  write(*,*) i

end program main
```

## Tipos de datos

- Variable: `i`
- Constante: `n`



## Características de un tipo de dato

- Identificador de tipo

`integer :: i`

- Rango válido de valores

$\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$

- Forma de denotar esos valores

`i = 10`

- Conjunto de operaciones válidas

`i + j`



## Tipos de datos

- **Datos de tipo intrínseco**
  - **Datos numéricos**
    - integer
    - real
    - complex
  - **Datos no numéricos**
    - Datos lógicos: `logical`
    - Datos carácter: `character`
- **Datos de tipo derivado**



# Reglas básicas de un dato



## Declaración

- Todo dato debe ser declarado

Identificador de tipo `:: nombre`

---

```
program main  
  integer :: n
```

```
n = 20
```

```
n = -2
```

```
n = 0
```

```
n = 9999
```

```
end program main
```

---



# Identificador (nombre) de un dato



- No debe exceder de 31 caracteres



Letra

Letra, número o “\_”

- No distingue mayúsculas y minúsculas
- Evitar palabras “clave” de Fortran

```
allocate  write  call  max
matmul   open   do    ...
```



# Reglas básicas de un dato



## Definición

- Todo dato debe tener un valor antes de ser utilizado

---

```
program main
  integer :: n
```

```
n = 20
n = -2
n = 0
n = 9999
```

```
end program main
```

---



# Reglas básicas de un dato



## Definición

- Todo dato debe tener un valor antes de ser utilizado
- **Inicialización:** asignación del valor en la misma línea de declaración

```
integer           :: n = 10  
real, parameter  :: pi = 3.14159
```



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos**
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter



## ● Declaración

```

program enteros
implicit none
integer :: i
integer :: j, k, numero

    i = 2
    j = -1
    k = 0
    numero = -999999

end program enteros

```

```

integer :: i
integer :: j, k, numero

```

## ● Definición

```

i = 2
j = -1
k = 0
numero = -999999

```



## Declaración

### Simple Precisión

- 4 bytes en memoria
- 6 decimales significativos

```
real          :: X
real*4        :: X
real(4)       :: X
real(kind=4)  :: X
```

## Declaración

### Doble Precisión

- 8 bytes en memoria
- 16 decimales significativos

```
real*8        :: X
real(8)       :: X
real(kind=8)  :: X
```



## Definición

### Simple Precisión

- Modo Decimal
- Modo exponencial

$$x = 20.508$$

$$x = 20.508e0$$

$$x = 205.08e-1$$

$$x = 0.20508e+2$$

## Definición

### Doble Precisión

- Modo exponencial

$$x = 2d0$$

$$x = 2.0d0$$



# Datos numéricos. Complejos



```

program complejos
implicit none
complex      :: z
complex(8)   :: z1

      z = (2.0, 0.0)
      z1 = (2.0d0, 0.0d0)

end program complejos

```

## ● Declaración

```

complex      :: z
complex(8)   :: z1

```

## ● Definición

```

z = (2.0,0.0)
z1 = (2d0, 0d0)

```



## Parámetros (o constantes)

- Dato cuyo valor no puede variar a lo largo de la ejecución del programa.
- Se deben inicializar en la misma línea de declaración.

```
complex, parameter :: unidad_imag = (0.,1.)  
integer, parameter :: n = 3  
real*8 , parameter :: avogadro = 6.022d23  
real(8), parameter :: pi = acos(-1.d0)
```



# Datos numéricos. Definición



## Importante

El valor asignado a un dato debe ser acorde con su tipo

```

program main
implicit none
real(8), parameter :: pi = 3.14159265358979d0
real(8), parameter :: pi_mal = 3.14159265358979
real(8), parameter :: pii = acos(-1.0d0)
real(8), parameter :: pii_mal = acos(-1.0)

write(*,*) 'pi = ', pi
write(*,*) 'pi_mal = ', pi_mal
write(*,*) 'pii = ', pii
write(*,*) 'pii_mal = ', pii_mal

end program main
  
```



# Datos numéricos. Definición



El valor asignado a un dato debe ser acorde con su tipo

```

program main
implicit none
real(8), parameter :: pi = 3.14159265358979d0
real(8), parameter :: pi_mal = 3.14159265358979
real(8), parameter :: pii = acos(-1.0d0)
real(8), parameter :: pii_mal = acos(-1.0)

write(*,*) 'pi = ', pi
write(*,*) 'pi_mal = ', pi_mal
write(*,*) 'pii = ', pii
write(*,*) 'pii_mal = ', pii_mal

end program main

```

```

pi =      3.14159265358979
pi_mal =   3.14159274101257
pii =      3.14159265358979
pii_mal =   3.14159274101257
Presione una tecla para continuar . . .

```



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter



*operando1 operador operando2*

## Operandos numéricos

- **Expresiones aritméticas**

$$x * 5.5$$

$$x + y$$

- **Expresiones relacionales**

$$x \leq 6.8$$

$$x == y$$



## Operadores aritméticos

+	Suma
-	Resta
*	Producto
/	Cociente
**	Potenciación

## Orden en ausencia de Paréntesis

2 + 3 \* 2 = ????



## Operadores aritméticos

+	Suma
-	Resta
*	Producto
/	Cociente
**	Potenciación

## Orden en ausencia de Paréntesis

$$2 + 3 * 2 = 10$$




## Orden en ausencia de Paréntesis

### Operadores aritméticos

+	Suma
-	Resta
*	Producto
/	Cociente
**	Potenciación

$$2 + 3 * 2 = 8$$




# Expresiones aritméticas



$$2 + 4 * 1 - 2 ** 2 + 8 = 10$$

## Orden en ausencia de Paréntesis

- 1º Potenciación
- 2º Producto y Cociente
- 3º Suma y Resta

- El orden en el que se realizan las operaciones de igual precedencia es de izquierda a derecha.

$$a * b / c \text{ equivale a } (a * b) / c$$

- **Excepción:** la potenciación

$$3 ** 2 ** 2 \text{ equivale a } 3 ** (2 ** 2) \quad \boxed{81}$$



$$2 + 4 * 1 - 2 * *(2 + 8) = -1018$$

## Orden con Paréntesis

- 1º Paréntesis
- 2º Potenciación
- 3º Producto y Cociente
- 4º Suma y Resta



# Expresiones aritméticas



## Tipo del resultado

- Cuando se opera con datos numéricos de distinto tipo y *kind*,  
*“el resultado es del tipo con mayor rango de los involucrados en la operación”*

$$(a + - * /) b$$

Tipo de a	Tipo de b	Tipo de resultado
<code>integer</code>	<code>integer</code>	<code>integer</code>
<code>integer</code>	<code>real</code>	<code>real</code>
<code>real</code>	<code>integer</code>	<code>real</code>
<code>real</code>	<code>real</code>	<code>real</code>



# Expresiones aritméticas. Ejemplo 1



```
program operaciones
```

```
integer   :: i,j
real      :: x,y
real*8    :: xd,yd
```

```
i = 2
j = 4
x = 2.
y = 4.
xd = 2.d0
yd = 4.d0
```

```
write(*,*) i/j
write(*,*) x/y
write(*,*) xd/yd
write(*,*) xd/j
```

```
end program
```

```
0
0.500000
0.5000000000000000
0.5000000000000000
```

```
Press RETURN to close window ...
```



## Tipo del resultado

- Cuando se opera con datos numéricos de distinto tipo y *kind*,  
*“el resultado es del tipo con mayor rango de los involucrados en la operación”*
- Antes de evaluar la expresión se convierten los operandos al tipo más fuerte.
  - Excepto cuando se eleva un dato de tipo **real** o **complex** a una potencia de tipo **integer**, en cuyo caso no se convierte el exponente.



# Expresiones aritméticas. Ejemplo 2



```
program operaciones

write(*,*) 2**(-3)
write(*,*) 2.0**(-3)
write(*,*) 2.0**(-3.0)
write(*,*) (-2)**3
write(*,*) (-2.)**3
write(*,*) (-2.)**3.5
write(*,*) (-2.)**(2/3)
write(*,*) (-2.)**2/3
write(*,*) (-2)**2/3
write(*,*) (-2.)**(2./3)

end program
```



# Expresiones aritméticas. Ejemplo 02



```
program operaciones
```

```
write(*,*) 2**(-3)
write(*,*) 2.0**(-3)
write(*,*) 2.0**(-3.0)
write(*,*) (-2)**3
write(*,*) (-2.)**3
write(*,*) (-2.)**3.5
write(*,*) (-2.)**(2/3)
write(*,*) (-2.)**2/3
write(*,*) (-2)**2/3
write(*,*) (-2.)**(2./3)
```

```
end program
```

```
G:\Informatica\Primer cuatrimestre 14-15\Clase 2>gfortran -o ejemlo2 ejemplo2.f95
ejemplo2.f95:8.16:
      write(*,*) (-2.)**3.5
      1
Error: Raising a negative REAL at (1) to a REAL power is prohibited
ejemplo2.f95:12.16:
      write(*,*) (-2.)**(2./3)
      1
Error: Raising a negative REAL at (1) to a REAL power is prohibited
G:\Informatica\Primer cuatrimestre 14-15\Clase 2>_
```



# Expresiones aritméticas. Ejemplo 2



```
program operaciones
```

```
write(*,*) 2**(-3)
write(*,*) 2.0**(-3)
write(*,*) 2.0**(-3.0)
write(*,*) (-2)**3
write(*,*) (-2.)**3
write(*,*) (-2.)**(2/3)
write(*,*) (-2.)**2/3
write(*,*) (-2)**2/3
```

```

0
0.125000
0.125000
-8
-8.00000
1.00000
1.33333
1
Press RETURN to close window..._
```

```
end program
```



# Expresiones aritméticas. Ejemplo 2



```
program operaciones
```

```
write(*,*) 2**(-3)
write(*,*) 2.0**(-3)
write(*,*) 2.0**(-3.0)
write(*,*) (-2)**3
write(*,*) (-2. )**3
write(*,*) (-2. )** (2/3)
write(*,*) (-2. )**2/3
write(*,*) (-2)**2/3
```

```
end program
```



## Importante

- Siempre que el exponente sea entero, su valor debe darse como entero.
- No se puede elevar un número negativo a un real.



# Expresiones aritméticas. Ejemplo 3



```
program operaciones
```

```
real      :: x,y
real*8    :: xd,yd
```

```
x = 1.
y = 1.e-6
xd = x
yd = y
```

```
write(*,*) x+y
write(*,*) xd+yd
```

```
end program
```

```
1.00000
1.00000100000
Press RETURN to close window..._
```



## Expresiones relacionales

Se utilizan para comparar datos numéricos entre sí o con expresiones aritméticas.

## Operadores relacionales

$>$	Mayor que
$<$	Menor que
$>=$	Mayor o igual que
$<=$	Menor o igual que
$==$	Igual que
$\neq$	Distinto que



## Expresiones relacionales

- El resultado de evaluar una expresión relacional es de tipo **logical**
- Solo puede tomar uno de los valores `.true.` o `.false.`
- Generalmente estas expresiones se utilizan en las estructuras de control que gobiernan el flujo de un programa.
- Si al menos uno de los operandos es de tipo **complex** los únicos operadores relacionales disponibles son `==` y `/=`



# Expresiones relacionales. Ejemplo 4



```
program ejemplo
  integer :: i, j
  real    :: a, b

  i = 0
  j = -2
  a = 5.5
  b = 0.5

  write(*,*) i <= 0
  write(*,*) a < b
  write(*,*) (a+b) > (i-j)
  write(*,*) a+b > i-j
end program ejemplo
```



# Expresiones relacionales. Ejemplo 4



```

program ejemplo
  integer :: i, j
  real    :: a, b

  i = 0
  j = -2
  a = 5.5
  b = 0.5

  write(*,*) i <= 0           ! T
  write(*,*) a < b           ! F
  write(*,*) (a+b) > (i-j)   ! T
  write(*,*) a+b > i-j       ! T
end program ejemplo

```



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos**
- 6 Datos carácter



# Datos de tipo lógico




---

```

program main
  logical :: flag

  flag = .true.
  flag = .false.

end program main

```

---

- **Declaración**

```
logical :: flag
```

- **Definición**

```
flag = .true.
flag = .false.
```



## Expresiones relacionales

- Con 1 operando:**      *operador operando*  
                                   *.not. flag*
- Con 2 operandos:**      *operando1 operador operando2*  
                                   *flag1 .and. flag2*  
                                   *flag1 .or. flag2*



# Expresiones relacionales. Ejemplo 5



```
program logicos
```

```
  implicit none
```

```
  real   :: a
```

```
  real   :: b
```

```
  logical :: flag
```

```
  a = 1.0
```

```
  b = 0.0
```

```
  write(*,*) a == 0d0
```

```
  write(*,*) a/= b
```

```
  b = -(b + 1.0)
```

```
  write(*,*) (a >= 0.0) .and. (b >= 0)
```

```
  write(*,*) (a >= 0.0) .or. (b >= 0)
```

```
  flag = ((a >= 0.0) .and. (b >= 0)) .or. (b < 0)
```

```
  write(*,*) flag
```

```
  write(*,*) .not.flag
```

```
end program logicos
```

```
F
T
F
T
T
F
Presione una tecla para continuar . . .
```



- 1 Introducción
- 2 Tipos de datos
- 3 Datos numéricos
- 4 Operaciones básicas con datos numéricos
- 5 Datos lógicos
- 6 Datos carácter**



# Datos de tipo carácter



- **Declaración**

```
character           :: nombre1
character(10)       :: nombre2
character(len=10)   :: nombre3
```

- **Definición**

```
nombre1 = 'A'
nombre2 = ' aaa AAAA'
nombre3 = 'Me duermo en esta clase'
```

- **Valor asignado**

```
nombre1 ← 'A'
nombre2 ← ' aaa AAAA '
nombre3 ← 'Me duermo '
```



# Datos carácter. Operaciones



## ● Concatenación //

```
program caracteres
  character(len=9)   :: Ag1
  character(len=12)  :: Ag2
  character(len=7)   :: Ag3
  character(len=6)   :: nombre
  character(len=3)   :: numero
  nombre = 'Agente'
  numero = '007'
  Ag1 = nombre//numero
  Ag2 = nombre//numero
  Ag3 = nombre//numero
```



# Datos carácter. Operaciones



## ● Concatenación //

```

program caracteres

  character(len=9)   :: Ag1
  character(len=12)  :: Ag2
  character(len=7)   :: Ag3
  character(len=6)   :: nombre
  character(len=3)   :: numero

  nombre = 'Agente'
  numero = '007'

  Ag1 = nombre//numero   ! Ag1 = Agente007
  Ag2 = nombre//numero   ! Ag2 = Agente007
  Ag3 = nombre//numero   ! Ag3 = Agente0
  
```



# Datos carácter. Operaciones



## ● Extracción //

```
program caracteres
implicit none

character(len=9) :: nombre

    nombre = 'Agapito'
    write(*,*) nombre(1:1)
    write(*,*) nombre(1:6)
    write(*,*) nombre(5:9)

end program caracteres
```



## ● Extracción //

```

program caracteres
implicit none

character(len=9) :: nombre

    nombre = 'Agapito'
    write(*,*) nombre(1:1)
    write(*,*) nombre(1:6)
    write(*,*) nombre(5:9)

end program caracteres
  
```

```

A
Agapit
ito
  
```



# Estructura básica de un programa



```
program main
implicit none
    ! Declaración de datos

    ! Definición de datos

    ! Sentencias de ejecución

    ! Visualización de resultados

end program main
```



# Estructura básica de un programa



```
program main
implicit none
    ! Declaración de datos
real :: x
real :: y

    ! Definición de datos
x = 2.5

    ! Sentencias de ejecución
y = x**0.5

    ! Visualización de resultados
write(*,*) ' La raíz cuadrada de', x, 'es', y
end program main
```