

Introducción al Fortran

Bruno Juliá-Díaz (brunojulia@ub.edu)

Dpto. Física Quàntica i Astrofísica

Facultat de Física

Universitat de Barcelona

Curso 2019/2020

Sources: *Mètodes Numèrics per a la Física*, Guardiola, Higón y Ros (U. Valencia)

<http://www.chem.ox.ac.uk/fortran>

<http://www.math.hawaii.edu/~hile/fortran/fortmain.htm>

Fortran

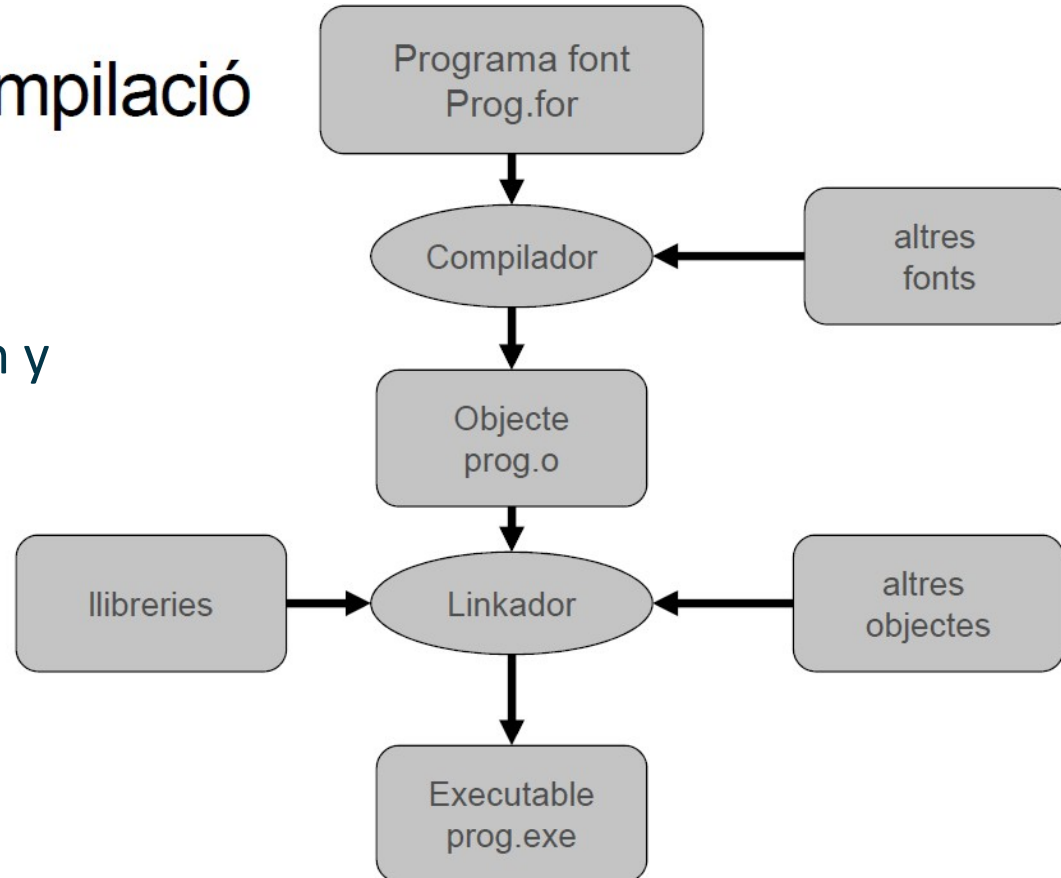
- > Lenguaje muy utilizado en física computacional:
 - + Primer lenguaje completo
 - + Todas las revisiones han mantenido la compatibilidad con las versiones previas
 - + Fue desarrollado para trabajar con expresiones matemáticas
 - + Permite un gran control de la precisión de los números utilizados
 - + Existen multitud de subrutinas bien testadas disponibles
 - + Permite trabajar con caracteres

- > Limitaciones, no está particularmente adaptado a trabajar con:
 - bases de datos
 - programas a través de WWW
 - integración de gráficos

Requiere compilación

Compilació

> La mayoría de los compiladores compilan y linkan directamente



Programa

- > El programa lo componen uno o varios ficheros ascii.
`Program.f`, `program.for`, `program.f90`, `rutinas1.f`, etc
- > Cada fichero contiene una serie de lineas con las instrucciones Fortran
- > Fortran **no distingue** mayúsculas de minúsculas
- > Utilizaremos formato “clásico”, 72 columnas útiles:
 - + **columna 1**, si tiene una “C” o “*” es una linea de comentario
 - + **columnas 1-5**, se utilizan para poner etiquetas a las líneas y poder volver a ellas
 - + **columna 6**, un carácter en esta columna indica que es continuación de la anterior
 - + **columna 7-72**, instrucciones normales de Fortran

Hello World

(fichero hello.f)

```
      print*, "Hello World"  
end
```

Compilación:

```
gfortran hello.f -o hello.exe
```

Ejecución:

```
hello.exe
```

Hello World

Programa

```
C          1          2          3          4          5          6          7
C2345678901234567890123456789012345678901234567890123456789012
C          PROGRAM PRUEBA 1
C          PRUEBA 1
C          PROGRAMA QUE ESCRIBE EL CUADRADO DE UN NÚMERO
C          BJULIA SEP 2015
C
C          IMPLICIT NONE
C          REAL A,A2
C
C          LEE EL NÚMERO DEL TECLADO
C          PRINT*,"ESCRIBA UN NUMERO"
C          READ*,A
C
C          CALCULA EL CUADRADO
C          A2=A**2
C
C          ESCRIBE EL CUADRADO
C          WRITE(*,111) A2
111  FORMAT('EL CUADRADO DEL NUMERO ES:',F9.6)
C
C          END
```

Programa

```
C      PROGRAM MEDIA1
C
C      Programa que lee 3 números y calcula su valor medio
C
C      INPUT
C      x1, x2, x3: numeros reales
C
C      OUTPUT
C      xm          : valor medio
C
```

- > **Encabezado:**
 - + Nombre del programa
 - + Descripción de lo que hace
 - + Explicación de las variables

```
IMPLICIT NONE
```

- > Línea que indica que todas las variables tendrán que definirse explícitamente

```
REAL X1,X2,X3
REAL xm
```

- > Especificación de las variables

```
WRITE(*,*) 'Introduce 3 numeros'
READ(*,ie) X1,X2,X3
print*,ie
XM= (X1+X2+X3)/3.
WRITE(*,*) 'El valor medio es:',xm
END
```

- > **Cuerpo del programa**

Modulos

> PROGRAM

+ Indica el programa principal, **solo puede haber uno**

> SUBROUTINE

+ Realiza una tarea específica.

> FUNCTION

+ Devuelve un valor como función de una serie de parámetros

> BLOCK DATA

+ variables compartidas entre diversas partes del programa

> El programa principal, las subrutinas y funciones pueden estar definidas en distintos ficheros.

```
PROGRAM PRUEBA 2
C      PRUEBA 2
C      PROGRAMA QUE ESCRIBE EL CUADRADO DE UN NÚMERO
C      BJULIA SEP 2015
C
C      IMPLICIT NONE
C      REAL A,A2

C LEE EL NÚMERO DEL TECLADO
      PRINT*, "ESCRIBA UN NUMERO"
      READ*, A

C CALCULA EL CUADRADO
      call cuadrado(A,A2)

C ESCRIBE EL CUADRADO
      WRITE(*,111) A2
111   FORMAT('EL CUADRADO DEL NUMERO ES:',F9.6)

      END

      SUBROUTINE CUADRADO(X,X2)
C
C      input: X
C      output: X2, cuadrado de X

      x2=x*x

      END
```


Tipos de variables

> **INTEGER*2** (2 bytes=16 bits, un bit para el signo, 15 para el numero)

+ Desde -2^{15} a $2^{15}-1 = 32767$

> **INTEGER*4**

+ Desde -2^{31} a $2^{31}-1 = 2147483647$

> **REAL O REAL*4**

+ 0.42, 4.31E-10, 0.3141592E+1

+ precisión de una parte en 10^7 , entre 10^{-38} a 10^{38}

> **DOUBLE PRECISION O REAL*8**

+ 0.42d0, 4.31D-10, 0.3141592d+1

+ precisión de una parte en 10^{15} , entre 10^{-308} a 10^{308}

> **COMPLEX**

+ (0.1 , 0.42)

+ real y imag

> **LOGICAL**

+ true o false

> **CHARACTER**

+ CHARACTER*3 PATATA

PATATA="abc"

> **DECLARACIONES POR DEFECTO**

IMPLICIT REAL (A-H,O-Z), INTEGER (I-N)

MUY RECOMENDABLE

IMPLICIT NONE

Fuerza a definir explícitamente todas las variables utilizadas

Matrices y vectores

C DECLARACIÓN

```
REAL A,B  
INTEGER C  
CHARACTER*3 PALABRA
```

```
DIMENSION A(40)  
DIMENSION B(20,30)  
DIMENSION C(100)
```

Índice del 1 al numero, ej, de 1 a 40
2 indices, 1 a 20 y 1 a 30

C USO

```
A(3)= 4.0  
A(39)=3.2
```

```
B(10,20)=4.
```

```
A(5)= B(10,20)+A(39)  
C(4)=3
```

C DEFINICION COMPACTA

```
REAL A(40),B(20,30)  
INTEGER C(100)
```

C INDICES PERSONALIZADOS

```
REAL EQUIS(-30:30), Y(0:10)  
REAL TIEMPOS(0:100)
```

Operaciones y expresiones relacionales

C OPERACIONES

$A * B$

$A + B$

A / B

$A ** B$

$A - B$

C EXPRESIONES RELACIONALES

$(A.EQ.B)$

$.GE.$ Greater equal

$.GT.$ Greater than

$.LE.$ Lower equal

$.LT.$ Lower than

$.NE.$ not equal

$.AND.$ y

$.OR.$ o

$.NOT.$ not

$.EQV.$ Equivalente (lógico)

$.NEQV.$ No Equivalente (lógico)

Las operaciones relacionales van entre paréntesis Si se combinan varias, la expresión completa también ha de ir entre paréntesis.

EJEMPLOS:

$IF (A.EQ.B) THEN$
 $ENDIF$

$IF ((A.EQ.B).OR.(C.EQ.D)) THEN$
 $ENDIF$

$.NOT.$ SE USA COMO NEGADOR
 $IF (.NOT.(A.GE.B)) THEN$

Operaciones y expresiones relacionales

C FUNCIONES INTRINSECAS

SQRT(X),DSQRT(X)
SIN(X),DSIN(X)
COS(X)
TAN(X)
ASIN(X)
ATAN(X)
SINH(X)
COSH(X)
EXP(X)
LOG(X)

C

ABS(X)
DBLE(X)
INT(X)
NINT(X)
REAL(X)

.... (HAY UNAS 40)

Abs(3.2)	=	3.20000005
Abs(-3.2)	=	3.20000005
dabs(3.2d0)	=	3.200000000000000002
dabs(-3.2d0)	=	3.200000000000000002
Int(3.6)	=	3
Nint(3.6)	=	4
Dble(4)	=	4.000000000000000000
Dble(4.0)	=	4.000000000000000000

Valor absoluto

Pasa a doble precisión

Parte entera de X

Entero más cercano

Pasa a real

Prioridad en las operaciones

Las expresiones aritméticas se evalúan (En este orden)

- 1 De dentro a fuera, primero los paréntesis
- 2 De derecha a izquierda, exponenciación **
- 3 De izquierda a derecha, multiplicación, división
- 4 De izquierda a derecha, suma, resta

Ejemplos:

$$-A^{**}N+B^{*}C-D/E$$

Se evalúa:

1> $A^{**}N$

2> $B^{*}C$

3> D/E

4> el $-$ frente a $A^{**}N$

5> el $+$ entre $A^{**}N$ y $B^{*}C$

6> el $-$ entre $B^{*}C$ y D/E

$$A+((B^{**}M)^{**}N)/(C-D)$$

1> $B^{**}M$

2> $B^{**}M$ a la N

3> la división $/$

4> el $+$

Aritmética (*)

Si se utilizan reales en toda la expresión no habrá problema y la aritmética real funciona correctamente

REAL A,B,C

A=3.

B=4.

C=10.

Por ejemplo:

$A*B*C$ valdrá 120.

Si todos los números son enteros, la suma, resta y multiplicación funcionan como se espera.

La **división** no, en lugar de dar el real correcto, truncará los decimales dejando sólo la parte entera, por ejemplo,

$1/3$ da como resultado 0

$4/3$ da como resultado 1

$5/2$ da como resultado 2

Aritmética mixta y asignación

Si alguno de los operandos es real, el resultado será real. Aun así, hay que tener mucho cuidado porque el orden sigue importando

$5/2*3.0= 6.00000000$ primero realiza $5/2=2$, y después $2*3.0=6.0$
 $3.0*5/2= 7.50000000$ primero realiza $3.0*5=15.0$ y después $15.0/2.=7.5$

Es recomendable utilizar comandos como

`xnn=real(nn)`

que convierten el entero `nn` en el real `xnn` antes de operar

Para **asignar un valor** a una variable se utiliza:

Variable= expresion

Ejemplos. `X=X+1`

`Y=2.35`

`pi=atan(1.0)`

`cosa=x*y`

Se pueden definir constantes, cuyo valor no se puede modificar:

`Real pi`

`Integer ndim`

`Parameter (pi=3.141592)`

`Parameter (ndim=40)`

Control de flujo

Para controlar el flujo dentro del programa existe el comando

IF

Se puede utilizar :

IF (npasos.eq.40) mensaje=1

IF (b.eq.c) c=3.

IF (expresión condicional) THEN

.....

.....

ENDIF

Es bueno indentar el contenido del IF. Ayuda a entender el flujo del programa

IF (expresión condicional) THEN

.....

ELSE IF (otra expresion condicional) THEN

.....

ELSE IF (otra expresion condicional) THEN

.....

ELSE

ENDIF

Tests de igualdad en números reales

Imaginemos que queremos incluir una determinada condición del tipo “Calcula la trayectoria de un móvil, cuando su velocidad sea 5 m/s escribe su posición”

La primera opción (algo drástica) sería añadir una línea de código tipo:

```
REAL VELOCIDAD  
IF (VELOCIDAD.EQ.5.) THEN  
    PRINT*,POSICION  
ENDIF
```

Esto presenta el **GRAVE** inconveniente de que, trabajando con precisión fija, es prácticamente imposible que la condición se cumpla exactamente y el código nunca cumplirá su cometido.

Lo más apropiado es comparar los números dentro de la precisión deseada, algo como

```
IF (abs(VELOCIDAD-5.).LT.1.e-7) THEN  
    PRINT*,POSICION  
ENDIF
```

Bucles

El comando **DO** tiene dos formas de funcionar:

Se puede utilizar :

```
DO 40 I = INICIO, FINAL, PASO
```

```
.....
```

```
40      CONTINUE
```

O de este modo:

```
DO I=INICIO,FINAL,PASO
```

```
.....
```

```
ENDDO
```

Comando **GO TO**

```
GOTO 40
```

```
.... COSAS
```

```
40      CONTINUE
```

ESTE CÓDIGO NO PASARÁ POR
"COSAS"

No se recomienda el uso de GOTO
porque dificulta considerablemente
entender el flujo del programa.

EJEMPLOS (el paso por defecto es
1):

```
DO 40 ITIEMPOS=1,40
```

```
PRINT*,ITIEMPOS
```

```
40      CONTINUE
```

```
DO NCOSAS=1,1000,10
```

```
NC=NCOSAS**2
```

```
ENDDO
```

EJEMPLO (sumatorio)

```
INTEGER NDIM
```

```
REAL SUMA, FUNC1(NDIM)
```

```
SUMA=0.
```

```
DO I=1,NDIM
```

```
SUMA=SUMA+ FUNC1(I)
```

```
ENDDO
```

Nuevo
valor

Antiguo
valor

Subroutine

Utilizan una o varias variables, cuyo valor pueden modificar y devolver.

```
INTEGER SEC, X, HOUR, MIN
```

```
HOUR=4
```

```
MIN=3
```

```
SEC=40
```

```
Call TSECS(x, HOUR,MIN,SEC)
```

```
PRINT*,X
```

```
END
```

```
SUBROUTINE TSECS(TS,HH,MM,SS)
```

```
INTEGER HH,MM,SS,TS
```

```
TS=((HH*60)+MM)*60+SS
```

```
RETURN
```

```
END
```

No hay una diferencia formal entre entrada (input) y salida (output). Se recomienda no cambiar el valor de las variables de entrada

El orden de las variables en la llamada y en la definición ha de ser el mismo.

Acaba con

```
RETURN
```

```
END
```

Function

Devuelven un sólo valor de un tipo determinado (real, integer, etc) a partir de una serie de variables.

El orden de las variables ha de ser el mismo en la llamada a la function que en su definición. Veamos un ejemplo:

```
INTEGER SEC, X, HOUR, MIN
```

```
INTEGER TSECS
```

```
    HOUR=4
```

```
    MIN=3
```

```
    SEC=40
```

```
    X=TSECS(HOUR,MIN,SEC)
```

```
    PRINT*,X
```

```
END
```

```
INTEGER FUNCTION TSECS(HH,MM,SS)
```

```
INTEGER HH,MM,SS
```

```
    .....
```

```
    TSECS=((HH*60)+MM)*60+SS
```

```
RETURN
```

```
END
```

Hay que definir el tipo de la propia función, en el ejemplo, **integer**

El nombre interno de las variables no tiene que coincidir con el externo. El orden es lo importante. Esta función espera un hora, minuto y segundo, en ese orden.

La función tiene que tener un valor asignado al finalizar.

Acaba con

RETURN

END

Function, external

Permiten utilizar nombres de funciones como argumento en subroutines

```
REAL X1,X2,AREA
EXTERNAL FCN
...
CALL INTEGRA( FCN, X1, X2, AREA )
...
END

FUNCTION FCN( X )
...
    FCN = ....
RETURN
END

SUBROUTINE INTEGRA ( F, X0, X1, A )
...
RETURN
END
```

Common blocks

Las variables definidas en el programa principal, y las diversas **subroutines** o **functions** son locales. Esto es, sólo están definidas dentro de cada estructura.

Se pueden compartir datos utilizando COMMON BLOCKS

```
PROGRAM COSA1
  INTEGER A,B,C
  REAL D,P,Q
  COMMON/NUMEROS/D,A,B,C
  D=3.
  A=1
  B=2
  C=4
  CALL SUBRUT(P,Q)
  END
  SUBRUT(P,Q)
  REAL P,Q, POSI
  INTEGER HORA,MINU,SEGU
  COMMON/NUMEROS/POSI,HORA,MINU,SEGU
  PRINT*,POSI
  RETURN
  END
```

Cada COMMON BLOCK debe llevar un nombre.

Se pueden hacer varios.

El orden vuelve a ser fundamental.

Escritura en fichero

El comando es **WRITE (UNIT= , FMT=) A**

UNIT es el lugar donde se escribe.

- + En **pantalla** (o estandar output), *
- + en un fichero abierto anteriormente

```
OPEN(14,FILE="DATOS.DAT")  
WRITE(14,*) X,Y
```

```
CLOSE(14)
```

El **formato (FMT)** es importante, puede ser

- + estandar, *
- + con un control de los decimales que tenemos en el numero

```
X=3.0
```

```
Y=4.0
```

```
WRITE(*,100) X,Y
```

```
WRITE(*,200) X,Y
```

```
100    FORMAT(F12.6,2X,F12.4)
```

```
200    FORMAT(E12.3,2X,E3.2)
```

Salida:

```
3.000000    4.0000
```

```
0.300E+01 ***
```

```
PRINT*, "el valor es:", A
```

PRODUCE ALGO PARECIDO A

```
WRITE(*,*) "el valor es",A
```

Formatos:

Ejemplo

f coma flotante

f9.3

4.000

E precision simple

e12.4

0.4000E+01

D doble precisión

d12.6

0.400000E+01

I entero

I4

4

A carácter

A4

abcd

X espacio en blanco

/ espacio vertical, rotura de linea

T tabulador

Ejemplos:

Frase completa con resultados intercalados

```
300    FORMAT('la parte entera de x=',F12.6,' multiplicado',1x,  
1 'por y=',F12.4,2x,'es igual a',2x,i4)
```

10 columnas de números en precisión simple

```
400    format(10(e12.6,2x))
```

Lectura de datos a partir de un fichero

El comando es `READ (UNIT= , FMT=) A`

>**UNIT** es el lugar de donde se lee

- + desde `pantalla/teclado` (o estandar input), *
- + de un fichero abierto anteriormente

```
OPEN(14,FILE="DATOS.DAT")
```

```
READ(14,*) X,Y
```

```
CLOSE(14)
```

De esta manera leeríamos dos números `x` e `y` de la primera línea del fichero `DATOS.DAT`

Si no hubiera dos números, retornaría un error.

> **formato (FMT)** Normalmente es preferible leer sin formato, entendiendo que los números en el fichero han de estar separados por tabulador o espacios para que fortran los lea como distintos.

> **Read** lee números en una línea, si hay más números para leer seguirá en la siguiente línea.

> Si no conocemos cuántas líneas tiene el fichero, podemos decirle que cuando llegue al final salte a una etiqueta del código (en lugar de dar error).

```
READ(14,*,END=400) A,B
```

Salta a la línea 400 si llega al final del fichero.

Lectura de datos a partir de un fichero

```
program lectura
c ilustra el uso de READ
c el fichero datos.dat contiene
c      2.  4.  5.
c      3.  6.  8.
c      4.  8. 11.
c      5. 10. 14.
c variables que leemos, x,y,z
  real x,y,z
c abre el fichero datos en la unidad 15
  open(15,file="datos.dat")
c lee los dos primeros numeros
c y escribe su valor
  read(15,*) x,y
  write(*,100) x,y
100  format('los dos primeros números son:',f8.3,2x,f8.3)
c si ahora volvemos a leer leeremos la segunda linea
  read(15,*) x,y
  write(*,200) x,y
200  format('los dos números siguientes son:',f8.3,2x,f8.3)
end
```

Salida

```
los dos primeros números son:  2.000  4.000
los dos números siguientes son: 3.000  6.000
```

Lectura de datos a partir de un fichero

```
READ ( UNIT= , FMT= )
```

Si queremos realizar entradas desde el teclado:

```
READ(*,*) X,Y
```

> El programa en ejecución esperará a que el usuario introduzca dos números de dos formas:

```
+ 3.,4. (enter)
```

```
+ 3.(enter), 4.(enter)
```

Errores y problemas habituales

Problemas en el uso de número reales

Errores de:

- > Redondeo
- > Overflow/Underflow
- > Pérdida de dígitos significativos

Redondeo

Redondeo

Proceden del hecho de utilizar un número de dígitos significativos fijo.

```
X1=2.0E30
X2=2.0E8
X3=X1+X2
WRITE(*,*) ' Redondeo'
WRITE(*,*) ' NUMEROS  ',x1,x2
WRITE(*,*) ' X3-X1=X2=',X3-X1
WRITE(*,*) ' X2=',X2
END
```

Redondeo

```
NUMEROS    2.00000003E+30  200000000.
X3-X1=X2=  0.00000000
X2=  200000000.
```

Guardiola et al, Mètodes numèrics per a la física, p. 21-25

Overflow

Overflow

En real, tendremos problemas si el número supera 10^{38}

El compilador dará un “infinity” al correr o “arithmetic overflow” al compilar

Ex0104.for (Guardiola et al)

```
C    OVERFLOW FLOTANT
      x1=2.0E30
      x2=x1+2.0E20
      WRITE(*,*) ' Overflow flotant'
      WRITE(*,*) ' Nombres  ',x1,x2
      WRITE(*,*) ' 1/(x1*x2)',(1.0/x1)*(1.0/x2)
      WRITE(*,*) ' x1*x2  ',x1*x2
      X3=X1*X2
      PRINT*,X3+1
      END
```

Overflow flotant

Nombres	2.00000003E+30	2.00000003E+30
1/(x1*x2)	0.00000000	
x1*x2	Infinity	
Infinity		

El error de redondeo es mayor que la diferencia entre los números!

Mejora si utilizamos doble precisión

Overflow flotant

Nombres	2.0000000000000000E+030	2.0000000002000001E+030
1/(x1*x2)	2.4999999997499994E-061	
x1*x2	4.0000000004000002E+060	
4.0000000004000002E+060		

Guardiola et al, Mètodes numèrics per a la física, p. 21-25

Perdida de dígitos significativos

Suele aparecer al restar números próximos.

$$X1=4.100111\pm0.000001$$

$$X2=4.100222\pm0.000001$$

Si los restamos tenemos,

$$x3=x2-x1=0.000111\pm0.000002$$

Con lo que tenemos al final un número conocido con un 1% de precisión aunque partíamos de números con mucha más precisión

Guardiola et al, Mètodes numèrics per a la física, p.25

NaN (Not a Number)

```
PRINT*,SQRT(-3.0)  
END
```

```
AA=-3.0  
PRINT*,SQRT(AA)  
PRINT*, "Hola"  
END
```

Dependiendo del compilador determinadas operaciones pueden dar lugar a NaN.

En este caso el compilador detecta el problema y devuelve al compilar un **Error**

a.f:1.25:

```
print*,sqrt(-3.0)  
1
```

Error: Argument of SQRT at (1) has a negative value

En este caso el compilador no da ningun error, pero al ejecutar obtenemos:

NaN

Hola

Guardiola et al, Mètodes numèrics per a la física, p.25

Errors y Warnings

Para compilar el programa haremos:

Linux/MacOS>

```
gfortran programa1.f -o ejecutable.out
```

Windows>

```
gfortran programa1.f -o ejecutable.exe
```

Si la compilación ha funcionado satisfactoriamente el sistema no dirá nada. Habrá generado un fichero ejecutable.

Comprobad siempre que se ha generado un NUEVO ejecutable (mirando la hora de creación)

El proceso de compilación puede producir:

> **Warnings (avisos)**

- + El compilador avisa de posibles problemas

Warning: line ...

- + Los problemas no son tan graves (aparentemente) como para detener la compilación
- + Es bueno que el compilador no de Warnings

> **Errors (Errores)**

- + El compilador detiene el proceso y no genera ningún ejecutable

- + Escribe:

Error: Descripción del error y posible localización

Errors y Warnings

Mirad siempre los mensajes del compilador en orden

+ En muchos casos basta resolver el primer error detectado

```
implicit none
  integer i
  do i=1,30
    print*,i
  enddo
end
```

+ El compilador dice:

```
codig.f:3:8:
      integer i
      1
Error: Unclassifiable statement at (1)
codig.f:4:12:
      do i=1,30
      1
Error: Symbol 'i' at (1) has no IMPLICIT type
```

Errores típicos (0)

program err0

c ilustramos algunos errores tipicos
c el numero y tipo de argumentos con
c los que se llama (CALL) a una
c subrutina o funcion
c ha de ser el correcto
c bjd Sep 2015

implicit none

integer i,j,k

i=2

call suma2(i,j)

print*,i,j

end

subroutine suma2(x)

implicit none

integer x,xp2

xp2=x+2

return

end

program err0b

implicit none

integer i,j,k

i=2

call suma2(i)

print*,i,j

end

subroutine suma2(x,xp2)

implicit none

integer x,xp2

xp2=x+2

return

end

call suma2(i)

1

Warning: Missing actual argument for argument 'xp2' at (1)

err0.f:11.72:

call suma2(i,j)

1

Warning: More actual than formal arguments in procedure call at (1)

Errores típicos (1)

```
program err1
```

```
c ilustramos algunos errores tipicos  
c el tipo de argumentos con  
c los que se llama (CALL) a una  
c subrutina o funcion ha de ser el  
c correcto  
c bjd Sep 2015
```

```
implicit none
```

```
integer i,j,k
```

```
real p
```

```
i=2
```

```
call suma2(i,p)
```

```
print*,"i=",i," p=",p
```

```
end
```

```
subroutine suma2(x, xp2)
```

```
implicit none
```

```
integer x, xp2
```

```
    xp2=x+2
```

```
return
```

```
end
```

Compilación

```
call suma2(i,p)
```

```
1
```

Warning: Type mismatch in argument 'xp2' at (1); passed REAL(4) to INTEGER(4)

Salida

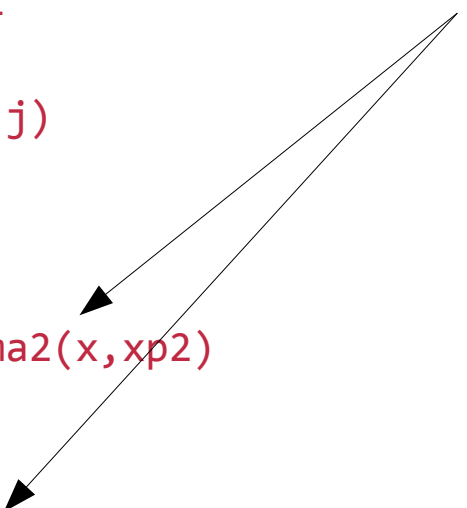
```
i=      2 p= 5.60519386E-45
```

Errores típicos (2)

```
program err2
c ilustramos algunos errores tipicos
c no podemos pasar como argumento
c a una subrutina una variable que
c ya estamos pasando por un common
c bjd Sep 2015
```

```
implicit none
integer i,j,k
common/equis/i
i=2
call suma2(i,j)
print*,i,j
end
```

```
subroutine suma2(x,xp2)
implicit none
integer x,xp2
common/equis/x
xp2=x+2
return
end
```



err1.f:18.20:

```
common/equis/x
1
```

Error: COMMON attribute conflicts with DUMMY attribute in 'x' at (1)

Errores típicos (3), durante la ejecución

Si no utilizamos

IMPLICIT NONE

El compilador **supone** que las variables que comienzan por

(a-h)-(o-z) son REAL y las
(i-h) INTEGER

Fuente:

<http://www.physics.usyd.edu.au/guides/f77errors.html>

<http://kb.mit.edu/confluence/display/istcontrib/Common+Fortran+Error+messages>

Errores de segmentación (al intentar asignar una posición no definida)

IMPLICIT NONE

INTEGER I(10),J

DO J=1,20

I(J)=J

ENDDO

END

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Backtrace for this error:

#0 0x7FB79AE24777

#1 0x7FB79AE24D7E

#2 0x7FB79AA7CD3F

Segmentation fault (core dumped)

Divide by zero

IMPLICIT NONE

INTEGER I,J,K

I=3

J=0

K=I/J

END

Program received signal SIGFPE: Floating-point exception - erroneous arithmetic operation.

Backtrace for this error:

#0 0x7F5B98A8E777

#1 0x7F5B98A8ED7E

#2 0x7F5B986E6D3F

#3 0x400683 in MAIN__ at err4.f:?

Floating point exception (core dumped)

Errores típicos (4)

Recordemos: (hay muchos formatos diferentes)

$4000. = 4E+03 = 0.4E+04$

$0.0005 = 0.5E-03$

Un número escrito como:

4. o 4.E0 o 4.0000

Es **REAL**

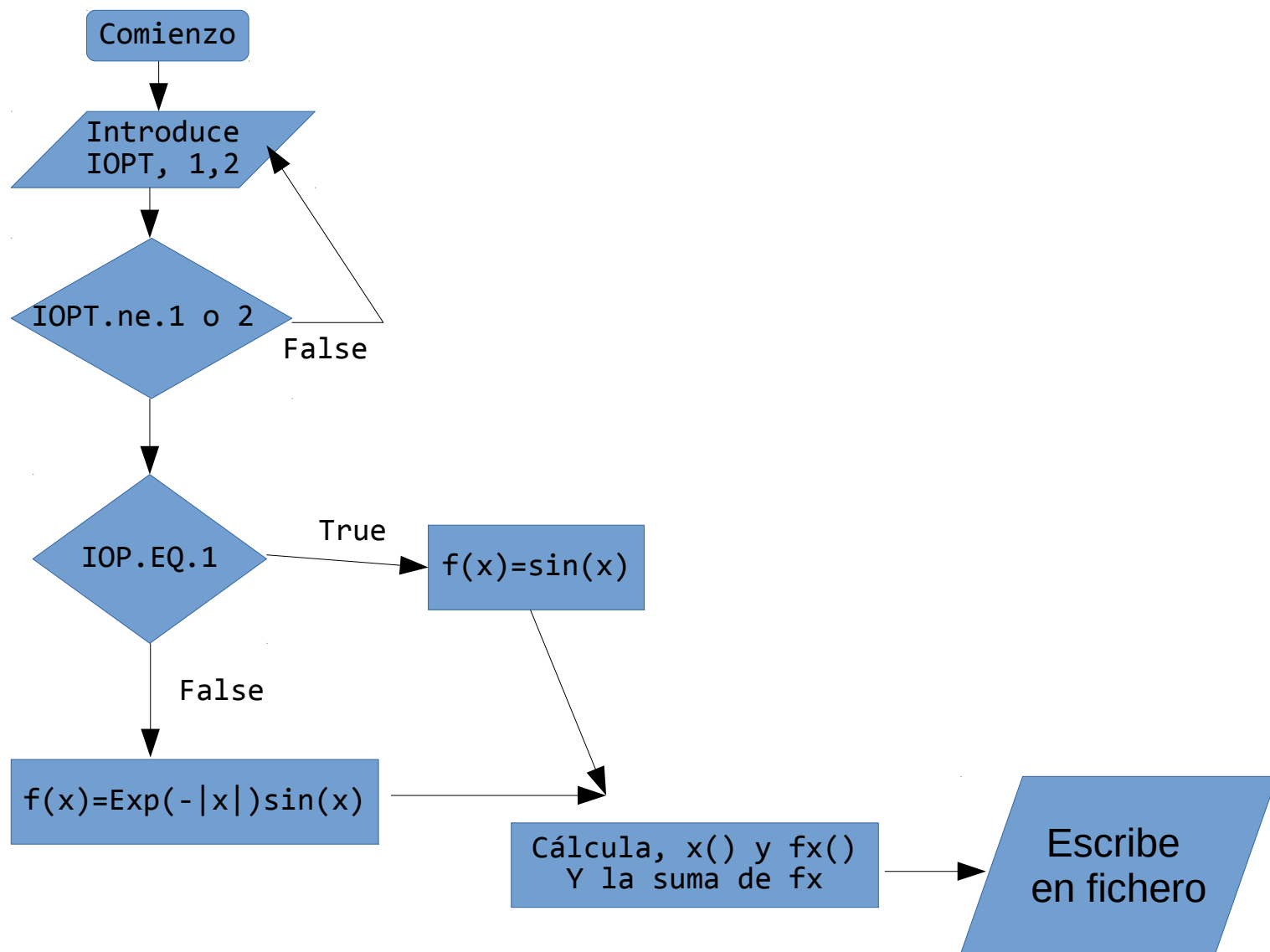
Un número escrito como:

4.D0

Es **DOUBLE PRECISION**

Ejemplo de programa sencillo

Ejemplo (programa1.f), diagrama flujo



Ejemplo (programa1.f), encabezado

```
C PROGRAMA SENCILLO PARA PROBAR EL COMPILADOR
C
C REALIZA ALGO DE LECTURA Y ESCRITURA
C CONTIENE: IF
C CONTIENE: DO
C
C CONSTRUYE UNA TABLA DE POSICIONES (ENTRE -5 Y 5) Y VALORES DE DOS
FUNCIONES.
C CALCULA LA SUMA DE LOS VALORES DE LA FUNCION
C
C LAS FUNCIONES SON SIN(X), EXP(-|X|) * COS(X)
C
C BJD SEP 2015
C
C LAST MODIFIED: 21 SEP 2015
C
```

Ejemplo (programa1.f), declaración variables

CODIGO

```
IMPLICIT NONE

C NUMERO DE POSICIONES
  INTEGER NDIM
  PARAMETER (NDIM=100)

  REAL XX(NDIM), FUNC1(NDIM), SUMA
  INTEGER IOPT, ERR, IESTADO
  INTEGER I

C INICIALIZA IOPT, SERVIRA PARA ESCOGER
  IOPT=0
```

Comentarios

Todas las variables necesitaran declararse explícitamente

Un entero, el número de puntos
No podremos modificarlo

X, func1, vectores de posiciones y valores de $f(x)$
Enteros, para controlar el flujo y gestionar errores de lectura
Entero usado como contador en un loop

Ejemplo (programa1.f), entrada datos

Busca que el usuario introduzca una opción controlando los posibles errores. Por ejemplo, si en lugar de 1 o 2 escribe un carácter u otro número.

```
10  WRITE(*,*) 'QUE FUNCION QUIERES ESCRIBIR:  
1  (1) SIN(X), (2) EXP(-X)*SIN(X) '  
    READ(*,*,ERR=20,IOSTAT=IESTADO) IOPT  
  
20  IF ((IESTADO.NE.0).OR.(IOPT.NE.1.AND.IOPT.NE.2)) THEN  
    WRITE(*,*) 'OPCION INCORRECTA, ESCRIBA 1 o 2 '  
    GOTO 10  
ENDIF
```

Ejemplo (programa1.f), cálculos

```
C CALCULA LOS VALORES DE LA FUNCION
```

```
DO I=1,NDIM
```

```
C POSICIONES DE -5 A 5, CON PASO CONSTANTE
```

```
XX(I) = -5. + REAL(I)/NDIM*10.
```

```
IF (IOPT.EQ.1) THEN
```

```
  FUNCI(I) = SIN(XX(I))
```

```
ELSE
```

```
  FUNCI(I)=EXP(-ABS(XX(I)))*SIN(XX(I))
```

```
ENDIF
```

```
ENDDO
```

```
C CALCULA LA SUMA DE LOS VALORES DE LA FUNCION
```

```
SUMA=0.
```

```
DO I=1,NDIM
```

```
  SUMA=SUMA+FUNCI(I)
```

```
ENDDO
```

Bucle principal, para calcular los valores de "x" y f(x) en un conjunto finito de puntos

Ejemplo (programa1.f), escritura

C ESCRIBE LOS DATOS EN UN FICHERO

```
OPEN(14,FILE='salida.dat')
DO I=1,NDIM
    WRITE(14,100) XX(I),FUNCI(I)
100    FORMAT(F9.3,2X,F12.6)
ENDDO
CLOSE(14)
```

Loop para escribir
los datos en un
fichero

C ESCRIBE EL VALOR DE LA SUMA EN PANTALLA

```
WRITE(*,*) ' LA SUMA VALE ',SUMA
WRITE(*,*) ' POR QUE NO VALE ZERO LA SUMA?'
END
```

END, finaliza el
programa

Algunas instrucciones de fortran>77 (fortran90, etc)

Cambios y comandos útiles

Si quereis utilizar fortran 90:

- > extension del fichero .90, e.g. programa.f90
- > No hace falta respetar la columna 7, etc.

C BUCLE DO WHILE

```
DO WHILE (condicion)
  COSAS
ENDDO
```

C Allocate / deallocate

```
Real, allocatable :: matriu(:, :)
integer ix, iy
```

```
read (*, *) ix, iy
allocate(matriu(ix, iy))
```

...

```
deallocate(matriu)
```


Modules

Una alternativa elegante a los common blocks, que a veces puede ser mas complicado de debuggear es el uso de modules. Ejemplo (de <http://fortranwiki.org/fortran/show/Compiling+and+linking+modules>)

```
C !> \file testModule.f
module mathModule

    implicit none
    private
    real, public, parameter :: &
        pi = 3.1415 , &
        e = 2.7183 , &
        gamma = 0.57722

end module mathModule
```

```
!> \file mainProg.f
program testing
    use mathModule
    implicit none

    print *, "pi:", pi, "e:", e, "gamma:", gamma

end program testing
```

Sobre common vs modules, <http://iprc.soest.hawaii.edu/users/furue/improve-fortran.html>