

ELEMENTOS DE FORTRAN 90





CAPÍTULO 1 INTRODUCCIÓN





Conjunto de caracteres

- Letras:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

- Dígitos:

0 1 2 3 4 5 6 7 8 9

- Caracteres especiales:

= + - * / () , . ' : ! " % & ; < > ? \$ _ espacio



Formas de escritura del código fuente

● FORMA FIJA

● FORMA LIBRE





FORMA FIJA

LAS COLUMNAS DE LA 1 A LA 5 ESTÁN RESERVADAS PARA ETIQUETAS.

LA COLUMNA 6 ES USADA PARA INDICAR LA CONTINUACIÓN DE LA LÍNEA ANTERIOR.

LAS COLUMNAS DE LA 7 A LA 72 SON USADAS PARA LAS INSTRUCCIONES FORTRAN.

LAS COLUMNAS 73 EN ADELANTE SON IGNORADAS



FORMA LIBRE

```
Lahey ED4w - [d_f.f90 (Edited) Window 1]
File Edit Block Buffer Goto Search Macro Tool Options Window Help
[Icons]
! Last change: RWV 14 Feb 2005 9:45 pm

SUBROUTINE delphi_fortran(valor,sat,sat1)
dll_export delphi_fortran
integer:: i=1,num_dat=21
REAL(kind=8),DIMENSION(21)::sat,sat1
REAL(KIND=8)::valor

do i=1,num_dat,1
sat(i)= i-valor
sat1(i)=i+valor
end do
END SUBROUTINE

SUBROUTINE wyllie( sfm_i, wyllie_for, seleccion_sistema,S,k1,k2)
dll_export wyllie
INTEGER:: i=1,num_dat=21, wyllie_for, seleccion_sistema
REAL(KIND=8),DIMENSION(21)::sat,s_eff,krfnm,krfm,S,k1,k2
REAL(KIND=8) sfm_i,exp_krfnm, exp_krfm

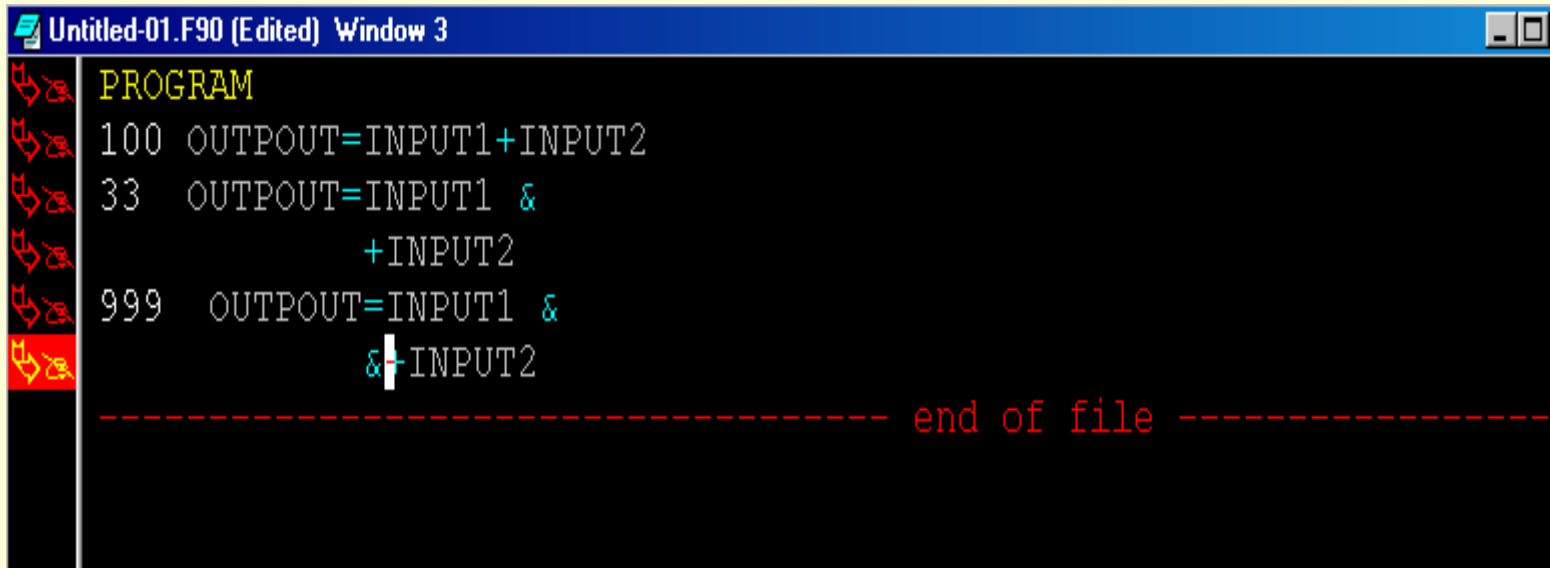
sat=(/0.,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95,1./)
s_eff=0.
k1=0.
k2=0.

IF (seleccion_sistema==1) THEN
DO i=1,num_dat,1
s_eff(i) = (sat(num_dat+1-i)-sfm_i)/(1-sfm_i)
END DO
END IF
IF (seleccion_sistema==2) THEN
DO i=1,num_dat,1
s_eff(i) = (sat(num_dat+1-i))/(1-sfm_i)
END DO
END IF

SELECT CASE (wyllie_for)
CASE (1)
exp_krfnm=3.
exp_krfm=3.
CASE (2)
```

1-132

FORMA LIBRE



The image shows a screenshot of a text editor window titled "Untitled-01.F90 (Edited) Window 3". The editor has a dark background and a light blue title bar. On the left side, there is a vertical toolbar with several icons. The code is written in a monospaced font. The first line is "PROGRAM" in yellow. The second line is "100 OUTPUT=INPUT1+INPUT2" in white. The third line is "33 OUTPUT=INPUT1 &" in white, followed by "+INPUT2" on the next line, also in white. The fourth line is "999 OUTPUT=INPUT1 &" in white, followed by "&+INPUT2" on the next line, also in white. A red dashed line is drawn across the width of the editor, with the text "end of file" in red centered below it.

```
PROGRAM
100 OUTPUT=INPUT1+INPUT2
33  OUTPUT=INPUT1 &
      +INPUT2
999  OUTPUT=INPUT1 &
      &+INPUT2
----- end of file -----
```

Se recomienda usar siempre el estilo de forma libre con programas
FORTRAN 90/95

La estructura de un programa FORTRAN

```
Lahey ED4W - [c:\ejemplos_clase\primer_programa.f90 Window 1]
File Edit Block Buffer Goto Search Macro Tool Options Window Help
! Last change: RVW 7 Jan 2005 3:46 pm
PROGRAM primer_programa
!Proposito
!Ilustras las formas basicas de un programa FORTRAN

!Declaracion de las variables usadas en este programa
INTEGER::i,j,k !todas las variables son enteras

!Lee dos variables
WRITE (*,*)'Introduzca dos numeros'
READ (*,*) i,j

!Multiplica los dos numeros
k=i*j

!Escribe el resultado
WRITE (*,*)'resultado=',k

!Termina el programa
STOP
END PROGRAM
----- end of file -----
```

Declaración

Ejecución

Terminación



Nombres en FORTRAN

(variables y constantes)

Nombres validos

Tiempo

Distancia

Z234545654

Presión_vapor

Nombres no válidos

este_nombre_es_muy_pero_muy_largo

3_dias

a\$



Integer

intervalo: -2147483647 a 2147483647

Valid integer

0

-999

123456789

+17

Not valid integer

1,000,000

-100.



Real

Valid real

10.

-999.9

123.45E20

0.12E+1

Not valid real

1,000,000.

111E3

-12.0E1.5



Character

Valid character

‘Esta es una prueba’

‘ ’

{^}

“3.1415”

Not valid character

Esta es una prueba

‘Esta es una prueba’

”Esta es una prueba’

Logical

Valid Logical

Not valid Logical

.TRUE.

TRUE

.FALSE.

.FALSE





CAPÍTULO
2
ESTRUCTURAS DE
DECISIÓN



Operadores Lógicos

OPERADOR		SIGNIFICADO
Nuevo estilo	Estilo anterior	
.EQ.	==	igual
.NE.	/=	diferente
.LT.	<	menor que
.LE.	<=	menor o igual que
.GT.	>	mayor que
.GE.	>=	mayor o igual que



OPERACIÓN

RESULTADO

3<4

.TRUE.

3<=4

.TRUE.

3==4

.FALSE.

3>4

.FALSE.

4<=4

.TRUE.

'A' < 'B'

.TRUE.



FUNCIONES LÓGICAS

OPERADOR	FUNCION	DEFINICIÓN
A.AND.B	Y	RESULTADO .TRUE. SI A Y B SON VERDADERAS
A.OR.B	O	RESULTADO .TRUE. SI CUALQUIERA A O B ES VERDADERA
.A.EQV.B	EQUIVALENCIA	RESULTADO .TRUE. SI A=B
.A.NEQV.B	NO EQUIVALENCIA	
.NOT.B	NO	



Arithmetic operator

illegal expression

$a*-b$

$a** -2$


$x(y+z)$

legal expression

$a*(-b)$

$a**(-2)$

$x*(y+z)$





Integer Arithmetic

Integer arithmetic always produces an integer result

$$\frac{3}{4} = 0 \quad \frac{5}{4} = 1 \quad \frac{8}{4} = 2$$






Real Arithmetic

$$\frac{3.}{4.} = 0.75$$

$$\frac{8.}{4.} = 2.$$

$$\frac{5.}{4.} = 1.25$$




JERARQUÍA DE OPERACIONES

- El contenido de los paréntesis es evaluado primero, de adentro hacia afuera
 - Todos los exponenciales se evalúan de derecha a izquierda
 - Las multiplicaciones y divisiones se evalúan de izquierda a derecha
 - Las sumas y restas se evalúan de izquierda a derecha
-



DECLARACIÓN DE VARIABLES

El tipo de variable debe de ser declarada en la sección de declaración al comienzo del programa, con el uso de las siguientes instrucciones no ejecutables.

INTEGER:: var1, var2, var3

REAL:: var1, var2, var3

LOGICAL:: var1, var2, var3





DECLARACIÓN DE VARIABLES

El tipo de variable debe de ser declarada en la sección de declaración al comienzo del programa, después de la instrucción **PROGRAM**, ejemplo:

PROGRAM ejemplo

INTEGER:: dia

REAL:: segundo



DECLARACIÓN DE DATOS TIPO CHARACTER

CHARACTER(len=<len>)::var1, var2

CHARACTER(len=10)::uno, dos

CHARACTER::inicial

CHARACTER(15)::id





ALMACENAMIENTO DE CONSTANTES EN UN PROGRAMA

Se le asigna un nombre constante a un valor para garantizar que su valor sea el mismo en todo el programa mediante el atributo **PARAMETER**, la forma de declararlo es:

Type, **PARAMETER**:: nombre=valor

Ejemplo:

REAL, PARAMETER :: pi= 3.141593





INSTRUCCIONES DE ASIGNACIÓN Y VARIABLES DE TIPO CHARACTER


character_nombre_variable= expresión tipo carácter

CHARACTER (len=3)::archivo

Archivo='f'

CHARACTER (len=3)::archivo2

Archivo2='file01'





SUBCADENAS ESPECIFICADAS

Selecciona una porción de una variable tipo carácter y la trata como si fuera una variable tipo carácter independiente, ejemplo:

```
PROGRAM prueba
```

```
CHARACTER (len=8)::a,b,c
```

```
a='ABCDEFGHIJ'
```


```
b='12345678'
```

```
c=a(5:7)
```

```
b(7:8)=a(2:6)
```

```
END PROGRAM
```

```
c=EFG; b=123456BC
```





OPERADOR DE CONCATENACIÓN (//)

Permite combinar dos o más cadenas o subcadenas para formar otra cadena, ejemplo

```
PROGRAM prueba  
CHARACTER (len=10)::a  
CHARACTER (len=8)::b,c  
a='ABCDEFGHIJ'  
b='12345678'  
c=a(5:7)// b(4:5)//a(6:8)  
END PROGRAM
```

```
c='EFG45FGH'
```





Variables: Default y explícitas

● Default:

Integer: nombres de variables comienzan con las letras **I,J,K,L,M,N**

Reales: Las restantes





FUNCIONES INTRINSECAS


Una función FORTRAN calcula un valor de salida a partir de uno o más valores de entrada. Los valores de entrada a la función se conocen como el **argumento**:

$y = \text{SIN}(\text{alfa})$ //argumento en radianes

$y = \text{SIN}(\text{alfa} * (3.141593/180.))$ //factor de conversión de grados a radianes

REAL, PARAMETER::grados_a_radianes=3.141593/180.

$y = \text{SIN}(\text{alfa} * \text{grados_a_radianes})$





FUNCIONES INTRINSECAS

El argumento de una función puede ser una constante, una variable, una expresión o el resultado de otra función, ejemplo:

$$y = \text{SIN} (3.141593)$$

$$y = \text{SIN} (x)$$

$$y = \text{SIN} (\text{pi} * x)$$

$$y = \text{SIN} (\text{SQRT}(x))$$





FUNCIONES INTRINSECAS

Dependiendo del tipo de dato que sea requerido por una función, éstas pueden ser funciones genéricas o funciones específicas:

Funciones genéricas: pueden usar uno o más tipos de datos,
ejemplo: `ABS(x)`

Funciones específicas: solo pueden usar un específico dato de entrada.





INSTRUCCIONES DE ENTRADA Y SALIDA DE DATOS

INSTRUCCIÓN READ(*.*)

Ejemplo:

```
PROGRAM ejemplo_datos_entrada
```

```
INTEGER::i,j
```

```
REAL::a
```

```
CHARACTER(len=12)::chars
```

```
READ(*,*)i,j,a,chars
```

```
END PROGRAM
```

Los datos tienen que ser escritos en el orden que se pidan en el programa:

1,2,3.,'número'

Los datos pueden estar separados por comas o espacios, o por líneas





INSTRUCCIONES DE ENTRADA Y SALIDA DE DATOS

INSTRUCCIÓN READ(*.*)

Ejemplo:

```
PROGRAM ejemplo_datos_entrada2  
INTEGER::i,j,k,l; READ(*,*)i,j;READ(*,*)k,l
```

```
END PROGRAM
```


Si los datos de entrada del programa son

1,2,3,4

5,6,7,8

Los valores que leerá el programa son

$i=1, j=2, k=5, l=6$



INSTRUCCIONES DE ENTRADA Y SALIDA DE DATOS

INSTRUCCIÓN WRITE(*.*)

Ejemplo:

```
PROGRAM ejemplo_datos_salida
```

```
INTEGER::ix
```

```
LOGICAL::prueba
```

```
REAL::beta
```

```
ix=1
```

```
Prueba=.TRUE.
```

```
Beta=3.141593
```

```
WRITE(*,*) ' ix =          ', ix
```

```
WRITE(*,*) ' beta =          ', beta
```

```
WRITE(*,*) ' coseno beta =          ', COS(beta)
```

```
WRITE(*,*) ' prueba =          ', prueba
```

```
WRITE(*,*) REAL(ix), NINT(beta)
```

```
WRITE(*,*) IX, ' ', BETA, ' '
```

```
13.141593
```

```
END PROGRAM
```



INICIALIZACIÓN DE VARIABLES

```
PROGRAM inicializacion
```


```
INTEGER::i
```

```
WRITE(*,*)i
```

```
END PROGRAM
```

El valor de *i* varía según el compilador, En LF90 las variables se inicializan en cero.

Siempre se deben de inicializar las variables





INICIALIZACIÓN DE VARIABLES

PROGRAM inicializacion

INTEGER::i

i= 1

WRITE(*,*)i

END PROGRAM

PROGRAM inicializacion

INTEGER::i

READ(*,*) i

WRITE(*,*)i

END PROGRAM

PROGRAM inicializacion

INTEGER::i=1

WRITE(*,*)i

END PROGRAM

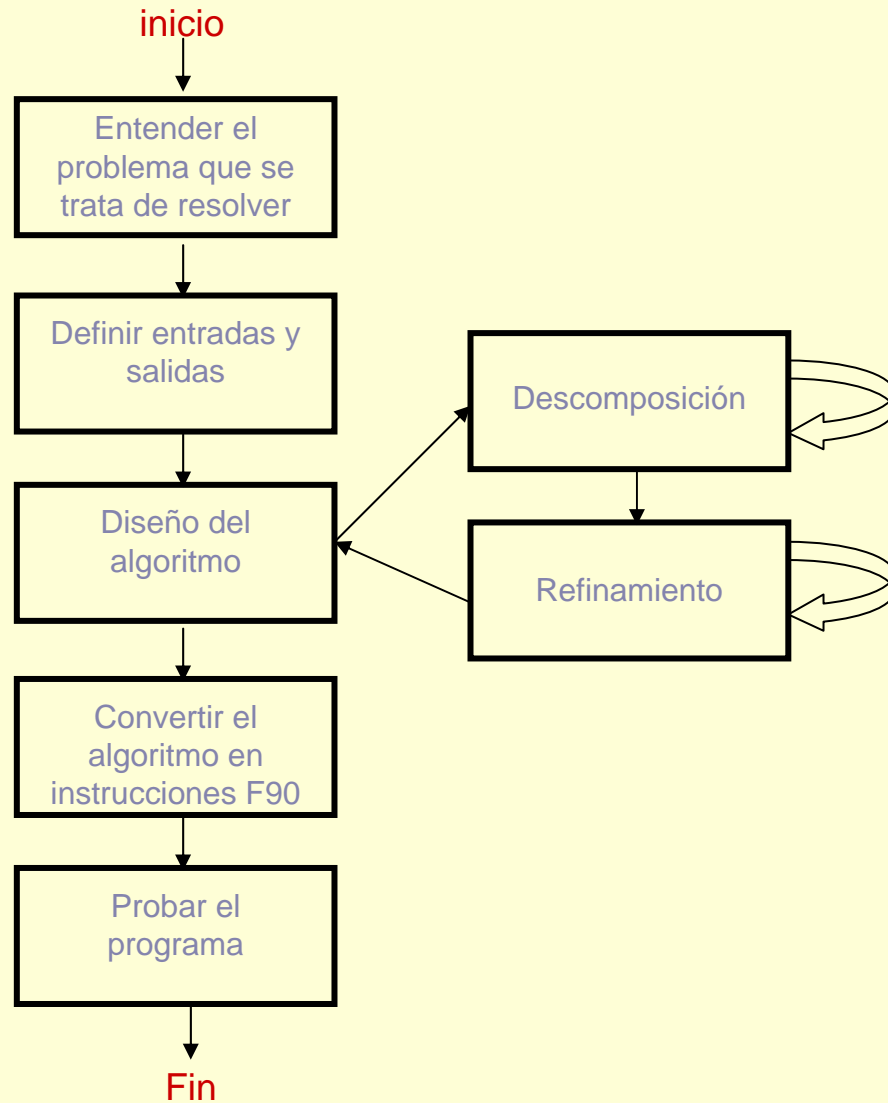


Compilando, enlazando y ejecutando un programa FORTRAN



- Batch mode
- Interactive mode

DISEÑO DE PROGRAMAS





PSEUDOCÓDIGO

Ejemplo:

El usuario introduce un valor de temperatura en grados Fahrenheit

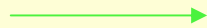
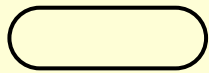
Leer temperatura en grados Fahrenheit (temp_F)

Temp_K en kelvin ← $(5./9.)*(temp_F-32)+273.15$

Escribir temperatura en Kelvin



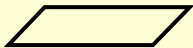
DIAGRAMAS DE FLUJO



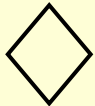
Indica el comienzo o final de un algoritmo



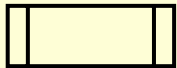
Indica un cálculo, el resultado del cálculo se le asigna a una variable



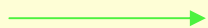
Indica una operación de entrada o salida



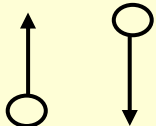
Indica un punto donde se escoge que hacer entre dos alternativas



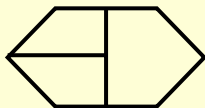
Indica una referencia a un procedimiento



Indica la dirección del programa

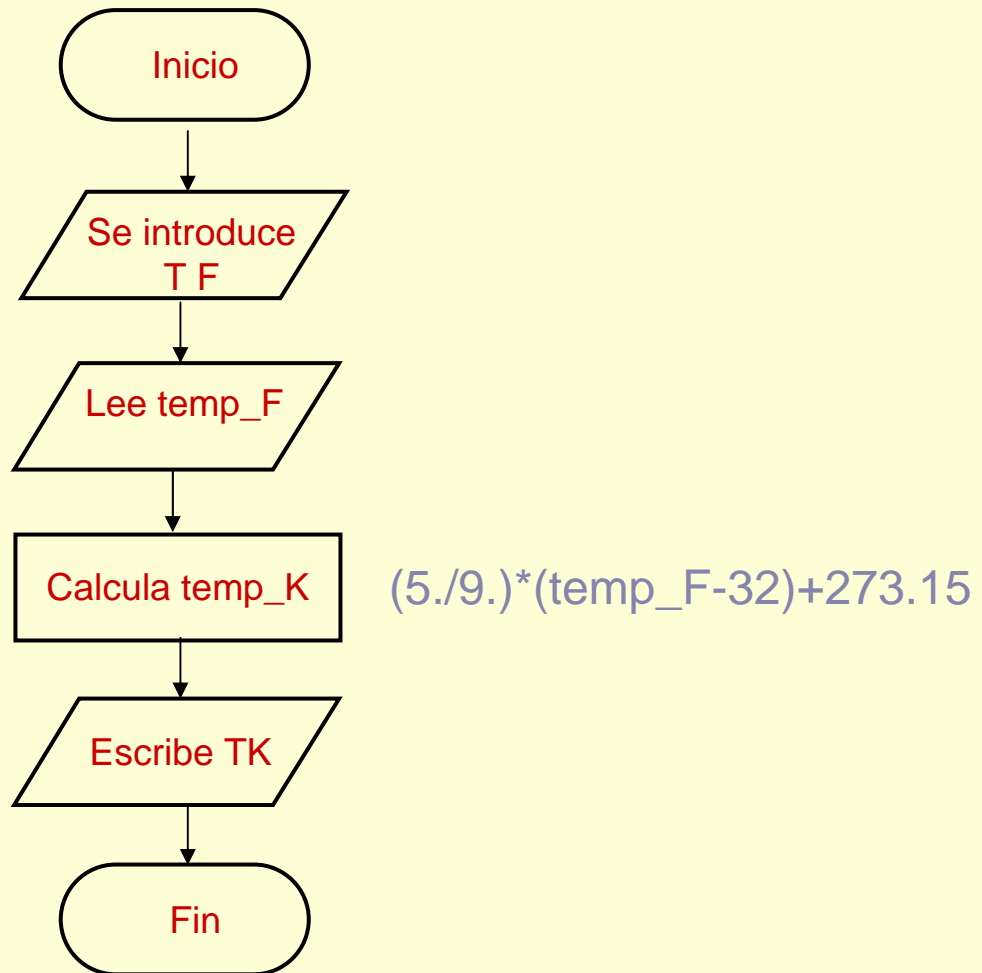


Indica procesos iterativos



DIAGRAMAS DE FLUJO

Ejemplo:





ESTRUCTURAS DE CONTROL

INSTRUCCIÓN **IF**

INSTRUCCIÓN **SELECT CASE**

Estas instrucciones nos permiten seleccionar y ejecutar secciones específicas de código (llamadas bloques)





INSTRUCCIÓN IF

IF (expresión lógica) THEN

Instrucción 1

Instrucción 2

.

.

.

END IF

```
PROGRAM prueba_3
```

```
REAL::a=3,b=2
```

```
IF (a>b) THEN
```

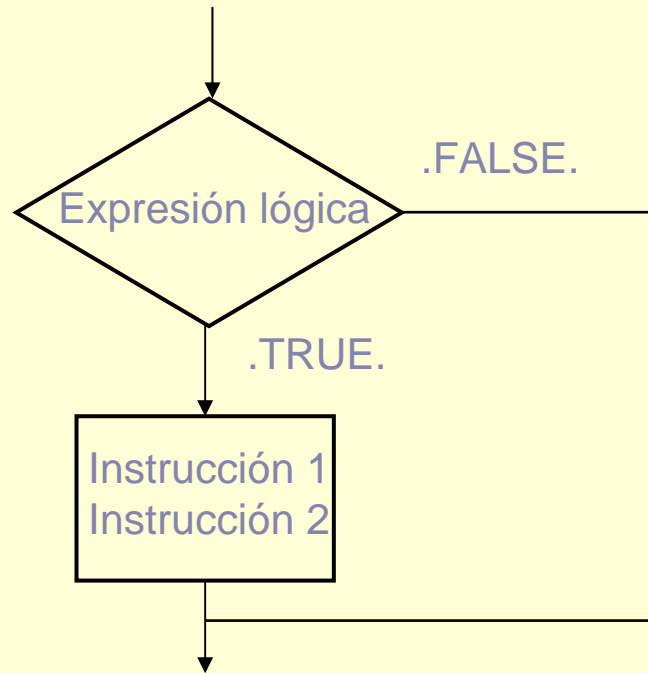
```
WRITE(*,*) a
```

```
END IF
```

```
END PROGRAM
```



DIAGRAMA DE FLUJO DE LA INSTRUCCIÓN IF





IF – ELSE IF- ELSE

IF (expresión lógica 1) **THEN**

Instrucción 1

Instrucción 2

ELSE IF (expresión logica 2) **THEN**

Instrucción 1

Instrucción 2

ELSE

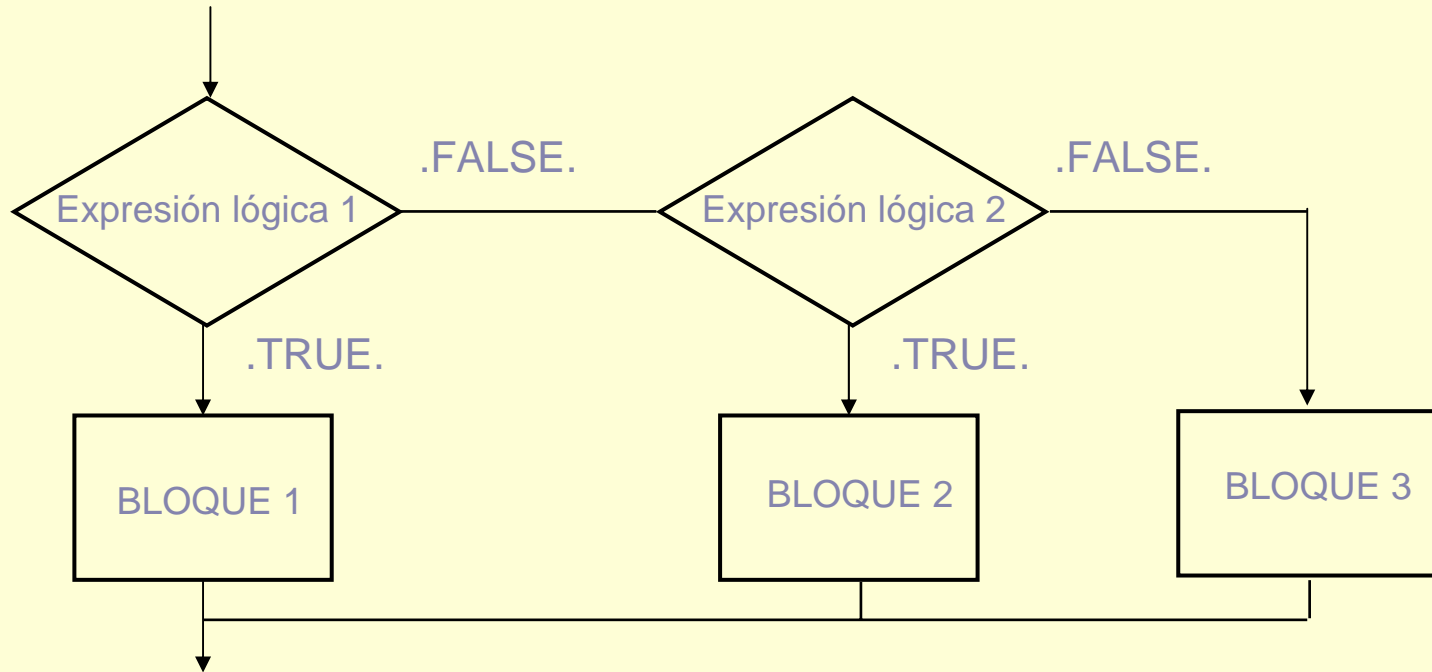
Instrucción 1

Instrucción 2

END IF



DIAGRAMA DE FLUJO DE LA INSTRUCCIÓN **IF-ELSE IF- ELSE**





CONSTRUCCIÓN CASE

SELECT CASE (case expresión)

CASE(case expresión 1)

Instrucción 1

Instrucción 2

.

.

CASE(case expresión 2)

Instrucción 1

Instrucción 2

.

.

CASE DEFAULT

Instrucción 1

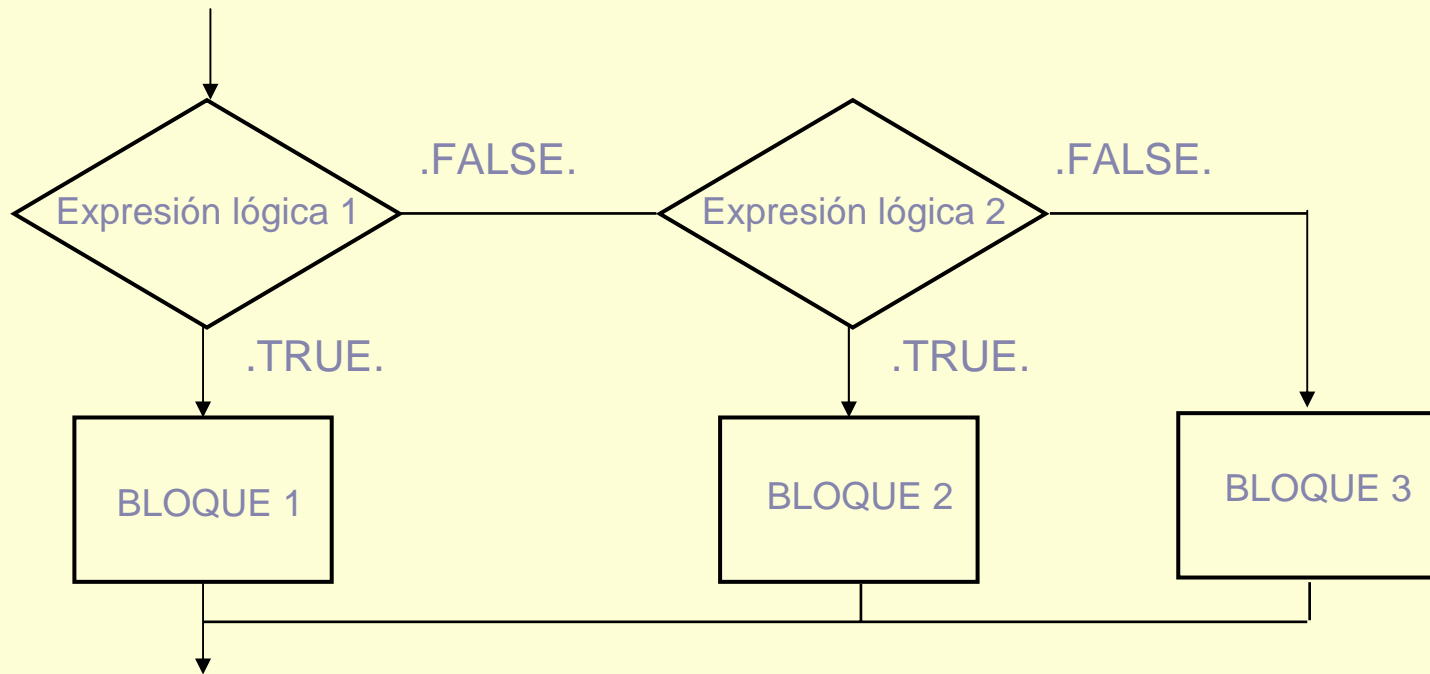
Instrucción 2


.

.


END SELECT

DIAGRAMA DE FLUJO DE LA INSTRUCCIÓN CASE





CAPÍTULO
3
CICLOS



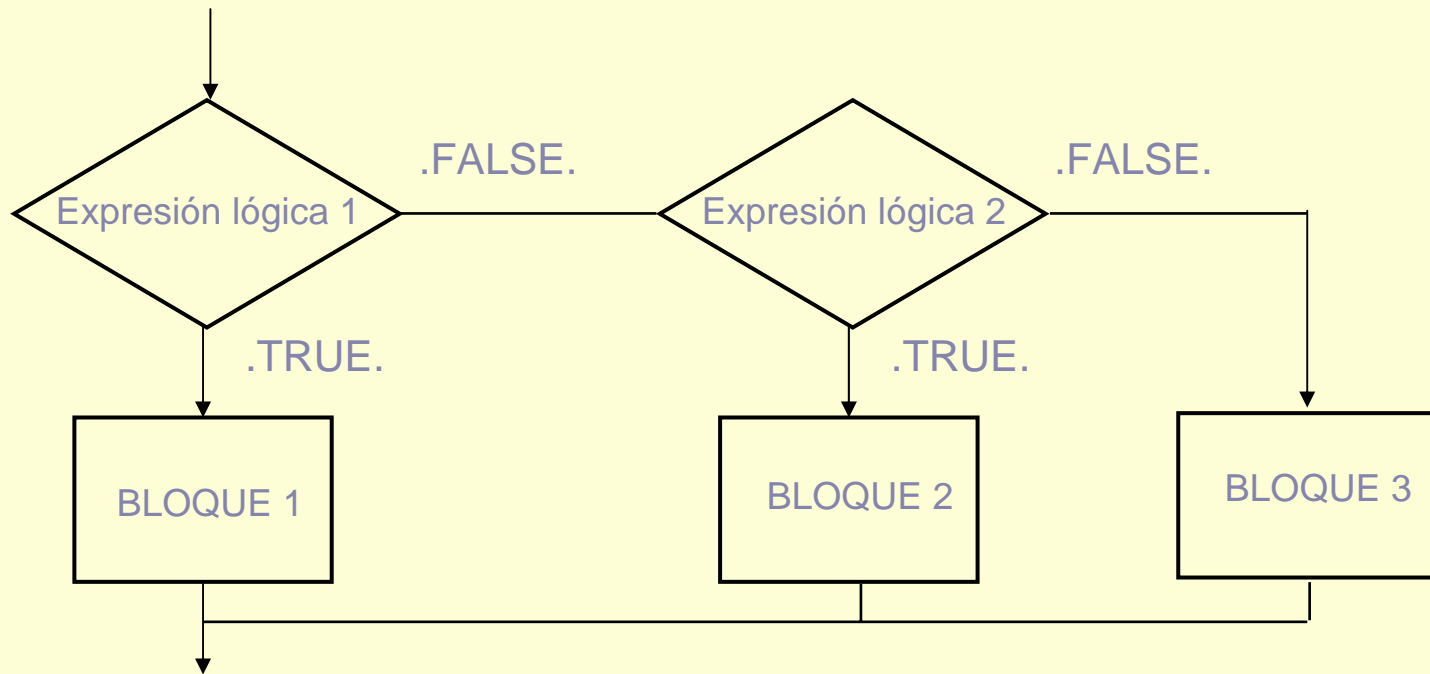


CICLOS WHILE

```
DO  
.....  
IF (expresión lógica) EXIT  
.....  
END DO
```



DIAGRAMA DE FLUJO CICLO WHILE





CICLOS DO WHILE

DO WHILE (expresión lógica)

.....

.....

.....

END DO





CICLOS ITERATIVOS

DO índice=valor inicial,valor final,incremento

Instrucción 1

Instrucción 2

Instrucción n

END DO

Ejemplos:

DO 10 I=1,5

DO 80 J=1,13,2

DO 5,ABC=-1,1,.1

DO 999 X=1.5,12,.5

DO 80 n=10,1,-1

DO 50 R=SIN(0.0), COS(0.0), .1

DO 710 L=N, K+3, J-1



CAPÍTULO
4
ARREGLOS






ARREGLOS

UN ARREGLO ES UN CONJUNTO DE DATOS, TODOS DEL MISMO TIPO Y PARÁMETROS DE TIPO .
SON ESPECIFICADOS CON EL ATRIBUTO DIMENSION

EJEMPLO:

```
real (kind=selected_real_kind(5,30)), &  
dimension(0:5,1:7)::matriz,matriz_alfa
```

```
real,dimension(10,10)::coeficientes
```





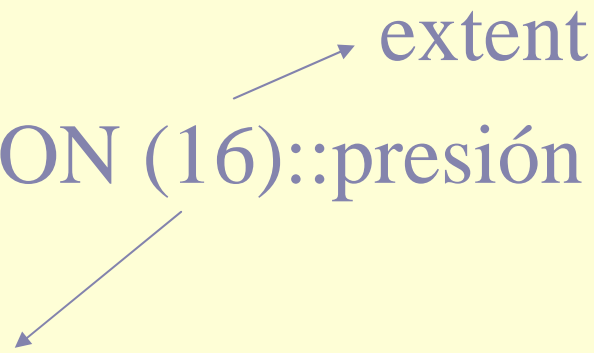
Declaración de Arreglos

Se debe de indicar el número de elementos del arreglo:

REAL, DIMENSION (16)::presión

extent

Rank del arreglo



Rank = 1, extent=16





SUBÍNDICES

ESPECIFICAN A UN ELEMENTO EN PARTICULAR DE UN ARREGLO

EJEMPLOS:

Real,dimension (10)::x,y

.

.

!un subíndice puede ser una constante, una variable o una expresion


$$y(4)=x(k)$$

$$y(5)=x(1+n-1)/150$$

$$y(6)=x(1)+2$$

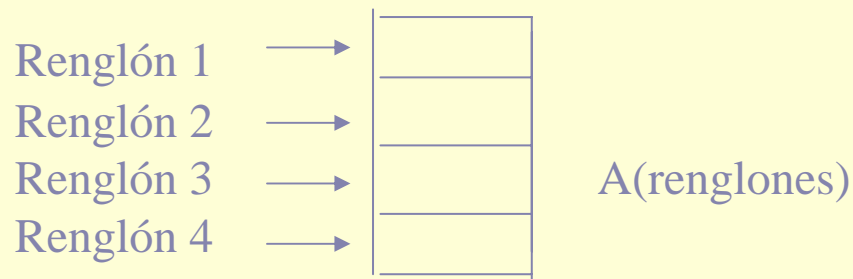
!el valor numérico de la expresión a-b es truncado, porque los subíndices deben

!ser enteros

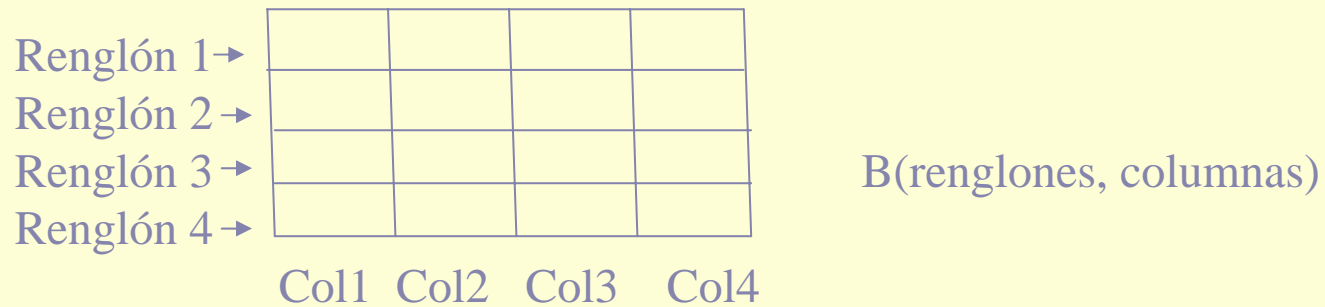
$$y(7)=x(a-b)$$


Arreglos de Rango 2

Vector



Matriz

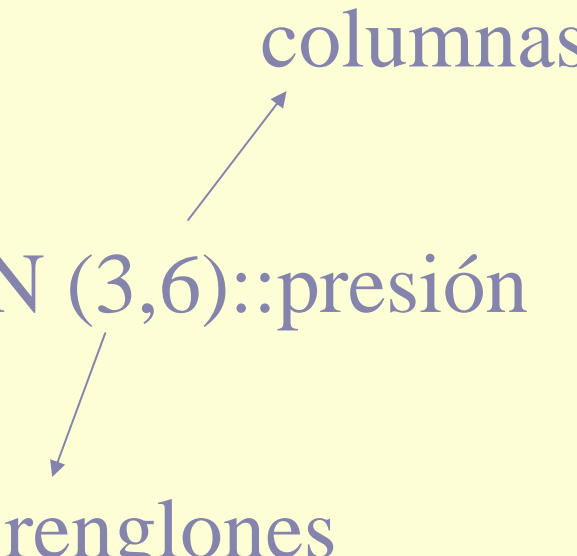




DECLARACIÓN DE ARREGLOS (Rank=2)

Se debe de indicar el número de elementos del arreglo:

REAL, DIMENSION (3,6)::presión



columnas

renglones



CONSTRUCTORES DE ARREGLOS

Un constructor de arreglo es una lista de valores encerrados entre los delimitadores (/ y /). Ejemplos:

!en el constructor de arreglos para el vector c, i toma
!un valor inicial de 4, un valor final de 6 con un incremento
!de 1

integer,dimension(3)::a,b=(/13,-5,14/),c=(/(i,i=4,6)/)

b(1)	-----
	13
b(2)	-----
	-5
b(3)	-----
	14

c(1)	-----
	4
c(2)	-----
	5
c(3)	-----
	6

CONSTRUCTORES DE ARREGLOS

La función RESHAPE se usa para dar valores a los arreglos de rango mayor que 1:

!el argumento shape=(/2,2/) indica que se trata

!de un arreglo de 2 filas y 2 columnas

```
real,dimension(2,2)::matriz_ejemplo=&
```

```
reshape((/35.1,-12.4,5.7,4.21/),shape=(/2,2/))
```

```
          |-----|  
matriz_ejemplo(1,1) | 0.3510000E+02|
```

```
          |-----|  
matriz_ejemplo(2,1) |-0.1240000E+02|
```

```
          |-----|  
matriz_ejemplo(1,2) | 0.5700000E+01|
```

```
          |-----|  
matriz_ejemplo(2,2) | 0.4210000E+01|
```

```
          |-----|  
character,dimension(2)::carac=(/'A','B','C','D'/)
```

```
type prueba
```

```
logical::a
```

```
real::b,c
```

```
end type prueba
```

```
type(prueba),dimension(2)::var=&
```

```
(/prueba(.true.,1.0_4,2.9_4),prueba(.false.,3.7,-4e-2)/)
```



Arreglos ALLOCATABLE

Indican al compilador que la forma y el tamaño del arreglo serán determinados en tiempo de ejecución.

Ejemplos:

```
real (kind=selected_real_kind(12,25)),allocatable::y(: , :)
```

```
complex
```

```
allocatable f50 (: , :)
```






OPERACIONES CON ARREGLOS

Un elemento de un arreglo puede ser empleado de la misma forma que una variable escalar es usada en expresiones, asignaciones o en instrucciones de lectura/escritura:

```
a(5)=b(7)-c(m-1)  
print *,x(12)
```

Al cambiar el valor de uno o más subíndices de un arreglo, es posible hacer referencia a varios elementos del mismo:

```
do i=1,20  
c(i)=a(i)*b(i)  
end do
```





SECCIONES DE ARREGLOS

Una sección de un arreglo es indicada mediante una lista de subíndices:

$a(2:8:2)$! $a(2),a(4),a(6)$ y $a(8)$

$(a(k),k=2,8,2)$!equivalente a la instrucción anterior

$matriz(6,1:n_columnas)$!sexta fila de matriz

$x(1:n_filas,1)$!primera columna de x

$b(1,3:1:-1)$! $b(1,3),b(1,2)$ y $b(1,1)$

$vect(8:5)$!esta sección es un arreglo de tamaño cero





Construcción WHERE

La construcción WHERE decide qué elementos de un arreglo serán afectados por un bloque de instrucciones de asignación (asignación de arreglos mediante una máscara).

WHERE(máscara)

·
instrucciones de asignación

·
ELSEWHERE

·
instrucciones de asignación

·
END WHERE





CAPÍTULO
5
LECTURA/ESCRITURA





Instrucción OPEN

Conecta o reconecta un archivo externo a una unidad de lectura/escritura. Ejemplos de instrucciones OPEN:

!el archivo `c:\lf90\src\equi.dat` se conecta a la unidad 19
`open(unit=19,file='c:\lf90\src\equi.dat')`





Instrucción READ

La instrucción READ transfiere los valores proporcionados a través de una unidad de lectura a las entidades especificadas en una lista de lectura o en un grupo NAMELIST. Los siguientes son ejemplos del uso de la instrucción READ:

!100 es una etiqueta que identifica a una instrucción

!FORMAT

```
read(unit=5,fmt=100)a,i
```

```
read(5,100)a,i !instrucción equivalente a la anterior
```





Instrucción READ

Un * en el lugar de la etiqueta correspondiente a un FORMAT, indica que la lectura se llevará a cabo con un formato predefinido; los valores se proporcionarán separados por una , o por espacios:

```
read(5,*)a,i
```

Las cláusulas FMT= y FILE= pueden ser dadas en cualquier orden:

```
read(fmt=827,unit=38)k,l,m,n
```




Instrucción WRITE

La instrucción WRITE transfiere valores a una unidad de escritura desde las entidades especificadas en una lista de escritura o en un grupo NAMELIST. PRINT es equivalente a un WRITE

!100 es la etiqueta de la instrucción FORMAT

```
write(unit=6,fmt=100)a,b,nombre
```

!equivalente a la instrucción anterior

```
write(6,100)a,b,nombre
```

!la lista de escritura puede contener constantes, variables

!o expresiones

```
write(fmt=1000,unit=315)5,a+b,xyz
```





Instrucción WRITE

Un * en vez del número de la unidad indica que se trata de la unidad estándar de salida. Otro * en el lugar de la etiqueta de la instrucción FORMAT, ocasiona que la información sea escrita de acuerdo a un formato predefinido por el compilador.

```
write(*,*)a,f,'constante'
```

!195 es la etiqueta del formato y la unidad de escritura

!es la estándar

```
print 195,a,j+k,l
```

!el formato es predefinido y la unidad de escritura

!es la estándar (escritura dirigida por lista)

```
print *,a,j+k,l
```





Instrucción WRITE

La cláusulas ERR, IOSTAT y ADVANCE también son aplicables a un WRITE:

```
write(6,125,err=17)x,y,z
```

```
w=(x+y+z)/70.0
```

```
print *,w
```

```
stop
```

```
17 print *,'error en la escritura de las variables x,y y z'
```

```
a=17.5e-3;b=7.8 4;mm=17
```

```
!(3f10.4) es el formato de escritura
```

```
write(*,'(3f10.4)',iostat=ierror)a,b,mm
```

```
select case(ierror)
```

```
case(-1)
```

```
print *,'condición de fin de archivo o fin de registro'
```

```
case(1:)
```

```
print *,'error en la escritura'
```

```
end select
```





Instrucción FORMAT

Provee información explícita para dirigir la edición entre la representación interna de los datos y los caracteres que son leídos o escritos. Ejemplo:

```
read(5,89)a,j
```

```
89 format(f10.2,i4)
```

donde el formato (f10.2,i4) especifica que el valor de a será leído en las primeras 10 columnas del registro y que el valor de b será leído en las siguientes 4 columnas de ese mismo registro.






Instrucción FORMAT

La siguiente es una muestra de una instrucción FORMAT aplicada en una operación de escritura

```
write(6,127)a,j  
127 format(1x,f10.2,i4)
```

en la cual se utiliza al trazador de control de edición `nx` para avanzar `n` espacios dentro de un registro. Por lo tanto, en el último ejemplo, el registro de impresión consistirá de una columna en blanco, del valor de la variable `a` escrito en las siguientes 10 columnas y del valor de la variable `b` impreso en las siguientes 4 columnas.





Instrucción FORMAT

Una instrucción FORMAT puede ser proporcionada a través de una cadena de caracteres:

```
read(5,'(f10.2,i4)')a,j
```

```
write(6,'(1x,f10.2,i4)')a,j
```

```
character *10 formato
```

```
read *,formato
```

```
read(5,formato)a,j
```

```
character *10 formato
```

```
read *,formato
```

```
write(6,formato)a,j
```





Edición de la lectura/escritura de datos (Trazador Fw.d)

El trazador Fw.d permite la edición de datos reales, donde w es el ancho del campo en columnas y d son los dígitos después del punto decimal. La edición de datos de tipo entero es con el trazador Iw, donde w también es el ancho del campo en columnas. A manera de ilustración; el par READ-FORMAT


```
read(unit=5,fmt=100)a,j
```


.

.

```
100 format(f10.2,i3)
```

implica que la variable a será leída en las 10 primeras columnas del registro de lectura, de las cuales las 2 últimas se consideran reservadas para los decimales. Las siguientes 3 columnas serán usadas para leer el valor de la variable j.





Edición de la lectura/escritura de datos (Trazador Fw.d)


Por lo tanto, si los datos contenidos en el registro de lectura son:

17438 2
1234567890123456789012345678901234567890

entonces a tendrá el valor 174.38 y j el valor entero 2. En caso de que el valor correspondiente a la variable a se represente con el punto decimal, la especificación de que la últimas 2 columnas son decimales será ignorada:

174.38 2
1234567890123456789012345678901234567890





Edición de la lectura/escritura de datos (Trazador Ew.d)

El trazador de edición Ew.d es empleado para leer o escribir datos reales con un formato de notación normalizada:


```
read(5,717)z,x
```

```
717 format(e10.2,e12.4)
```

```
-317E+02 +6.03E23
```

```
1234567890123456789012345678901234567890
```

Aquí, para la variable z son utilizadas las primeras 10 columnas, de las cuales, las 2 anteriores a la letra E están reservadas para los decimales. Las siguientes 12 columnas son para la variable x. Entonces, z tendrá el valor -3.17×10^2 y x será igual a 6.03×10^{23} .



Edición de la lectura/escritura de datos (Trazador Ew.d)

El trazador Dw.d es empleado para leer o escribir datos de doble precisión:

```
double precision x  
read(5,10)x,j,k
```

.


.

```
10 format(d11.5,2i3)
```

```
77.8D2      5-20
```

```
1234567890123456789012345678901234567890
```

El número 2 que antecede al trazador i3 es un factor de repetición, es decir, 2i3 es equivalente a i3,i3. En este ejemplo, x será 77.8×10^2 , j 5 y k -20. Sin embargo, los datos REAL (KIND=8) o DOUBLE PRECISION también pueden ser leídos o escritos con el trazador Fw.d o Ew.d.





CAPÍTULO
6
SUBPROGRAMAS





Unidades de Programa

Las unidades de programa son los elementos más pequeños que pueden ser compilados en forma separada. Hay cinco clases de unidades de programa:

- 1) Programas principales
 - 2) Subprogramas externos FUNCTION
 - 3) Subprogramas externos SUBROUTINE
 - 4) BLOCK DATA
 - 5) Módulos (no soportados en FORTRAN 77)
-



FUNCIONES

- FUNCIONES ELEMENTALES

Sin, cos, tan, exp, log, log10, mod, sqrt

Ejemplo:

```
DO i=1,10  
  A(i)=sin(x(i))  
END DO
```





FUNCIONES

- FUNCIONES INQUIRY

ALLOCATED (array)

Determina la condición del almacenamiento

LBOUND (ARRAY)

El límite inferior del arreglo

SHAPE(ARRAY)

La forma del arreglo

UBOUND(ARRAY)

El límite superior del arreglo

SIZE(ARRAY)

Tamaño del arreglo





FUNCIONES

- Transformational functions

ALL(MASK), ANY(MASK), COUNT(MASK),
DOT_PRODUCT(Vector a, vector b),
MATMUL(matriz a, matriz b),
MAXLOC(ARRAY, MASK),
MAXVAL(ARRAY, MASK),
MINLOC(ARRAY, MASK),
MINVAL(ARRAY, MASK),
PRODUCT(ARRAY, MASK),
RESHAPE(ARRAY, MASK),
SUM(ARRAY, MASK),
TRANSPOSE(MATRIZ)



SUBROUTINAS

Una subrutina es un procedimiento FORTRAN que es solicitado con la instrucción CALL y que recibe valores de entrada y regresa resultados mediante el argumento. La forma general de una subrutina es:

```
SUBROUTINE subroutine_name(argument_list)
```

```
Sección de declaración
```


```
Sección de ejecución
```


```
RETURN
```

```
END SUBROUTINE
```

Para llamar a una subrutina se utiliza la instrucción CALL:

```
CALL subroutine_name(argument_list)
```





```
subroutine hipotenusa(lado1,lado2,hip)
implicit none
```

```
REAL, INTENT(IN)::lado1
```

*atributo que indica que es un valor de entrada
y que no cambiará su
valor durante la ejecución de la subroutine*

```
REAL, INTENT(IN)::lado2
```

```
REAL, INTENT (OUT)::hip
```

atributo que indica que es un valor de salida

```
REAL:: temp
```

```
temp=lado1**2+lado2**2
```

```
hip=SQRT(temp)
```

variables locales

```
end subroutine
```





Proposiciones Intrinsic y External

El atributo EXTERNAL indica el nombre de un procedimiento externo. Al declarar a una función o a una subrutina como externa, es posible usarla como un argumento actual:


```
program muestra_uno
real,external::hipotenusa
read *,a,b
!hipotenusa es una función externa
call proc(a,b,result,hipotenusa)
print *,result
end program muestra_uno
subroutine proc(x,y,res,funci)
real,external::funci
res=x*y-funci(x,y)
end subroutine proc
function hipotenusa(a,b)
hipotenusa=sqrt(a**2+b**2)
end function hipotenusa
```




Proposiciones Intrinsic y External

El atributo INTRINSIC especifica que una función es intrínseca. El declarar a una función como intrínseca, permite usarla como un argumento actual:

```
program muestra_dos
real,intrinsic::alog !la función intrínseca debe ser
read *,a,b, !específica
!
!se pasa la función intrínseca alog como un argumento
!actual
call proc(a,b,result,alog)
print *,result
end program muestra_dos
subroutine proc(x,y,res,funci)
res=x+y-funci(x)
end subroutine procedimiento
```





Atributo INTENT

- INTENT (IN)


El argumento es dato de entrada

- INTENT (OUT)

El argumento es dato de salida

- INTENT (INOOUT)

El argumento actúa como
entrada y salida



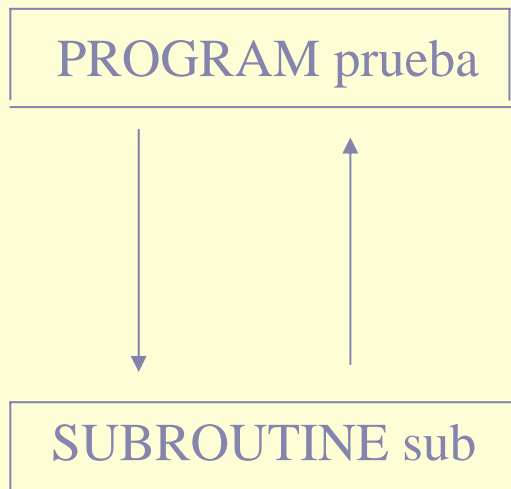


Atributo OPTIONAL

Un argumento actual no necesita ser provisto para corresponder a un argumento fingido, si es que se especifica el atributo OPTIONAL.



Esquema de referencia “Pass-by”



```
PROGRAMA prueba  
Real::a,b(4)  
Integer::sig  
CALL sub(a,b,sig)  
END PROGRAM
```

```
SUBROUTINE sub(x,y,i)  
Real, intent (out)::x  
Real, dimension(4), intent(in)::y  
Integer::i  
END subroutine
```



Pasando arreglos a subrutinas

```
Subroutine proceso(d1,d2,n,nv)
```

```
Integer, intent (in)::n,nv
```

```
Real, intent (in), dimension(n)::d1
```

```
Real, intent (out), dimension(n)::d2
```

```
End subroutine
```

```
Subroutine proceso(d1,d2,nv)
```

```
Integer, intent (in)::nv
```

```
Real, intent (in), dimension(*)::d1
```

```
Real, intent (out), dimension(*)::d2
```


```
End subroutine
```





Funciones de proposición


Una función de proposición es una función definida por el usuario, en una sola línea y con una sola expresión. La función de proposición solamente puede ser llamada dentro de la unidad de programa en la cual está definida.





Procedimientos genéricos

Una proposición INTERFACE con un nombre específica una interfase genérica para cada uno de los procedimientos en el bloque de interfase. De esta manera se pueden crear procedimientos genéricos, análogos a los procedimientos intrínsecos genéricos.





CAPÍTULO
7
MÓDULOS





Subprograma MODULE

Un módulo es una unidad de programa que permite "empaquetar" información como:

- 1)datos globales
- 2)definiciones de tipos
- 3)subprogramas
- 4)bloques de interfase
- 5)grupos de NAMELIST.

Los módulos deben aparecer antes de cualquier otra unidad de programa en un archivo fuente, o en su defecto, ser compilados por separado previamente.

Para que la información contenida en un módulo esté disponible en una unidad de programa, se emplea la instrucción USE seguida por el nombre del módulo. El proceso de tener acceso a la información de un módulo se conoce como asociación por uso (use association).



Procedimientos de módulo

Los procedimientos que son definidos dentro de un módulo se conocen como procedimientos de módulo (module procedures) y son escritos después de un **CONTAINS**



Instrucción COMMON

La instrucción COMMON (declarativa) permite que diferentes variables en distintas unidades de programa compartan físicamente las mismas localidades de memoria. El uso de la instrucción COMMON permite definir una área de memoria común, llamada bloque COMMON, que es compartida por cualquier unidad de programa que contenga la instrucción COMMON adecuada:




Proposición EQUIVALENCE

La proposición EQUIVALENCE (declarativa) permite que diferentes variables compartan las mismas localidades de memoria dentro de una unidad de programa. Por ejemplo, en equivalence (a,x,f1) se declara que una localidad de memoria es conocida como a, x o f1.



Proposición DATA

La proposición DATA proporciona valores iniciales para variables y arreglos dentro de una unidad de programa. DATA es una instrucción no ejecutable y puede colocarse en cualquier parte del programa, después de las declaraciones de tipo correspondientes a las variables que intervienen en ella.





Subprograma BLOCK DATA

Es una unidad de programa que se emplea para proporcionar valores iniciales a las variables y/o arreglos de un bloque COMMON etiquetado, a través de una proposición DATA. A las variables en un COMMON blanco no es factible darles valores con la proposición DATA, ni siquiera en un BLOCK DATA.



CAPÍTULO
8
ARCHIVOS



