

Programación en FORTRAN

Nivel Básico - Sesión 3

Martin Josemaría Vuelta Rojas

23 de enero de 2018

SoftButterfly

1. Declaración de arreglos
2. Asignación en arreglos
3. Instrucciones y operaciones exclusivas de arreglos
4. Operadores matriciales
5. Lectura y escritura de archivos

Declaración de arreglos

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.
 - Arreglo unidimensional: indexado de valores a un índice

$$x_{ni}, x_{ni+1}, \dots, x_{ns-1}, x_{ns}$$

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.
 - Arreglo unidimensional: indexado de valores a un índice

$$x_{ni}, x_{ni+1}, \dots, x_{ns-1}, x_{ns}$$

ubicados por el índice i y acotados por $ns \geq ni$.

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.

- Arreglo unidimensional: indexado de valores a un índice

$$x_{ni}, x_{ni+1}, \dots, x_{ns-1}, x_{ns}$$

ubicados por el índice i y acotados por $ns \geq ni$.

- Arreglo bidimensional: indexado de valores a dos índices.

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.

- Arreglo unidimensional: indexado de valores a un índice

$$x_{ni}, x_{ni+1}, \dots, x_{ns-1}, x_{ns}$$

ubicados por el índice i y acotados por $ns \geq ni$.

- Arreglo bidimensional: indexado de valores a dos índices.

ubicados por los índices i, j ("filas y columnas") y acotados por ni y ns , mi y ms respectivamente.

$$A = \begin{pmatrix} x_{ni,mi} & x_{ni,mi+1} & \dots & x_{ni,j} & \dots & x_{ni,ms} \\ : & : & & : & & : \\ x_{ns,mi} & x_{ns,mi+1} & \dots & x_{ns,j} & \dots & x_{ns,ms} \end{pmatrix}$$

Definición y tipos de arreglos

- Un arreglo es el conjunto de una variable con un identificador a los elementos de una lista o grupo.
- Se pueden formar arreglos con distinta cantidad de dimensiones.

- Arreglo unidimensional: indexado de valores a un índice

$$x_{ni}, x_{ni+1}, \dots, x_{ns-1}, x_{ns}$$

ubicados por el índice i y acotados por $ns \geq ni$.

- Arreglo bidimensional: indexado de valores a dos índices.

ubicados por los índices i, j ("filas y columnas") y acotados por ni y ns , mi y ms respectivamente.

$$A = \begin{pmatrix} x_{ni,mi} & x_{ni,mi+1} & \dots & x_{ni,j} & \dots & x_{ni,ms} \\ : & : & & : & & : \\ x_{ns,mi} & x_{ns,mi+1} & \dots & x_{ns,j} & \dots & x_{ns,ms} \end{pmatrix}$$

- Arreglo n -dimensional: lista indexada a n índices i_1, i_2, \dots, i_n acotados por un valor inferior y uno superior.

Declaración de arreglos

- La forma más simple de declarar un arreglo es la estática, al definir el número y rango de índices en la declaración.

Declaración de arreglos

- La forma más simple de declarar un arreglo es la estática, al definir el número y rango de índices en la declaración.
- La notación para el rango de un arreglo es
 $ni : ns$ que significa $ni \leq i \leq ns$.

Declaración de arreglos

- La forma más simple de declarar un arreglo es la estática, al definir el número y rango de índices en la declaración.

- La notación para el rango de un arreglo es

$ni : ns$ que significa $ni \leq i \leq ns$.

- La declaración de un arreglo, de valores de un tipo dado, se realiza mediante el atributo DIMENSION o directamente sobre la variable de acuerdo a lo siguiente:

`<tipo>, DIMENSION(<decl. índices>), [<otros atrib.>]::<lista var>`

`<tipo>, [<otros atrib.>]::<variable>(<decl. índices>), <otras var>`

Declaración de arreglos

- La forma más simple de declarar un arreglo es la estática, al definir el número y rango de índices en la declaración.
- La notación para el rango de un arreglo es
 $ni : ns$ que significa $ni \leq i \leq ns$.
- La declaración de un arreglo, de valores de un tipo dado, se realiza mediante el atributo DIMENSION o directamente sobre la variable de acuerdo a lo siguiente:

`<tipo>, DIMENSION(<decl. índices>), [<otros atrib.>] :: <lista var>`
`<tipo>, [<otros atrib.>] :: <variable>(<decl. índices>), <otras var>`

```
1  INTEGER , PARAMETER :: NR=5
2  INTEGER , PARAMETER :: NC=10
3  INTEGER , PARAMETER :: NF=3
4  INTEGER :: Row,Column,Floor
5  CHARACTER*1 , DIMENSION(1:NR,1:NC,1:NF) :: Seats=' '
```

Véase seatplan.f95

Declaración de arreglos

Asignación de valores a arreglos

Asignación de valores a arreglos

- La definición de los valores de un arreglo pueden darse componente por componente o de manera global.

Asignación de valores a arreglos

- La definición de los valores de un arreglo pueden darse componente por componente o de manera global.
- La asignación componente por componente se da la misma forma que una variable independiente.

Declaración de arreglos

Asignación de valores a arreglos

- La definición de los valores de un arreglo pueden darse componente por componente o de manera global.
- La asignación componente por componente se da la misma forma que una variable independiente.

```
1  REAL(kind=8), DIMENSION::vector
2  REAL(kind=4)::matriz(2,2)
3  :
4  vector(1)=1.d0; vector(2)=2.d0; vector(3)=3.d0
5  matriz(1,1)=1.; matriz(2,1)=0
6  matriz(1,2)=0.; matriz(2,2)=vector(2)*vector(3)
```

Declaración de arreglos

- La asignación de manera global, en el caso de arreglos unidimensionales, se denota con delimitadores (/y/) de la forma
$$\langle \text{arreglo} \rangle = (/ \langle \text{listado de } (n_s - n_i) + 1 \text{ expresiones} \rangle /)$$

Declaración de arreglos

- La asignación de manera global, en el caso de arreglos unidimensionales, se denota con delimitadores (/y/) de la forma
$$\langle \text{arreglo} \rangle = (/ \langle \text{listado de } (ns-ni)+1 \text{ expresiones} \rangle /)$$
donde $\langle \text{arreglo} \rangle$ es de DIMENSION(ni:ns).

Declaración de arreglos

- La asignación de manera global, en el caso de arreglos unidimensionales, se denota con delimitadores (/y/) de la forma
 $\text{<arreglo>} = (/ \text{<listado de (ns-ni)+1 expresiones>} /)$
donde <arreglo> es de DIMENSION(ni:ns).

```
1 REAL , DIMENSION(12) :: RainFall = &  
2 (/3.1,2.0,2.4,2.1,2.2,2.2,1.8,2.2,2.7,2.9,3.1,3.1/)
```

Véase rainfalloperations.f95

Declaración de arreglos

Declaración Dinámica de tableros

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).

Declaración de arreglos

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).
- La creación de estos arreglos pueden resumirse en tres momentos:

Declaración de arreglos

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).
- La creación de estos arreglos pueden resumirse en tres momentos:
 - La notificación, asignando un tipo al arreglo; la forma (# de índices) a través de DIMENSION junto con la opción ALLOCATABLE.

Declaración de arreglos

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).
- La creación de estos arreglos pueden resumirse en tres momentos:
 - La notificación, asignando un tipo al arreglo; la forma (# de índices) a través de DIMENSION junto con la opción ALLOCATABLE.

```
1 REAL, DIMENSION(:), ALLOCATABLE::vector
2 COMPLEX, DIMENSION(:, :), ALLOCATABLE::matriz
3 CHARACTER(len=8), DIMENSION(:, :, :), ALLOCATABLE::grilla
```

Declaración de arreglos

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).
- La creación de estos arreglos pueden resumirse en tres momentos:
 - La notificación, asignando un tipo al arreglo; la forma (# de índices) a través de DIMENSION junto con la opción ALLOCATABLE.

```
1 REAL, DIMENSION(:), ALLOCATABLE::vector
2 COMPLEX, DIMENSION(:, :), ALLOCATABLE::matriz
3 CHARACTER(len=8), DIMENSION(:, :, :), ALLOCATABLE::grilla
```

- La creación del arreglo con el rango de índices asignado, se realiza mediante la instrucción ALLOCATE.

Declaración de arreglos

Declaración Dinámica de tableros

- Asignación de memoria de manera dinámica durante la ejecución de un programa (Fortran 90 en adelante).
- La creación de estos arreglos pueden resumirse en tres momentos:
 - La notificación, asignando un tipo al arreglo; la forma (# de índices) a través de DIMENSION junto con la opción ALLOCATABLE.

```
1  REAL, DIMENSION(:), ALLOCATABLE::vector
2  COMPLEX, DIMENSION(:, :), ALLOCATABLE::matriz
3  CHARACTER(len=8), DIMENSION(:, :, :), ALLOCATABLE::grilla
```

- La creación del arreglo con el rango de índices asignado, se realiza mediante la instrucción ALLOCATE.

```
1  :
2  n=10
3  :
4  ALLOCATE(vector(3:n), matriz(1:n, 0:n-1), grilla(3, 4, 2))
```

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

- Finalmente la destrucción o anulación del arreglo, se realiza mediante la instrucción `DEALLOCATE`

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

- Finalmente la destrucción o anulación del arreglo, se realiza mediante la instrucción `DEALLOCATE`

```
1 DEALLOCATE(vector, matriz)
2 DEALLOCATE(grilla)
```

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

- Finalmente la destrucción o anulación del arreglo, se realiza mediante la instrucción `DEALLOCATE`

```
1 DEALLOCATE(vector, matriz)
```

```
2 DEALLOCATE(grilla)
```

- Es posible saber si el espacio de memoria para el arreglo es suficiente, agregando la opción `stat` a la instrucción `ALLOCATE`.

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

- Finalmente la destrucción o anulación del arreglo, se realiza mediante la instrucción `DEALLOCATE`

```
1 DEALLOCATE(vector, matriz)
```

```
2 DEALLOCATE(grilla)
```

- Es posible saber si el espacio de memoria para el arreglo es suficiente, agregando la opción `stat` a la instrucción `ALLOCATE`.

```
1 ALLOCATE(vector(1:n), stat=error)
```

Declaración de arreglos

La instrucción `ALLOCATE` puede crear varios arreglos al mismo tiempo.

- Finalmente la destrucción o anulación del arreglo, se realiza mediante la instrucción `DEALLOCATE`

```
1 DEALLOCATE(vector, matriz)
```

```
2 DEALLOCATE(grilla)
```

- Es posible saber si el espacio de memoria para el arreglo es suficiente, agregando la opción `stat` a la instrucción `ALLOCATE`.

```
1 ALLOCATE(vector(1:n), stat=error)
```

entonces `error = 0` y el arreglo se habrá creado.

Asignación en arreglos

Asignación en arreglos

- La asignación de valores a un arreglo también puede darse por subarreglos, es decir, bloques de arreglos o arreglos locales.

- La asignación de valores a un arreglo también puede darse por subarreglos, es decir, bloques de arreglos o arreglos locales.

```
1  REAL, DIMENSION(3,2)::matriz1
2  REAL, DIMENSION(-1:1,0:1)::matriz2
3  :
4  matriz1(1,1)=1           ! asignacion a un solo elemento
5  matriz1=1                ! asignacion a todo el tablero con el valor 1
6  matriz2=matriz(2,2)      ! asignacion de todo el tablero matriz2 con
7                           ! el valor de matriz(2,2)
8  matriz2=matriz1          ! asignacion, copiando los valores de la matriz
```

Subarreglos

Subarreglos

- Los subarreglos se identifican mediante el uso de la sintaxis
 <inicio>:<final>:<incremento>

Subarreglos

- Los subarreglos se identifican mediante el uso de la sintaxis
 <inicio>:<final>:<incremento>
- Si <inicio> se omite (:), toma el primer valor del arreglo.

Subarreglos

- Los subarreglos se identifican mediante el uso de la sintaxis
 <inicio>:<final>:<incremento>
 - Si <inicio> se omite (:), toma el primer valor del arreglo.
 - Si <final> se omite (:), toma el último valor del arreglo.

Subarreglos

- Los subarreglos se identifican mediante el uso de la sintaxis
 <inicio>:<final>:<incremento>
 - Si <inicio> se omite (:), toma el primer valor del arreglo.
 - Si <final> se omite (:), toma el último valor del arreglo.
 - Si <incremento> se omite (:), toma el valor de 1.

Subarreglos

- Los subarreglos se identifican mediante el uso de la sintaxis
 <inicio>:<final>:<incremento>
 - Si <inicio> se omite (:), toma el primer valor del arreglo.
 - Si <final> se omite (:), toma el último valor del arreglo.
 - Si <incremento> se omite (:), toma el valor de 1.

```
1  REAL, DIMENSION(1:3)::vector
2  REAL, DIMENSION(1:3,1:3)::matriz
3  :
4  WRITE(*,*)vector(1:2)      !los dos primeros elementos
5  WRITE(*,*)matriz(1:1,1:3)  !la primera fila
6  WRITE(*,*)matriz(1:3:2,1:3) !la primera y tercera fila
7  WRITE(*,*)matriz(3:1:-2,1:3) !la tercera y primera fila
8  WRITE(*,*)matriz(:,2)      !el primer bloque 2x2
9  :
```

Asignación en arreglos

Expresiones de asignación y operaciones aritméticas

Asignación en arreglos

Expresiones de asignación y operaciones aritméticas

- Las expresiones entre arreglos deben crear arreglos de la misma forma.

Expresiones de asignación y operaciones aritméticas

- Las expresiones entre arreglos deben crear arreglos de la misma forma.
- Los elementos de los arreglos están relacionados por relaciones de orden, preestablecida sobre los índices, como se evidencia en una matriz por las columnas y filas.

Expresiones de asignación y operaciones aritméticas

- Las expresiones entre arreglos deben crear arreglos de la misma forma.
- Los elementos de los arreglos están relacionados por relaciones de orden, preestablecida sobre los índices, como se evidencia en una matriz por las columnas y filas.
- Las operaciones aritméticas y funciones intrínsecas (cos, exp, etc.) operan por elemento.

Asignación en arreglos

Expresiones de asignación y operaciones aritméticas

- Las expresiones entre arreglos deben crear arreglos de la misma forma.
- Los elementos de los arreglos están relacionados por relaciones de orden, preestablecida sobre los índices, como se evidencia en una matriz por las columnas y filas.
- Las operaciones aritméticas y funciones intrínsecas (cos, exp, etc.) operan por elemento.

```
1  REAL, DIMENSION(1:3)::vect           !vector
2  REAL, DIMENSION(1:3,1:3)::matr1      !matriz 1
3  REAL, DIMENSION(0:2,-1:1)::matr2     !matriz 2
4  :
5  matr1(:,1)=matr2(0,:)+vect           !El resultado va a la columna1 la de matr1, de
6                                         !la suma de la fila0 de la matr2 con el vector.
7  matr1=matr1*matr2                    !Los productos son calculados individualmente
8  matr1=cos(matr2)
9  matr1=exp(matr2)
10 matr2(:,0)=sqrt(vector)
```


Vectores sub-índices

Vectores sub-índices

- Es posible también identificar los elementos de un arreglo a través de un vector índice, de elementos tipos INTEGER.

Vectores sub-índices

- Es posible también identificar los elementos de un arreglo a través de un vector índice, de elementos tipos INTEGER.

```
1  INTEGER, DIMENSION(1:3)::indice::(/2,4,6/)
2  REAL, DIMENSION(1:10,1:10)::matriz
3  :
4  PRINT*,matriz(5,indice)      !escribe los elementos (5,2), (5,4) y (5,6)
5  PRINT*,matriz(indice,indice) !los elementos en el orden (2,2), (4,2), (6,2)
6                               !(2,4), (4,4), (6,4), (2,6), (4,6) y (6,6)
7  matriz(indice,5)=matriz(1:5:2,6) !Se asigna a matriz (2,5) el
8                                   !valor de matriz(1,6),
9                                   !a matriz(4,5) el valor matriz(3,6)
10                                  !y a matriz(6,5) el valor matriz(5,6)
```

Instrucciones y operaciones exclusivas de arreglos

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción `WHERE`.

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción WHERE.
- La sintaxis es la siguiente

WHERE (<arreglo control>) <arreglo>=<expresión>

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción WHERE.
- La sintaxis es la siguiente

WHERE (<arreglo control>) <arreglo>=<expresión>

donde los elementos de <arreglo control> y <arreglo> son tipo LOGICAL.

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción WHERE.
- La sintaxis es la siguiente

WHERE (<arreglo control>) <arreglo>=<expresión>

donde los elementos de <arreglo control> y <arreglo> son tipo LOGICAL.

Por ejemplo: Sea A un arreglo de 2×2

$$A = \begin{pmatrix} 100. & 10. \\ 1. & 0. \end{pmatrix}$$

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción WHERE.
- La sintaxis es la siguiente

WHERE (<arreglo control>) <arreglo>=<expresión>

donde los elementos de <arreglo control> y <arreglo> son tipo LOGICAL.

Por ejemplo: Sea A un arreglo de 2×2

$$A = \begin{pmatrix} 100. & 10. \\ 1. & 0. \end{pmatrix}$$

```
1  :  
2  REAL, DIMENSION(2,2)::A,B  
3  :  
4  WHERE(A>0) B=log10(A)  
5  :
```

Instrucciones de control

- Fortran permite asignar de valores en elementos específicos de un arreglo, bajo una determinada condición, con la instrucción WHERE.
- La sintaxis es la siguiente

WHERE (<arreglo control>) <arreglo>=<expresión>

donde los elementos de <arreglo control> y <arreglo> son tipo LOGICAL.

Por ejemplo: Sea A un arreglo de 2×2

$$A = \begin{pmatrix} 100. & 10. \\ 1. & 0. \end{pmatrix}$$

```
1  :  
2  REAL, DIMENSION(2,2)::A,B  
3  :  
4  WHERE(A>0) B=log10(A)  
5  :
```

dará como resultado

$$A = \begin{pmatrix} 2. & 1. \\ 0. & 0. \end{pmatrix}$$

Instrucciones de control

- Es posible permitir asignaciones a valores de tipo LOGICAL, del arreglo de control, utilizando la instrucción ELSE WHERE.

Instrucciones de control

- Es posible permitir asignaciones a valores de tipo LOGICAL, del arreglo de control, utilizando la instrucción ELSE WHERE.
- La sintaxis es la siguiente:

```
1  WHERE (<arreglo control>)  
2      <bloque de instrucciones>  
3  ELSE WHERE  
4      <bloque de instrucciones>  
5  END WHERE
```

Instrucciones de control

- Es posible permitir asignaciones a valores de tipo LOGICAL, del arreglo de control, utilizando la instrucción ELSE WHERE.
- La sintaxis es la siguiente:

```
1  WHERE (<arreglo control>)  
2      <bloque de instrucciones>  
3  ELSE WHERE  
4      <bloque de instrucciones>  
5  END WHERE
```

Del ejemplo anterior, tenemos:

```
1  :  
2  REAL, DIMENSION(2,2)::A,B  
3  :  
4  B=A  
5  :  
6  WHERE(A>0)  
7      B=log10(A)  
8  ELSE WHERE  
9      B=-100  
10 ELSE WHERE  
11 :
```

Instrucciones de control

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

Instrucciones de control

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

Instrucciones de control

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

Algunas funciones de asignación de control

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

Algunas funciones de asignación de control

- `all`

```
1  all(Acontrol)      !valor .true. si todos los elementos de Acontrol  
2                      !tienen valor .true.. Sino el valor es .false.
```

Instrucciones de control

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

Algunas funciones de asignación de control

- **all**

```
1 all(Acontrol)      !valor .true. si todos los elementos de Acontrol
2                   !tienen valor .true.. Sino el valor es .false.
```

- **any**

```
1 any(Acontrol)      !valor .true. si al menos un elemento de Acontrol
2                   !tiene valor .true.. Sino el valor es .false.
```

que dará como resultado

$$B = \begin{pmatrix} 2. & 1. \\ 0. & -100. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

Algunas funciones de asignación de control

- **all**

```
1 all(Acontrol)      !valor .true. si todos los elementos de Acontrol
2                   !tienen valor .true.. Sino el valor es .false.
```

- **any**

```
1 any(Acontrol)      !valor .true. si al menos un elemento de Acontrol
2                   !tiene valor .true.. Sino el valor es .false.
```

- **count**

```
1 count(Acontrol)    !valor INTEGER indicando el número de elementos
2                   !de Acontrol, cuyos valores son .true.
```

- Agregando la opción *dim* a las funciones de control, es posible reducir los arreglos a unos de forma unidimensional (vectores).

Instrucciones de control

- Agregando la opción *dim* a las funciones de control, es posible reducir los arreglos a unos de forma unidimensional (vectores).

Por ejemplo:

$$A_{control} = \begin{pmatrix} .true. & .false. \\ .false. & .false. \\ .true. & .true. \end{pmatrix}$$

Instrucciones de control

- Agregando la opción *dim* a las funciones de control, es posible reducir los arreglos a unos de forma unidimensional (vectores).

Por ejemplo:

$$Acontrol = \begin{pmatrix} .true. & .false. \\ .false. & .false. \\ .true. & .true. \end{pmatrix}$$

- Considerando un arreglo de control *Acontrol* de tipo LOGICAL, tenemos:

```
1  all(Acontrol,dim=1)      !da (/false.,.false.,.true./)
2  all(Acontrol,dim=2)      !da (/false.,.false./)
3  any(Acontrol,dim=1)      !da (/true.,.false.,.true./)
4  any(Acontrol,dim=2)      !da (/true.,.true./)
5  count(Acontrol,dim=1)    !da (/1,0,2/)
6  count(Acontrol,dim=2)    !da (/2,1/)
```

Funciones intrínsecas

- Fortran, en sus versiones 90 y posteriores, cuenta con funciones especiales para arreglos.

Funciones intrínsecas

- Fortran, en sus versiones 90 y posteriores, cuenta con funciones especiales para arreglos.

Tomando como ejemplo el siguiente arreglo:

$$A = \begin{pmatrix} 5. & 3. & 1. \\ 8. & 12. & 10. \\ 9. & 11. & 7. \\ 4. & 6. & 2. \end{pmatrix}$$

cuya declaración es:

```
1  REAL, DIMENSION(0:3,2:4)::A
```


Funciones intrínsecas

- Fortran, en sus versiones 90 y posteriores, cuenta con funciones especiales para arreglos.

Tomando como ejemplo el siguiente arreglo:

$$A = \begin{pmatrix} 5. & 3. & 1. \\ 8. & 12. & 10. \\ 9. & 11. & 7. \\ 4. & 6. & 2. \end{pmatrix}$$

cuya declaración es:

```
1 REAL, DIMENSION(0:3,2:4)::A
```

lbound

Funciones intrínsecas

- Fortran, en sus versiones 90 y posteriores, cuenta con funciones especiales para arreglos.

Tomando como ejemplo el siguiente arreglo:

$$A = \begin{pmatrix} 5. & 3. & 1. \\ 8. & 12. & 10. \\ 9. & 11. & 7. \\ 4. & 6. & 2. \end{pmatrix}$$

cuya declaración es:

```
1 REAL, DIMENSION(0:3,2:4)::A
```

lbound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los mínimos que pueden tomar los índices del arreglo.

```
1 lbound(A) !asigna los valores (/0,2/)
```

Funciones intrínsecas

- Fortran, en sus versiones 90 y posteriores, cuenta con funciones especiales para arreglos.

Tomando como ejemplo el siguiente arreglo:

$$A = \begin{pmatrix} 5. & 3. & 1. \\ 8. & 12. & 10. \\ 9. & 11. & 7. \\ 4. & 6. & 2. \end{pmatrix}$$

cuya declaración es:

```
1 REAL, DIMENSION(0:3,2:4)::A
```

lbound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los mínimos que pueden tomar los índices del arreglo.

```
1 lbound(A) !asigna los valores (/0,2/)
```

Agregando la opción *dim* se obtiene el mínimo valor de cada índice.

```
1 lbound(A,dim=1) !se obtiene: 0
```

```
2 lbound(A,dim=2) !se obtiene: 2
```

`ubound`

ubound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los máximos que pueden tomar los índices del arreglo.

1 ubound(A) *!asigna los valores (/3,4/)*

ubound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los máximos que pueden tomar los índices del arreglo.

```
1 ubound(A) !asigna los valores (/3,4/)
```

Agregando la opción *dim* se obtiene el máximo valor de cada índice.

```
1 ubound(A,dim=1) !se obtiene: 3
```

```
2 ubound(A,dim=2) !se obtiene: 4
```

ubound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los máximos que pueden tomar los índices del arreglo.

```
1 ubound(A) !asigna los valores (/3,4/)
```

Agregando la opción *dim* se obtiene el máximo valor de cada índice.

```
1 ubound(A,dim=1) !se obtiene: 3
```

```
2 ubound(A,dim=2) !se obtiene: 4
```

size

ubound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los máximos que pueden tomar los índices del arreglo.

```
1 ubound(A) !asigna los valores (/3,4/)
```

Agregando la opción *dim* se obtiene el máximo valor de cada índice.

```
1 ubound(A,dim=1) !se obtiene: 3
```

```
2 ubound(A,dim=2) !se obtiene: 4
```

size

- Proporciona el número de elementos de un arreglo valor tipo INTEGER.

```
1 size(A) !se obtiene: 12
```


ubound

- Da como resultado un vector de tipo INTEGER, cuyos valores son los máximos que pueden tomar los índices del arreglo.

```
1 ubound(A) !asigna los valores (/3,4/)
```

Agregando la opción *dim* se obtiene el máximo valor de cada índice.

```
1 ubound(A,dim=1) !se obtiene: 3
```

```
2 ubound(A,dim=2) !se obtiene: 4
```

size

- Proporciona el número de elementos de un arreglo valor tipo INTEGER.

```
1 size(A) !se obtiene: 12
```

Agregando la opción *dim* se obtiene la longitud del rango del índice indicado.

```
1 size(A,dim=1) !se obtiene: 4
```

```
2 size(A,dim=2) !se obtiene: 3
```

shape

shape

- Proporciona la forma del arreglo en un vector de tipo INTEGER cuyo tamaño es el número de índices y donde cada elemento del vector representa la longitud del rango del índice.

```
1  shape(A) !se obtiene: (/4,3/)
```

shape

- Proporciona la forma del arreglo en un vector de tipo INTEGER cuyo tamaño es el número de índices y donde cada elemento del vector representa la longitud del rango del índice.

```
1 shape(A) !se obtiene: (/4,3/)
```

minval, maxval

shape

- Proporciona la forma del arreglo en un vector de tipo INTEGER cuyo tamaño es el número de índices y donde cada elemento del vector representa la longitud del rango del índice.

```
1 shape(A) !se obtiene: (/4,3/)
```

minval, maxval

- Proporciona el mínimo o máximo valor, respectivamente, de un arreglo de tipo numérico.

```
1 minval(A) !se obtiene: 12
```

```
2 maxval(A) !se obtiene: 1
```

shape

- Proporciona la forma del arreglo en un vector de tipo INTEGER cuyo tamaño es el número de índices y donde cada elemento del vector representa la longitud del rango del índice.

```
1 shape(A) !se obtiene: (/4,3/)
```

minval, maxval

- Proporciona el mínimo o máximo valor, respectivamente, de un arreglo de tipo numérico.

```
1 minval(A) !se obtiene: 12
```

```
2 manval(A) !se obtiene: 1
```

Agregando la opción *dim* se obtiene un vector cuyos valores son los mínimos o máximos fijados por el valor del índice.

```
1 minval(A, dim=1) !se obtiene: (/1.,8.,7.,2./)
```

```
2 manval(A, dim=1) !se obtiene: (/5.,12.,11.,6./)
```

```
3 minval(A, dim=2) !se obtiene: (/4.,3.,1./)
```

```
4 manval(A, dim=2) !se obtiene: (/9.,12.,10./)
```

minloc, maxloc

minloc, maxloc

- Proporcionan los índices del valor mínimo y máximo, respectivamente, de los elementos del arreglo.

```
1 minloc(A) !se obtiene: (/1,3/)
```

```
2 maxloc(A) !se obtiene: (/2,2/)
```


Funciones intrínsecas

minloc, maxloc

- Proporcionan los índices del valor mínimo y máximo, respectivamente, de los elementos del arreglo.

1 minloc(A) *!se obtiene: (/1,3/)*

2 maxloc(A) *!se obtiene: (/2,2/)*

sum, product

minloc, maxloc

- Proporcionan los índices del valor mínimo y máximo, respectivamente, de los elementos del arreglo.

```
1 minloc(A) !se obtiene: (/1,3/)
```

```
2 maxloc(A) !se obtiene: (/2,2/)
```

sum, product

- La función sum nos da la suma de todos los elementos del arreglo, mientras la opción product, el producto de los mismos, siempre y cuando el arreglo sea de tipo numérico.

```
1 sum(A) !se obtiene: 4.7900160E+08
```

```
2 product(A) !se obtiene: 78.00000
```

Funciones intrínsecas

minloc, maxloc

- Proporcionan los índices del valor mínimo y máximo, respectivamente, de los elementos del arreglo.

```
1 minloc(A) !se obtiene: (/1,3/)
2 maxloc(A) !se obtiene: (/2,2/)
```

sum, product

- La función sum nos da la suma de todos los elementos del arreglo, mientras la opción product, el producto de los mismos, siempre y cuando el arreglo sea de tipo numérico.

```
1 sum(A)      !se obtiene: 4.7900160E+08
2 product(A)  !se obtiene: 78.00000
```

Agregando la opción *dim*, se restringen las operaciones de suma y multiplicación a los índices.

```
1 sum(A, dim=1)      !se obtiene: (/26.0000,32.0000,20.0000)
2 sum(A, dim=2)      !se obtiene: (/9.000000,30.0000,27.00000,12.00000)
3 product(A, dim=1)   !se obtiene: (/1440.000,2376.000,140.0000)
4 product(A, dim=2)   !se obtiene: (/15.00000,960.00000,693.0000,48.00000)
```

Operadores matriciales

Operadores matriciales

- Fortran, en sus versiones 90 y posteriores, permite realizar operaciones con matrices y vectores (arreglos bidimensionales y unidimensionales, respectivamente) como la adición, sustracción y multiplicación por escalar.

Operadores matriciales

- Fortran, en sus versiones 90 y posteriores, permite realizar operaciones con matrices y vectores (arreglos bidimensionales y unidimensionales, respectivamente) como la adición, sustracción y multiplicación por escalar.

matmul

Operadores matriciales

- Fortran, en sus versiones 90 y posteriores, permite realizar operaciones con matrices y vectores (arreglos bidimensionales y unidimensionales, respectivamente) como la adición, sustracción y multiplicación por escalar.

matmul

- La función *matmul* permite efectuar la multiplicación entre estos arreglos.

Operadores matriciales

- Fortran, en sus versiones 90 y posteriores, permite realizar operaciones con matrices y vectores (arreglos bidimensionales y unidimensionales, respectivamente) como la adición, sustracción y multiplicación por escalar.

matmul

- La función *matmul* permite efectuar la multiplicación entre estos arreglos.
- La sintaxis es la siguiente:

```
1  matmul(<matriz>,<matriz>)    !se obtiene una matriz
2  matmul(<matriz>,<vector>)    !se obtiene un vector
```


Operadores matriciales

- Fortran, en sus versiones 90 y posteriores, permite realizar operaciones con matrices y vectores (arreglos bidimensionales y unidimensionales, respectivamente) como la adición, sustracción y multiplicación por escalar.

matmul

- La función *matmul* permite efectuar la multiplicación entre estos arreglos.
- La sintaxis es la siguiente:

```
1 matmul(<matriz>,<matriz>)    !se obtiene una matriz
2 matmul(<matriz>,<vector>)    !se obtiene un vector
```

Por ejemplo:

Sean los arreglos

$$A = \begin{pmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \end{pmatrix}, B = \begin{pmatrix} 1. & 2. \\ 3. & 4. \\ 5. & 6. \end{pmatrix}, u = \begin{pmatrix} 1. \\ 2. \\ 3. \end{pmatrix}, v = \begin{pmatrix} 1. \\ 2. \end{pmatrix}.$$

Operadores matriciales

El siguiente ejemplo muestra el uso de *matmul*:

```
1 PROGRAM matmul
2 REAL, DIMENSION(3)::u=(/1.,2.,3./),x
3 REAL, DIMENSION(2)::v=(/1.,2./),y
4 REAL, DIMENSION(3,2)::A=reshape((/ (1.*i,i=1,6)/), (/3,2/))
5 REAL, DIMENSION(2,3)::B=reshape((/ (1.*i,i=1,6)/), (/2,3/))
6 REAL, DIMENSION(3,3)::C
7 REAL, DIMENSION(2,2)::D
8 C=matmul(A,B)
9 D=matmul(B,A)
10 y=matmul(B,u)
11 x=matmul(A,v)
12 :
```

Operadores matriciales

El siguiente ejemplo muestra el uso de *matmul*:

```
1 PROGRAM matmul
2 REAL, DIMENSION(3)::u=(/1.,2.,3./),x
3 REAL, DIMENSION(2)::v=(/1.,2./),y
4 REAL, DIMENSION(3,2)::A=reshape((/(1.*i,i=1,6)/),(/3,2/))
5 REAL, DIMENSION(2,3)::B=reshape((/(1.*i,i=1,6)/),(/2,3/))
6 REAL, DIMENSION(3,3)::C
7 REAL, DIMENSION(2,2)::D
8 C=matmul(A,B)
9 D=matmul(B,A)
10 y=matmul(B,u)
11 x=matmul(A,v)
12 :
```

dando como resultado

$$C = \begin{pmatrix} 9,000000 & 19,000000 & 29,000000 \\ 12,000000 & 26,000000 & 40,000000 \\ 15,000000 & 33,000000 & 51,000000 \end{pmatrix}, D = \begin{pmatrix} 22,000000 & 49,000000 \\ 28,000000 & 64,000000 \end{pmatrix}$$

$$x = \begin{pmatrix} 9,000000 \\ 12,000000 \\ 15,000000 \end{pmatrix}, y = \begin{pmatrix} 22,000000 \\ 28,000000 \end{pmatrix}$$

`dot_product`

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*:

Operadores matriciales

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*:
- La sintaxis es la siguiente:

```
1 dot_product(<vector>,<vector>)
```

Operadores matriciales

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*..
- La sintaxis es la siguiente:

```
1 dot_product(<vector>,<vector>)
```

Por ejemplo:

```
1 :  
2 REAL::valor  
3 REAL, DIMENSION(8)::vector=(/(1.*i,i=1,8)/)  
4 :  
5 valor=dot_product(vector(1:7:2),vector(2:8:2)) !se obtiene: 100  
6 :
```

Operadores matriciales

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*..
- La sintaxis es la siguiente:

```
1 dot_product(<vector>,<vector>)
```

Por ejemplo:

```
1 :  
2 REAL::valor  
3 REAL, DIMENSION(8)::vector=(/(1.*i,i=1,8)/)  
4 :  
5 valor=dot_product(vector(1:7:2),vector(2:8:2)) !se obtiene: 100  
6 :
```

transpose

Operadores matriciales

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*:
- La sintaxis es la siguiente:

```
1 dot_product(<vector>,<vector>)
```

Por ejemplo:

```
1 :  
2 REAL::valor  
3 REAL, DIMENSION(8)::vector=(/(1.*i,i=1,8)/)  
4 :  
5 valor=dot_product(vector(1:7:2),vector(2:8:2)) !se obtiene: 100  
6 :
```

transpose

- La transposición de una matriz, el intercambio de posición simétrica entre filas y columnas, es posible mediante la función *transpose*.

Operadores matriciales

dot_product

- El producto escalar de dos vectores del mismo tamaño se realiza empleando la función *dot_product*:
- La sintaxis es la siguiente:

```
1 dot_product(<vector>,<vector>)
```

Por ejemplo:

```
1 :  
2 REAL::valor  
3 REAL, DIMENSION(8)::vector=(/(1.*i,i=1,8)/)  
4 :  
5 valor=dot_product(vector(1:7:2),vector(2:8:2)) !se obtiene: 100  
6 :
```

transpose

- La transposición de una matriz, el intercambio de posición simétrica entre filas y columnas, es posible mediante la función *transpose*.
- La sintaxis es la siguiente:

```
1 transpose(<matriz>)
```

Lectura y escritura de archivos

Archivos

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.
- Entre ellos destacan:

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.
- Entre ellos destacan:
 - Archivos de texto. Son la fuente de los programas, listas de compilación, etc. Pueden ser impresos y directamente leídos.

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.
- Entre ellos destacan:
 - Archivos de texto. Son la fuente de los programas, listas de compilación, etc. Pueden ser impresos y directamente leídos.
 - Archivos de datos. Existen en dos formas: los que pueden ser usados en un editor (archivo de texto) y aquellos que pueden ser usados un paquete o un programa (codificado).

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.
- Entre ellos destacan:
 - Archivos de texto. Son la fuente de los programas, listas de compilación, etc. Pueden ser impresos y directamente leídos.
 - Archivos de datos. Existen en dos formas: los que pueden ser usados en un editor (archivo de texto) y aquellos que pueden ser usados un paquete o un programa (codificado).
 - Archivos binarios. Secuencias de bits. No pueden ser impresos o examinados directamente, por ello necesitan de programas para su propósito.

Archivos

- Un archivo es un objeto que guarda o contiene datos, información o instrucciones en un espacio de memoria.
- Entre ellos destacan:
 - Archivos de texto. Son la fuente de los programas, listas de compilación, etc. Pueden ser impresos y directamente leídos.
 - Archivos de datos. Existen en dos formas: los que pueden ser usados en un editor (archivo de texto) y aquellos que pueden ser usados un paquete o un programa (codificado).
 - Archivos binarios. Secuencias de bits. No pueden ser impresos o examinados directamente, por ello necesitan de programas para su propósito.

Archivo de datos en Fortran

- Fortran permite asociar un número como unidad lógica con cualquier archivo para su creación, conexión y acceso.

Declaración OPEN

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración CLOSE

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración CLOSE

- *CLOSE* permite borrar o desvincular la unidad numérica del archivo.

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración CLOSE

- *CLOSE* permite borrar o desvincular la unidad numérica del archivo.
- La sintaxis para la declaración es la siguiente sintaxis:
CLOSE (unidad de control, [opciones])

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración CLOSE

- *CLOSE* permite borrar o desvincular la unidad numérica del archivo.
- La sintaxis para la declaración es la siguiente sintaxis:
CLOSE (unidad de control, [opciones])

Declaración READ

Declaración OPEN

- *OPEN* permite crear o conectar una unidad numérica como especificador con un archivo.
- La sintaxis para la declaración es la siguiente:
OPEN (unidad de control, [lista de salida])

Declaración CLOSE

- *CLOSE* permite borrar o desvincular la unidad numérica del archivo.
- La sintaxis para la declaración es la siguiente sintaxis:
CLOSE (unidad de control, [opciones])

Declaración READ

- *READ* permite leer datos desde una referencia. Puede contener especificaciones del proceso de retorno de datos.

Declaración WRITE

Declaración `WRITE`

- `WRITE` permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.

Declaración *WRITE*

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.

Declaración *WRITE*

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.
- Puede tomar cuatro valores:

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.
- Puede tomar cuatro valores:
 - STATUS = 'OLD'

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.
- Puede tomar cuatro valores:
 - STATUS = 'OLD'
 - STATUS = 'NEW'

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.
- Puede tomar cuatro valores:
 - STATUS = 'OLD'
 - STATUS = 'NEW'
 - STATUS = 'SCRATCH'

Declaración **WRITE**

- *WRITE* permite mandar datos o escribirlos en unidad numerada que puede ser la pantalla, un archivo de disco u otro dispositivo externo.
- La opción *PRINT* es una forma limitada de *WRITE*.
- Tanto *READ* como *WRITE* tienen la siguiente sintaxis:
READ/WRITE (unidad de control, [lista de entrada/salida])

Declaración **STATUS**

- *STATUS* permite conocer el estado del archivo.
- Puede tomar cuatro valores:
 - STATUS = 'OLD'
 - STATUS = 'NEW'
 - STATUS = 'SCRATCH'
 - STATUS = 'UNKNOWN'

Lectura y escritura de archivos

```
1  PROGRAM File01
2  IMPLICIT NONE
3  REAL :: X
4  CHARACTER (7) :: WHICH
5      OPEN(UNIT=5,FILE='INPUT')
6      DO
7          WRITE(UNIT=6,FMT='('' DATA SET NAME, OR END'')')
8          READ(UNIT=5,FMT='(A)') WHICH
9          IF(WHICH == 'END') EXIT
10         OPEN(UNIT=1,FILE=WHICH)
11         READ(UNIT=1,FMT=100) X
12         !...
13     CLOSE(UNIT=1)
14     END DO
15 END PROGRAM File01
```

Opciones en declaraciones de entrada/salida

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.
 - Un entero positivo, indicando una condición de error.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.
 - Un entero positivo, indicando una condición de error.
 - Cero, indicando que no existe un error, el término de un archivo o condición de registro durante la ejecución de *READ*.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.
 - Un entero positivo, indicando una condición de error.
 - Cero, indicando que no existe un error, el término de un archivo o condición de registro durante la ejecución de *READ*.
- **FILE:** Especifica el nombre del archivo.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.
 - Un entero positivo, indicando una condición de error.
 - Cero, indicando que no existe un error, el término de un archivo o condición de registro durante la ejecución de *READ*.
- **FILE:** Especifica el nombre del archivo.
- **STATUS:** Especifica el estado del archivo.

Opciones en declaraciones de entrada/salida

- **UNIT:** El número del archivo que será abierto.
- **IOSTAT:** Especifica el estado I/O y designa una variable numérica para guardar un valor de acuerdo al estado del archivo.
 - Un entero positivo, indicando una condición de error.
 - Cero, indicando que no existe un error, el término de un archivo o condición de registro durante la ejecución de *READ*.
- **FILE:** Especifica el nombre del archivo.
- **STATUS:** Especifica el estado del archivo.
- **ACCESS:** Especifica la forma en la que se usará el archivo: secuencial (*SEQUENTIAL*) por defecto o aleatoria (*DIRECT*).

- FORM:

- **FORM:**
- **RECL:** Variable entera o constante que especifica

- **FORM:**
- **RECL:** Variable entera o constante que especifica
- **BLANK:** Especifica el reconocimiento de un espacio en blanco tomando los valores...

- **FORM:**
- **RECL:** Variable entera o constante que especifica
- **BLANK:** Especifica el reconocimiento de un espacio en blanco tomando los valores...
 - NULL, si son ignorados en la lectura.

- **FORM:**
- **RECL:** Variable entera o constante que especifica
- **BLANK:** Especifica el reconocimiento de un espacio en blanco tomando los valores...
 - NULL, si son ignorados en la lectura.
 - ZERO, si los espacios son tratados como cero.

Lectura y escritura de archivos

A partir del siguiente programa:

```
1  PROGRAM matriz3x2
2  INTEGER, DIMENSION(3,2)::A
3  WRITE(*,*) "Introduzca los valores de la matriz A de talla 3x2"
4  READ(*,*)A
5  WRITE(*,*) "La matriz que usted ha introducido es:"
6  WRITE(*,*) "Primera Fila",A(1,:)
7  WRITE(*,*) "Segunda Fila",A(2,:)
8  WRITE(*,*) "Tercera Fila",A(3,:)
9  END PROGRAM matriz3x2
```

Lectura y escritura de archivos

A partir del siguiente programa:

```
1 PROGRAM matriz3x2
2 INTEGER, DIMENSION(3,2)::A
3 WRITE(*,*) "Introduzca los valores de la matriz A de talla 3x2"
4 READ(*,*)A
5 WRITE(*,*) "La matriz que usted ha introducido es:"
6 WRITE(*,*) "Primera Fila",A(1,:)
7 WRITE(*,*) "Segunda Fila",A(2,:)
8 WRITE(*,*) "Tercera Fila",A(3,:)
9 END PROGRAM matriz3x2
```

podemos obtener, meditante la ejecución:

```
1 [softbutterfly\@SB-PC] ~\$
2 Introduzca los valores de la matriz A de talla 3x2
3 1,2,3,4 5 6
4 La matriz que usted ha introducido es:
5 Primera Fila          1          4
6 Segunda Fila          2          5
7 Tercera Fila           3          6
8 [softbutterfly\@SB-PC]
```