**¿QUE ES UN ARRAY?**

Un array(arreglo) es un grupo de variables/constantes del mismo *tipo* asociados a una sola etiqueta (nombre).

Cada variable del arreglo se llama elemento

Memoria de la máquina

| A |
| --- |
| B |
| M(1) |
| M(2) |
| M(n-1) |
| M(n) |

**Array M:** Para distinguir entre variables se les asocia con un índice

A cada elemento del arreglo se le asocia un número entero que lo distingue

* El uso de arreglos evita la declaración excesiva de variables

**DECLARACIÓN DE ARRAYS**

* Arreglos de tamaño fijo

1. (TIPO), DIMENSION(40) **cantidad de elementos para partir el arreglo** :: A **(nombre del arreglo)**

Forzosamente van a haber varios arreglos (A y B) con 40 elementos

1. Tipo :: A(20), B(60)

Los arreglos pueden tener diferentes cantidades de elementos (distinto tamaño)

**¿CÓMO ACCEDER A LOS ELEMENTOS DEL ARREGLO?**

Como cada arreglo posee un elemento que lo distingue, para acceder a ellos basta con especificar dicho número.

* a) EJEMPLO

PROGRAM Ejemplo\_a

IMPLICIT NONE

REAL, DIMENSION(3) :: T

T(1) = 6

T(2) = 8

T(3) =15

indicar elemento por elemento de forma manual; indica el valor de cada elemento del arreglo

END PROGRAM Ejemplo\_a

* b) EJEMPLO

También podemos utilizar [ ]

PROGRAM Ejemplo\_b

IMPLICIT NONE

REAL, DIMENSION(3) :: T

REAL, DIMENSION(5) :: A =[1, 2, 3, 4, 5]

END PROGRAM Ejemplo\_b

! También se pueden indicar en la misma línea en la que se declara el arreglo

T = [6, 8, 15]

! Almacenamos todos los elementos del arreglo dentro de los corchetes, sin especificarlos de forma manual (después de la declaración)

* c) EJEMPLO

Uso de [ ] y DO Loops implícitos

Un loop implícito funciona igual que un loop iterativo

DO i = 1, 3, 1

. . .

END DO

PROGRAM Ejemplo\_c

IMPLICIT NONE

REAL :: x(100)

x = [1, 2, 3, ....., 99, 100] (correcto pero impráctico; escribir todos los elementos)

x = [(i, i=1,100)] (uso de loop implícito)

! Hacen lo mismo, pero uno es mejor que el otro

**SE PUEDEN UTILIZAR LOOPS DENTRO DE OTROS (anidados)**

EJEMPLO : x = [((0, i=1,4), 5\*j, j=1,5)]

**Loop interno**

**Loop externo**

EJEMPLO

Uso de loops iterativos

PROGRAM Ejemplo\_d

IMPLICIT NONE

REAL :: X(25)

INTEGER :: i, j

DO i = 1, 3

DO j = 1, 3

x(i,j) = T(i)\*V(j)

END DO

x(5\*i) = 5\*i

END DO

END PROGRAM

**USO DE LOOPS IMPLÍCITOS**

INTEGER :: i

REAL, DIMENSION(25) :: w

w = [((0, i=1,4), 5\*j, j=1,5)]

loops implícitos anidados

manera de construir el siguiente arreglo

w = [0,0,0,0,5,0,0,0,0,10,0,0,0,0,15, ….. etc]

DO j=1,5

DO i=1,4

w(i+5\*(j-1)=0

END DO

w=(5\*j)=5\*j

END DO

Lo que representa el loop implícito anidado

PROGRAM prueba

! uso de loops implícitos para mostrar los siguientes elementos

! w = [0,0,0,0,5,0,0,0,0,10,0,0,0,0,15, ….. etc]

IMPLICIT NONE

REAL, DIMENSION(25) :: w, z

INTEGER :: i, j

w = [((0, i=1,4), 5\*j, j=1,5)]

! loops implícitos anidados

DO j=1,5

DO i=1,4

z(i+5\*(j-1)) = 0

! permite guardar los 0 en los lugares entre los múltiplos de 5

END DO

z(5\*j)=5\*j

END DO

! loops explícitos

DO i=1,25

PRINT\*, w(i), z(i)

END DO

END PROGRAM prueba

Al declarar arreglos, por default Fortran le asigna un número entero a cada elemento. La numeración estándar inicia en 1.

El programador también ṕuede decidir desde qué número empieza la numeración y en donde termina.

EJEMPLO:

REAL, DIMENSION(20) :: x

! estándar, la numeración va del 1 al 20

Para modificar la manera en que Fortran asocia los números

REAL, DIMENSION(0:19) :: Z

! numeración de elementos establecidos por el programador (va de 0 a 19)

REAL, DIMENSION(-30:100) :: z

! numeración con 131 elementos

**CARACTERÍSTICAS DE LOS ARREGLOS**

En estas clases de han utilizado arreglos de rango 1, mejor conocidos como vectores.

Se pueden construir arreglos de rango superior

1. **Rango:** está determinado por la cantidad de índices necesarios para acceder a todos los elementos del arreglo.

**Tenemos un máximo de 7 índices disponibles**

| x | 3 | 1.5 | 7.8 | 9 |
| --- | --- | --- | --- | --- |

DO i=1,5

PRINT\*, x(i)

END DO

se imprimirán los elementos del arreglo x

Si

z =

| 1,1 | 1,2 | 1,3 | 1,4 |
| --- | --- | --- | --- |
| 2,1 | 2,2 | 2,3 | 2,4 |
| 3,1 | 3,2 | 3,3 | 3,4 |

Este arreglo es de rango 2 ya que contiene dos índices por elemento.

Declaración de arreglos de rango superior

REAL, DIMENSION(3,4) :: z RANGO 2 (matrices)

REAL, DIMENSION(3,4,5) :: w RANGO 3

1. **Extensión:** la cantidad de elementos por cada índice del arreglo
2. **Tamaño:** es la cantidad total de elementos en el arreglo.

Cuando el arreglo es de rango 1, la extensión y el tamaño coinciden.

1. **Forma:** combina la extensión y el rango de un arreglo. (extensión por cada indica¿?)

PRINT\*, SHAPE(Z) ¡¡matriz!!

3 4

PRINT\*, SIZE(Z) total de elementos

12

PRINT\*, RANGE(Z) numero de indices por elemento

2

**OPERACIONES ENTRE ARREGLOS**

Arreglos deben de ser conformes; aquellos que tienen la misma forma.

Para operar arreglos directamente, sin necesidad de indicar explícitamente elemento a elemento, es necesario que los arreglos sean conformes.

EJEMPLO:

REAL, DIMENSION(3) :: A, B, C

C = A + B

En este caso A,B,C son conformes ya que poseen el mismo rango (1) y la misma extensión (3)

Una cantidad escalar es conforme a cualquier arreglo.

REAL :: C=5

REAL, DIMENSION(3) :: M, N

M = 4\*N M = C- N

M = 4+N M = C/N

Cualquier operación entre escalar y arreglo se puede hacer sin tener que indicarse. (escalar x vector, en este caso)

**USO DE SUBCONJUNTOS DE UN ARREGLO**

En ocasiones es necesario utilizar solo algunos de los elementos de un arreglo inicial.

En Fortran podemos elegir un subconjunto del arreglo original de varias maneras.

1. Uso de un triplete de subíndices
2. Uso de un arreglo de subíndices

EJEMPLO

REAL, DIMENSION(10) :: A = [3,8,9,4,7,1,6,5,4,2]

1. Triplete de subíndices

Queremos imprimir en pantalla algunos de los elementos

PRINT\*, A(inicio : final : paso)

De cuanto a cuanto se seleccionan los elementos

PRINT\*, A(3 : 9 : 2)

se empezaria en el tercer número, y finaliza en el noveno, con un incremento de dos en dos.

EN PANTALLA 9 7 6 4

Cualquiera de estas tres cantidades puede omitirse

A = [3,8,9,4,7,1,6,5,4,2]

PRINT\*, A(3 : 9 : )

El paso es de uno en uno

EN PANTALLA 9 4 7 1 6 5 4

PRINT\*, A( : 9 : 2)

El conjunto inicia en el primer elemento del arreglo

EN PANTALLA 3 9 7 6 4

Si se omite el 2do subíndice, el subconjunto termina en el último elemento del arreglo original

PRINT\*, A(6 : : 2)

EN PANTALLA 1 5 2

PRINT\*, A( : : 5)

EN PANTALLA 7 2

PRINT\*, A( : )

EN PANTALLA **se escriben todos**

1. Arreglo de subíndices

En este caso para seleccionar los elementos se debe de utilizar un arreglo de tipo INTEGER que contenga los índices de los elementos deseados

En esta forma, los elementos elegidos no necesitan seguir un orden determinado.

EJEMPLO

INTEGER, DIMENSION(4) :: W=[7,8,1,6], ! A = [3,8,9,4,7,1,6,5,4,2]

! W = los subíndices de A

PRINT\*, A(W)

EN PANTALLA 6 5 3 1

Se imprimen los elementos del arreglo A con las posiciones de W

**¿CÓMO ASIGNAR VALORES A LOS ELEMENTOS DE UN ARREGLO DE RANGO SUPERIOR?**

Cuando el arreglo es de rango 1

EJEMPLO

Matriz M

| 4 | 8.3 | 6 | 8 |
| --- | --- | --- | --- |
| 3 | 4.5 | 7 | 4 |
| 6 | 3.1 | 9 | 2 |

SI se desea construir un arreglo que representa la matriz

Fortran tiene preferencia por las columnas.

REAL, DIMENSION(3,4) :: M

! Llenado de M

M = RESHAPE([4,3,6,8.3,4.5,3.1,6,7,9,8,4,2],[3,4] )

! Arreglo 1: este arreglo posee los valores a guardar de M, por columnas (valores a guardar)

! Arreglo 2: indica cómo arreglar los valores proporcionados (distribución de cantidades)/(forma de la matriz)

! RESHAPE requiere de dos argumentos(arreglos) que contengan las cantidades que queremos guardar de la matriz

**RESHAPE** = se usa para rellenar arreglos de rango superior

**CONSTRUCTO WHERE (Fortran 2001 +)**

Nos permite realizar operaciones con algunos de los elementos de un arreglo principal.

Forma de WHERE

WHERE(exp. logica 1)

Bloque de instrucciones 1

ELSEWHERE(exp. lógica 2) equivalente al ELSE IF

Bloque de instrucciones 2

ELSEWHERE si no hay exp. lógica entonces las instrucciones son para todo lo demás (equivalente a ELSE)

Bloque de instrucciones 3

END WHERE

EJEMPLO

PROGRAM ejemplo

IMPLICIT NONE

REAL, DIMENSION(7) :: y=[4,6,-1,0,-3,7,5], z

! se calcula el algoritmo del arreglo y, los resultados se almacenan en z

! como y es conforme a z, entonces se podría realizar la operación de forma global

z = log(y)

PRINT\*, z

END PROGRAM

EN PANTALLA

log(4) log(6) NAN[log(-10)] NAN[log(-3))] log(7) log(5)

NAN: números que no se pueden representar

**¿CÓMO CALCULAMOS EL LOG?**

La manera evidente es utilizando un loop

z = y

DO i=1,7

IF (y(i)>0) THEN

Z(i) = log(yi)

END IF

! El programa solo calculara el log cuando el número sea mayor que 0

EN ṔANTALLA

log(4) log(6) -1 0 -3 log(7) log(5)

Usando LOOPS podemos elegir los elementos de manera explícita

**¡PROBLEMAS!**

Si el arreglo es de rango superior, se deben de realizar varios loops anidados y además señalar explícitamente la cantidad de subíndices adecuada

Para disminuir el problema…

WHERE (y < 0)

z = y

* ELSEWHERE (y == 0)

z = y

* Puede omitirse

ELSEWHERE

z = log(y)

END WHERE

! Todos los números que cumplan con las expresiones lógicas se guardaran en z tal y como están.

! Se pueden usar tantos ELSEWHERE como son necesarios

**CONSTRUCTO FOR ALL**

Fortran cuenta con un constructo más que permite operar con subconjuntos de un arreglo

Estructura

FORALL (i = inicio: final) : paso, expresión lógica

! Triplete de subíndices

Bloque de Instrucciones

END FORALL

El equivalente al WHERE del ejemplo anterior usando FORALL es:

z = y

FORALL (i = 1:7:1, y(i)>0)

! Para todos los elementos de y mayores a 0 se calcularán el logaritmo y se guardará el resultado en z

! La desventaja de FORALL sobre WHERE:

* Se debe de decir explícitamente el elemento que nos interesa

z(i) = log(y)

END FORALL

El DO Loop no le pide nada al FORALL. (al menos no en el curso de Fortran xd)

***VENTAJAS:*** [programación en paralelo](http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html)

programar y ejecutar en todos los procesadores disponibles

**ARREGLOS DE MEMORIA DINÁMICA**

**Arreglos estáticos:** indicamos su forma en el momento en que se declaran.

REAL, DIMENSION (2,3) :: x

! Desde aquí la memoria se parte en la cantidad de elementos señalados

! Estas cantidades se mantienen fijas en el resto del código

Cuando la cantidad de elementos que se usarán es conocida, los arreglos estáticos son más que suficientes xd

En ocasiones el programador desea realizar un código donde no se conoce a priori la cantidad de elementos o la forma del arreglo.

EJEMPLO

Si se desean ordenar n numeros de menor a mayor, ¿cómo declarar un arreglo de tamaño suficiente?

Tenemos en Fortran arreglos de **memoria dinámica**

En estos arreglos se reserva memoria para su posterior división. Guarda un pedazo de memoria y cuando obtenemos el valor de n, se parte el pedazo de memoria en n pedazos.

**DECLARACIÓN DE ARREGLOS DINÁMICOS**

Tipo, ALLOCATABLE :: x( : ), w( : , : )

! Reservando memoria para un arreglo x para cuando se le diga en cuantos pedazos se va a dividir.

! Hay dos índices

Para dividir la memoria en n elementos:

ALLOCATE(x(1:n), w(1:n , 1:n), STAT = error, ERRMSG = mensaje\_error)

! cuando n ya fue declarada

* STAT funciona de manera similar a IOSTAT del open
* En la variable error se puede almacenar un número entero

Si error=0 Todo salió bien, exito en la partición

Si error/=0 Algo salio mal

* ERRMSG produce un mensaje que indica el posible error

**POSIBLES ERRORES EN EL ALLOCATE**

1. *Memoria insuficiente*

Cuando hay muchos arreglos en un programa, puede que no haya suficiente memoria almacenada para todos ellos.

Solución: liberando la memoria, saber cuándo se puede usar un arreglo y cuando no.

1. *Intentar la división de una memoria que ya está dividida*

Es recomendable liberar la memoria de aquellos arreglos que ya no son requeridos en el resto del programa (reciclado de arreglos).

DEALLOCATE(arreglo)

Cuando el arreglo está partido el deallocate lo regresa a un estado no dividido, y ya se podrá usar para otro arreglo.

Después de borrar la memoria puedo utilizar el ALLOCATE de nuevo y construir un arreglo distinto

Para saber si la memoria se ha liberado:

IF ( .NOT. ALLOCATED(x)) ALLOCATE (x(1:3))

SI el resultado es true, significa que no está ALLOCATED y se puede usar para otro arreglo.

Si el resultado es false, sigue ALLOCATED

Por si un arreglo ya está ALLOCATED, que se libere su memoria para poder usarla en otro arreglo

IF (ALLOCATED(x)) DEALLOCATE(x)

[**ALGORITMO COMB SORT**](https://en.wikipedia.org/wiki/Comb_sort)

algoritmo para ordenar/comparar números dependiendo de una cierta distancia.

Requiere de una brecha/gap entre números

7 1 4 8 5 3

Gap puede iniciar de varias maneras

* Gap -> n-1=5
* Gap -> n/1.3=4.61=4

Si el resultado del gap es un decimal, solo se toma en cuenta el entero.

Se comparan los números que tengan la distancia del gap dependiendo de cómo se inició, y se cambia el orden de los números (se cambian las posiciones de los números). Ya que están todos calculados (se termina la vuelta) se obtienen un nuevo gap.

Se utilizan Loops para hacer un gap en fortran.

EJEMPLO

Gap = n/1.3 = 4.61 = 4

5 1 4 8 5 3 o

El tres está a 4 espacios del uno pero es mayor.

Gap = gap/1.3 = 4/1.3 = 3

5 1 4 8 5 3

Se cambia si el primer número es mayor que el que se está comparando

5 1 3 8 7 4

Gap = 3/1.3 = 2

5 1 3 8 7 4

3 1 5 4 7 8

Gap = 2/1.3 = 1

3 1 5 4 7 8

1 3 4 5 7 8

El proceso acabará cuando el gap sea igual a 0

declaramos gap como integer y iniciamos un do while, GAP=GAP/1.3

**subrutina de el programa**

**SUBROUTINE orden(n,x)**

**!esta subrutina se usara para acomodar a los numeros que el usuario ingrese por el archivo de texto**

**!en esta subrutina nos dara los numeros de manera ascendete**

**INTEGER::Gap,n,i,j**

**REAL,DIMENSION(1:n)::x**

**REAL::temp**

**!combsort empeiza**

**Gap=n/1.3**

**DO WHILE (Gap>0)**

**DO i=1,n**

**j=i+gap**

**if(j<=n)then**

**IF(x(i)>x(j))THEN**

**temp=x(i)**

**x(i)=x(j)**

**x(j)=temp**

**END IF**

**END IF**

**END DO**

**Gap=Gap/1.3**

**END DO**

**! anidamos un if dentro de otro, para que no nos de errores de compilación**

**END SUBROUTINE orden**

**RESUMEN PDF PARA PROGRAMA MATRICES**

**MATRICES EN FORTRAN (DECLARACIÓN DINÁMICA)**

Dimensiones de una tabla se decide en la ejecución del programa (se le piden al usuario), pero los índices(dimensión) que va a tener la tabla se deciden en la compilación.

* Para declarar la matriz/tabla

TIPO, ALLOCATABLE :: nombre\_tabla(índices : dimensión)

* Para crear la tabla/matriz sin declarar

ALLOCATE(tabla(n,n))

El usuario introduce las dimensiones de la matriz/tabla (n)

**DEALLOCATE:** usado para destruir una tabla que ya ha sido creada

Para saber si una tabla/matriz ya ha sido creada usamos *ALLOCATED(nombre de la tabla)* que nos da un valor lógico.

* .TRUE. si ya está creada
* .FALSE. no ha sido creada

ascendente

print\*,y(i)

descendente

DO i=n,1,-1

PRINT\*,y(i)

END DO

DISTINTOS CASOS Para OPERAR ARREGLOS (corranlos para que vean lo que hacen pueden servir en algun momento)

PROGRAM arreglo0

IMPLICIT NONE

REAL,DIMENSION(3)::y

Y(1)=1.5

Y(2)=3.4

Y(3)=2.5

END PROGRAM

PROGRAM arreglo1

IMPLICIT NONE

INTEGER:: i

REAL,DIMENSION(10):: w

PRINT\*,"Introduce los 10 elementos"

DO i=1, 10

READ\*, w(i)

END DO

END PROGRAM

PROGRAM arreglo2

IMPLICIT NONE

REAL:: z(5),z1(3)

Z = [1,2,3,4,5]

Z1= [6.2, 4.4, 66.7]

END PROGRAM

PROGRAM arreglo3

IMPLICIT NONE

INTEGER:: i

REAL,DIMENSION(1000)::B

B= [(i,i=1, 1000)]

END PROGRAM

PROGRAM arreglo4

IMPLICIT NONE

INTEGER:: i,j

REAL,DIMENSION(25)::W,Z

W= [((0,i=1, 4),5\*j,j=1,5)]

Do j=1,5

DO i=1,4

z(1+5\*(j-1))=0

END DO

z(5\*j)=5\*j

END DO

DO i=1,25

PRINT\*,W(i),Z(i)

END DO

END PROGRAM

PROGRAM veloo

IMPLICIT NONE

REAL,DIMENSION(3):: x, v=[32,67,9]

INTEGER:: t

OPEN(unit=54,file="A1.dat",status="unknown",action="write")

x=[0,0,0]

DO t=0, 1440/2, 30

x(1)= x(1) + v(1)\*t

x(2)= x(2) + v(2)\*t

x(3)= x(3) + v(3)\*t

WRITE(54,\*) x(1),x(2),x(3)

WRITE(54,\*) x(1)/2,x(2)/2,x(3)/2

END DO

END PROGRAM