

Introducción a Fortran95

Diego Rosales

Instituto de Física de Líquidos y Sistemas Biológicos.



Departamento de Ciencias Básicas, Fac. de Ingeniería, UNLP



FACULTAD
DE INGENIERÍA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Herramientas **C**omputacionales para **C**ientíficas/os
Curso 2024

¿Qué es Fortran?

Fortran, acrónimo de **FOR**mula **TRAN**slation, es uno de los lenguajes de programación más antiguos y aún en uso. Desarrollado originalmente por IBM en la década de 1950, fue diseñado específicamente para cálculos científicos y de ingeniería. Su nombre refleja su propósito principal: **traducir fórmulas matemáticas en instrucciones comprensibles para una computadora.**

Fuente: <https://en.wikipedia.org/wiki/Fortran>

¿Por qué Fortran?

- **Cálculos numéricos:** Es un **lenguaje de programación** de elección para realizar cálculos numéricos complejos (ejemplos: de **física, la ingeniería, la meteorología y la finanzas**).
- **Optimización:** Los programas pueden **optimizarse** para ejecutarse en computadoras de alto rendimiento. El lenguaje es adecuado para producir código donde el **rendimiento es importante**.
- **Legado:** Muchas **aplicaciones científicas y de ingeniería de gran escala** aún están escritas en Fortran: gran cantidad de código existente que debe ser mantenido y actualizado.
- **Paralelismo:** Fortran ha evolucionado para soportar **cómputo en paralelo**.

Fuente: <https://en.wikipedia.org/wiki/Fortran>

Lenguaje compilado

Un **COMPILADOR** traduce el código fuente a código máquina, creando un archivo **EJECUTABLE**. Este archivo puede ejecutarse directamente en la máquina sin la necesidad del código fuente original.

Desventajas de los lenguajes compilados:

- **Velocidad:** El código máquina es ejecutado directamente por el hardware.
- **Eficiencia:** Los compiladores pueden realizar optimizaciones en el código.
- **Independencia:** Una vez compilado, el programa puede ejecutarse en cualquier máquina que tenga el entorno de ejecución adecuado.
- **Seguridad:** Los errores de compilación suelen ser detectados antes de la ejecución.

Lenguaje compilado

Ventajas de los lenguajes compilados:

- **Proceso de desarrollo más lento:** Cada cambio en el código requiere una nueva compilación, lo que puede ralentizar el desarrollo.
- **Dependencia de la plataforma:** En algunos casos, un programa compilado para una plataforma (por ejemplo, Windows) puede no ejecutarse en otra (por ejemplo, Linux).
- **Mayor tamaño de los archivos:** Los archivos ejecutables suelen ser más grandes que el código fuente original.

[Lenguaje interpretado: son aquellos que se ejecutan directamente, sin ser traducidos previamente a código máquina. Suelen ser más lentos, ya que el código fuente debe ser interpretado por el software del intérprete pero más fáciles de aprender y utilizar. Ejemplos: Python, JavaScript, PHP, R.]

Características de Fortran95

Fortran 95 representa una actualización significativa del lenguaje, introduciendo características modernas como:

Fortalezas de Fortran

- **Programación estructurada:** Permitiendo una mejor organización y legibilidad del código.
- **Arreglos:** Facilitando el manejo de grandes conjuntos de datos.
- **Módulos:** Promoviendo la reutilización de código.
- **Objetos:** Introduciendo conceptos de programación orientada a objetos.

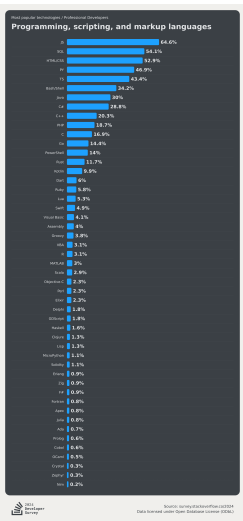
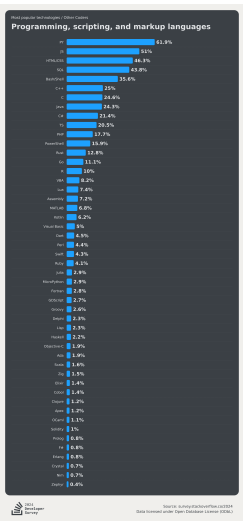
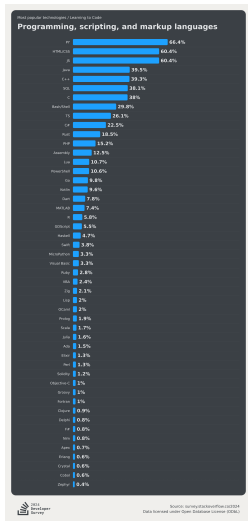
Fortalezas de Fortran

- **Rendimiento:** Excelencia en cálculos numéricos y optimización de código.
- **Gran comunidad:** Una comunidad de usuarios activa y una amplia base de conocimientos.
- **Bibliotecas:** Disponibilidad de numerosas bibliotecas científicas y de ingeniería.

Fortalezas de Fortran

- **Sintaxis:** Puede parecer anticuada en comparación con lenguajes más modernos.
- **Curva de aprendizaje:** Puede ser más desafiante para principiantes que no estén familiarizados con la programación estructurada.
- **Menos popular:** Ha perdido popularidad en comparación con otros lenguajes como C++ y Python, lo que puede limitar la disponibilidad de herramientas y recursos.

Stackoverflow: 2024 survey



Fuente: <https://survey.stackoverflow.co/2024/>

Tipos de programas en Fortran

- **Programa fuente (ejemplo.f95)**: Es el programa que editamos y que contiene las instrucciones del lenguaje FORTRAN. Para su edición, utilizaremos cualquier editor de texto.

⇓ **Compilación** ⇓

- **Programa objeto**: Se traduce el programa al programa equivalente escrito en lenguaje de máquina. A ese proceso se lo llama **compilar** y al traductor se lo conoce como **compilador**.

⇓ **Linkado** ⇓

- **Programa ejecutable (algunnombre.out)**: enlazar (link en inglés) el código objeto producido en el primer anteriormente, eventualmente con otros códigos que se encuentran en archivos biblioteca.

Software que vamos a (o podemos) usar:

- Compilador **GNU Fortran** o Intel Fortran
- Editores de texto plano: **nano**, emacs, vi, pico,...
- Para windows: GNU Mingw-w64
<https://www.mingw-w64.org/>
- Graficos:
 - 1 **Gnuplot** (free, portable)
 - 2 Octave (free, similar a Matlab)

Instalando Fortran

GFortran es el compilador Fortran del proyecto GNU
(<https://gcc.gnu.org/wiki/GFortran>).

Pasos básicos para instalar GFortran en Windows y Linux

WINDOWS

- <http://www.equation.com> ejecutables en 32 and 64-bit x86 para GCC.
- TDM GCC, ejecutables en 32 and 64-bit x86 para GCC.
- MinGW-w64 ejecutables en 64-bit x86 para GCC.

LINUX

- `sudo add-apt-repository ppa:ubuntu-toolchain-r/test`
- `sudo apt update`
- `sudo apt install gfortran-10`

Escribiendo y compilando código

Opciones

- Linux: Editor de texto plano (kwrite,nano) + Terminal



- Linux y Windows: Visual Studio Code
<https://code.visualstudio.com/>



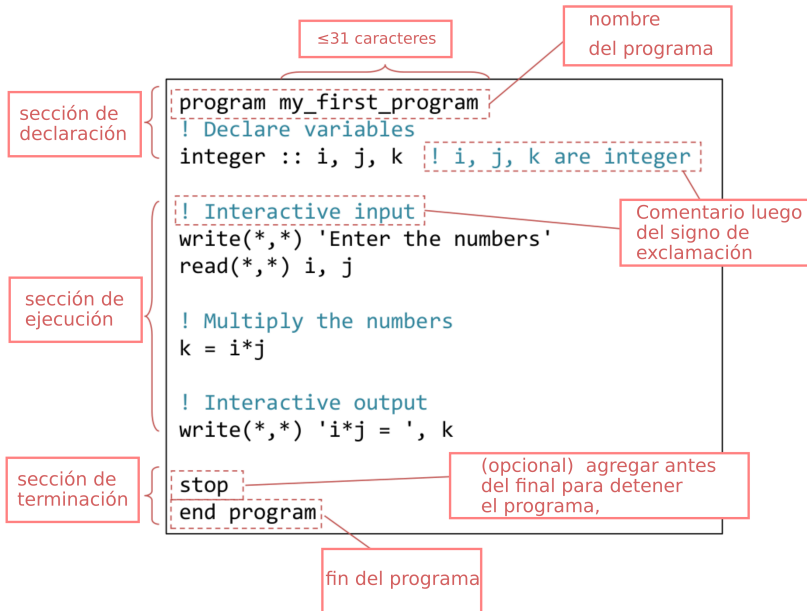
Contenido del curso

Temas a cubrir en el curso

- **Sintaxis básica:** Variables, tipos de datos, operadores, expresiones.
- **Estructura de control:** Sentencias condicionales, bucles.
- **Subprogramas:** Funciones y subrutinas.
- **Arreglos:** Declaración, manipulación y operaciones.
- **Entrada y salida:** Lectura y escritura de datos.
- **Módulos:** Organización y reutilización de código.
- **Programación orientada a objetos:** Introducción a los conceptos básicos. Herencia. Polimorfismo. Diseño Orientado a Objetos.
- **Breve idea de técnicas de paralelización en Fortran:** OpenMP, MPI, Co-arrays.



Ejemplo de un programa en Fortran:Ejemplo0



Elementos básicos de un programa en FORTRAN

Un programa en FORTRAN tiene los siguientes elementos básicos:

- **Nombre del programa.** El nombre del programa es en realidad opcional, pero es muy buena idea tenerlo.
- **Declaraciones de variables** utilizadas en el programa.
- **Cuerpo del programa.** Comandos a ejecutar en el código. Los comandos se ejecutan en orden de aparición. El programa siempre debe terminar con el comando **END PROGRAM**.
- **Subprogramas.** El cuerpo del programa puede llamar a subprogramas que realicen tareas específicas. Es buena práctica de programación separar un programa en bloques y poner cada bloque en diferentes subprogramas.

Elementos básicos de un programa en FORTRAN

La estructura de un comando en FORTRAN 90 tiene las siguientes propiedades:

- Los comandos se escriben en **líneas de no más de 132 caracteres**.
- **Espacios en blanco** al principio de una línea se ignoran. Esto ayuda a mejorar visualmente la estructura del programa.
- **Un signo & al final de una línea** indica que el comando continúa en la línea siguiente.
- Todo lo que siga de un **signo !** se considera un **comentario** y es ignorado por el compilador.
- Importante: **FORTRAN no distingue entre mayúsculas y minúsculas** en un programa, también ignora más de un espacio en blanco y líneas en blanco.

Compilando nuestro primer programa Fortran

```
program prueba
  ! imprime un mensaje
  write (*,*) 'Hola, _hoy_es_viernes_1_de_septiembre&
          _chann!!! '
end program prueba
```

- guardar en archivo de texto 01-print_hola.f95
- compilar [home]\$ gfortran 01-print_hola.f95 -o sasa
- ejecutar [home]\$./sasa

PARTE 1

Objetivo de la parte 1:

Escribir un programa en Fortran 90/95 y adquirir conocimientos en la invocación y aplicación de operaciones fundamentales.

Ejercicio propuesto: Leer dos archivos de datos con varias filas y columnas (dos matrices), hacer una operación entre filas y columnas (multiplicarlas) y determinar si el resultado es un archivo (matriz) con todos sus elementos positivos.

¿Qué necesitamos saber para poder resolver el problema?

- | | |
|--------------------------|--------------------------------------|
| I Leer matrices → | ● E/S, Memoria |
| II Multiplicar → | ● Operaciones, Memoria, Repeticiones |
| III Imprimir resultado → | ● E/S |
| IV ¿Positivos? → | ● Decisiones |

Ejercicios extras: cubren los conceptos básicos de E/S, variables, operaciones y decisiones

- 1b Leer un archivo de texto y contar el número total de palabras que contiene. Luego, imprimir el resultado en la pantalla.
- 1c Escriba un programa que lea un número y determine si es un número primo.
- 1d Escriba un programa que lea una lista de 10 números y los ordene en orden ascendente.
- 1e Leer un archivo que contiene una lista de números. Encontrar y mostrar el número más grande en la lista.

CONSTANTES Y VARIABLES

Constantes y variables

Los programas en FORTRAN manejan datos de dos tipos: constantes y variables.

Tipos de datos básicos

- **Strings:** `character :: 'sarasa', 'pepino'`
- **Números enteros:** `integer :: name1=3, name2=-55`
- **Números reales:** `real :: r1=3.0, r99=-23.01`
- **Complejos:** `complex :: pop=(1.5,2.0)`
- **Lógicos:** `logical :: visita=.true., canaste=.false.`
- **Arreglos de cualquiera de los tipos anteriores.**
Ejemplo: `real :: AA=(/1.2,-2.35,4.59/)`
- **Parameter (atributo)-** Sirve para almacenar un valor constante en una variable de forma que no cambie a lo largo de todo el programa.

Notar la diferencia entre Complex y Array.

Constantes y variables

Tipos de datos básicos

Los tipos de datos se definen como combinación de un tipo de **base** y un **tamaño**. El tipo de base puede ser **INTEGER**, **REAL**, **COMPLEX** o **LOGICAL**. El tamaño especifica la cantidad de memoria que se reserva para la variable.

- **Enteros (1, 2, 4 o 8 bytes)**
 - ▶ `INTEGER*1` → -128 a 127
 - ▶ `INTEGER*2` → -32,768 a 32,767
 - ▶ `INTEGER*4` → 2,147,483,648 a 2,147,483,647 (defecto)
- **Reales (4 o 8 bytes.)**
 - ▶ `REAL*4` → 7 decimales $10^{\pm 38}$ (defecto)
 - ▶ `REAL*8` → 15 decimales $10^{\pm 308}$
- **Complejas: se almacenan como pares de números reales (8 o 16 bytes)**
 - ▶ `COMPLEX*8` → 7 decimales $10^{\pm 38}$ (defecto)
 - ▶ `COMPLEX*16` → 16 decimales $10^{\pm 308}$
- **Caracteres: 1 byte por carácter**
- **Lógicas (1, 2, 4 bytes etc, pocas veces utilizadas):** `LOGICAL*1`, `LOGICAL*2`, `LOGICAL*4`

Declaración de variables

Para evitar usar declaraciones implícitas se debe poner al principio del programa la siguiente línea:

- `'implicit none'` obliga a declarar todo lo que se vaya a usar

```
program prueba1  
  implicit none  
  .  
  .  
  .  
end program prueba1
```

Si no se agrega `'implicit none'`, se asume que variables no declaradas tienen un tipo implícito de acuerdo a la siguiente regla: variables cuyos nombres empiezan con `{i, j, k, l, m, n}` se asumen enteras, y todas las demás se asumen reales.

Declaración de variables

- `implicit none` obliga a declarar todo
- Asignar nombres (variables) a los espacios de memoria.
- Palabras de hasta 31 letras
- Incluir números “0-9” y guión bajo “_” (no al comienzo)
- Mayúsculas y minúsculas no se diferencian: `numero_de_planetas` es idéntico a `Numero_de_PLANETAS`

Ejemplos

- `character:: mi_nombre, tu_nombre`
- `integer:: i, j, k, u1, pantuflas`
- `real:: x, y, z, Jota`
- `logical:: respuesta2, test`
- `real,dimension(3):: vector1, vv, conejo`
- `integer,dimension(10,10):: matriz, mm, caso56`

Asignación: ejemplos

Los valores se asignan a las variables en Fortran mediante el **operador de asignación (=)**. El **operador de asignación** copia el valor del lado derecho del operador al lado izquierdo del operador.

```
character :: mi_nombre  
integer :: i, j  
real :: x, Jota  
logical :: respuesta2, test  
real, dimension(3) :: vector1  
integer, dimension(10,10) :: mosca
```

```
mi_nombre='Diego'  
i=1789  
x=1.08  
vector1=(/1.0,-0.59,6.98/)  
vector1(2)=-9.59  
sapo(:, :)=0.0  
tortuga(1,1)=1.59
```

```

program variables
  implicit none
  ! Declaro variables
  logical :: isAlive
  integer :: i
  real :: a
  character(30) :: texto
  !-----
  i = 1
  a = 2.5
  texto = 'Estas_son_las_variables'
  isAlive=1>0
  !-----
  write(*,*) texto
  write(*,*) isAlive , i , a
end program variables

```

Código: 02-variables.f95

E/S

Entrada/Salida

La **entrada** y **salida** en Fortran se realiza mediante las sentencias `READ` y `WRITE`. La sentencia `READ` lee un valor desde un dispositivo de entrada y lo asigna a una variable. La sentencia `WRITE` escribe un valor en un dispositivo de salida.

Por default, la entrada de datos es desde el teclado y la salida es a la pantalla.

- `read(,)`
- `write(,)`

Ambos comandos tienen dos argumentos, el primero de los cuales indica la “unidad” de entrada o salida, y el segundo el formato en el que están los datos. La versión más simple es:

- `read(*,*)`
- `write(*,*)`

Aquí, el primer asterisco indica entrada o salida estandar (teclado y pantalla respectivamente), y el segundo formato libre. El comando `write(*,*)` puede substituirse por la forma equivalente `print *` seguido de una coma.

```
PROGRAM mengano
  !Declarar variables .
implicit none
character(20) :: nombre
print *           !Esta linea imprime un lugar en blanco
write (*,*) 'Cómo_te_llamas_?'

print *           !Esta linea imprime un lugar en blanco

read (*,*) nombre !Leemos desde el teclado
print *
write (*,*) 'Hola' , nombre
print *
END PROGRAM mengano
```

Código: 03-entrada_salida.f95

Entrada/Salida

Además de utilizar el teclado y la pantalla, los datos también pueden **leerse** o **enviarse** a un **archivo**. Para ello:

- 1 abrir el archivo con el comando `open`
- 2 leer (`read`) o escribir (`write`)
- 3 cerrar el archivo con el comando `close`

El prototipo sería el siguiente

```
PROGRAM testES
  ! codigo
  ! codigo
open (unit=|número, \verb| file =|"nombre_de_archivo"\verb
read (|de donde, en que formato\verb|)
write (|a donde, en que formato\verb|)
close (unit=|número\verb|)

END PROGRAM testES
```


Ejemplos

- `write(*,*) a`
- `write(*,*) 'pelota'`
- `read(*,*) b`
- `open(unit=1,file='mate.txt')`
- `write(1,*) a`
- `close(unit=1)`

```
program abrofile
  implicit none
  integer :: edad
  character(20) :: nombre
  !---
  open(10, file = 'entrada.dat')
  read(10,*) nombre, edad
  close(10)
  !---
  open(11, file='salida.dat')
  write(11,*) 'Me llamo' , nombre
  write(11,*) 'Tengo' , edad , 'a_\_~_\_nos'
  close(11)
  !---
end program abrofile
```

Código: 04-entrada_salida_files.f90

OPERACIONES BÁSICAS

Operaciones en FORTRAN

Las operaciones aritméticas en FORTRAN involucran los siguientes operadores:

- **Asignaciones:** =

Es muy importante recalcar que este operador **NO** significa igualdad. En FORTRAN, tiene sentido escribir líneas que **algebraicamente** son absurdas como por ejemplo:

$$a=a+1$$

Esto significa tomar la variable a , sumarle 1, y asignar el resultado de nuevo a la variable a .

Operaciones

- Reales: +, -, *, /, **
- Enteros: +, -, *, / entera, **
- Complejos: +, -, *, /, ** (raíz principal)
- Lógicos: .and., .or., .not.
- Caracteres: // (concatena o une strings)
- Comparaciones: ==, <, >, <=, >=, /= (resultados lógicos)

Ejemplos

- `a = 'clase de Fortran'`
- `x = 2.53`
- `w = x**2 + 5.1`
- `b = a // a`
- `x > w`

Algunas funciones intrínsecas útiles

Function name & argument	Argument type	Result type
<code>sqrt(x)</code> <code>SQRT(X)</code> ($x \geq 0$)	R	R
<code>abs(x)</code>	R/I	R/I
<code>sin(x)</code> (x in radians)	R	R
<code>cos(x)</code> (x in radians)	R	R
<code>tan(x)</code> (x in radians)	R	R
<code>exp(x)</code>	R	R
<code>alog(x)</code>	R	R
<code>alog10(x)</code>	R	R
<code>int(x)</code> integer part of x	R	I
<code>nint(x)</code> nearest integer to x	R	I
<code>real(i)</code>	I	R
<code>fraction(X)</code> $= x - \text{int}(x)$	R	R
<code>mod(a,b)</code> $= a - \text{int}(a/b)*b$ remainder	R/I	R/I
<code>max(a,b)</code>	R/I	R/I
<code>min(a,b)</code>	R/I	R/I
<code>asin(x)</code>	R	R
<code>acos(x)</code>	R	R
<code>atan(x)</code>	R	R

Funciones pre-definidas para manipular arrays

- `maxval(array [,dim])` : Calcula el valor máximo.
- `minval(array [,dim])` : Calcula el valor mínimo.
- `maxloc(array [,mask])` : Localización del mayor elemento.
- `minloc(array [,mask])` : Localización del menor elemento.
- `product(array [,dim] [,mask])` : Producto de los elementos del array.
- `sum(array [,dim] [,mask])` : Suma de los elementos del array.
- `dot product(vec 1, vec 2)` : Producto escalar.
- `matmul(matriz a, matriz b)` : Multiplicación matricial.
- `transpose(matriz)` : Matriz transpuesta.

BUCLES

Repeticiones/Bucles

En Fortran 90, los bucles **DO** y **DO WHILE** son dos tipos de bucles utilizados para la **repetición de código**, pero difieren en su estructura y cómo se evalúan las condiciones.

Ejemplo más básico: DO

```
DO i=inicio , fin , paso
    comando 1
    comando 2
    .
    .
END DO
```

- `i, inicio, fin, paso` que deben ser enteros.
- la variable `i` inicia en `inicio`. Al terminar, el valor de `i` se incrementa en `paso`.
- siempre que `i <= fin` se vuelven a ejecutar los comandos dentro del loop.
- cuando `i > fin` el loop ya no se ejecuta. El programa continua en el comando que siga después de `end do`.

Repeticiones 1: DO

do variable entera = inicio, fin, paso

Bloque de instrucciones que deben repetirse. La variable entera puede usarse para adaptar las operaciones a la pasada correspondiente.

enddo

Ejemplo 5: suma de números pares

```
program pares
integer :: i, j
j = 0
do i = 2, 100, 2
    j = j + i
enddo
write (*,*) j
endprogram pares
```

Código: 05-repeticiones_DO.f95

Repeticiones 2: DO WHILE

`do while` (condición)

Bloque de instrucciones
que deben repetirse
mientras la condición sea
VERDADERA.

`enddo`

CUIDADO: ¿Qué problema puede haber con este bucle?

Ejemplo 6: suma de números pares

```
program pares
integer :: i , j
j = 0
i = 0
do while (i <= 100)
    i = i + 2
    j = j + i
enddo
write (* , *) j
endprogram pares
```

Código: 06-repeticiones_DO_WHILE.f95

DECISIONES

Decisiones

En ocasiones uno desea que una parte del programa **sólo sea ejecutada si** cierta condición específica se satisface.

“Condicionales”, en FORTRAN:

- `if`
- `select case`

Ambos tienen sus propios usos y ventajas, y la elección entre ellos depende de la situación específica

Decisiones

IF-ELSEIF-ELSE-ENDIF

cuando necesites manejar **condiciones complejas o evaluar expresiones booleanas múltiples**.

SELECT CASE

cuando estés **comparando una sola variable con varios valores posibles**, especialmente si esos valores forman un rango o conjunto de valores discretos

Decisiones 1

if (condición 1) then

Hacer si condición 1 es
VERDADERA.

elseif (condición 2) then

Hacer si condición 1 es
FALSA y 2 es VERDADE-
RA.

else

Hacer si las condiciones 1
y 2 son FALSAS.

endif

Ejemplo 7: clima

PROGRAM ClasificarClima

REAL :: temp

!Asignar un valor

!a la temperatura

temp = 25.0

IF(temp < 10.0) **then**

print *, "Hace_frío"

ELSEIF (temp>=10.0 .and. temp<30.0)

print *, "Temperaturatemplada"

ELSE

print *, "Hace_calor"

ENDIF

END PROGRAM ClasificarClima

Código: 07-decisiones_IF_ELSE.f95

Decisiones 2

`select case (expresión) case
(valor 1)`

Hacer si expresión==valor 1.

`case (valor 2, valor 3, ...)`

Hacer si expresión==valor 2, o valor 3, ...

`case default`

Hacer si expresión/=valores listados.

`endselect`

Ejemplo 8: días de semana

```
integer :: dia
!Asignar un valor a un día
dia = 3
select case(dia)
  case(1)
    print *, "Lunes"
  case(2)
    print *, "Martes"
  case(3)
    print *, "Miércoles"
  case(7)
    print *, "Domingo"
  case default
    print *, "Valor_no_válido"
end select
```

Código: 08-decisiones_SELECT_CASE.f95

ARRAYS

Arreglos

Son variables que pueden almacenar un conjunto de datos del mismo tipo. Los arreglos se pueden declarar de forma **estática** o **dinámica**.

Arreglos estáticos

Tienen un tamaño fijo que se especifica **en el momento de la declaración**. Por ejemplo, el siguiente código declara un arreglo estático de 10 enteros:

```
integer :: a(10)
```

El tamaño del arreglo a es 10. Esto significa que el arreglo puede almacenar 10 enteros

Arreglos dinámicos

No tienen un tamaño fijo. El tamaño del arreglo se puede especificar en tiempo de ejecución mediante la sentencia **ALLOCATE**. Por ejemplo, el siguiente código declara un arreglo dinámico de enteros:

```
integer :: a  
allocate(a(10))
```

La sentencia **ALLOCATE** reserva 10 bytes de memoria para el arreglo **a**.

Arreglos

FORTRAN puede almacenar en memoria vectores y matrices utilizando variables llamadas “arreglos”. Los “arreglos” pueden ser de cualquiera de los tipos aceptados por FORTRAN.

Arreglos de tamaño fijo (desde el principio):

Por ejemplo, para declarar un vector de 3 componentes y una matriz de 4×5 se escribe:

```
real,dimension(3) :: v
```

```
real,dimension(4,5) :: m
```

- También se puede dar explícitamente el rango permitido para los índices:

```
real,dimension(0:8),v1
```

```
real,dimension(2:5),v2
```

Arrays

- Los arrays (o matrices) contienen una colección de diferentes valores al mismo tiempo.
- Se accede a los elementos individuales subindicando el array.
- Un array de 10 elementos se visualiza como

1	2	3	...	8	9	10
---	---	---	-----	---	---	----

mientras que un array de 4×3 como

	Dimension 2		
	→		
Dimension 1 ↓	(1,1)	(1,2)	(1,3)
	(2,1)	(2,2)	(2,3)
	(3,1)	(3,2)	(3,3)
	(4,1)	(4,2)	(4,3)

Arrays: algunos detalles

Fortran

- Almacenamiento por columnas
- Índices basados en 1
- Arreglos multidimensionales: simples
- Necesidad de gestionar la memoria: solo dinámicos

C

- Almacenamiento por filas
- Índices basados en 0
- Arreglos multidimensionales: maso
- Necesidad de gestionar la memoria: todos los arreglos, ya sean estáticos o dinámicos

Arrays

Declaración: ejemplos equivalentes

```
real , dimension(100) :: a
```

```
real :: a(100)
```

```
real :: a(1:100)
```

```
real :: a(-39:60)
```

```
integer , parameter :: nn = 100  
real :: a(nn)
```

Arrays

Inicialización: ejemplos equivalentes

```
real :: a(5)
do i = 1, 5
    a(i) = i
end do
```

```
real :: a(5)
a = (/ 1.0, 2.0, 3.0, 4.0, 5.0 /)
```

```
real :: a(5) = (/1.0, 2.0, 3.0, 4.0, 5.0/)
```

```
real :: a(5) = (/ (i, i = 1, 5) /) ! DO implícito
```

Más sobre Arrays

```
real :: a(10)
real :: b(10)
real :: c(10)
...
do i = 1, 10
  c(i) = a(i) + b(i)
end do
```

→

```
real :: a(10)
real :: b(10)
real :: c(10)
...
c=a+b
```

I/O de Arrays

```
write(*,*) a(1), a(2), a(3), a(4), a(5)
write(*,*) (a(i), i = 1, 5)
write(*,*) a(:)
write(*,*) a(2:4)
```


Arreglos

- Declaración: `integer, dimension(1:5) :: x`
- Asignación directa: `x = (/2,4,6,8,10/)`
- Do implícito 1: `x = (/ (2*i, i=1, 5) /)`
- Do implícito 2: `write (*,*) (x(i), i=1, 5)`
- Elementos: `x(1)`, `x(2)`, etc.
- Secciones: `x(2:4)`, `x(:4)`, `x(2:)`, `y(1:3,8:15)`, `x(:)`
- Elementos de secciones: `x(1:4)(2)` **NO ES VALIDO**
- Operaciones elemento a elemento: `+`, `-`, `*`, `/`, `==`, etc.
- Asignación de secciones: `x(2:4) = (/56,76/)`
- Funciones intrínsecas: `sum(x)`, `dot_product(x,y)`, `matmul(a,b)`

Código: 09-arrays.f95

Arreglos y asignación dinámica de memoria

Fortran 90/95 permite la **asignación dinámica de memoria**. Esto quiere decir que podemos dimensionar los arreglos durante la ejecución del programa, sin tener que asignarles un tamaño al momento de declararlos. Para esto, los arreglos **deben declararse** con el atributo `ALLOCATABLE`.

```
REAL, ALLOCATABLE :: V(:) , A(:, :) !V vector y A matriz
```

Para asignarles un tamaño, usamos la sentencia `ALLOCATE`, por ejemplo:

```
ALLOCATE(V(5))  
ALLOCATE(A(N,M))
```

Usando la sentencia `DEALLOCATE` liberamos la memoria cuando dejamos de utilizar los arreglos:

```
DEALLOCATE(V)  
DEALLOCATE(A)
```

Arreglos dinámicos

Ejemplo

```
program multiplica
  real , dimension (:,:) , allocatable :: x

  allocate (x(2,2))
  x = 2.0
  write (*,*) matmul(x,x)
  deallocate (x)

  allocate (x(10,10))
  x = 1.0
  write (*,*) sum(x)
  deallocate (x)

endprogram multiplica
```

Código: 10-arrays_dinamicos.f95

Más sobre cadenas de caracteres

Para representar cadenas de caracteres (texto). Las constantes se escriben delimitando los caracteres entre apóstrofes (comilla simple) o comillas dobles:

Ejemplo

```
character ( len = 5 ) : : a  
character ( len = * ) , parameter :: b = 'Ah_no!'
```

- Secciones: `a(1:3)`, `a(1:1)`, `a(5:)`, `a(:3)`
- Operaciones: `a//b`
- Funciones intrínsecas:
 - ▶ `len(a)`: longitud de la cadena
 - ▶ `len_trim(a)`: longitud sin espacios a la derecha
 - ▶ `ichar('e')`: código entero correspondiente a la letra e
 - ▶ `char(65)`: letra correspondiente al código 65
 - ▶ `iachar(65)`, `achar(65)`: ídem pero del ASCII
 - ▶ `adjustl(a)`, `adjustr(a)`: justifica izquierda o derecha
 - ▶ `trim`, `scan`, `index`, `verify`, `repeat`

Más sobre E/S: formato

- `read(unit=*,fmt='(formato)') a, b, c, ...`
`write(unit=*,fmt='(formato)') a, b, c, ...`
- Enteros: `Iw` (base 10), `Bw` (base 2), `ow` (base 8), `zw` (base 16)
- Reales: `Fw.d` (sin exponente), `Ew.d`, `Ew.dEe`, `ESw.dEe`
- Complejos: como dos reales
- Lógicos: `Lw`
- Caracteres: `A`, `Aw`, `"texto"`
- Control: `Tn`, `TRn` ó `nX`, `TLn`, `/`, repeticiones

Ejemplos

```
write (*, '(2X,I4,F8.3,A)') int, real, char12
write (*, '("Resultado=",I4,E8.3,/,A3)') int, real, char12
read (1, '(I4,5X,F10.7,A15)') int, real, char12
read (1, '(3I4)') int1, int2, int3
```

Código 11-ES_formato.f95

Códigos de formato

Código	Descripción	Significado de los valores w, m, d, e, k, n, r:
Iw[.m]	Entero base 10 (decimal)	w establece la anchura del campo
Bw[.m]	Entero base 2 (binario)	m indica al menos m cifras en el campo
Ow[.m]	Entero base 8 (octal)	d indica el número de cifras decimales en el campo
Zw[.m]	Entero base 16 (hexadecimal)	e indica el número de cifras del exponente
Fw.d	Real sin exponente	k es un factor de escala
Ew.d[Ee]	Real con exponente e	n indica la posición en el registro desde su principio (para el descriptor T)
ENw.d[Ee]	Real en notación ingeniera	n número de espacios a mover (para los descriptores X, TR, TL)
ESw.d[Ee]	Real en notación científica	r factor opcional de repetición, por defecto vale 1
Dw.d	Real con exponente d	Restricciones: $w > 0$, $e > 0$, $0 \leq m \leq w$, $0 \leq d \leq w$, $0 < e \leq w$, $n \geq 1$, $r \geq 1$, $k \geq 0$
Lw	Lógico	
A[w]	Carácter	
Gw.d[Ee]	Dato general	
' '	Literal	
" "	Literal	
nX	Avanzar n espacios	
Tn	Tabulación absoluta	
TRn	Tabulación a la derecha	
TLn	Tabulación a la izquierda	
[r]/	Salto de registro	
BN	Ignora espacios en campos numéricos	
BZ	Convierte espacios a ceros en campos numéricos	
S	El signo + opcional se imprime o no según el procesador	
SP	Se imprime el signo + opcional	
SS	No se imprime el signo + opcional	
kP	Factor de escala	
:	Final de formato	

Más sobre E/S

- IOSTAT: Es una especificación opcional que se usa para **manejar errores durante la operación de lectura**. IOSTAT captura el código de estado de la operación de entrada/salida.

Ejemplo

```
read(unit,fmt,iostat=i) a, b, c, ...
```

- ▶ $i < 0$: fin de archivo
- ▶ $i = 0$: OK
- ▶ $i > 0$: error

- ADVANCE:

```
write(unit,fmt,advance='no') a, b, c, ...
```

- Archivos internos:

```
read(character,fmt) real write(character,fmt) real
```

Códigos `ES_IOSTAT.f95` y `ES_ENDFILE.f95`

Recapitulación

- Entrada/Salida
- Variables
- Operaciones
- Repeticiones
- Decisiones

Ejercitación

- Probar los ejemplos dados
- Inventar variantes: anidar bucles, anidar decisiones, abortar bucles, etc.
- Resolver el Ejercicio 1 (no usar la función MATMUL)

Recordemos:

Ejercicio propuesto: Leer dos archivos de datos con varias filas y columnas (dos matrices), hacer una operación entre filas y columnas (multiplicarlas) y determinar si el resultado es un archivo (matriz) con todos sus elementos positivos.

¿Qué necesitamos saber para poder resolver el problema?

- | | |
|--------------------------|--------------------------------------|
| I Leer matrices → | ● E/S, Memoria |
| II Multiplicar → | ● Operaciones, Memoria, Repeticiones |
| III Imprimir resultado → | ● E/S |
| IV ¿Positivos? → | ● Decisiones |

Definición de lista de datos (matrices)

Sean dos archivos A y B de la siguiente forma:

$$\text{Archivo1} \rightarrow \begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}, \text{Archivo2} \rightarrow \begin{matrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{matrix}$$

Los vamos a identificar como "dos matrices"

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mk} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{k1} & b_{k2} & \dots & b_{kn} \end{pmatrix}$$

La idea es leer esta información guardada en archivos y luego realizar operaciones matemáticas entre sus elementos: multiplicar/sumar.

Operación matemática elegida: producto de matrices

Repaso producto de matrices:

El producto de la tabla (matriz) A por la tabla (matriz) B y que da como resultado la tabla C , se calcula como:

$$C = A \cdot B$$

cuyos elementos son:

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & \mathbf{c_{22}} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}$$

con

$$c_{ij} = \sum_{k=1}^N a_{ik} b_{kj}$$

Ej:

$$c_{23} = \sum_{k=1}^N a_{2k} b_{k3} = a_{21} b_{13} + a_{22} b_{23} + \dots + a_{2N} b_{N3}$$