– **Typeset by GMNI &** FoilTEX –

PROGRAMACIÓN EN LENGUAJE FORTRAN

GMNI — GRUPO DE MÉTODOS NUMÉRICOS EN INGENIERÍA

Departamento de Métodos Matemáticos y de Representación
Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos
Universidade da Coruña

GMNI - Grupo de Métodos Numéricos en Ingeniería
http://caminos.udc.es/gmni

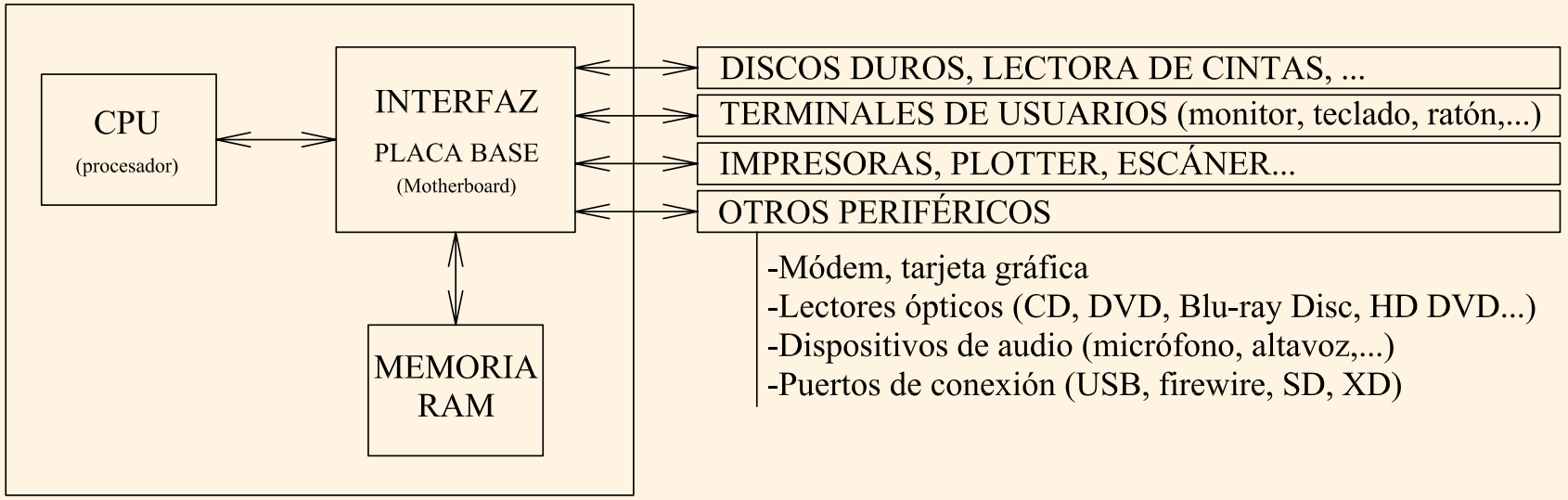
Índice

Organización de un ordenador convencional (I)

- Un ordenador es un aparato capaz de almacenar datos y procesarlos de acuerdo con una secuencia de órdenes establecidas que se le proporcionan con la finalidad de obtener una cierta información
- La secuencia de instrucciones que se aplica se conoce como "programa". A la elaboración de estos programas dedicaremos gran parte de este tema.
- Aunque existen algunos precedentes los ordenadores como tal surgieron en los años 40 del siglo XX
- Un ordenador se compone esencialmente de:
  - "Hardware": componentes físicos de funcionamiento(módulos de memoria, placa base, procesador,...)
  - "Software": componentes o aplicaciones implementadas sobre el "hardware" que han sido elaboradas para realizar una secuencia de órdenes determinada.

Organización de un ordenador convencional (II)

- Hardware:



- CPU (Central Processing Unit): es un componente del ordenador que se encarga de realizar las operaciones que se le indican con el software.
  - Unidad lógica que realiza las operaciones con los datos.
  - Unidad de control que interpreta la secuencia de operaciones y gestiona los dispositivos asociados.
- Memoria principal (RAM-"Random Access Memory"): gestiona los datos y especificaciones que está utilizando el procesador en cada momento.
  - Es una memoria volátil.
  - De acceso más rápido que la memoria secundaria.

Organización de un ordenador convencional (III)

- Memoria secundaria (Disco duro, ...): Es una memoria permanente para almacenar datos, software, ... Por lo tanto, es de acceso más lento.
- Placa base: es la interfaz encargada de conectar todos los elementos de hardware con el procesador (CPU)
- Periféricos: son los dispositivos externos que se acoplan al ordenador a través de la placa base

  -Teclado, ratón, monitor,...
  -Lectores y grabadores ópticos
  -Dispositivos de audio
  -...

▶ Software:

- Sistema operativo (Windows, Linux, Unix, MAC OS, MS-DOS, VMS,...)
- Editores de texto (Word, OpenOffice, Wordpad, Scite, vi, emacs,...)
- Hojas de cálculo (Excel, OpenOffice,...)
- ...

Organización de un ordenador convencional (IV)

- Potencia de un ordenador:
  - Frecuencia de reloj de la CPU (GHz, MHz,...) medidos en la práctica en Gigaflops (operaciones en coma flotante por segundo)
  - Velocidad del bus de conexión entre el procesador y la memoria RAM (FSB-Front Side Bus) medido normalmente en MHz.
  - Memoria RAM: tanto en tamaño (Mb, Gb,...) como en velocidad de acceso (MHz)
- Tratamiento de la información:
  - Concepto de bit (BInary digiT): base de numeración 2 $\rightarrow \square \left\{ \begin{array}{c} 0 \\ 1 \end{array} \right.$. Es la unidad de memoria más pequeña
  - CPU: procesadores de 8, 16, 32, 64, 128 bits
  - Unidades de medida:
    - 1 byte=8 bits=1 octeto
    - 1 kocteto=10^3 octetos, 1 Mocteto=10^6 octetos, 1 Gocteto=10^9 octetos
    - 1 kbyte=2^{10} bytes $= 1024$ bytes, 1 Mb=2^{20} bytes $= (1024)^2$ bytes, 1 Gb=2^{30} bytes $= (1024)^3$ bytes
  - Importancia de la base binaria:
    - Estructura del ordenador
    - Almacenamiento de información:

      Ej. 23 en Base decimal $= \left\{ \begin{array}{l} \mathbf{1} \quad \mathbf{0} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \\ \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0 \end{array} \right.$ en base binaria

Algoritmos, programas y lenguajes de programación (I)

- Algoritmo:
  - Conjunto de instrucciones que describen las distintas etapas de un método para conseguir ciertos resultados. Existen desde mucho antes de la aparición de los ordenadores.
    (Mohammed Ibn Musa abu Djafar **Al-Khwarizmi**, matemático siglo VIII-IX).
  - Si el algoritmo se hace empleando un lenguaje de ordenador, entonces ese conjunto de instrucciones recibe el nombre de "programa de ordenador"
- Tipos de lenguajes de programación:
  - Lenguaje máquina: combinación de 0 y 1 que el ordenador es capaz de entender e interpretar directamente
  - Lenguaje ensamblador (Assembler): sustituye el lenguaje máquina por códigos nemónicos y nombres simbólicos. La aplicación que traduce estos códigos en lenguaje máquina es el "ensamblador"
  - Lenguaje de alto nivel: presenta una sintaxis más sencilla para el usuario. En ocasiones, existe un lenguaje intermedio de apoyo para realizar la transición. Algunos de estos lenguajes de alto nivel son: Fortran, C, C++, Python, Java, Cobol, Lisp, Basic, Pascal...
  - ♡ Cada nivel de lenguaje de programación está soportado por programas desarrollados en alguno de los niveles inferiores

- Traducción de código fuente:
  - Intérpretes: traducen instrucción por instrucción las secuencias de operaciones durante la ejecución. Son interactivos y modificables pero son lentos en la ejecución. (Ej. Basic, Python)
  - Compiladores: traducen el programa ("código fuente") en bloque antes de la ejecución de las operaciones. No son modificables de modo interactivo pero son muy rápidos en la ejecución de las operaciones. (Ej. Fortran)

    Por ejemplo, las fases de desarrollo de un programa Fortran son:

PROGRAMA FUENTE FORTRAN (*.for,*.f)

⇓ Compilación

PROGRAMA OBJETO (*.obj)

⇓ Linkado

PROGRAMA EJECUTABLE (*.exe)

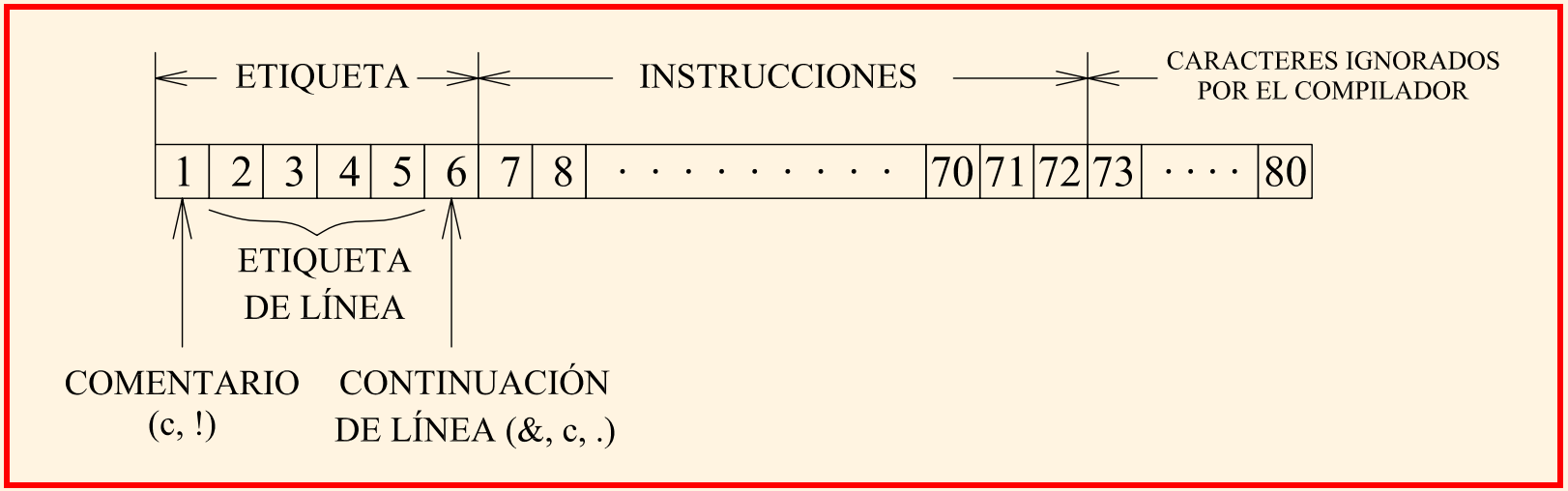
⇓ Ejecución

RESULTADOS

Lenguaje Fortran (I)

▶ Introducción:

- Fortran es el acrónimo de FORmula TRANslator y fue creado en 1954 por la compañía IBM en contraposición a otros lenguajes muy próximos al lenguaje máquina de entonces.
- Es un lenguaje estándar, de fácil utilización, muy extendido, muy bien adaptado a los problemas de ingeniería y muy perfeccionado a lo largo de sus diferentes versiones: I, II, III, IV, 66, 77, 90, 95, HPF (High Performance Fortran), 2000.
- Se trata de un lenguaje de programación secuencial cuyas sentencias se incorporan en un fichero de texto plano con extensión (*.f, *.for).
- El uso de mayúsculas y minúsculas en el fichero de texto es indiferente.
- El formato de escritura de cada línea de código fuente es:



Lenguaje Fortran (II)

▶ Elementos de un programa Fortran

- Sentencias:
  - No ejecutables — describen { -tipo, -características, -valor } de los datos
  - Ejecutables — describen una acción a realizar
- Comentarios: No afectan al procesamiento del programa (código fuente), pero favorecen la comprensión del mismo por parte del programador.

▶ Organización de un programa Fortran

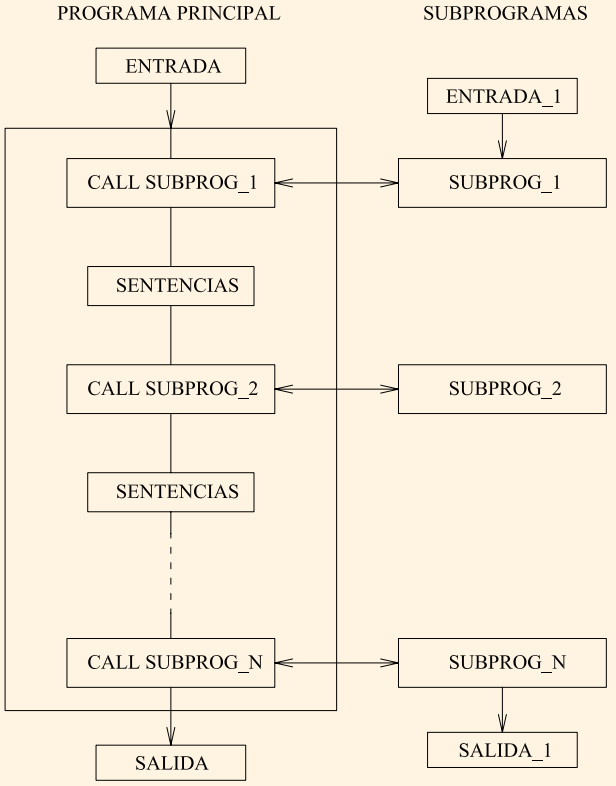
-Declaración de variables { Common, Dimension, implicit, Data, parameter }

.
.
.

-Sentencias ejecutables

.
.
.

-END



Lenguaje Fortran (III)

Sentencias Fortran

- Datos:
  - Constantes:
    - Enteras: 12, -37, ...
    - Reales { Coma fija: 6.5, -7.3, -0.12, 12.5 ... / Coma flotante: 0.12E+04, 0.13E-01, ...
    - Complejas: (-3.7,5.4),(7E-3,5.1), ...
    - Lógicas: { `.TRUE.` / `.FALSE.`
    - Alfanuméricas ("character"): 'Diego','problema1', ...

Lenguaje Fortran (IV)

- Variables: Son nombres simbólicos que corresponden a una determinada posición de memoria en la que se almacena un valor (numérico, lógico, alfanumérico, ...). El primer caracter que define el nombre debe ser una letra, pero los restantes pueden ser otros símbolos. Pueden declararse de forma explícita (una a una) o de forma implícita (mediante un criterio general aplicable a todas ellas)
  - Enteras: Almacenan números enteros.
    - Por defecto, su nombre empieza por I, J, K, L, M, N (I-N). Pero la configuración por defecto puede cambiarse fácilmente.
    - Pueden ocupar 2 bytes (simple precisión) → INTEGER*2
      Rango = (-32768, 32767)
    - Pueden ocupar 4 bytes (doble precisión) → INTEGER*4
      Rango = (-2147483648, 2147483647)
    - Declaración explícita: `integer*2 ind, num`
    - Declaración implícita: `implicit integer*4(i-n)`

    Atención:

    - Se recomienda dejar como enteras I-N y utilizarlas sólo como contadores enteros.
    - No se puede operar directamente con números reales. Hay que transformarlas previamente

Lenguaje Fortran (V)

- Reales:

  Se almacenan en coma flotante. Por defecto, su nombre empieza por (A-H,O-Z)

  Pueden almacenarse en 4 bytes (simple precisión) → REAL*4

  Rango ≈(-1.7E38,-2.9E-39), (2.9E-39,1.7E38)

  $$\underbrace{\pm 0.1234567}_{\text{MANTISA (24 bits)}} \quad \text{E} \quad \underbrace{\pm\ 123456}_{\text{EXPONENTE (8 bits)}}$$

  Pueden almacenarse en 8 bytes (doble precisión) → REAL*8

  Rango ≈(-1.0E+307,-1.E-309), (1.E-309,1.0E+307)

  Si se han de hacer operaciones entre variables de distinto tipo es necesario transformar uno de ellos para que sean del mismo tipo.

  Declaración explícita: `real*8 coord, temp`
  Declaración implícita: `implicit real*8(a-h,o-z)`

- Complejas:

  No existe declaración por defecto.

  Declaración explícita: `complex*8 a1,a2` ó `complex*16 a3`

  Declaración implícita: `implicit complex*8 (h-k)`

Lenguaje Fortran (VI)

- Lógicas: { `.true.` `.false.`

  No existe declaración por defecto

  Declaración explícita: `logical var1, var2`

  Declaración implícita: `implicit logical (a-c)`

  Operaciones: `.NOT.`, `.AND.`, `.OR.`

  Relaciones lógicas (asignan valores lógicos a variables numéricas)

  `.LT.` $\rightarrow <$ (o bien <)
  `.LE.` $\rightarrow \leq$ (o bien <=)
  `.EQ.` $\rightarrow =$ (o bien ==)
  `.NE.` $\rightarrow \neq$ (o bien /=)
  `.GE.` $\rightarrow \geq$ (o bien >=)
  `.GT.` $\rightarrow >$ (o bien >)

- Alfanuméricas: cualquier conjunto de caracteres comprendidos entre ’

  Declaración explícita: `character nombre*20, apellido*30`

  Declaración implícita: `implicit character*20 (h-m)`

  Operador concatenación:

  ```
  a='PEDRO '
  b='GONZALEZ'
  c=a//b  →  c='PEDRO GONZALEZ'
  ```

Lenguaje Fortran (VII)

- Matrices: Las matrices se almacenan por columnas en un vector

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \Rightarrow (a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{3,2}, a_{13}, a_{23}, a_{33})$$

```
integer i, j, k
real a, b, c
dimension i(10),a(3,4), b(10,10,10)
```

- Data: Asigna valores iniciales a variables antes de ejecutarse el programa

```
dimension a(3)
data a /1.0, 2.0, 3.0/
```

- Parameter: Asigna un nombre simbólico a una constante.

PARAMETER (identificador1=cte1, identificador2=cte2)

```
parameter (PI=3.14159265, ALFA=2.7)
```

▶ Operaciones básicas:

Las operaciones elementales se interpretan de derecha a izquierda. Se realizan las operaciones indicadas en las sentencias a la derecha del símbolo "=" y se almacenan sobre la variable indicada en la parte izquierda.

```
factor=x*y
```

- Suma: `x=a+b` `a, b` y `c` deben ser del mismo tipo
- Resta: `x=a-b` `a, b` y `c` deben ser del mismo tipo
- Producto: `x=a*b` `a, b` y `c` deben ser del mismo tipo
- Cociente: `x=a/b` `a, b` y `c` deben ser del mismo tipo

  Precaución al dividir entre 0.d+00. Se producen errores de "overflow" y dan lugar a salidas tipo (NaN)
- Potencia: `x=a**b`

  Si es posible el exponente `b` debe ser una variable entera. La base será normalmente una variable real.
  Si se necesita calcular potencias de exponente real, el cálculo será más lento e impreciso.

Prioridad de operaciones (de menos a más): $\left(\begin{smallmatrix} + \\ - \end{smallmatrix} \right)$, $\left(\begin{smallmatrix} * \\ / \end{smallmatrix} \right)$, $(**)$

Y en caso de conflicto de izquierda a derecha.

Lenguaje Fortran (IX)

▶ Funciones

- Externas: funciones aritméticas incorporadas en librerías del sistema

$sin(x)$	$asin(x)$	$log(x)$	$abs(x)$
$cos(x)$	$acos(x)$	$exp(x)$	$aint(x) \equiv E(x)$
			$cosh(x)$
$tan(x)$	$atan(x)$	$sqrt(x)$	$senh(x)$
			$tanh(x)$

- Intrínsecas del compilador: funciones de conversión
  - ▷ nint(x) (Nearest INTeger): (`real*4`) → Entero por aproximación
  - ▷ int(x) (parte entera): (`real*4` ó `real*8`) → Entero por truncamiento
  - ▷ ifix(x) (parte entera): (`real*4`) → Entero por truncamiento
  - ▷ float(x) (coma flotante): entero → real por defecto, normalmente 4 bytes
  - ▷ dfloat(x) (coma flotante): entero → real en doble precisión
  - ▷ dble(x) cualquier variable → real en doble precisión
- Aritméticas: definidas por el usuario

▶ Instrucciones de control:

• GOTO

♡ GOTO INCONDICIONAL:

GOTO ET_1

Transfiere el curso del programa a la línea de código que tiene como etiqueta (label) el número ET_1

Es recomendable que la línea con etiqueta ET_1 tenga la instrucción CONTINUE porque algunos compiladores lo exigen

```
      goto 47
        Sentencias no incluidas
47    continue
```

Lenguaje Fortran (XI)

♡ GOTO CALCULADO:

GOTO (ET_1,ET_2,...,ET_N), INDICE

Si INDICE vale 1, 2, ..., N el curso del programa va a las líneas de código con número de etiqueta ET_1,ET_2,...,ET_N.

Si no hay coincidencia se continúa la ejecución normal posterior a GOTO

```
      goto(11,12,13), I
        Sentencias para I ≠ 1, 2, 3
      goto 15
11    continue
        Sentencias para I = 1
      goto 15
12    continue
        Sentencias para I = 2
      goto 15
13    continue
        Sentencias para I = 3
15    continue
```

Lenguaje Fortran (XII)

- IF

  ♡ IF ARITMÉTICO:

  IF (INDICE)ET_1,ET_2,ET_3

  - Si INDICE<0 se dirige a la línea de etiqueta número ET_1
  - Si INDICE=0 se dirige a la línea de etiqueta número ET_2
  - Si INDICE>0 se dirige a la línea de etiqueta número ET_3

  ♡ IF LÓGICO:

  IF (ILOGIC) EXPRESION

  ILOGIC= Expresión o variable lógica
  EXPRESION= Expresión de cualquier tipo o instrucción

  Ejemplos:

```
if ((a.eq.b).and.(c.eq.d))e=a+c

if ((a.eq.b).and.(c.eq.d))goto 10
```

Lenguaje Fortran (XIII)

♡ BLOQUES IF:

```
if (ilogic) then
  Sentencias a ejecutar si se cumple ilogic
endif
```

```
if (ilogic) then
  Sentencias a ejecutar si se cumple ilogic
else
  Sentencias a ejecutar si no se cumple ilogic
endif
```

```
if (ilogic1) then
  Sentencias a ejecutar si se cumple ilogic1
elseif (ilogic2) then
  Sentencias a ejecutar si se cumple ilogic2 y no se cumple ilogic1
else
  Sentencias a ejecutar si no se cumple ni ilogic1 ni ilogic2
endif
```

```
if (modelo.eq.1) then
  a=x*y
elseif (modelo.gt.0) then
  a=x+y
else
  a=x-y
endif
```

Lenguaje Fortran (XIV)

♡ BLOQUES IF (Bucles anidados):

```
if (cond1) then
  if (cond2) then
  ... sentencias a ejecutar si se cumplen cond1 y cond2
  else
  ... sentencias a ejecutar si se cumple cond1 y no se cumple cond2
  endif
else
  if (cond3) then
  ... sentencias a ejecutar si no se cumple cond1 y se cumple cond3
  else
  ... sentencias a ejecutar si no se cumple cond1 y no se cumple cond3
  endif
endif
```

```
if (a.eq.1) then
  if (b.eq.2) then
    x=a+b
  else
    x=a-b
  endif
else
  if (b.eq.1) then
    x=a*b
  endif
endif
```

Programa de ejemplo: Cálculo del factorial de 10

```
      program factorial
      implicit integer*4(i-n)

      ifact=1
      i=0

10    continue

      i=i+1

      ifact=ifact*i

      if(i.lt.10)then
        goto 10
      endif

      end
```

Lenguaje Fortran (XVI)

- DO
  - ♡ Permite repetir una secuencia de operaciones un número determinado de veces:

```
do icontador=imin,imax,is
  Secuencia a repetir desde "imin" hasta "imax" de "is" en "is"
enddo
```

imin=valor mínimo de comienzo del contador
imax=valor máximo que podrá alcanzar el contador
is=intervalo de salto entre cada valor del contador (opcional, si no se indica → is=1)

```
program factorial
integer*4 i,fact
fact=1
 do i=1,10
   fact=fact*i
 enddo
end
```

```
do i=n,1,-1
  do j=1,m
    Secuencia a repetir
  enddo
enddo
```

- DO WHILE: Repite la secuencia mientras se cumpla la condición establecida

```
do while(condicion_logica)
  Secuencia a repetir
enddo
```

- CALL: Desvía la ejecución a un subprograma (se verá más adelante)
- RETURN: Desvía la ejecución desde un subprograma al programa principal
- STOP: Detiene la ejecución del programa

▶ INSTRUCCIONES DE LECTURA Y ESCRITURA DE DATOS. FORMATOS

- READ: Instrucción de lectura de datos (el programa lee datos)

  READ(NL,NF) Variables

  donde: { NL = número de unidad lógica de lectura (teclado=5)
  NF = número de etiqueta de línea donde se especifica el formato de lectura

```
      read(5,100)a,b,c
100   format(3d15.6)    → El formato se explica en el siguiente apartado
      read(5,*)a,b,c    → Lectura con formato libre
```

- WRITE: Instrucción de escritura de datos (el programa escribe datos)

  WRITE(NL,NF) Variables

  donde: { NL = número de unidad lógica de escritura (pantalla=6)
  NF = número de etiqueta de línea donde se especifica el formato de escritura

```
      write(6,100)a,b,c
100   format(3d15.6)    → El formato se explica en el siguiente apartado
      write(6,*)a,b,c   → Escritura con formato libre
```

  Los formatos se pueden escribir en cualquier punto del programa, pero es aconsejable escribirlos a continuación de la instrucción READ o WRITE

  Nunca deben introducirse dentro de instrucciones de control (goto, if, ...) porque pueden no ser accesibles desde otros puntos del programa

Lenguaje Fortran (XVIII)

- FORMAT: Establece la forma en la que se van a hacer la lectura y escritura de datos (cifras, decimales, tipo, ...)
  - ♡ Especificaciones:
    - $ → No salto de líneal al final de la instrucción
    - / → Salto de línea
    - , → Separación de especificaciones
  - ♡ Var. enteras: n**I**m siendo
    - n = número de variables con esa especificación (opcional)
    - m = dígitos totales del número entero (signo incluido)

    `15i5` → 15 números enteros con 5 dígitos (`-210`, `-1234`)
  - ♡ Var. reales: n**F**$m.d$
    - n = número de variables reales (opcional)
    - m = total de cifras del número real (incluido signo, coma y "0.")
    - d = número de cifras decimales

    n**E**$m.d$
    - n = número de reales (opcional)
    - m = total de cifras del número real (incluido signo, coma y 0, de la mantisa, letra E, signo y cifras del exponente)
    - d = cifras decimales de la mantisa

    n**D**$m.d$ Igual que el anterior pero para `real*8`

    `17f5.1` → 17 números reales de 5 cifras con 1 decimal (`12.1`, `-12.1`)
    `3e12.5` → 3 reales de 12 caracteres y 5 decimales (`0.12345E+05`, `-0.12345E-05`)

Lenguaje Fortran (XIX)

♡ Variables alfanuméricas: nam $\left\{ \begin{array}{l} n = \text{n}^\circ \text{ de variables (opcional)} \\ m = \text{n}^\circ \text{ de caracteres de cada una (opcional)} \end{array} \right.$

15a5 → 15 variables de 5 caracteres (`Diego`, `serie`, ...)

♡ Espacios: n`x` donde n es el número de espacios

♣ Normas generales sobre formatos:

- ♡ Dar un formato adecuado a cada variable
- ♡ Si el formato se agota antes que las variables se repite de nuevo
- ♡ Si la lista de variables se agota antes que el formato se obvian el resto de especificaciones
- ♡ Cuidado con la capacidad de los formatos (el número 1000 no cabe en un formato i3), aparecen $(*)$

♣ Definición alternativa de formatos:

Se puede especificar directamente un formato propio para cada sentencia READ/WRITE

WRITE(6,'(FORMATO)') → `write(6,`**`'(i5,5e15.6)'`**`)`

♣ Ejemplos

`-1234.123` → { **10** `read(5,`**`10`**`)a` **`format`**`(f9.3)` **11** `write(6,`**`11`**`)3.1416d+00` **`format`**`(d10.3)` } → `0.314E+01`

`Serie 123` → `read(5,`**`'(a5,x,i3)'`**`)a,i` `write(6,*)' Serie'` → `Serie`

- FICHEROS DE ENTRADA/SALIDA DE DATOS

OPEN(UNIT=$\left\{\text{n}^\circ\right\}$,FILE=$\left\{\begin{array}{c}\text{'entrada.txt'}\\ \text{'salida.txt'}\\ \vdots\end{array}\right\}$, STATUS=$\left\{\begin{array}{l}\text{'old'}\\ \text{'new'}\\ \text{'unknown'}\end{array}\right\}$)

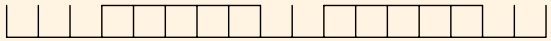
Normalmente, el número de unidad lógica $\left\{\text{n}^\circ\right\} \in [10, 99]$

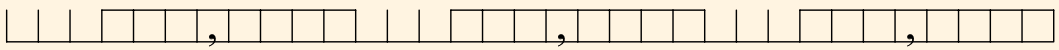
Cuando finaliza la lectura o escritura se deben cerrar con CLOSE($\left\{\text{n}^\circ\right\}$)

Ejemplos:

```
      open(unit=11,file='datos.txt',status='old')
      do i=1,100
        read(11,'(i5,3e15.6)')ipunto,xpunto,ypunto,zpunto
      enddo
      close(11)

      open(unit=12,file='entrada.txt',status='unknown')
      read(12,10)i,j,x,y,z
10    format(3x,2(i5,2x),/,3(3x,f8.4))
      close(12)
```





▶ VECTORES Y MATRICES (Conjunto ordenado de datos o *"array"*)

Declaración → Igual que para el resto de variables.
Se indica el tipo de información que se almacena.

Dimensionamiento "estático" → DIMENSION nombre_1(k_1,k_2,...,k_m)

- m indica en este caso el número de dimensiones de la matriz (*array*)
  - ♡ $m = 1$ indica que la información se almacena con estructura de vector
  - ♡ $m = 2$ indica que la información se almacena como en una matriz en dos dimensiones
  - ♡ $m > 2$ indica que la información se almacena como en una hipermatriz en m dimensiones
- k_i $(i = 1, ..., m)$ es el número de componentes que contiene el *array* en cada dimensión. Con esta información se reserva la memoria necesaria.

```
dimension v(20),a(3,4) ! vector v de 20 componentes y
                       ! matriz a de 3 filas y 4 cols.
```

- La información se almacena internamente por columnas:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{se almacena como} \quad [a_{11},\ a_{21},\ a_{12},\ a_{22}]$$

Lenguaje Fortran (XXII)

Vectores (n)

Lectura/Esc. por fila

```
dimension a(10)

read(5,*)(a(i),i=1,10)
```

Lectura/Esc. por columna

```
   dimension a(10)
   do i=1,10
     read(5,*)a(i)
   enddo
```

Matrices (n×m)

Lectura/Esc. en 1 fila

```
dimension b(10,20)

write(6,*)((b(i,j),j=1,20),i=1,10)
```

Lectura/Esc. en filas y columnas

```
   dimension b(10,20)
   do i=1,10
     write(6,*)(b(i,j),j=1,20)
   enddo
```

Lectura/Esc./modificación por filas

```
dimension b(10,20)
do i=1,10
  do j=1,20
    read(5,*)b(i,j)
  enddo
enddo
```

Lectura/Esc./modificación por columnas

```
   dimension b(10,20)
   do j=1,20
     do i=1,10
       write(6,*)b(i,j)
     enddo
   enddo
```

Consideraciones importantes:

- El almacenamiento de grandes matrices es muy costoso porque requiere mucha capacidad de almacenamiento en memoria

```
implicit real*8(a-h,o-z)
dimension a(10000,10000)
```

  $10000 \times 10000 \times 8$ bytes $\approx$ 763Mb

- Es necesario ser muy cuidadoso con el dimensionamiento de las matrices:
  - Si nos pasamos dimensionando podemos exceder los límites de memoria del ordenador
  - Si nos quedamos demasiado cortos podemos sobreescribir otras variables (OJO: Fortran no avisa de este problema)
  - **SOLUCIÓN: DIMENSIONAMIENTO DINÁMICO.**

Dimensionamiento dinámico:

- Se realiza en dos pasos:
  1. En la declaración de variables se indica que una variable será un *array*:

     ```
     implicit integer*4(i-n), real*8(a-h,o-z)
     allocatable nombre1(:,:,...), nombre2(:,:,...)
     ...
     ```

     El número de dimensiones del *array* se indica con el número de veces que aparecen ":" entre paréntesis
  2. En las sentencias del programa se indicará el tamaño de ese *array*:

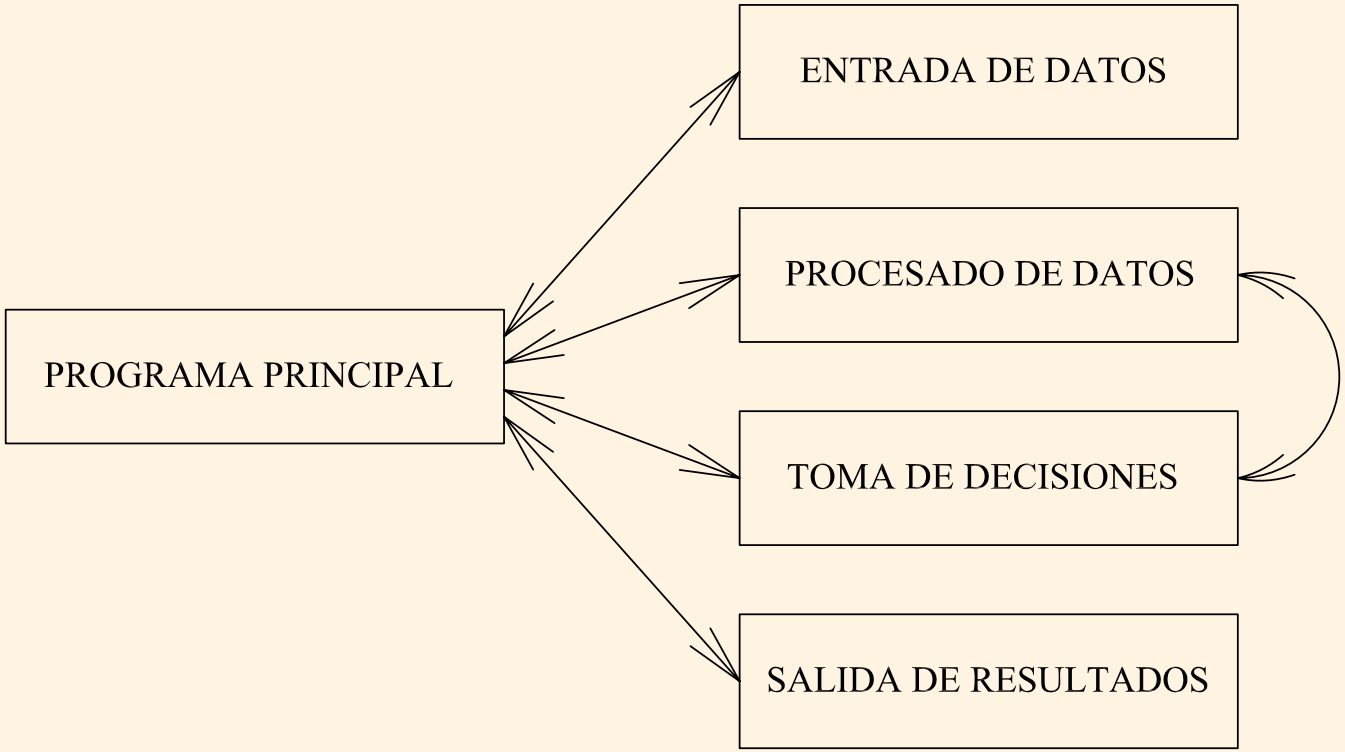
     ```
     ...
     allocate (nombre1(k1, k2, ...), nombre2(l1, l2, ...))
     ...
     ```

     siendo $k1,\ k_2, \ldots$ y $l_1,\ l_2, \ldots$ el número de componentes de los *array* en cada dimensión.
- El tipo de datos (INTEGER, REAL, ...) que contiene el *array* se indica de la misma forma que para las restantes variables: de forma explícita ó implícita.
- NOTA: Se recomienda utilizar esta forma de dimensionamiento dinámico sólo en el programa principal.

▶ SUBPROGRAMAS:

- La utilización de subprogramas permite disponer de módulos de cálculo separados del programa principal a los que se puede acceder desde cualquier parte del mismo.
- Una vez realizadas sus operaciones devuelven el control al programa principal
- Los distintos módulos pueden, a su vez, llamarse entre sí unos a otros
- Permiten realizar una programación modular y estructurada en la que es posible que en el programa principal sólo figuren las llamadas a subprogramas que realicen las operaciones correspondientes $\rightarrow$ programas mucho más sencillos
- Cada módulo se puede compilar por separado (cada uno contiene un END) que se conecta al programa principal durante el linkado
- La transferencia de información entre el subprograma y el programa principal es fundamental. (OJO: las variables en FORTRAN indican la posición en memoria donde se almacenan)

- Esquema básico de programa.



Lenguaje Fortran (XXVII)

- FUNCIONES:

  ♡ Son subprogramas elementales que prepara el usuario para evaluar funciones

  Tipo FUNCTION nombre_funcion (lista de argumentos)
  Lista de 'COMMON'
  Lista de 'DIMENSION' y 'ALLOCATABLE'
  ...Instrucciones
  RETURN
  END

  ♡ Definición

  -Tipo = INTEGER, REAL, ... Si no se especifica depende de la 1ª letra del nombre

  -nombre_funcion = nombre de la función

  -Lista de argumentos = (opcional) nombre de las variables transferidas mediante argumentos separados por comas

  -Lista de common = (opcional) nombre de las variables transferidas mediante bloques common (se verá más adelante)

  -Instrucciones = Instrucciones que realiza la función. Debe aparecer al menos 1 vez el nombre de la función como variable

  -RETURN = devuelve el control a la instrucción de llamada.

- FUNCIONES:

  ♡ Llamada desde el programa principal

  Es una asignación directa del tipo:

  variable = nombre_funcion (lista de argumentos)

  donde:

  -variable = almacena el valor que devuelve la función

  -Lista de argumentos: lista de variables separadas por comas que se envían a la función. Deben coincidir en número, tipo y orden de aparición. El nombre puede ser distinto

Lenguaje Fortran (XXIX)

▷ Ej. Función que calcula el módulo de un vector de m componentes (m $\leq$ 10)

Programa ppal.

```
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension v(10)
n=10
do i=1,n
  v(i)=1.d+00
enddo
vmod=vecmod(v,n)
end
```

Función

```
function vecmod(w,m)
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension w(m)
vecmod=0.d+00
do i=1,m
  vecmod=vecmod+w(i)*w(i)
enddo
vecmod=sqrt(vecmod)
return
end
```

♡ Las variables transmitidas como argumentos ocupan la misma posición de memoria que en el programa principal.

♡ Por lo tanto, si se modifican dentro de la función se modifican para los restantes módulos y operaciones.

♡ Si no se indican en la lista de argumentos, las variables son locales para la función y aunque tengan el mismo nombre corresponden a posiciones de memoria distintas.

♡ Al finalizar la ejecución de la función, las variables locales de la función desaparecen.

- SUBRUTINAS:

Subprogramas que permiten devolver al programa principal no sólo el valor de una función sino varios conjuntos de resultados

SUBROUTINE nombre(lista de argumentos separados por comas)
lista de "dimension"

... Instrucciones

RETURN
END

Las variables transmitidas como argumentos ocupan la misma posición de memoria que en el programa principal y son globales para todo el programa.

Las variables definidas dentro de la subrutina no transmitidas como argumento son locales y se eliminan al salir de la misma

OJO: El nombre de la subrutina no puede aparecer como variable

Llamada desde el programa principal:

CALL nombre (lista de argumentos)	(argumentos de entrada y de salida)

Lenguaje Fortran (XXXI)

▷ Ejemplo de programa con subrutina:

Programa ppal.

```
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension v(10)
n=10
do i=1,n
  v(i)=1.d+00
enddo
call vmod(v,n,vmodulo)
end
```

Subrutina

```
subroutine vmod(w,m,wmod)
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension w(m)
wmod=0.d+00
do i=1,m
  wmod=wmod+w(i)*w(i)
enddo
if (wmod.eq.(0.d+00))stop
wmod=sqrt(wmod)
do i=1,m
  w(i)=w(i)/wmod
enddo
return
end
```

Lenguaje Fortran (XXXII)

- Bloque COMMON:

  Se suelen colocar al inicio de la declaración de variables

  COMMON /nombre/ lista_de_variables

  - En todos los módulos en los que aparezca declarado el bloque COMMON de nombre dado, las variables de la lista (que deben coincidir en tipo) ocuparán la misma posición de memoria
  - Suelen combinarse con las listas de argumentos de subrutinas o funciones
  - El bloque COMMON realiza de forma implícita el DIMENSION de las variables de la lista.

- Ejemplo:

Programa ppal.

```
implicit real*8(a-h,o-z)
common /vector/ v(10),n
implicit integer*4(i-n)
n=10
do i=1,n
  v(i)=1.d+00
enddo
call vmod(vmodulo)
end
```

Subrutina

```
subroutine vmod(wmod)
common /vector/ w(10),m
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
wmod=0.d+00
do i=1,m
  wmod=wmod+w(i)*w(i)
enddo
wmod=sqrt(wmod)
return
end
```

Lenguaje Fortran (XXXIV)

- Bibliografía:

  - *Fortran 77 for engineers and scientists with an introduction to Fortran 90*, Larry Nyhoff y Sandford Leestma, Prentice Hall, Upper Saddle River, NJ, USA, 1996
  - *Aprenda Fortran 8.0 como si estuviera en primero*, Javier García de Jalón, Franciso de Asís de Ribera, E.T.S. Ingenieros Industriales, Universidad Politécnica de Madrid, 2005