Introducción

- Características del lenguaje Fortran
  - → Lenguaje para el tratamiento de datos numéricos
  - → Poca variedad de tipos de datos
  - → No permite recursividad
  - → Eficiente
- Elementos básicos de Fortran
  - → Constantes
  - → Variables
  - → Instrucciones
    - Aritméticas y lógicas
    - De control
    - De entrada y salida
  - → Funciones
- Hoja de codificación Fortran

Columnas

1	2 5	6	7 72	73 80
C/*			Comentario	
Nº	Nº		Instrucción	Ignoradas
		car	Continuación	Ignoradas

Constantes

- Cadena cuyo valor no puede cambiar durante la ejecución del programa.
- Tipos de constantes
  - → Enteras

```
-95675
```

  - → Reales (hasta 7 cifras significativas)

```
+17.0
-15E-8
```

  - → Doble precisión (hasta 17 cifras significativas)

```
-17.8D+7
```

  - → Complejas

```
(2.0, -56.78)
```

  - → Lógicas

```
.TRUE.
.FALSE.
```

  - → Alfanuméricas

```
'cadena ejemplo'
```

- Constantes simbólicas o parámetros:
  - → Pueden utilizarse identificadores para las constantes
  - → Sintaxis

PARAMETER (nombre$_1$=valor$_1$, nombre$_2$=valor$_2$, ...)

  - → Ejemplo

```
PARAMETER (PI=3.141592, CADENA='HOLA')
```

Variables

- Identificador que se utiliza para representar cierto tipo de información
  - → Nombre de la variable: Identificador que representa la información (hasta 6 caracteres alfanuméricos, el primero alfabético)
  - → Valor de la variable: Información que se almacena bajo el nombre de la variable
  - → Tipo de la variable: Tipo del dato que representa la variable: real, entero, carácter
- Con las variables podemos
  - → Declararlas: definir el nombre y el tipo
  - → Asignarles valor inicial, con la instrucción DATA
  - → Asignarles un valor en ejecución: Cambiar su valor
  - → Operar con la información que guardan
- Ejemplo

```
PROGRAM ejemplo
   CHARACTER caract
   INTEGER ent1, ent2
   DATA caract, ent1 /'A',1/, ent2 /3/

   ent1 = 3
   ent2 = 5 * ent1
   caract = 'a'
END
```


Tipos de datos simples

- Indican el tipo de información que se puede almacenar en una variable
- Los tipos simples disponibles en Fortran son:

Identificador de tipo	*Significado*	*Espacio estándar en memoria*
CHARACTER	carácter	4 bytes
INTEGER	entero	4 bytes
REAL	real	4 bytes
DOUBLE PRECISION	real con doble precisión	8 bytes
COMPLEX	complejo	8 bytes
LOGICAL	lógico	4 bytes

- Existen otros tipos más complejos que veremos más adelante
- La declaración de variables puede ser:
  - → Implícita (No recomendable. Sólo para enteros y reales): la letra inicial del nombre de la variable indica el tipo:
    - i, j, k, l, m, n: Variables enteras
    - otras: Variables reales
  - → Explícita: indicando el tipo. Por ejemplo:
    - INTEGER edad
    - DOUBLE PRECISION radio

Operadores aritméticos

- Componentes léxicos para operar con variables y constantes numéricas y crear expresiones aritméticas
- Operadores binarios

Símbolo	Significado	Ejemplo	Resultado
+	suma	result = 3 + 5	result ← 8
-	resta	result = 3 – 5	result ← -2
*	multiplicación	result = 3 * 5	result ← 15
/	división	result = 15 / 3	result ← 5
**	potencia	result = 10 ** 2	result ← 100

- Cambios de tipos: se producen al operar datos de diferentes tipos.
  - → entero ***op*** entero → entero
  - → entero ***op*** real → real
  - → real ***op*** entero → real
  - → real ***op*** real → real
- Operador de asignación

Símbolo	Significado	Ejemplo	Resultado
=	asignación	x = y	x ← y

- Cambios de tipos: cuando el tipo del valor no concuerda con el de la variable, se adapta a ella.

Operadores relacionales y lógicos

- Componentes léxicos para crear expresiones lógicas
- Operadores relacionales

Símbolo	Significado	Ejemplos verdaderos	Ejemplos falsos
.GT. (>)	mayor que	(7 + 4) .GT. 3	(6 / 2) .GT. 20
.GE. (>=)	mayor o igual que	(3*4) .GE. 8	40 .GE. 41
.LT. (<)	menor que	12 .LT. 12*2	9 – 4 .LT. 10 – 8
.LE. (<=)	menor o igual que	24 .LE. 12*2	12 + 4 .LE. 12/4
.EQ. (==)	igual que	5 – 4 .EQ. 6	2 +3 .EQ. 5 + 6
.NE. (<>)	distinto de	5 .NE. 2	4 .NE. 8/2

- Operadores lógicos

Símbolo	Significado	Ejemplos
.AND.	conjunción	(x .GT. 2) .AND (y .LT. 4)
.OR.	disyunción	(x .EQ. 4) .OR. (x .LT. 0)
.NOT.	negación	.NOT. (x .LT. 5)

- Prioridad entre operadores
  1. ()
  2. **
  3. * /
  4. + -
  5. .GT. .GE. .LT. .LE. .EQ. .NE.
  6. .NOT.
  7. .AND.
  8. .OR.

  → Cuando hay conflicto: de izquierda a derecha

Instrucciones o sentencias

- Instrucción

  Componente léxico que indica al computador que efectúe una acción

- Tipos de instrucciones
  - → **Instrucción de expresión**: Una expresión aritmética o lógica provoca su propia evaluación
  - → **Instrucción de control**: Instrucciones utilizadas para cambiar el flujo de control o hilo de ejecución del programa: bucles, bifurcaciones, saltos ...
  - → **Instrucción secuencial**: Conjunto de instrucciones simples separadas por saltos de línea. Se ejecutan secuencialmente, una tras otra.
- Instrucciones de inicio y fin de un programa

PROGRAM nombre
...
END

- Ejemplo

```
PROGRAM circulo
REAL radio, area, PI
PI = 3.14159
radio = 10
area = PI * (radio ** 2)
END
```


Instrucción de salto: go to

- Son instrucciones fuera de uso y deben evitarse
- Instrucción de salto incondicional
  - → Permite realizar un salto directo a cualquier instrucción del programa
  - → La instrucción a la que se realiza el salto debe estar etiquetada (debe estar numerada)
- Instrucción de salto calculada
  - → Permite realizar un salto condicionado por el valor de una variable o expresión
  - → Las instrucciones a las que se realiza el salto deben estar etiquetadas (deben estar numeradas)
- Sintaxis

GOTO n	***GOTO (n_1, n_2, ...), i***
Saltar a la instrucción número ***n***	*Saltar a la instrucción número* ***n_1*** *si* ***i*** *vale 1, a* ***n_2*** *si* ***i*** *vale 2...* ***i:*** *variable o expresión entera*

- Ejemplo

```
      GO TO (1, 10), i
    1 z = 3.1416 / radio
      GO TO 100
   10 Z = 2.718 / radio
  100 ...
```

Instrucción condicional: if-else

- Instrucción condicional aritmética
  - → Evalúa una expresión aritmética entre paréntesis
  - → Realiza saltos a diferentes instrucciones dependiendo de si la expresión es menor, igual o mayor que cero.

IF (expr_arit) n_1, n_2, n_3	*Si la expresión es < 0 salta a **n_1**, si es 0 a **n_2** y si es > 0 a **n_3***

- Instrucción condicional lógica simple
  - → Evalúa una expresión lógica entre paréntesis
  - → Si la expresión es verdadera, se realiza una instrucción y, si no, se continúa

IF (expr_log) instrucción	*Si se cumple la expresión se ejecuta la instrucción. Si no, no se ejecuta.*

- Instrucción condicional lógica compuesta
  - → Evalúa una expresión lógica entre paréntesis
  - → Si la expresión es verdadera, se realiza una o varias instrucciones y, si no, se realizan otra u otras

IF (expr_log) THEN ***instrucción(es) 1*** ***[ELSE*** ***instrucción(es) 2]*** ***END IF***	*Si se cumple la expresión se ejecuta instrucción(es) 1 [Si no, se ejecuta instrucción(es) 2]*

Instrucción condicional múltiple

- Instrucción condicional anidada
  - → Evalúa una expresión aritmética entre paréntesis
  - → Permite anidar varias instrucciones condicionales.

IF (expr_log) THEN ***instrucción(es) 1*** ***ELSEIF (expr_log2) THEN*** ***instrucción(es) 2*** ***…*** ***ELSE*** ***instrucción(es) n*** ***END IF***	*Si se cumple la expresión se ejecuta instrucción(es)1 Si no, si se cumple la segunda expresión se ejecuta instrucción(es) 2* *…* *Si no, se ejecuta instrucción(es) n*

- Instrucción condicional múltiple
  - → Evalúa una expresión lógica, entera o de carácter entre paréntesis.
  - → Dependiendo del valor de la expresión, se realiza una acción u otra.

SELECT CASE (expr) ***CASE (valor 1)*** ***instrucción(es) 1*** ***…*** ***CASE DEFAULT*** ***instrucción(es) n*** ***END SELECT***	*Si la expresión toma el valor 1, se ejecuta instrucción(es)* *…* *En el resto de casos se ejecuta instrucción(es) n*

Instrucción repetitiva: do

- Instrucción repetitiva o bucle **do**
  - → Ejecuta iterativamente un bloque de instrucciones teniendo en cuenta tres valores
    - El primero especifica el valor inicial de un índice
    - El segundo el valor final del índice
    - El tercero el incremento del índice en cada iteración (si se omite vale 1)
  - → Se puede terminar con una etiqueta y **CONTINUE**, o con **ENDDO** (recomendable).
- Sintaxis

DO [n], i = m_1, m_2, [m_3]
...
[n CONTINUE]
ENDDO

*Las instrucciones se ejecutan para un valor inicial de **i** = m_1, hasta que **i** vale m_2, incrementándose en m_3. **n** es una constante entera positiva que indica la etiqueta de la última instrucción del bucle.*

- Ejemplo

```
DO j = 1, 3, 1
    a = b + c
    d = a + 1
ENDDO
...
```


Instrucción repetitiva: while

- Instrucción repetitiva o bucle while
  - → Ejecuta iterativamente un bloque de instrucciones mientras se cumpla una condición
- Sintaxis

DO WHILE (expresión_lógica)
instrucción(es)
...
ENDDO

Las instrucciones se ejecutan interiores al bucle se ejecutan mientras se cumpla la condición marcada por la expresión lógica

- Ejemplo

```
DO WHILE (b + c <= 7)
    a = b + c
    d = a + 1
ENDDO
...
```

Vectores y matrices

- Vector y matrices (también llamados arrays o conjuntos)

  Estructura homogénea de datos, de tamaño constante, a cuyos elementos se accede mediante un identificador común y un índice

- Características de un vector
  - → Todos los elementos son del mismo tipo
  - → Cada elemento es a su vez una variable
  - → El número de ellos no varía durante la ejecución
  - → Se accede a los elementos mediante un identificador -nombre del vector- y uno o varios índices que indican su posición
  - → Los índices son consecutivos → deben ser de tipo entero
  - → El número de índices necesarios se denomina dimensión
  - → El número de elementos se denomina rango

- Ejemplo: dimensión 1, rango n

23	12	11	45	....	37	Elementos
1	2	3	4	...	n	Indices

Vectores y matrices

- Elementos de la declaración
  - → Tipo de los elementos del vector
  - → Nombre del vector
  - → Dimensión y rangos

- Sintaxis
  - → Versión completa

DIMENSION nombre (rango$_1$, rango$_2$, …, rango$_n$)
tipo nombre

  - → Versión abreviada

tipo nombre (rango$_1$, rango$_2$, …, rango$_n$)

  - → Los rangos pueden ser:
    - Un entero ***m***: los índices van de ***0*** a ***m***
    - Un par ***inferior:superior***: los índices van de ***inferior*** a ***superior***
  - → La dimensión del vector es ***n***

- Acceso a un elemento

nombre (índice$_1$, índice$_2$, …, índice$_n$)

Vectores y matrices

- Ejemplo

```
PROGRAM ejemplo

  DIMENSION A(10), B(10), C(10,15)
  DIMENSION D(2:12), E(10, -1:20)
  REAL A, E
  INTEGER B, C, D

  INTEGER F(10:30), G(-5, 8)
  COMPLEX H(4)
  ...
  DO j = 1, 10, 1
    A(j) = j;
    B(j) = A(j)+2;
    D(j+2) = 0.0
  ENDDO
  ...
  DO i = 1, 10, 1
    DO 20 k = -1, 20, 1
      E(i,k) = 0
    ENDDO
  ENDDO
  ...

END
```


Cadenas de caracteres

- Son vectores de caracteres que se manejan de forma especial, como cadenas
- Sintaxis

CHARACTER*n nombre
*Define una variable de nombre **nombre**, de tipo cadena de caracteres con tamaño **n***

- Ejemplo

```
CHARACTER*12 cadena
```

- Operaciones con cadenas de caracteres
  - → Asignación: puede realizarse directamente

```
cadena = 'Ejemplo'
```

  - → Acceso y asignación de partes de la cadena (los elementos no pueden coincidir)

```
cadena(3) = 'h'
cadena(3:5) = 'ijk'
cadena(:7) = 'ejemplo'
cadena(1:3) = cadena(5:7)
cadena(1:3) = cadena(2:4)
```

  - → Concatenación: operador / /

```
cadena = 'ejemplo' // ' ' // '1'
```

Instrucciones de E/S

- Las instrucciones de entrada/salida pueden ser:
  - → Sin formato: el ordenador elige un formato predeterminado
  - → Con formato: el formato lo elige el programador
- Entrada/salida sin formato
  - → Instrucciones READ y PRINT
  - → Puede leerse o escribirse una lista de variables de cualquier tipo
  - → Lectura

READ *, lista_variables
Lee de teclado una lista de valores y los asigna en orden a las variables de la lista

  - → Escritura

PRINT *, lista_variables
Escribe en orden en pantalla los valores de la lista de variables

- Ejemplo

```
READ *, ent, cad, vector(3), matr(2:4,3:6)
PRINT *, ent, cad, vector(4), matr(1:4,2:5)
```


Instrucciones de E/S

- Entrada/salida con formato
  - → El programador define el formato
  - → Instrucciones `READ`, `PRINT` y `WRITE`
  - → Las variables de la lista son de cualquier tipo
  - → Lectura

READ formato, lista_variables
*Leer lista de variables. El **formato** puede ser una etiqueta de línea (en la línea aparecerá una sentencia FORMAT) o una cadena con el formato*

  - → Escritura

PRINT formato, lista_variables ***WRITE (u, formato), lista_variables)***
*Escribir la lista de variables. El **formato** puede ser una etiqueta de línea (en la línea aparecerá FORMAT) o una cadena con el formato. **u** es el dispositivo (1: terminal, 2: impresora)*

  - → Formato

etiqueta FORMAT (formato)
*Escribir la lista de variables. El **formato** puede ser una etiqueta de línea (en la línea aparecerá una sentencia FORMAT) o una cadena con el formato*

Instrucciones de E/S

- Códigos de formato para datos

Código	*Tipo*	*Descripción*
In	Entero	n: nº máximo de dígitos
Fn.d	Real	n: nº total de dígitos d: nº de dígitos de la parte decimal
En.d	Real con exponente	n: nº total de dígitos d: nº de dígitos de la mantisa
Ln	Lógico	n: nº de posiciones
An	Cadena	n: nº de caracteres

- Códigos de posicionamiento

Código	*Descripción*
nX	Saltar n caracteres
Tn	Tabular hasta la posición n
TRn	Tabular n caracteres a la derecha de posición actual
TLn	Tabular n caracteres a la izquierda de posición actual
/	Salto de línea
'literal'	Escribir un literal

- Ejemplos

```
10   FORMAT ('X:',I3,'Y:I3)
20   FORMAT (A10,/,T4,F5.2)
     PRINT 10, 2, 3
     PRINT 20, 'Valor', 3.2
     PRINT '(I2)', 3
```

- Salida

```
X:__2Y:__3
Valor_____
______3.20
_3
```

(_: blancos)

Otras instrucciones

- Instrucción de parada

STOP n

Terminar la ejecución del programa. ***n****: constante entera o cadena que se visualiza al terminar*

- Instrucción de pausa

PAUSE n

Detener temporalmentela ejecución del programa. ***n****: constante entera o cadena que se visualiza al parar*

- Instrucción de asignación de etiqueta

ASSIGN n TO m

Asignar a una variable ***m*** *una etiqueta* ***n.*** ***n*** *puede utilizarse con* `GO TO`*, ...*

Programación modular

- Un módulo o procedimiento es un segmento de programa que realiza tareas bien definidas.
- Divide las tareas grandes de computación en otras más pequeñas.
- Viene definido por un nombre y una lista de argumentos
- Los procedimientos pueden ser de dos tipos:
  - → ***Funciones:*** Al nombre se le asigna un valor de salida
  - → ***Subrutinas:*** Al nombre no se le asigna valor.
- Durante la ejecución del programa principal, para llamar a un módulo se especifica su nombre y la lista de valores para los argumentos
- Al ejecutar el programa, cuando el flujo llega a la llamada a un módulo, el control se transfiere a éste, se ejecuta su código para los valores concretos de los argumentos y se devuelve el control a la línea siguiente a la de llamada
- Ventajas de la utilización de procedimientos:
  - → ***Modularidad***: dividir las tareas en otras más pequeñas
  - → ***Reutilización***: utilizar código ya escrito

Funciones

- Tipos de funciones
  - → Intrínsecas
  - → De sentencia
  - → Externas
- Funciones intrínsecas: son las del sistema

Función	***Descripción***
INT	Parte entera
REAL	Convertir a real
DBLE	Convertir a doble precisión
CMPLX	Convertir a complejo
ICHAR	Obtener código ASCII
CHAR	Obtener carácter a partir de ASCII
ABS	Valor absoluto
MOD	Resto de la división entera
LEN	Longitud de una cadena
SQRT	Raíz cuadrada
EXP	Exponencial
LOG	Logaritmo neperiano
LOG10	Logaritmo decimal
SIN, COS, TAN	Seno, coseno, tangente
ASIN, ACOS, ATAN	Arcoseno, arcococoseno, arcotangente
SINH, COSH, TANH	Seno, coseno, tangente hiperbólicos
MAX	Máximo
MIN	Mínimo

Funciones de sentencia

- Son funciones que se escriben en una única sentencia
- Se incluyen en un módulo y son locales a dicho módulo (sólo pueden utilizarse en él)
- Sintaxis

nombre* (arg_1, arg_2, …, arg_n) = *expresión

*Define una función de nombre **nombre**, con los argumentos **arg_1**, **arg_2**, …**arg_n**, y cuyo resultado se obtiene al evaluar la expresión **expresión***

- Para llamarlas, se utiliza el nombre y los argumentos
- Se puede alterar el tipo del valor que devuelve
- Ejemplo

```
INTEGER SUMA
SUMA (X, Y, Z) = X + Y + Z + 3.0
...
RESULTADO = SUMA (3, 4, 5)
```


Funciones externas

- Definición de funciones externas (definidas por el usuario)
  - → Palabra clave FUNCTION
  - → Nombre de la función (identificador)
  - → Tipo del dato que devuelve la función
  - → Lista de argumentos con sus tipos de datos correspondientes
- Sintaxis

***[tipo] FUNCTION nombre* (arg_1, arg_2, …, arg_n)**
***tipo* arg_i**
…
***nombre* = …**
arg_i = …
…
RETURN
END

*Define una función de nombre **nombre**, con los argumentos **arg_1**, **arg_2**, …**arg_n**. En el cuerpo de la función debe asignarse el valor de salida al nombre. También pueden alterarse los argumentos durante la ejecución. Los tipos de la función y de los argumentos son opcionales (aunque es recomendable ponerlos)*

Funciones externas

- Llamada a la función

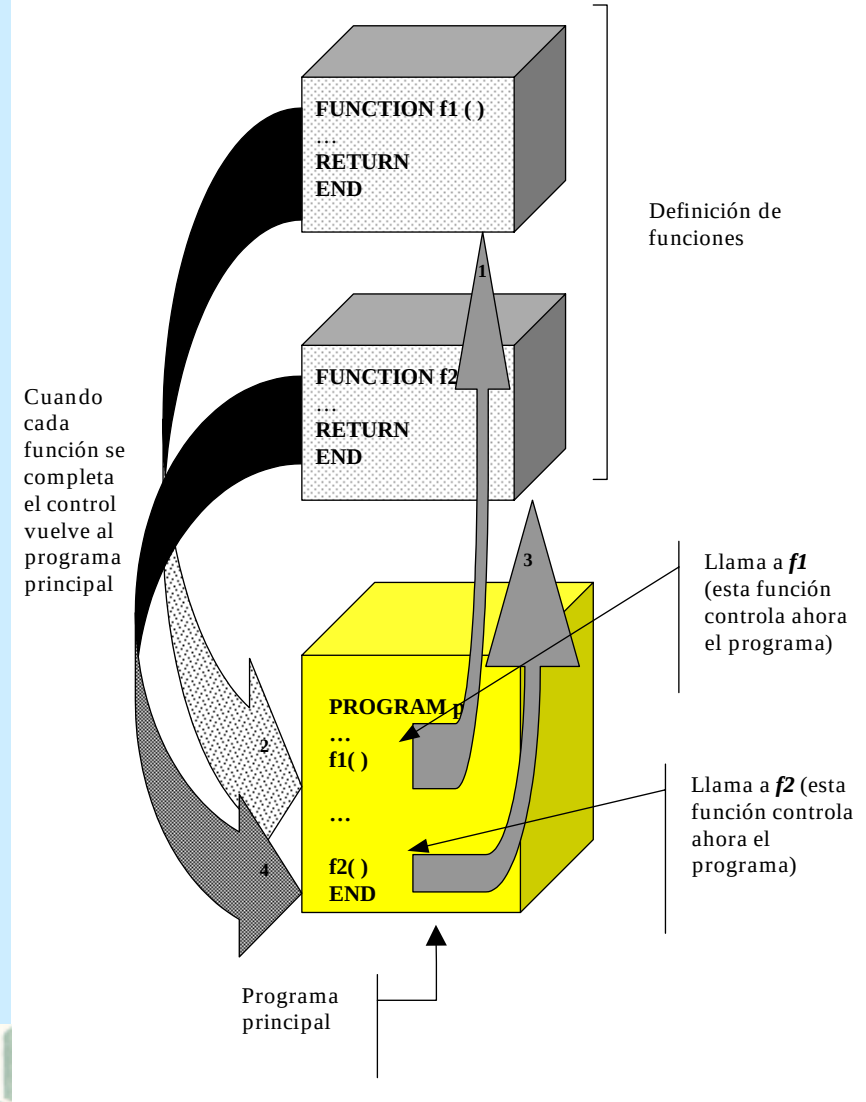
nombre (valor$_1$, valor$_2$, …, valor$_n$)
Llama a la función ***nombre.*** *Los argumentos toman los valores* ***valor$_1$****,* ***valor$_2$****, …****valor$_n$***

- Ejemplo

```
REAL FUNCTION parabola (x)
REAL x
parabola = A*x**2 + B*x + C
RETURN
END

PROGRAM parabola
...
y = parabola (3.5);
...
END
```


Llamadas a funciones



Subrutinas

- Similar a las funciones, pero no devuelven un valor en el nombre
- Definición de subrutinas
  - → Palabra clave SUBROUTINE
  - → Nombre de la subrutina (identificador)
  - → Lista de argumentos con sus tipos de datos correspondientes
- Sintaxis

SUBROUTINE nombre (arg$_1$, arg$_2$, …, arg$_n$)
tipo arg$_i$
…
***arg$_i$* = ...**
…
RETURN
END

*Define una subrutina de nombre **nombre**, con los argumentos **arg$_1$**, **arg$_2$**, …**arg$_n$**. En el cuerpo de la función pueden alterarse los argumentos durante la ejecución. Los tipos de los argumentos son opcionales (aunque es recomendable ponerlos)*

Subrutinas

- Llamada a la subrutina

CALL nombre (valor$_1$, valor$_2$, …, valor$_n$)

*Llama a la subrutina **nombre.** Los argumentos toman los valores **valor$_1$**, **valor$_2$**, …**valor$_n$***

- La transferencia del flujo del programa en las rutinas es igual que en las funciones
- Ejemplo

```
SUBROUTINE misub (a, b)
REAL a
INTEGER b
a = a + b
b = 0
RETURN
END

PROGRAM ejemplo
...
CALL misub (3.5, 7);
...
END
```

Argumentos de funciones y subrutinas

- Argumentos o parámetros formales

  Nombres que aparecen en la lista de argumentos de la definición. Dentro del módulo (función o subrutina) se tratan como cualquier variable

- Argumentos o parámetros reales

  Valor que toman los argumentos formales en la llamada. Pueden ser variables, constantes o funciones y deben coincidir en número, orden y tipo con los formales

- Ambito
  - → El ámbito de una variable es la parte del programa en que es conocida
  - → Una variable definida en un módulo es **local** a ese módulo (sólo es conocida en ese módulo). Podemos hacerlas globales a todas las instancias del módulo con la instrucción `SAVE(lista_var)`
  - → Los argumentos son **locales** al módulo
  - → Una variable definida en el programa principal es ***local*** a éste. Para que sea conocida en los módulos debe pasarse como argumento
- Efectos colaterales
  - → El paso de argumentos es siempre por referencia (salvo para constantes), por lo que los cambios en los argumentos formales repercuten en los reales y producen efectos colaterales.

Funciones como argumentos

- Las funciones pueden ser argumentos de otras funciones o de subrutinas
- Para ello deben especificarse en el programa principal con una de las siguientes órdenes
- Para funciones externas

EXTERNAL función$_1$, función$_2$, …, función$_n$

- Para funciones intrínsecas

INTRINSIC función$_1$, función$_2$, …, función$_n$

- Algunas funciones intrínsecas no pueden utilizarse como argumentos:
  - INT
  - FLOAT
  - REAL
  - DBLE
  - CMPLX
  - ICHAR
  - CHAR
  - MAX
  - MIN

Vectores y cadenas como argumentos

- Es posible utilizar vectores y cadenas como argumentos en funciones
- El número de elementos del argumento formal debe ser menor o igual que el del argumento real
- Los índices pueden no coincidir. En ese caso se asigna el 1º al 1º, el 2º al 2º, …
- Si algún rango no se conoce, puede sustituirse por un asterisco (*)
- Un argumento puede ser índice de otro argumento tipo vector
- Ejemplo

```
SUBROUTINE misub (a, b, c, d)
REAL a(3)
INTEGER b(2:*,*:*), d, c(d)
...
RETURN
END

PROGRAM ejemplo
REAL aa(3)
INTEGER bb(1:3, 5:8)
INTEGER dd, cc(5)
...
CALL misub (aa, bb, cc, dd)
END
```


Ejemplos

- Cálculo del factorial

```
FUNCTION factorial (n)
   INTEGER n
   INTEGER i, fact

   fact = 1
   DO i = 1, n
      fact = fact*i
   ENDDO
   factorial = fact
   RETURN
END

PROGRAM calcular_factorial
   INTEGER n, f
   ...
   f = factorial (n)
   ...
END
```

Ejemplos

- Ordenación de un vector por selección

```
SUBROUTINE orden_sel (v, n)
   REAL v(n)
   INTEGER n
   INTEGER pos_min, i, j
   REAL aux

   DO i = 1, n-1
      pos_min = i
      DO j = i+1, n
         IF (v(j) .LT. v(pos_min))
*             pos_min = j
      ENDDO
      aux = v(i)
      v(i) = v(pos_min)
      v(pos_min) = aux
      ENDDO
   RETURN
END

PROGRAM ordenacion_seleccion
   REAL v(7)
   ...
   CALL orden_sel (v, 7)
   ...
END
```


Ejemplos

- Ordenación de un vector por inserción

```
SUBROUTINE orden_ins (v, n)
   REAL v(n)
   INTEGER n
   INTEGER i, j
   REAL elemento

   DO i = 2, n
      j = i-1
      elemento = v(i)
      DO WHILE (j>0 .AND. Elemento<v[j])
         v(j+1) = v(j)
         j = j-1
      ENDDO
      v(j+1) = elemento
   ENDDO
   RETURN
END

PROGRAM ordenacion_insercion
   REAL v(7)
   ...
   CALL orden_ins (v, 7)
   ...
END
```

Ejemplos

- Sucesión de Fibonacci

```
FUNCTION fibonacci (n)
   INTEGER fibonacci
   INTEGER n
   INTEGER i, j, k

   i = 1
   j = 0
   DO k = 1, n
      j = i+j
      i = j-i
   ENDDO
   fibonacci = j
   RETURN
END

PROGRAM sucesion_fibonacci
   INTEGER n, fib
   ...
   fib = fibonacci (n)
   ...
END
```


Ejemplos

- Multiplicación de matrices

```
SUBROUTINE mult_matr (A,B,C,m,n,p)
   REAL A(m,n), B(n,p), C(m,p)
   INTEGER m, n, p
   INTEGER i, j, k

  DO i = 1, m
    DO j = 1, n
    C(i,j) = 0.0
      DO k = 1, p
       C(i,j)=C(i,j)+A(i,k)*B(k,j)
      ENDDO
    ENDDO
  ENDDO
  RETURN
END

PROGRAM multiplicacion_matrices
   REAL A(5,7), B(7,3), C(5,3)
   ...
   Mult_matr (A, B, C, 5, 7, 3)
   ...
END
```