Lenguaje FORTRAN. Estructuras y sentencias básicas

Programación en Fortran

Valentín Moreno

ÍNDICE

1. INTRODUCCIÓN A FORTRAN

- FORmula TRANslator (Traductor de Fórmulas).
- Es un lenguaje de alto nivel, compilado y linkado
- Se crea a mediados de los años cincuenta
- Nace para cálculos científicos
- 1966
  - primera estandarización por el ANSI, quedando fijadas las reglas del lenguaje y recibiendo el nombre de FORTRAN 66
- 1977
  - nueva versión FORTRAN77
- 1990
  - FORTRAN se adapta a las nuevas tecnologías, con el nuevo estándar elaborado por ANSI y aparece el FORTRAN90

- Alfabeto de Fortran
  - Alfabeto = símbolos que utiliza
  - Letras
    - exceptuando ñ y letras con tilde, aunque se pueden usar en las líneas de comentarios
  - Números
  - Caracteres especiales
  - Se usa el símbolo ampersand (&) como signo de continuación línea
  - Si una línea acaba con &, significa que continúa en la siguiente
  - Los comentarios empiezan con !

Carácter	Nombre
	Espacio
=	Signo igual
+	Signo más
-	Signo menos
*	Asterisco
/	Slash
(	Paréntesis de apertura
)	Paréntesis de cierre
,	Coma
.	Punto
' o "	Apóstrofe o comillas
&	Continuación línea
!	Comentario

- SENTENCIAS
  - Tipos de sentencias
    - Ejecutables
      - especifican acciones
      - modifican contenidos, alteran el orden de ejecución, ..
    - No ejecutables
      - describen características
      - asocian tipos, asignan nombres, ….

2. ESTRUCTURA DE UN PROGRAMA EN FORTRAN

- La estructura de un programa en Fotran es:

```
PROGRAM nombre_programa
    Sentencias de especificación y declaración...
    ...
    Sentencias ejecutables ... ...
END PROGRAM nombre_programa
```

- El código del programa aparece entre las palabras reservadas **PROGRAM y END PROGRAM**

- Excepción: código correspondiente a funciones y procedimientos

- PROGRAM
  - Todo programa principal en FORTRAN debe comenzar con PROGRAM
  - Nombre del programa
    - Ha de empezar por una letra
    - Puede contener números y letras
    - NO puede contener espacios
    - Su longitud máxima es de 31 caracteres
    - NO puede coincidir con ningún nombre de variable que se use en el programa
- END PROGRAM
  - indica al compilador que ya no hay más instrucciones

3. VARIABLES Y CONSTANTES

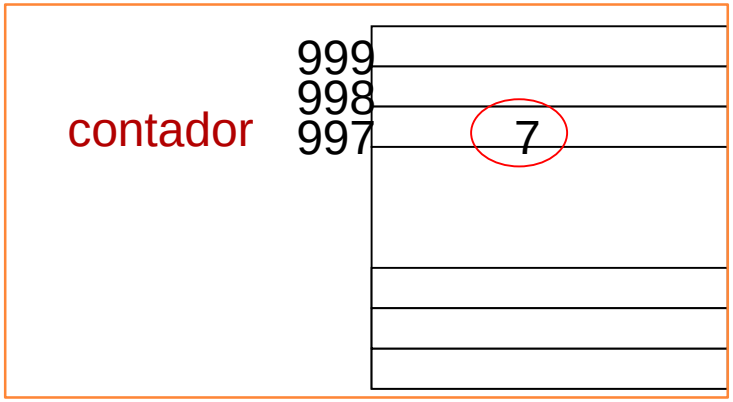
- Las CONSTANTES designan un valor específico y determinado que se define al escribir un programa y que no cambia a lo largo del mismo
- Se pueden definir de dos maneras, utilizando siempre la palabra PARAMETER
  - Definiendo primero el tipo de dato y a continuación asignando el valor a la constante (método estándar)

```
INTEGER max
PARAMETER (max=100)
```

  - Asignando directamente el valor a la constante (no aceptado por todos los compiladores)

```
PARAMETER pi=3.141592
```

- Las VARIABLES designan valores que irán cambiando a lo largo de la ejecución de un programa y que han de ser definidas (declaradas) para reservarles un espacio en memoria
- Una variable representa un campo o dirección en la memoria
  - cuando se usa el nombre de la variable se utiliza el valor que se encuentre almacenado en ese momento en dicha dirección



- Declaración de variables
  - Necesitamos reservar espacio en la memoria para guardar datos
  - La variable tiene asociado un tipo de datos, que determinará cuánto espacio se guarda en memoria para almacenarlos
    - integer, real, character, logical
  - Una variable sólo puede definirse una vez en el programa
- Sintaxis para declaración de variables

```
tipo identificador
```

  - Ejemplos

```
INTEGER contador
REAL temperatura
LOGICAL aprobado
```

- Reglas para asignar nombres válidos a las variables:
  - El primer carácter debe ser alfabético
  - El resto de caracteres deben ser letras o números
  - NO pueden contener espacios
  - El número máximo de caracteres depende del compilador
    - nombres válidos: MAX2, XR2D2, etc.
    - nombres no válidos: 321, 3PO, etc.
  - Es aconsejable elegir nombres autoexplicativos

```
contador = contador + 1
num_alumnos = 56
```

- Dos formas de definir una variable en el programa
  - Implícita
    - No declaro la variable
    - Directamente uso su nombre en una sentencia de asignación (le asigno un valor)
  - Explícita
    - Declaro la variable antes de usarla, con una sentencia de declaración

- Definición implícita de variables
  - Los tipos de datos están asociados por defecto
    - toda variable cuyo nombre comience por I, J, K, L, M, N, se asocia al tipo entero
    - el resto de variables se consideran reales
  - Hacen difícil detectar los errores
- Definición explícita de variables
  - Antes de la sección de declaraciones se añade la sentencia

    IMPLICIT NONE
  - Si el compilador encuentra un identificador que no conoce da un error en tiempo de compilación

4. TIPOS DE DATOS SIMPLES

Tipo de dato	Palabra reservada
Enteros	`INTEGER`
Lógicos	`LOGICAL`
Carácter	`CHARACTER`
Complejos	`COMPLEX`
Reales	`REAL`
Reales doble precisión	`DOUBLE PRECISION`

- **Tipo INTEGER**
  - Variables y constantes enteras
  - Son almacenados en memoria sin parte decimal
  - Valores validos:
    - número (sucesión de dígitos) sin decimales (sin punto decimal), precedido o no del signo + o -
    - Ejemplos

      +123 -65002 99999 0

- El rango de valores depende de la máquina y del compilador

Tipo INTEGER (cont)

- En nuestro compilador pueden definirse intervalos distintos
  - Por defecto está definida INTEGER*4
    - 4 bytes
    - Máximo valor
      - En 1 byte $255 = 2^{(8-1)}-1$
      - En 4 bytes $2^{(32-1)} -1$

	Rango
INTEGER*1	-128 … 127
INTEGER*2	-32,768...32767
INTEGER*4	-2,147,483,648 … 2,147,483,647
INTEGER*8	-9,223,372,036,854,775,808 … 9,223,372,036,854,775,807

Tipo REAL

- Variables y constantes reales : números con parte fraccionaria
- Se pueden escribir de dos formas:
  - a) reales sin exponente:

    `+3.2` `-0.676` `4.` (no es el entero 4)
  - b) reales con exponente (notación científica)
    - Tres partes:
      - base numérica (siempre con punto decimal),
      - base de potencia (decimal, representada por E)
      - exponente (tipo entero, sin punto decimal).

        `-23.044E+4` `0.23E-2` `0.022E2`
- Como en el caso de los enteros, el rango (valores máximo y mínimo) dependen de la máquina y del compilador

- **Tipo CHARACTER**
  - Variables y constantes de tipo carácter
  - Una constante alfanumérica es una cadena de caracteres encerrada entre apóstrofes
  - El número de caracteres que pueden contener, varía entre 1 y el máximo permitido por el compilador.
  - Para definir palabras de más de un carácter es necesario indicar el número de caracteres a continuación del la palabra reservada CHARACTER:

```
CHARACTER(20) nombre
(palabra de 20 caracteres como máximo)
```

- **Tipo LOGICAL**
  - Variables y constantes lógicas
  - Sólo pueden tomar dos valores, cierto y falso
  - Estos valores se representan en Fortran como

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

5. OPERADORES Y EXPRESIONES

- **OPERADORES ARITMÉTICOS**

OPERADOR	OPERACIÓN
+	suma
-	resta
*	multiplicación
/	división
**	potencia

- Reglas para construir expresiones
  - No puede haber dos operadores consecutivos

    X**-2 es incorrecto

    X**(-2) es correcto
  - Los operadores *, /, **, deben estar rodeados por dos variables o constantes.

    A*B D/3.0 F**3
  - Los operadores +, -, pueden afectar a dos operandos (suma, resta) o a una (signo)

    -5.0 +C-B
  - La multiplicación nunca está implícita

    ha de escribirse A*(B+C) en lugar de A(B+C)

- Los operadores aritméticos pueden usarse con operandos reales o enteros
  - Las operaciones con enteros, dan resultado entero
    - se usan sobre todo para operaciones con contadores o índices
  - Las operaciones con reales dan resultado real
    - los cálculos se hacen siempre con reales
  - En una expresión en la que haya número reales el resultado es real, aunque también participen enteros (el tipo real domina)

OPERADORES RELACIONALES

OPERADOR (Dos formas de escribirlo admitidas por F90)		OPERACIÓN
==	.EQ.	Igual a
/=	.NE.	Distinto de (No igual a)
>	.GT.	Mayor que
>=	.GE.	Mayor o igual que
<	.LT.	Menor que
<=	.LE.	Menor o igual que

- **OPERADORES ALFANUMÉRICOS**
  - Operandos: caracteres y cadenas
  - Resultado: cadena

OPERADOR	OPERACIÓN
//	Concatenación

- **OPERADORES LÓGICOS**
  - Operandos: lógicos
  - Resultado: lógico

OPERADOR	OPERACIÓN
.OR.	O lógico
.AND.	Y lógico
.NOT.	Negación lógica

- Los operadores en una expresión se evalúan siguiendo el **orden jerárquico** siguiente:
  - 1.º Paréntesis (comenzando por los más internos).
  - 2.º Signo (-,+ siempre que no tengan a su izquierda un valor numérico).
  - Luego los aritméticos:
    - 3.º Potencia **
    - 4.º Productos y divisiones. Con igual prioridad. Se resuelven de izquierda a derecha.
    - 5.º Sumas y restas. Con igual prioridad. Se resuelven de izquierda a derecha.
  - 6.º Concatenación.
  - Luego los relacionales:
    - 7.º Relacionales.
  - Luego los booleanos
    - 8.º Negación .NOT.
    - 9.º Conjunción .AND.
    - 10.º Disyunción .OR.

6. TIPOS DE SENTENCIAS

- Sentencias de procedimiento
  - PROGRAM, END, FUNCTION, SUBROUTINE
- Sentencias declarativas
  - Declaración de variables y constantes
- Sentencias de asignación
  - Asigna un valor a una variable (=)
- Sentencias de control
  - Cambian el flujo del programa
  - IF, CASE, DO
- Sentencias de entrada y salida
  - Entrada y salida de datos
  - READ, PRINT

- PROGRAM
  - Cualquier programa principal en FORTRAN debe comenzar con PROGRAM

```
PROGRAM nombre_programa
```

- END PROGRAM
  - indica al compilador que ya no hay más instrucciones

```
END PROGRAM nombre_programa
```

- SUBROUTINE, FUNCTION
  - Usadas para definir funciones y subrutinas (lo veremos mas adelante)

- No ejecutables
- Indican al compilador el nombre y tipos de datos que va a almacenar una variable o constante
- Normalmente se colocan al principio del programa
  - Pueden aparecer en cualquier orden, pero nosotros vamos a declarar primero las constantes y a continuación las variables

```
! Constantes
      PARAMETER  nombre_constante=valor
! Variables
      TIPO  nombre_variable
```

- Las asignaciones se hacen con el símbolo ”=“
- Tipos de asignación
  - Inicialización de variables (Inicialización = asignación valor inicial)

```
a=0
b=1.5
nombre = “sin definir”
validado = .FALSE.
```

  - Asignación del valor de una operación a una variable

```
contador = contador + 1
resultado = (a<=7)
```

- Entrada
  - asignamos a una variable un valor leído de un periférico (típicamente teclado) o de un archivo

READ*, lista_de_variables

- Salida
  - escribimos en un periférico (típicamente monitor) o archivo el valor de una variable
  - sólo para escribir en la salida estándar (monitor)

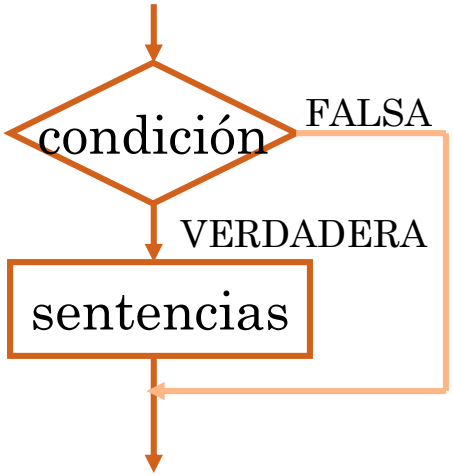
PRINT *, var1, var2,.., varN
PRINT *, “Paula”

- Estructuras de control
  - Programación estructurada: sólo puedo usar unas estructuras de control determinadas
    - Secuencial
    - Alternativas (IF)
    - Repetitivas (bucle)

- Sentencias de control alternativas
  - También llamadas de selección o condicionales
  - IF
    - Simple:
      - IF-THEN-END IF
    - Dobles
      - IF-THEN-ELSE- END IF
    - Múltiples
      - IF-THEN-ELSE IF-THEN-ELSE- END IF
  - SELECT CASE

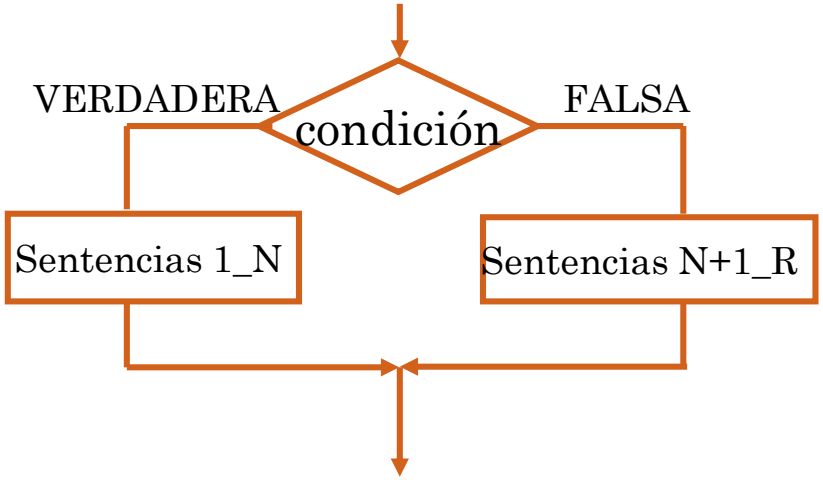
- IF
  - Se evalúa la expresión lógica. Si es verdadera se ejecuta la sentencia, si es falsa se continua con la siguiente instrucción
  - La expresión lógica debe ir entre paréntesis y puede estar compuesta por varias condiciones unidas por operadores lógicos
  - Simples

```
IF (condición) THEN
        sentencia_1
        ...
        sentencia_N
END IF
```



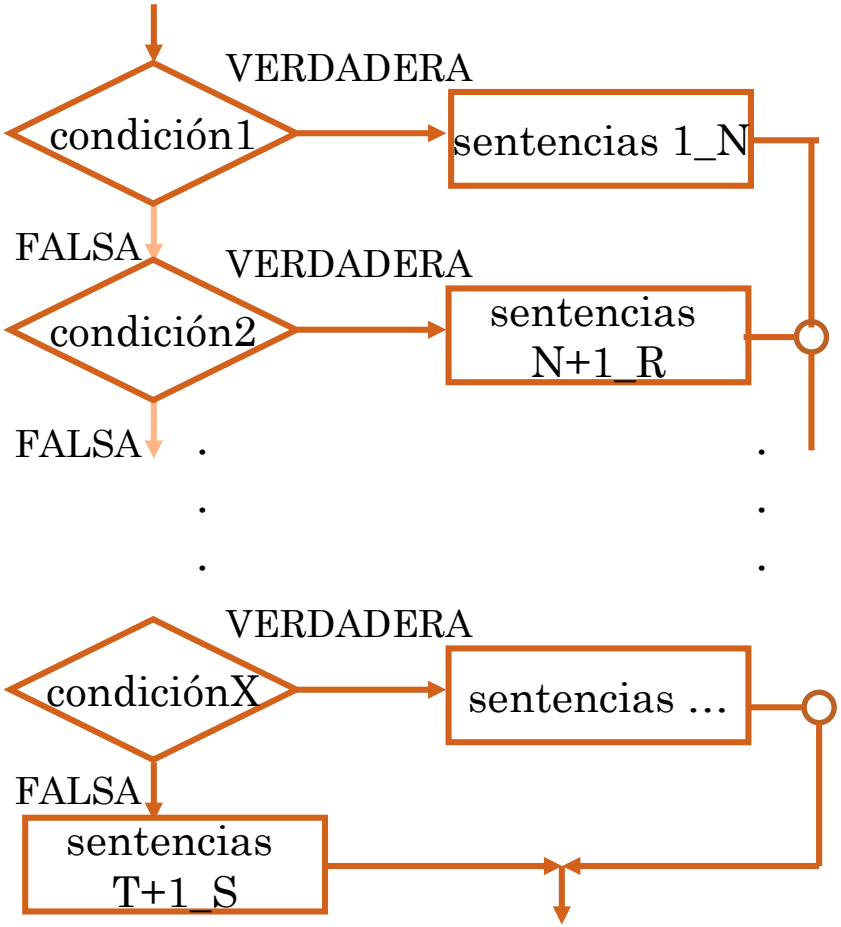
- IF (cont..)
  - Dobles

```
IF (condición) THEN
        sentencia_1
        ...
        sentencia_N
ELSE
        sentencia_N+1
        ...
        sentencia_R
END IF
```



- IF (cont..)
  - Múltiples

```
IF (condición1) THEN
   sentencia_1
   ....
   sentencia_N
ELSE IF (condición2) THEN
    sentencia_N+1
   ...
   sentencia_R
ELSE IF (condición3) THEN
 ...
ELSE
   sentencia_T+1
   ...
   sentencia_S
END IF
```



- CASE
  - Permite implementar estructuras de selección múltiple de forma sencilla, a partir de una variable selectora
  - El selector
    - Debe ser una variable o expresión de tipo entera, lógica, carácter
    - No puede ser una expresión real
  - Los valores de la lista
    - pueden sustituirse por expresiones indicando rangos

      ```
      valor_inferior:valor_superior
      ```

    - y también por listas de valores

      ```
      valor1,valor2,valor3
      ```

    - o por combinaciones de las dos cosas

      ```
      valor1,valor2,valor3:valor4
      ```

- CASE (cont.)

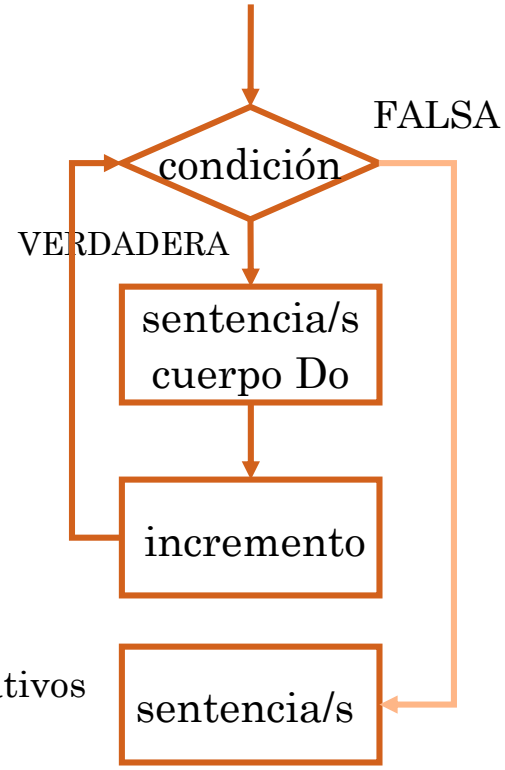
```
SELECT CASE (selector)
  CASE (valor1)
        bloque_sentencias_1
  CASE (valor2:valor_n)
        bloque_sentencias_2
  CASE (valor_r)
        bloque_sentencias_N
  .........
  CASE DEFAULT
        bloque_sentencias_R
END SELECT
```

- Sentencias de control repetitivas
  - También llamadas BUCLES
  - DO
  - DO WHILE

- DO
  - *'para'* en pseudocódigo
  - Se usa si se conoce el número de veces que ha de ejecutarse el bucle
  - Sintaxis

```
DO variable_control = valor_inicial, valor_final, incremento
   sentencia_1
   sentencia_2
   .....
    sentencia_n
END DO
```

  - donde
    - variable_control es una variable de tipo entero
    - valor_inicial, valor_final son valores o variables enteras tales que si el incremento:
      - es positivo valor_inicial ≤ valor_final
      - es negativo valor_inicial ≥ valor_final
    - Incremento:
      - puede ser cualquier valor entero, también negativos
      - si no se especifica, se considera que vale 1



- **DO**
  - *Ejemplo:*
    - *Sumar los 20 primeros números naturales*
      - *Bucle DO*
        - *Con incremento positivo +1*

```
DO i = 1, 20
    suma = suma + i
END DO
```

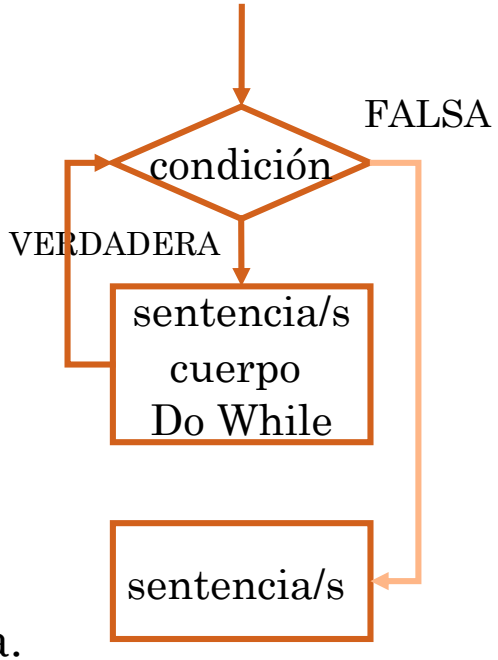
        - *Con incremento negativo -1*

```
DO i = 20, 1, -1
    suma = suma + i
END DO
```

DO WHILE

- *'mientras'* en pseudocódigo
- Se usa si no se conoce de antemano el número de veces que hay que ejecutar el bucle
- Sintaxis

```
DO WHILE (condición)
    sentencia_1
    sentencia_2
    .....
    sentencia_n
END DO
```



- Condición
  - es la negación de la condición de parada.
- Las sentencias en algún momento deben modificar la condición de forma que esta sea falsa para salir del bucle.

DO WHILE

- *Ejemplo:*
  - *Sumar números introducidos por teclado hasta introducir el 0*
    - *Condición de parada "numero = =0"*
    - *Bucle DO WHILE*

```
DO WHILE (numero/=0)
    suma = suma + numero
    READ *, numero
END DO
```