Capítulo 1

Los elementos del lenguaje

1.1 Elementos básicos

1.1.1 Juego de caracteres

Caracteres alfanuméricos Las 26 letras del alfabeto inglés y los 10 dígitos decimales:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
```

Caracteres especiales Los siguientes caracteres especiales:

```
+ * - / _ < > = ( ) ' " & $ % ! ? . , ; : espacio_blanco
```

1.1.2 Símbolos léxicos

Nombres o identificadores Identifican elementos de un programa. Pueden ser nombres de programas, nombres de variables, nombres de funciones, ... Pueden estar formados sólo por caracteres alfanuméricos y el signo _ (underline o subrayado), aunque deben necesariamente comenzar por una letra.

Palabras clave: Son palabras reservadas propias del lenguaje y representan acciones (órdenes) o declaraciones específicas. Por ejemplo, `READ` para ordenar la escritura de algo, `IF` para comenzar un bloque condicional, `PROGRAM` para declarar el comienzo de un programa, etc.

Operadores: Representan operaciones entre datos. Por ejemplo `+ **`

Delimitadores: Acotan parte de una instrucción. Por ejemplo `! ( )`

Etiquetas de instrucción: Son identificadores de una instrucción: números que se le ponen a algunas instrucciones para referirse a ellas en determinadas circunstancias.

1.1.3 Formato del código fuente FORTRAN

- FORTRAN no distingue entre mayúsculas y minúsculas.
- FORTRAN90 tiene dos formas posibles de escribir el código fuente: libre y fijo. El formato fijo, con normas muy rígidas, proviene del FORTRAN antiguo, en el que los sistemas de control de flujo del programa eran un tanto primitivos y, además, los programas se escribían en tarjetas perforadas. El formato libre de escritura del código, propio del FORTRAN90 es mucho más flexible. Por diversos motivos es conveniente (aunque no obligatorio) escribir el código en una formato compatible para ambos. Para ello hay que respetar las siguientes reglas:
  - Se deben escribir las instrucciones entre las columnas 7 y 72.
  - Se deben escribir las etiquetas en las columnas 1 a 5.
  - Los comentarios se preceden del símbolo `!`, **pero no se debe colocar este símbolo en la columna 6**.
  - Cuando una instrucción se escribe en más de una línea, se debe poner el símbolo `&` en la columna 73 de una línea y en la columna 6 de la línea siguiente para indicar que la segunda es continuación de la primera.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
      a partir de aqui se escriben las instrucciones
      por ejemplo:
      a=3.*angle       ! esto es un comentario
! esto es tambien un comentario
      si una instruccion ocupa mas de una linea                          &
     &se indica como en estas dos
```

1.2 Tipos de datos y forma de escribir las constantes

Los tipos de datos básicos que se pueden manipular en un programa FORTRAN son los que se describen a continuación. Hay otros, que no se mencionan aquí por evitar complejidades innecesarias por el momento.

INTEGER Este tipo de dato sirve para representar en el ordenador números enteros. Las constantes de tipo entero se escriben mediante los dígitos de su representación decimal, precedidos por el signo, que es opcional si el número es positivo: `(signo)ddddddddddd`.

```
243     +2     -15
```

REAL simple precisión Para representar números reales con precisión estándar. Las constantes de tipo real simple precisión se pueden escribir de las dos maneras siguientes:

notación decimal es la notación habitual, con punto decimal: `(+/-)ccccccc.dddddd`

notación científica o exponencial se escribe el número en notación decimal acompañado de una indicación sobre la potencia de 10 por la que va multiplicado:
`(+/-)cc.dddddE(+/-)ee`

```
-15.36    6.    -1e3(=-1000.)    33.E-9(=0.000000033)
```

REAL doble precisión Para representar números reales con precisión superior. Las constantes reales doble precisión se escriben en su notación cientítica, sustituyendo la `E` por una `D`: `(+/-)ccccccc.ddddddD(+/-)ee`

```
1.0D25    -.345677788D-3
```

COMPLEX simple precisión Para representar números complejos en simple precisión (también existe en doble precisión). Las constantes complejas simple precisión se escriben como un par de números reales, entre paréntesis y separados por una coma:
`(parte real,parte imaginaria)`

```
(-123.456,1.003E+2)
```

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

```
'abcdefghijklm'    "nopqrstuvwxyz"

(-123.456,1.003E+2)      'El resultado es:'
"Hola! : esto es un texto"
```

LOGICAL Para representar los valores lógicos, **verdadero** y **falso**:

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

1.3 Variables

Una variable es un nombre simbólico que representa una (o varias) direcciones de memoria en la que se almacena un dato (o varios), cuyo valor puede cambiar durante la ejecución del programa.

En FORTRAN es necesario especificar el TIPO de dato que va a contener una variable antes de usarla: **integer**, **real s.p.**, **character**, etc.

Para ello hay que **declararlas**: al principio de un programa hay que especificar el tipo de cada una de las variables que se van a utilizar. En FORTRAN hay dos formas de hacerlo:

1.3.1 Declaración explícita del tipo de las variables

Se declara **explícitamente** el tipo de cada variable, mediante alguna(s) de las siguientes instrucciones:

Declaración explícita de tipo entero

```
INTEGER variable(s)
INTEGER :: variable(s)
INTEGER(KIND=4) :: variable(s)
```

Todas las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo entero, es decir, todo dato que se almacene en esa variable será codificado como un número entero. Si se mencionan varias variables, hay que separarlas por comas.

```
integer numero1,numero2
integer :: number
```

Declaración explícita de tipo real simple precisión

```
REAL variable(s)
REAL*4 variable(s)
REAL :: variable(s)
REAL*4 :: variable(s)
REAL(KIND=4) :: variable(s)
```

Todas las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo real en simple precisión.

```
real*4 a,b,c
real :: var,x,y
```

Declaración explícita de tipo real doble precisión

```
DOUBLE PRECISION variable(s)
REAL*8 variable(s)
REAL*8 :: variable(s
REAL(KIND=8) :: variable(s)
```

Todas las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo real en doble precisión.

```
real*8 suma, prod
double precision variable,sol
```

Declaración explícita de tipo complejo simple/doble precisión

```
COMPLEX variable(s)
COMPLEX*8 variable(s)
COMPLEX(KIND=4) :: variable(s)
COMPLEX(KIND=8) :: variable(s)
```

Todas las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo complejo en simple/doble precisión.

Declaración explícita de tipo character

```
CHARACTER*n variable(s)
CHARACTER(LEN=n) :: variable(s)
CHARACTER(N) :: variable(s)
```

Las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo cadena de caracteres (**n** es el número de caracteres).

```
character(len=16) nombre,apellido
character*32 nombrefichero
```

Declaración explícita de tipo lógico

```
LOGICAL variable(s)
```

Las instrucciones anteriores sirven para declarar que la variable mencionada en la instrucción va a contener datos de tipo lógico.

```
logical test, log
```

Inicialización de variables en la declaración

Se puede aprovechar una instrucción de declaración de tipo para dar un valor inicial a una variable que luego puede, lógicamente, ser modificado en el curso de la ejecución del programa.

```
real*4 :: xcero=1.2, ycero=-.2e-3
integer :: m=10,n=30
```

Téngase en cuenta que cualquier variable debe ser inicializada, es decir, se le debe dar un valor inicial, antes de que sea usada como "dato", por ejemplo en una expresión. De lo contrario, su valor (contenido de la dirección de memoria que representa) es imprevisible. Así, no se puede suponer que una variable a la que no se ha dado previamente ningún valor valdrá cero: puede valer cualquier cosa.

1.3.2 Declaración implícita del tipo de las variables

En este tipo de declaración se declara el tipo de las variables en función de la PRIMERA LETRA DE SU NOMBRE: es decir, lo que se declara en realidad es que todas las variables cuyo nombre comience por una letra determinada van a ser del tipo que se indique. Se puede declarar, por ejemplo, que todas las variables cuyo nombre comience por `x` son del tipo real doble precisión.

```
IMPLICIT TIPO (letras)
IMPLICIT TIPO (rangos)
```

donde `letras = letras separadas por comas` y `rangos = rangos de letras`. Un `rango` se indica mediante dos letras separadas por un guión, y representa todas las letras del alfabeto entre ambas inclusive. Estas instrucciones declaran que todas las variables cuyos nombre comiencen por la(s) letra(s) que se menciona(n) en la lista o en los rangos serán del tipo indicado (`TIPO`).

```
implicit real*4 (a,b,x)
implicit real*8 (a-h)
implicit integer (i-k,r)
```

En FORTRAN existe una **declaración de tipo implícito por defecto** que hace que, a menos que se declare expresamente (de forma explícita o implícita) el tipo de una variable, esta tiene siempre un **tipo por defecto**, definido por la regla siguiente:

```
IMPLICIT REAL*4 (A-H,O-Z)
IMPLICIT INTEGER (I-N)
```

es decir, **por defecto**, todas las variables son reales simple precisión, excepto aquéllas cuyo nombre comience por `i`, `j`, `k`, `l`, `m` o `n` que son enteras. Obsérvese que las letras asociadas a variables enteras son las que habitualmente se utilizan en Matemáticas para designar los subíndices.

La orden

```
IMPLICIT NONE
```

declara que no hay ningún tipo implícito en el programa, es decir, anula la vigencia del tipo implícito por defecto. Si se usa esta orden en un programa es necesario declarar el tipo de TODAS las variables que en él aparecen.

En un programa, las declaraciones de tipo implícitas deben ir obligatoriamente antes de la declaraciones de tipo explícitas, ya que éstas últimas se sobreponen a las implícitas. Por ejemplo, con las siguientes declaraciones

```
implicit real*4 (z)
integer :: zonzo
```

todas las variables que comiencen por la letra `z` serían de tipo real s.p. excepto `zonzo` que sería entera. (Se recomienda vívamente no hacer este tipo de cosas).

1.3.3 ¿Declaración implícita o explícita?

Usar un tipo de declaración u otro presenta ventajas e inconvenientes:

- Aprovechar los tipos implícitos por defecto de FORTRAN (letras `i-n` para variables enteras y el resto para reales) resulta cómodo para el programador ya que puede cambiar los nombres y añadir nuevas variables sobre la marcha, sin tener que declararlas. Téngase en cuenta que, dado que el lenguaje FORTRAN se utiliza principalmente para realizar cálculos numéricos, la inmensa mayoría de las variables que se usan en los programas son, o bien enteras o bien reales.
- Usar la orden `IMPLICIT NONE` y obligarse así a declarar todas las variables puede disminuir el riesgo de cometer ciertos errores. Además, en muchos lenguajes, como por ejemplo C, es obligatorio declarar todas las variables.

En cualquier caso, es una buena costumbre utilizar nombres que respeten la norma implícita por defecto del FORTRAN, ya que eso facilitará la legibilidad del programa por el mismo u otros programadores.

1.3.4 `PARAMETER`: Constantes con nombre

FORTRAN permite dar un nombre a una constante, es decir, definir un **parámetro**. Es como usar una variable cuyo valor no se puede cambiar. Esto, como ya se verá, tiene utilidad en determinadas circunstancias.

Para declarar un parámetro se usa la orden `PARAMETER`, en alguna de las formas siguientes:

```
TIPO, PARAMETER :: nombre1=valor1, nombre2=valor2
PARAMETER (nombre1=valor1, nombre2=expresion)
```

```
      parameter (pi=3.1415926, e=2.71828)
      integer, parameter :: idias=30
      integer, parameter :: nn=6, nn2=2*nn
```

Como se ve en el ejemplo anterior, se puede usar, para dar valor a una variable `PARAMETER`, una expresión en la que figure el nombre de otra a la que se haya dado valor previamente.

1.4 La instrucción básica: asignación

La instrucción básica en un lenguaje de programación es la de **asignación**: orden más simple para guardar un valor en una variable. En FORTRAN dichas instrucciones son de la forma:

```
VARIABLE = VALOR (constante o expresión)
```

que debe ser interpretada como: guardar el valor `VALOR` (si es preciso, evaluar primero la `expresion`) en la variable `VARIABLE`.

```
a=1.
b=-3.
c=4.
raiz=sqrt(b*b-4*a*c)
```

Las tres primeras instrucciones ordenan guardar los valores (dados por constantes) en las variables (direcciones de memoria) correspondientes. La última es una instrucción de asignación con una **expresión** a la derecha del signo igual. Para ejecutarla primero se evalúa la expresión y luego se almacena el resultado en la dirección de memoria a que hace referencia la variable `raiz`.

Una **expresión** es una combinación de constantes, variables, funciones elementales y operadores que representa una determinada combinación de operaciones entre ellos.

1.4.1 Operadores aritméticos

Potenciación `**`

Multiplicación y división `*` `/`

Suma y resta `+` `-`

Si en una expresión aparecen varios de ellos, el orden de prioridad de los operadores es el de la lista anterior. A igual prioridad, los operadores se ejecutan de izquierda a derecha, salvo las potencias consecutivas, que se evalúan de derecha a izquierda. El orden se puede modificar mediante el uso de paréntesis: si en una expresión hay paréntesis, su interior es lo primero que se evalúa.

Así, por ejemplo,

`2.+3.**(3./2.)`	$\longrightarrow$	$2+3^{3/2}$	`2.+3.**3./2.`	$\longrightarrow$	$2+\dfrac{3^3}{2}$
`1/2*x`	$\longrightarrow$	$\dfrac{1}{2}x$	`1/(2*x)`	$\longrightarrow$	$\dfrac{1}{2x}$
`2.**(1./5**3)`	$\longrightarrow$	$2^{1/5^3}$	`2.**1./5**3`	$\longrightarrow$	$\dfrac{2^1}{5^3}=\dfrac{1}{2^2}$

1.4.2 Tipo de los resultados de operaciones aritméticas

Entre operandos del mismo tipo, las operaciones aritméticas se hacen en ese mismo tipo: si los dos datos son reales simple precisión el resultado será también real simple precisión, si los dos son enteros, el resultado será entero, etc. Si un operador está actuando entre dos operandos de distinto tipo, la operación se hace en la aritmética del tipo "más alto" (entero, real s.p., real d.p., complejo). No se pueden operar datos complejos con datos reales doble precisión.

Lo anterior implica que, por ejemplo, la expresión `2/3` da como resultado `0`(cero), ya que se hace la división entre números enteros, mientras que la expresión `1./4` da como resultado `0.25` al hacerse la operación entre números reales, por ser real uno de los operandos.

Debe tenerse en cuenta que las reglas anteriores se aplican **a cada operador**. Así, por ejemplo, la expresión `4/3*3.` daría como resultado `3.`(real) !!, ya que, en primer lugar se efectuará la operación `4/3` en la aritmética de números enteros al ser enteros los dos operandos, dando como resultado `1`(entero). Posteriormente se evaluará `1*3.` en la aritmética de números reales, por ser real al menos uno de los dos operandos.

1.4.3 Operador de concatenación de caracteres

Es la operación que consiste en "unir" dos cadenas de caracteres, poniendo una a continuación de la otra. En FORTRAN se indica mediante el signo `//`.

```
'im'//'presionante'  --->  'impresionante'
'un'//'a'//' n'//'ana n'//'apol'//'itana'  --->
'una nana napolitana'
```

1.4.4 Las funciones elementales

FORTRAN dispone de una serie de funciones ya programadas (intrínsecas) para evaluar las funciones elementales que se usan en Matemáticas: cos , exp, arctan, etc. La lista siguiente no es exhaustiva, pero incluye las funciones más habituales.

`ABS(X)`	$\|x\|$	entero,real o complejo	el mismo
`SQRT(X)`	$\sqrt{x}$	real o complejo	el mismo
`SIN(X)`	sen (x)	real o complejo	el mismo
`COS(X)`	cos (x)	real o complejo	el mismo
`TAN(X)`	$\tan(x)$	real o complejo	el mismo
`ASIN(X)`	arcsen (x)	real o complejo	el mismo
`ACOS(X)`	$\arccos(x)$	real o complejo	el mismo
`ATAN(X)`	$\arctan(x)$	real o complejo	el mismo
`EXP(X)`	e^x	real o complejo	el mismo
`LOG(X)`	$\ln x$	real o complejo	el mismo
`INT(X)`	Parte entera de x	real	entero
`NINT(X)`	Redondeo al entero más próximo	real	entero
`FLOOR(X)`	Mayor entero $\leq$ x	real	entero
`REAL(X)`	Conversión a real	entero	real
`MAX(X1,...,XN)`	Máximo entre los argumentos	entero o real	el mismo
`MIN(X1,...,XN)`	Mínimo entre los argumentos	entero o real	el mismo
`MOD(X,Y)`	Módulo: `x-INT(x/y)`	entero o real	el mismo

1.5 Leer e imprimir

Las operaciones básicas de entrada/salida de datos pueden hacerse mediante las versiones más simples de las órdenes `READ` y `PRINT`, que se describen a continuación. Más adelante se expondrán con más detalle las órdenes de lectura e impresión generales.

1.5.1 Lectura de datos

Para que un programa reciba, durante su ejecución, datos introducidos por el usuario a través del teclado, puede usarse la orden:

```
READ*, lista de variables separadas por comas
```

Cuando se ejecuta esta orden, el programa se detiene a esperar los datos que se le tecleen, que se irán asignando a las variables de la lista, en el mismo orden. Los datos se teclean separados por comas, espacios en blanco o saltos de línea.

```
character*32 :: nomfich
read*, a
read*, n,x,y
read*, nomfich
```

La ejecución de estas órdenes implican la asignación de valores a cinco variables: `a, n, x, y, nomfich`, en este orden. Salvo pocas excepciones, es obligatorio que los datos tecleados correspondan al tipo de la variable que los almacenará. Así, si en segundo lugar se teclea el valor (real) `4.` se producirá un error y se terminará la ejecución del programa, ya que se esperaba un entero. Por el contrario, en general, si se espera un número real y se encuentra un entero, se convertirá este último a real y se asignará a la variable correspondiente, sin

que se produzca error. Para asignar valor a una variable de caracteres hay que teclear una **constante** de caracteres es decir, una cadena de caracteres entre apóstrofes o comillas dobles, por ejemplo `"mi programa favorito"`.

Esta orden por sí sola no provoca la escritura de ningún mensaje avisando de que se esperan datos. Si se desea uno (recomendable), debe programarse.

1.5.2 Escritura de mensajes y resultados

Para imprimir datos de cualquier tipo en la pantalla del ordenador se puede usar la orden:

```
PRINT*, lista de variables o expresiones separadas por comas
```

Cuando se ejecuta esta orden, se imprimen en la pantalla, en un formato pre-definido por el compilador, los valores de las variables y/o expresiones de la lista, separados por espacios en blanco. Si no caben en una línea, se continuará en la siguiente.

```
character*32 :: nomfich
x=5.34
print*, " Valor de x: ",x," Valor de x**2: ",x*x
print*, " Nombre del fichero ? ::"
read*, nomfich
```

La porción de programa anterior producirá la salida:

```
valor de x:  5.340000 valor de x**2:  28.515600
    Nombre del fichero ?  ::
```

y a continuación hay que teclear el valor que se quiera dar a `nomfich`.

1.6 Detener la ejecución del programa

La ejecución de un programa se termina implícitamemte cuando se terminan sus instrucciones. Pero también se puede forzar explícitamente, cuando se desee. La instrucción

```
STOP
```

en cualquier sitio de un programa, detiene su ejecución. Cuando esto suceda, se obtendrá en la pantalla del ordenador, el siguiente mensaje:

```
STOP - program terminated
```

Como veremos más adelante, un programa puede tener diversos puntos de salida. Para poderlos distinguir, se puede añadir un mensaje a la orden STOP. Así se puede identificar el motivo de la terminación:

```
STOP "Mensaje"
```

1.7 Organización del programa

Nos referimos aquí sólo a la organización de un **programa principal** (main program). Más adelante se verá que hay otros tipos de *unidades* FORTRAN. Pero siempre tiene que haber un programa principal, que es el que actúa como controlador, eventualmente ordenando la ejecución de otros **subprogramas**.

La organización de un programa principal es la siguiente (lo que aparece entre corchetes es opcional):

```
      PROGRAM [nombre]
! Comentarios descriptivos del programa
      instrucciones no ejecutables
      instrucciones ejecutables
      END [PROGRAM [nombre]]
```

Es decir, un programa principal siempre comienza por la palabra-clave `PROGRAM` y a continuación se puede poner un nombre al programa. La ultima instrucción debe, necesariamente, comenzar por la palabra-clave `END`, y puede ser tambien `END PROGRAM`. Si se puso nombre al programa, se puede, también, incluir en la última instrucción: `END PROGRAM nombre`. A continuación de la primera instrucción (`PROGRAM`), es muy recomendable escribir unas lineas de comentarios que describan la función y utilización del programa.

En el *cuerpo* del programa hay que poner, en primer lugar, las *sentencias no ejecutables*, de las que sólo se han visto hasta ahora las **declaraciones de tipo** (`INTEGER, REAL, CHARACTER, ...`), incluyendo la declaración `PARAMETER`.

A continuación se ponen todas las instrucciones ejecutables, que incluyen las ya vistas instrucciones de asignación, de entrada/salida (`READ` y `PRINT`) y también la instrucción `STOP`.

No es obligatorio que un programa contenga una instrucción `STOP`. Si no la contiene, el programa se detendrá cuando se acaben las instrucciones. **Sí** tiene que contener, **siempre**, la palabra-clave `END`.

Es bien conocido que la parte más larga y dificil en la realización de un programa es la llamada depuración del programa, es decir **la corrección de errores**. El orden y un cierto método en la escritura del código ayudarán mucho en esta etapa.