

# **1 TIPOS DE DATOS Y LOS PRIMEROS PASOS: LEER, CALCULAR, ESCRIBIR**

## **1.1 Juego de caracteres Fortran**

- Fortran 90/95 tiene su propio alfabeto especial llamado *juego de caracteres Fortran*. Sólo los caracteres de su alfabeto pueden usarse en este lenguaje. Consta de los siguientes:
- Caracteres alfanuméricos:
  - Caracteres alfabéticos en mayúscula (26): 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.
  - Caracteres alfabéticos en minúscula (26): 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.
  - Caracteres numéricos (10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
  - Carácter de subrayado (1): \_
- Caracteres especiales (22): = + - \* \*\* / ( ) . , \$ ' : " % ; ! & < > ? <blanco>.
- Fortran no distingue entre caracteres alfabéticos en mayúsculas o en minúsculas, excepto cuando forman parte de cadenas de caracteres, como veremos en el capítulo 6.

## **1.2 Estructura de un programa Fortran**

- La estructura general de un programa Fortran, como todas las unidades de programa<sup>3</sup> Fortran, consta de cuatro partes:
- Cabecera:
  - PROGRAM *nombre\_programa*
  - Pequeña descripción del programa.
- Parte de especificaciones:
  - Define las variables e identificadores empleados en el programa.
- Parte ejecutable:

---

<sup>3</sup> Una unidad de programa es un trozo de código Fortran compilado separadamente. Se estudiarán en el capítulo 5.

- Describe las acciones que llevará a cabo el programa.
- Parte de terminación:
  - `END PROGRAM nombre_programa`
- Pueden insertarse comentarios libremente en cualquier punto del programa: dentro, antes o después del mismo.

### 1.3 PROGRAM

- La sentencia `PROGRAM` define el nombre del programa Fortran que comienza la ejecución.
- Sintaxis:

`[PROGRAM nombre_programa]`

- *Nombre\_programa* es el nombre del programa (y su punto de entrada).
- El nombre en una sentencia `PROGRAM` se usa única y exclusivamente con propósitos de documentación del programa.
- Si se utiliza la sentencia `PROGRAM`, deberá ser la primera sentencia no comentada del programa fuente.

### 1.4 STOP

- La sentencia `STOP` detiene la ejecución de un programa, y opcionalmente, imprime un mensaje en la salida estándar de errores.

`STOP [n]`

- *n* es una constante carácter o un entero de hasta 5 dígitos.
- `STOP` termina la ejecución de un programa Fortran, antes de que éste llegue al final de dicho programa. `STOP` también manda un mensaje a la salida estándar de errores si se ha especificado algo después de él (dígitos o cadena de caracteres).
- Un programa Fortran puede tener varias sentencias `STOP` (es decir, varios puntos de parada), por ello aunque el uso de *n* no es obligatorio, es conveniente, ya que nos dará una idea clara del punto en que ha parado el programa.
- Cuando la sentencia `STOP` precede inmediatamente a la sentencia `END PROGRAM` es opcional. El compilador genera automáticamente un comando `STOP` cuando alcanza la sentencia `END PROGRAM`.
- Entendiendo que un buen programa Fortran debe tener un único punto de entrada y otro de salida, sin ningún otro punto de parada intermedio, el uso de esta sentencia está desaconsejado.

## 1.5 Ejemplo de uso de STOP

CÓDIGO	DESCRIPCIÓN
STOP 7373	SE ESCRIBE "STOP 7373" EN LA SALIDA DE ERRORES
STOP 'ACABO'	SE ESCRIBE "STOP ACABO" EN LA SALIDA DE ERRORES
STOP	NO SE ESCRIBE NADA EN LA SALIDA DE ERRORES

Tabla 1.1: Ejemplo de uso de STOP

## 1.6 END PROGRAM

- Debe ser la última sentencia del programa.
- Esta sentencia indica al compilador que no hay más sentencias que compilar en el programa.
- Sintaxis:

**END PROGRAM** [nombre\_programa]

- Ej:

```
PROGRAM ejemplo
```

```
.....
```

```
.....
```

```
END PROGRAM ejemplo
```

O bien:

```
!Programa sin nombre
```

```
.....
```

```
.....
```

```
END PROGRAM
```

## 1.7 Formato de las líneas en Fortran 90/95

- Al contrario que en el 77, Fortran 90/95 es mucho más flexible en el formato de líneas:
  - Una línea puede tener hasta 132 caracteres.
  - Una línea puede tener más de una sentencia, separadas por “,”.

- Si una sentencia continúa en la línea siguiente, se debe terminar la línea con el carácter ampersand “&” y, opcionalmente, también se puede empezar la línea siguiente. Se permiten hasta 39 líneas de continuación.
- Si una cadena de caracteres debe continuar en una línea adicional, se debe poner el carácter “&” al final de cada línea y al principio de las líneas de continuación.
- Los caracteres que siguen al “!” hasta el final de la línea son comentarios.
- Si una sentencia requiere una etiqueta, ésta debe ser numérica (1...99999), debe preceder a la sentencia y estar separada por, al menos, un blanco.
- Los espacios en blanco son ignorados en Fortran 90/95, excepto dentro de las palabras clave, nombres de las funciones intrínsecas y cadenas de caracteres. Así, se pueden usar tantos espacios en blanco e intercalar tantas líneas en blanco en un programa como se quiera para mejorar el aspecto del mismo.

## 1.8 Tipos de datos

- Hay 5 tipos de datos predefinidos llamados *intrínsecos* para constantes y variables Fortran.
- Cada uno de estos tipos de datos intrínsecos admiten varias longitudes de memoria, dependiendo de la anchura de la palabra del computador, conocidas como *clases*. Una de esas clases, la más habitual, se le llama clase por defecto.

TIPO	DENOMINACIÓN EN FORTRAN	EN BYTES OCUPADOS <sup>4</sup>	INTERVALO DE VALORES
ENTERO	INTEGER	4	$-2^{32-1}$ a $-1$ 2147483648 $2^{32-1}$ a $1$ 1=2147483647

---

<sup>4</sup> Depende del computador.

REAL	REAL	4	-3.402823 10 <sup>+39</sup> -1.175495 10 <sup>-39</sup> Y 1.175495 10 <sup>-39</sup> 3.402823 10 <sup>+39</sup>
COMPLEJO	COMPLEX	8	IGUAL QUE REAL
LÓGICO	LOGICAL	4	.TRUE. O .FALSE.
CARÁCTER	CHARACTER [ {(len=n° caract de variab)  (n° caract de variab)} ]	len	CONJUNTO DE CARACTERES ASCII DE 8-BITS

**Tabla 1.2: Tipos de datos intrínsecos en Fortran**

- Suponiendo enteros de 4 Bytes, cualquier intento de usar un valor fuera del rango de representación dado en la Tabla 1.2, es decir, mayor o menor que sus valores límite, resulta en un llamado error de *overflow*.
- Para solucionar esta limitación, en parte, se incluye el tipo de dato real. Los números reales se caracterizan por dos cantidades: el *rango* que marca la diferencia entre los números mayor y menor que pueden representarse y la *precisión* o número de dígitos significativos que puede conservar un número. Para reales de 4 Bytes, el rango es aproximadamente de 10<sup>-38</sup> a 10<sup>38</sup> y la precisión de 7 dígitos. Así, 1000000.0 y 1000000.1 no pueden distinguirse en un programa Fortran en el que ambos son definidos de tipo real de 4 Bytes. Más adelante se aprenderá a usar reales de alta precisión.
- En cuanto a las variables declaradas de tipo carácter, su longitud viene dada por el número máximo de caracteres que van a almacenar. Si no aparece el paréntesis en la declaración de tipo, las variables de la lista sólo podrán almacenar un carácter.
- Además, Fortran 90/95 permite al programador definir tipos de datos *derivados*, que son tipos de datos especiales para resolver problemas particulares.

## 1.9 Constantes en Fortran

- Definición de constante:
  - Valor específico y determinado que se define al hacer un programa y que no cambia a lo largo del mismo.

- Cuando un compilador Fortran encuentra una constante, coloca el valor de la constante en una localización conocida de la memoria y hace referencia a esa localización cada vez que se usa la constante en el programa.

### 1.9.1 Constantes enteras

- Puede tomar únicamente un valor entero (positivo, negativo o cero). Se representa como un signo (opcional) seguido de una cadena no vacía de números.
- Ej:  
39, -6743, +8899, 0, -89, 0012, 27, 3.

### 1.9.2 Constantes reales

- Básica.

Se compone de:

- signo (opcional).
- parte entera (secuencia de dígitos).
- punto decimal.
- parte fraccional (secuencia de dígitos).
- Ej: 12.343 -0.0032 86. .3475.
- Básica con exponente.

El exponente real se compone de:

- carácter alfabético E.
- signo (opcional).
- constante entera (2 dígitos como máximo).
- Ej:  
+34.453E4  
12.323E+03  
-0.0034E-03  
.89E-2  
5.434E0  
-4344.49E-5

### 1.9.3 Constantes lógicas

- Una constante lógica puede tener únicamente los valores verdadero y falso.
  - .TRUE. Verdadero.
  - .FALSE. Falso.

### 1.9.4 Constantes complejas

- Una constante compleja representa un número complejo como un par ordenado de números reales. El primer número del par representa la parte real y el segundo la parte imaginaria. Cada parte vendrá codificada como un número real.
- Ej: (3.E34,0.332) (-3.329,-.3E9).

### 1.9.5 Constantes carácter

- Consiste en una cadena de caracteres.
- El carácter blanco es válido y significativo dentro de la cadena.
- La longitud de la constante carácter coincide con el número de caracteres que componen la cadena.
- Cada carácter de la cadena tiene una posición concreta que es numerada consecutivamente (comenzando en 1). El número indica la posición del carácter dentro de la cadena comenzando por la izquierda.
- Para codificar una constante carácter dentro de un programa deberá encerrarse entre dos caracteres delimitadores comilla simple o dobles comillas. La longitud de la cadena deberá ser mayor que 0.
  - Los delimitadores no forman parte de la constante en: ‘Hola que tal’ “hasta luego Lucas”.
  - Para que la constante carácter incluya comillas simples (dobles) rodearla de dobles (simples) comillas: “’Hola que tal’” “‘hasta luego Lucas’”.

### 1.10 Identificadores

- Un identificador es un nombre que se usa para denotar programas, algunas constantes, variables y otras entidades.
- Los identificadores deben empezar con una letra y pueden tener hasta 31 letras, dígitos o caracteres de subrayado \_.
  - Por ejemplo, identificadores válidos son:
    - Masa, MASA, Velocidad\_de\_la\_luz, Sueldo\_del\_ultimo\_mes, X007.
  - Y no válidos:
    - R2-D2, 34JULio, pepe\$.

### 1.11 Variables

- Una variable es una entidad que tiene un nombre y un tipo. Un nombre de variable es un nombre simbólico de un dato. Por tanto, ese dato podrá ser identificado, definido y referenciado a través de dicha variable.

- Cuando un compilador Fortran encuentra una variable, reserva espacio en memoria para esa variable de modo que, cada vez que se use la variable en el programa, se hace referencia a su ubicación en memoria.
- El nombre de las variables debe ser un identificador válido.
- El tipo de una variable puede venir determinado de forma explícita o implícita.

### 1.11.1 Declaración explícita

#### TIPO:: lista de variables

- TIPO es cualquiera de los tipos de datos Fortran válidos de la Tabla 1.2.
- *lista de variables* es un conjunto de variables separadas por comas cuyos nombres son identificadores válidos.

- Ej:

REAL:: radio, area

INTEGER:: mesas,sillas

COMPLEX:: z1,z2

LOGICAL:: testea

CHARACTER (len=20):: alumno1,alumno2

### 1.11.2 Declaración implícita

- Si una variable no ha sido definida explícitamente por una sentencia de definición de tipo, éste será determinado por la primera letra de su nombre:
  - I,J,K,L,M,N: entera.
  - resto de letras: real.
- Existe la posibilidad de gobernar los tipos implícitos. La sintaxis general es:

#### IMPLICIT {NONE | TIPO (lista de letras)}

- NONE significa que no hay nada implícito. Por lo tanto, todas las variables del programa deben declararse explícitamente.
- TIPO es cualquiera de los tipos de datos Fortran válidos de la Tabla 1.2.
- *lista de letras* es un conjunto de letras que corresponden a las iniciales de los nombres de las variables.
  - Si no son consecutivas en el alfabeto, van separadas por comas.
  - Si son consecutivas en el alfabeto, *letra\_inicial-letra\_final*.

- La declaración implícita puede ser una fuente de problemas. En su lugar, se recomienda usar la sentencia `IMPLICIT NONE` y declarar de forma explícita todas las variables y constantes que se vayan a emplear.

- Ej:

`IMPLICIT NONE`

!No hay nada implícito.

`IMPLICIT LOGICAL (H-J)`

!Variables que comiencen por H,I,J son lógicas.

## 1.12 Inicialización de variables

- A partir de Fortran 90, las variables pueden inicializarse al ser declaradas.

**TIPO:: var1=valor1[,var2=valor2]...**

- Ej: `REAL:: velocidad=3.25,aceleracion=0.75,espacio=10.9`
- Ej: `CHARACTER (len=10):: apell1,apell2,nombre='Santiago'`
- El valor de una variable no inicializada no está definido por el estándar de Fortran 90/95.
  - Algunos compiladores automáticamente cargan un cero en esa variable,
  - otros cargan cualquier valor existente en la posición de memoria de esa variable y
  - otros incluso causan un error de ejecución al usar una variable que no ha sido previamente inicializada.

## 1.13 Constantes con nombre: PARAMETER

- Cuando en un programa se utilizan constantes conocidas, ya sean: físicas como la aceleración de la gravedad o la constante de Planck, matemáticas como el número  $\pi$  o el número  $e$ , químicas como el número de Avogadro etc., es deseable escribirlas siempre igual y con la máxima precisión que acepte el computador.
- La mejor manera de conseguir consistencia y precisión en un programa en cuanto al uso de constantes conocidas es asignarles un nombre y usar ese nombre siempre para referirse a la tal constante a lo largo del programa.
- Las constantes con nombre se crean usando el atributo `PARAMETER` de una sentencia de declaración de tipo. La forma general de una sentencia de este estilo es:

**TIPO, PARAMETER:: nombre1 = valor1[, nombre2 = valor2]...**

- Permite dar un nombre simbólico a una expresión constante que será invariable en todo el programa
- En tiempo de compilación, se hace una sustitución del *nombre* por el *valor*.
- Cualquier intento de cambiar su valor con una sentencia de asignación o de lectura provoca un error de compilación.
- Ej:  
REAL, PARAMETER:: PI = 3.14159, E = 2.71828

## 1.14 Expresiones aritméticas

- Los operadores aritméticos son:

OPERADOR	DESCRIPCIÓN
OPERADORES BINARIOS	
+	SUMA
-	RESTA
*	MULTIPLICACIÓN
/	DIVISIÓN
**	POTENCIA
OPERADORES UNARIOS	
+	SIGNO POSITIVO
-	SIGNO NEGATIVO

**Tabla 1.3: Operadores aritméticos Fortran**

- No se pueden escribir dos operadores adyacentes.
- Ej:  
La expresión  $a^{**}-b$  es ilegal en Fortran y debe ser escrita como  $a^{**}(-b)$ .

### 1.14.1 Reglas de precedencia de operadores aritméticos

OPERADOR(ES)	PRECEDENCIA
( )	MAYOR
**	

+ , - (UNARIOS: SIGNOS)	
* , /	
+ , -	MENOR

**Tabla 1.4: Reglas de precedencia de operadores aritméticos**

- Si una expresión contiene dos o más operadores de la misma precedencia se siguen las siguientes reglas:
  - Cuando existen paréntesis anidados se evalúan desde el más interno hasta el más externo. Es conveniente usar tantos paréntesis como sean necesarios en una expresión para hacerla más clara y fácil de comprender.
  - Las operaciones de potencia se evalúan de derecha a izquierda.
  - Multiplicación/división y suma/resta se evalúan de izquierda a derecha.

### 1.14.2 Ejemplo de reglas de precedencia

- Sea la sentencia:  $A*B+C+D**(E*(F-6.25+G))**SIN(H)$ :

PARTE EVALUADA	EXPRESIÓN RESULTANTE
OP1=F-6.25	$A*B+C+D**(E*(OP1+G))**SIN(H)$
OP2=(OP1+G)	$A*B+C+D**(E*OP2)**SIN(H)$
OP3=(E*OP2)	$A*B+C+D**OP3**SIN(H)$
OP4=OP3**SIN(H)	$A*B+C+D**OP4$
OP5=D**OP4	$A*B+C+OP5$
OP6=A*B	$OP6+C+OP5$
OP7=OP6+C	$OP7+OP5$
RESULTADO=OP7+OP5	VALOR FINAL DE LA EXPRESIÓN

**Tabla 1.5: Ejemplo de reglas de precedencia de operadores aritméticos**

## 1.15 Aritmética con tipos mezclados

- Si se mezclan en una expresión operandos de distintos tipos, el resultado se eleva a la categoría del mayor, según el siguiente orden:

TIPO	PRECEDENCIA
COMPLEX	MAYOR
REAL	
INTEGER	
LOGICAL	MENOR

Tabla 1.6: Orden de precedencia de los tipos Fortran

+, -, *, /, **	INTEGER	REAL
INTEGER	INTEGER	REAL
REAL	REAL	REAL

Tabla 1.7: Aritmética con tipos mezclados

+, -, \*, /, \*\* INTEGER REAL DOUBLE P.

• Ej:

Sean:

a=8.0      b=4.0      c=3.0      (Reales)

i=8          j=4          k=3          (Enteros)

El resultado de:

- a/b      2.0      (real)
  - j/a      0.5      (real)
  - i/j      2      (entero)
  - b/c      1.33333... (real)
  - a/c      2.66667 (real)
  - j/k      1      (entero)
  - j/i      0      (entero)
  - j/c      1.33333... (real)
- Dado que la división entera puede producir resultados inesperados, los enteros deberían ser usados únicamente para cosas que son enteras intrínsecamente por naturaleza, como los contadores y los índices.
  - Debido a la longitud de palabra finita de un computador, algunos números reales no pueden representarse exactamente. Por ejemplo, la representación de 1./3. puede ser 0.333333 y, como resultado, algunas cantidades que son teóricamente iguales no lo son al ser evaluadas en un computador: 3.\*(1./3.)≠ 1.

## 1.16 Asignación aritmética

- Una sentencia de asignación aritmética asigna el valor de una expresión aritmética a una variable o un elemento de una matriz.
- El operador de asignación en Fortran es el símbolo “=”.

**variable = expresión\_aritmética**

- El funcionamiento es:
  - Se evalúa la expresión\_aritmética.
  - Se asigna el valor obtenido a la *variable*.
  - Si el tipo de la *variable* es diferente de la *expresión\_aritmética*, se produce una conversión de tipo: *expresión\_aritmética* es convertida al tipo de *variable* antes de ser asignada a *variable*.
- Pueden producirse problemas de truncamiento.
- Ej:

Si *i* es una variable entera:

*i* = 3./2. ! *i* se le asigna el valor 1

*i*=*i*+1 ! Incrementa en una unidad el valor de *i*

## 1.17 Funciones intrínsecas Fortran

- Hasta ahora hemos definido los operadores aritméticos Fortran. Pero, ¿cómo se codifican las funciones trigonométricas o el logaritmo de un número, su exponencial, raíz cuadrada, valor absoluto, etc.? y funciones más complicadas como las funciones hiperbólicas, funciones de Bessel etc.?
- Fortran incorpora todas las funciones matemáticas (y de otros tipos) en unas librerías a disposición del programador. Las funciones de esas librerías se denominan funciones *intrínsecas* del lenguaje. Otro tipo de funciones llamadas *funciones externas e internas* pueden crearse por el propio programador para resolver problemas específicos.
- Cada compilador Fortran viene provisto de las funciones intrínsecas y, en general, pueden verse también en los apéndices de cualquier libro de texto Fortran.
- En matemáticas, una *función* es una expresión que acepta uno o más valores de entrada y calcula un resultado único a partir de ellos. De la misma forma ocurre en Fortran para cualquier función.
- La sintaxis general de una función intrínseca Fortran es:

**NOMBRE (lista de argumentos)**

- NOMBRE es el nombre reservado para la función intrínseca Fortran.

- *Lista de argumentos* es una lista de variables, constantes, expresiones, o incluso los resultados de otras funciones, separadas por comas, en número y tipo fijado para cada función intrínseca.
- El resultado de la evaluación de la función en su lista de argumentos es de un tipo también fijado para cada función intrínseca.
- Hay dos tipos de funciones intrínsecas: las *genéricas* que pueden usar más de un tipo de datos de entrada y las *específicas* que sólo admiten un tipo de datos de entrada.
- Las funciones Fortran se usan escribiendo su nombre en una expresión y en cualquier caso, estarán a la derecha de una sentencia de asignación.
- Cuando aparece una función en una sentencia Fortran, los argumentos de la misma son pasados a una rutina separada que computa el resultado de la función y lo coloca en lugar de su nombre en la sentencia dada.

## **EJERCICIOS RESUELTOS**

Objetivos:

En este capítulo se construyen los primeros programas Fortran. Para ayudar al usuario a comprender los diversos conceptos, se introducen aquí las instrucciones de entrada/salida (READ/WRITE), si bien la teoría de las mismas se explica en el Capítulo 7.

Básicamente, se aprende a codificar fórmulas en este lenguaje de programación. Para ello, se usan algunas *funciones intrínsecas* matemáticas.

De cada programa mostramos su código Fortran, qué conseguimos al ejecutar el programa; alguna aclaración sobre su contenido y propuestas para conseguir ligeras modificaciones en su ejecución, teniendo en cuenta la teoría vista en el capítulo.

1. Escribir por pantalla el mensaje: HOLA A TODOS.

```
PROGRAM cap1_1

WRITE (*,*) 'HOLA A TODOS'
STOP 'FIN DE PROGRAMA'
END PROGRAM cap1_1
```

- ¿Es imprescindible la primera línea de código del programa y la penúltima?
- Comenta ambas líneas del código y analiza sus efectos.
- ¿Cómo modificarías el programa para escribir cada palabra del mensaje en una línea distinta?
- ¿Y para dejar una línea en blanco?

2. Escribir por pantalla ejemplos de los diferentes tipos de variables Fortran 90/95.

```
PROGRAM cap1_2

INTEGER :: a
REAL :: b1,b2,sueldo_del_ultimo_mes
LOGICAL :: d1,d2
COMPLEX :: e
CHARACTER (LEN=18) :: pal
CHARACTER (LEN=180) :: frase_larga
a=-123
b1=-2.98
b2=0.9E-8
sueldo_del_ultimo_mes=2850.75
d1=.true.
d2=.false.
e=(2.3,3.7)
pal='CONSTANTE CARACTER'
frase_larga=""CONSTANTE CARACTER dividida en dos lineas usando &
&el caracter & al final de la primera y al principio de la siguiente"
WRITE (*,*) 'CONSTANTE ENTERA',a
WRITE (*,*) 'CONSTANTES REALES (NOTAC NORMAL Y EXPON)',b1,b2
WRITE (*,*) 'IDENTIFICADOR DE VARIABLE REAL (MAX. 31 letras)',&
    sueldo_del_ultimo_mes,'EUROS'
```

```
WRITE (*,*) 'CONSTANTES LOGICAS',d1,d2  
WRITE (*,*) 'CONSTANTE COMPLEJA',e  
WRITE (*,*) paI !OBSERVAR QUE NO SALEN LOS APOSTROFES  
WRITE (*,*) frase_larga !AQUI SI SALEN LAS COMILLAS DOBLES  
END PROGRAM cap1_2
```

- El programa presenta declaraciones de todos los tipos de variables Fortran (ver Tabla 1.2). Se usan sentencias de asignación para asignar valores a esas variables, mostrándolos a continuación por monitor.
- Las sentencias demasiado largas se han dividido en dos líneas usando el operador & como marca de continuación de la línea anterior (ver sección 1.7).

3. Calcular la potencia de un número entero, leídos ambos previamente por teclado, y escribir el resultado por pantalla.

```
PROGRAM cap1_3  
  
IMPLICIT NONE  
INTEGER :: I,m,resul  
PRINT*,'TECLEA 2 NUMEROS ENTEROS SEPARADOS CON INTRO'  
READ*,I,m  
resul=I**m  
PRINT*,'EL RESULTADO ES:'  
PRINT*,resul  
STOP  
END PROGRAM cap1_3
```

- La tercera línea del programa incluye la declaración de variables que usamos en este programa. Notar que para escribir el contenido de la variable RESUL por pantalla se eliminan los apóstrofes en la sentencia PRINT\*.
- ¿Cómo cambia el programa si queremos operar con dos números reales?
- Ejecuta el programa para dos números grandes. ¿Qué tipos de variables debes usar para que funcione el programa?

4. Calcular el valor de la expresión  $3x^2 + 2y - \frac{z}{4}$  para tres números reales  $x, y, z$  leídos por teclado y escribir el resultado por pantalla.

```
PROGRAM cap1_4

REAL :: x,y,z,resul
WRITE(*,*) 'DAME 3 NUMEROS REALES'
READ (*,*) x,y,z
resul=3*x**2+2*y-z/4
WRITE(*,*) 'EL RESULTADO ES:',resul
END PROGRAM cap1_4
```

- Repite el ejercicio eliminando la declaración de variables, es decir, la segunda línea de código. Revisa el apartado de la declaración implícita de variables.
- ¿Qué ocurre si defines las variables de tipo INTEGER? Revisa los apartados de la aritmética con tipos mezclados y la asignación.

5. Resuelve la ecuación de segundo grado  $Ax^2 + Bx + C = 0$  únicamente para soluciones reales. Lee los coeficientes A, B y C por teclado y escribe las dos soluciones por pantalla.

```
PROGRAM cap1_5
IMPLICIT NONE
REAL :: a,b,c,x1,x2
WRITE(*,*) 'DAME A B Y C'
READ(*,*) a,b,c
x1=(-b+SQRT(b**2-4*a*c))/(2*a)
x2=(-b-SQRT(b**2-4*a*c))/(2*a)
WRITE(*,*) 'EL RESULTADO ES:',x1,x2
END PROGRAM cap1_5
```

- **SQRT(arg)** es una función intrínseca Fortran que permite calcular la raíz cuadrada del argumento escrito entre paréntesis. El argumento no puede ser de tipo INTEGER pero si REAL. El resultado es de tipo REAL.

- Comprueba que se obtiene el mismo resultado calculando la raíz cuadrada como el radicando elevado a un medio. ¿Qué operaciones se realizan cuando se quita el paréntesis en  $(2*A)$ ? ¿Y si se quita en  $(-B-SQRT(B**2-4*A*C))$ ?
- Ejecuta el programa para  $A=1$ ,  $B=1$  y  $C=1$ . ¿Qué tipo de error se obtiene? Reflexiona sobre la importancia de testear el funcionamiento de un programa para todos los posibles juegos de valores de prueba.

6. Codifica la expresión  $\frac{\text{sen}^2(x) + \text{cos}^2(x)}{\pi}$  para un ángulo  $x$  en radianes leído por teclado. Escribe el resultado por monitor.

```
PROGRAM cap1_6

IMPLICIT NONE
REAL :: x, pepe
REAL, PARAMETER :: pi=3.14159
WRITE(*,*) 'DAME EL VALOR DEL ANGULO EN RADIANES'
READ(*,*) x
pepe=(SIN(x)**2+COS(x)**2)/pi
WRITE(*,*) 'EL RESULTADO ES', pepe
END PROGRAM cap1_6
```

- Aquí se usan las funciones intrínsecas seno **SIN(arg)** y coseno **COS(arg)**. Para ambas, el argumento (ángulo) debe ser dado en radianes y puede ser REAL. El resultado es del mismo tipo que su argumento.
- Cambia el programa para que funcione con ángulos dados en grados sexagesimales.
- ¿Qué ocurre si escribimos la sentencia de asignación siguiente debajo de la de lectura de  $x$ :  $PI=3.14$ ?

### **EJERCICIOS PROPUESTOS**

- 1) Programa que muestre 'FORTRAN' y 'FORMula TRANslation' en líneas diferentes.
- 2) Programa que pida por teclado una longitud expresada en pulgadas y la muestre convertida en centímetros ( $1" = 2.54 \text{ cm}$ ).
- 3) Programa que pida una cantidad en pesetas y la muestre en € euros.
- 4) Programa que acepte por teclado el diámetro de una circunferencia, muestre la longitud de la circunferencia ( $L=2\pi R$ ) y el área del círculo encerrado ( $A=\pi R^2$ ).
- 5) Programa que pida por teclado el radio de la base y la altura de un cono. Calcula y escribe el valor de su volumen ( $V=\frac{1}{3}\pi R^2 H$ ).
- 6) Programa que pida por teclado el lado de un cuadrado. Calcula y escribe los valores de su área ( $A=L^2$ ) y diagonal (Teor. Pitágoras).
- 7) Programa que pida el radio de una esfera y calcule y muestre su superficie ( $S=4\pi R^2$ ) y volumen ( $V=\frac{4}{3}\pi R^3$ ).
- 8) Programa que pida los dos catetos de un triángulo rectángulo y muestre su hipotenusa, el área del rectángulo cuyos lados son los dos catetos y los dos ángulos del triángulo expresados en grados sexagesimales.
- 9) Programa que pida coordenadas polares de un punto ( $r, \alpha$ ) y muestre sus coordenadas cartesianas o rectangulares ( $x, y$ ).
- 10) Programa que pida coordenadas cartesianas de los extremos de dos vectores origen y calcule el vector suma, mostrando sus componentes, módulo y ángulo en grados.
- 11) Programa que pida una cantidad en euros y muestre la cantidad de monedas de cada tipo (2 euros, 1 euro, 50 céntimos, 20 céntimos, 10 céntimos, 5 céntimos, 2 céntimos y 1 céntimos).