7 FORMATOS Y ARCHIVOS

7.1 Entrada/salida en Fortran

- Al igual que en el resto de los lenguajes, se llama **entrada o lectura** de datos al proceso de pasar esos datos desde un dispositivo de entrada (teclado, archivo, etc) a un computador.
- El proceso simétrico se llama **salida o escritura** de datos.
- En Fortran, hay dos maneras de realizar entradas/salidas:
    - entrada/salida *dirigida por lista*. En este caso, el formato de los datos depende de sus tipos (enteros, reales, etc.) y del computador. Se dice que es una entrada/salida con *formato libre*.
    - entrada/salida *con formatos*. El programador define la manera exacta en que quiere leer/escribir los datos.
- Hay una sentencia de lectura: READ y dos de escritura: WRITE y PRINT. Se comienza estudiando las sentencias de escritura por ser más útiles que la de lectura.

7.2 Salida por pantalla

- Hasta ahora, la salida de datos se ha realizado siempre por pantalla con formato libre. Para ello, se ha empleado la sentencia:

```
WRITE (*,*) lista de variables
```

    - en la cual el primer asterisco se refiere al dispositivo de salida estándar (generalmente, la pantalla) y el segundo asterisco se refiere al formato libre con que se mostrarán las variables de la lista,
- o la sentencia:

```
PRINT *, lista de variables
```

    - en la cual PRINT significa escribir en el dispositivo de salida estándar y el asterisco indica con formato libre.
- Sin embargo, estas salidas no tienen generalmente el aspecto deseado, ya que aparecen demasiados blancos extra. En realidad, constituyen un caso particular de unas sentencias más generales en las que se puede indicar con exactitud cómo se quieren escribir los datos. La sintaxis general de una salida por pantalla con formatos es:

```
WRITE (*, formato) lista de variables
```
ó
```
PRINT formato, lista de variables
```

- donde *formato* puede ser un * (salida dirigida por lista), pero también puede ser una expresión carácter, variable o constante, que contiene los *descriptores de formato* de la lista, o la etiqueta de una

sentencia FORMAT, es decir, un entero entre 1 y 99999. En este último caso, debe existir además una sentencia de la forma:

etiqueta FORMAT (lista de descriptores de formato)

- En las secciones siguientes, se estudian en detalle los descriptores de formato disponibles en Fortran 90/95.
- La sentencia PRINT sólo funciona con el dispositivo de salida estándar y, por lo tanto, es mucho menos flexible que la sentencia WRITE estándar, como se verá en una sección siguiente.
- PRINT sobrevive en Fortran 90/95 por su extraordinario uso en versiones anteriores. Es útil reconocer esta sentencia, sin embargo, no debería usarse en los programas creados por el usuario.
- Ejemplos de salidas por pantalla.

WRITE (*, *) N,M ! formato libre

WRITE (*,‘2I3’) N,M !constante carácter especifica formatos de N y M

string=’2I3’

WRITE (*,string) N,M !variable carácter especifica formatos de N y M

WRITE (*, 200) N,M !etiqueta de sentencia FORMAT

200 FORMAT (2I3) !aqui se especifican formatos de N y M

7.3 Entrada por teclado

- Hasta ahora, la entrada de datos se ha realizado siempre por teclado con formato libre. Para ello, se ha empleado la sentencia:

READ (*,*) lista de variables

  - en la cual el primer asterisco se refiere al dispositivo de entrada estándar (generalmente, el teclado) y el segundo asterisco se refiere al formato libre con que se leerán las variables de la lista,

- o la sentencia:

READ *, lista de variables

  - en la cual READ significa leer del dispositivo de entrada estándar y el asterisco indica con formato libre.

- Sin embargo, estas entradas pueden no tener el resultado deseado y constituyen un caso particular de unas sentencias más generales en las que se puede indicar con exactitud cómo se quieren leer los datos de teclado. La sintaxis general de una entrada por teclado con formatos es:

READ (*, formato) lista de variables ó

READ formato, lista de variables

- donde *formato* puede ser un asterisco *, entrada dirigida por lista, pero también una expresión carácter, variable o constante, que contiene los descriptores de formato de la lista, o la etiqueta de una sentencia FORMAT, es decir, un entero entre 1 y 99999. En este último caso, debe existir además una sentencia de la forma:

etiqueta FORMAT (lista de descriptores de formato).

- La segunda forma de READ sólo funciona con el dispositivo de entrada estándar y, por lo tanto, es mucho menos flexible que la primera, cómo se verá en una sección siguiente. Sobrevive en Fortran 90/95 por su extraordinario uso en versiones anteriores. Es útil reconocer esta sentencia, sin embargo, no debería usarse en los programas creados por el usuario.

7.4 Descriptores de formato

- Hay 4 categorías básicas de descriptores de formato:
    - Los que describen la posición vertical de la línea de texto.
    - Los que describen la posición horizontal de los datos en una línea.
    - Los que describen el formato de entrada/salida de un valor particular.
    - Los que controlan la repetición de descriptores o grupos de descriptores de formato.
- La siguiente tabla contiene una lista de símbolos usados con los descriptores de formatos más comunes:

SÍMBOLO	SIGNIFICADO
c	número de columna
d	nº de dígitos a la derecha del punto decimal para entrada/salida de datos reales
m	mínimo nº de dígitos
n	nº de espacios saltados
r	factor de repetición: nº de veces que se usa un descriptor o grupo de descriptores
w	anchura del campo: nº de caracteres de entrada/salida

Tabla 7.1: Símbolos usados en los descriptores de formatos

7.4.1 Descriptor I de formato entero

- Sintaxis general para salida de datos enteros:

[r]Iw[.m]

- Los símbolos usados tienen el significado que se muestra en la Tabla 7.1. El valor se ajusta a la derecha del campo. Si el valor es demasiado grande para mostrarse con w caracteres, se muestran w asteriscos.
- Sintaxis general para entrada de datos enteros:

[r]Iw

- El valor puede estar en cualquier posición dentro del campo especificado.
- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	SALIDA
I4	452	•452
I2	6234	**
I5	-52	••-52
I4.3	3	•003
I2.0	0	••

Tabla 7.2: Formatos de escritura de enteros

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	VALOR LEÍDO
I4	•1••	1
I2	-1	-1
I4	-123	-123
I3	•12	12
I2	123	12

Tabla 7.3: Formatos de lectura de enteros

7.4.2 Descriptor F de formato real

- Sintaxis general para entrada/salida de datos reales:

[r]Fw.d

- Los símbolos usados tienen el significado que se muestra en la Tabla 7.1.
- Para salida, el valor se ajusta a la derecha del campo.
- Si d es menor que el número de dígitos decimales del número, el valor se redondea.
- Si d es mayor que el número de dígitos decimales del número, se añaden ceros hasta completarlo.
- Si el valor es demasiado grande para leerse/escribirse con w caracteres, el campo w se llena de asteriscos.
- Para evitar mal interpretaciones, es conveniente incluir siempre un punto decimal en cualquier valor real usado en una sentencia de lectura con formato.
- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	SALIDA
F9.3	25.338	•••25.338
F5.1	0.35247	••0.4
F6.2	0.089235	••0.09
F8.3	732.56	•732.560
F4.3	-12.345	****

Tabla 7.4: Formatos de escritura de reales

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	VALOR LEÍDO
F6.3	49.225	49.225
F6.2	49.225	49.23
F7.1	-1525.1	-1525.1

Tabla 7.5: Formatos de lectura de reales

7.4.3 Descriptor E de formato exponencial

- La notación científica es muy usada por científicos e ingenieros para representar números muy grandes o muy pequeños. En esta notación, los números se normalizan al rango de valores entre 1.0 y 10.0 y se representan, por un número entre 1.0 y 10.0 multiplicado por una potencia de diez. Por ejemplo: el número de Avogadro en notación científica es 6.023×10^{23}.

- El formato exponencial no se corresponde exactamente con la notación científica, pues los números se normalizan al rango de valores entre 0.1 y 1.0. En el ejemplo anterior, el número de Avogadro se representa en formato exponencial por 0.6023E+24.
- La sintaxis general para entrada/salida de datos reales con formato exponencial es:

[r]Ew.d

- Los símbolos usados tienen el significado que se muestra en la Tabla 7.1.
- Los mismos criterios que operan sobre un dato real con formato F se aplican a ese dato real con formato exponencial.
- Si un número se quiere leer/escribir en formato exponencial con d cifras decimales, la anchura de campo mínima w ≥ d+7, pues tal número se representa ±0.ddddE±ee y requiere como mínimo un carácter para representar el signo (sólo si es negativo), otro para la parte entera del número, el punto decimal, el carácter E, el signo del exponente y los dos dígitos del mismo.
- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	SALIDA
E8.2	83.456	0.83E+02
E7.2	83.456	*******
E10.3	8.3974	•0.840E+01
E10.4	0.83E2	0.8300E+02

Tabla 7.6: Formatos de escritura de reales en formato exponencial

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	VALOR LEÍDO
E11.2	•43.258E+03	0.43E+05
E11.5	•43.258E+03	0.43258E+05
E11.3	•••0.43E-02	0.430E-02

Tabla 7.7: Formatos de lectura de reales en formato exponencial

7.4.4 Descriptor ES de formato científico

- El formato científico coincide exactamente con la definición de notación científica dada más arriba.
- La sintaxis general para entrada/salida de datos reales con formato científico es:

[r]ESw.d

- Los símbolos usados tienen el significado que se muestra en la Tabla 7.1.
- Los mismos criterios que operan sobre un dato real con formato F se aplican a ese dato real con formato científico.
- Si un número se quiere leer/escribir en formato científico con d cifras decimales, la anchura de campo mínima w ≥ d+8, pues tal número se representa ±1.ddddES±ee y requiere como mínimo un carácter para representar el signo (sólo si es negativo), otro para la parte entera del número, el punto decimal, los dos caracteres ES, el signo del exponente y los dos dígitos del mismo.
- La diferencia entre el formato exponencial y el científico para un valor real dado con una anchura de campo dada es que en el formato científico se representa una cifra significativa más que en el formato exponencial.
- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	SALIDA
ES8.2	83.456	8.35E+01
ES7.2	83.456	*******
ES10.3	8.3974	•8.397E+00
E11.4	0.83ES2	8.3000ES+01

Tabla 7.8: Formatos de escritura de reales en formato científico

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	VALOR LEÍDO
ES11.2	•43.258E+03	4.33ES+04
ES11.5	•43.258E+03	4.32580ES+04
ES11.3	•••0.43E-02	4.300ES-01

Tabla 7.9: Formatos de lectura de reales en formato científico

7.4.5 Descriptor L de formato lógico

- Sintaxis general para entrada/salida de datos lógicos:

[r]Lw

- La salida de un dato lógico es T o F y su valor se ajusta a la derecha del campo.

- La entrada de un dato lógico se usa muy raramente pero puede ser T o F como primer carácter no blanco situado en cualquier posición dentro de la anchura de campo dada.
- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	SALIDA
L5	.FALSE.	••••F
L4	.TRUE.	•••T
L1	.TRUE.	T
L2	.FALSE.	•F

Tabla 7.10: Formatos de escritura de datos lógicos

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	VALOR LEÍDO
L5	•••T•	.TRUE.
L2	F1	.FALSE.
L4	•X•T	ERROR

Tabla 7.11: Formatos de lectura de datos lógicos

7.4.6 Descriptor A de formato carácter

- Sintaxis general para entrada/salida de datos carácter:

[r]A[w]

- Si w no aparece, el descriptor A lee/escribe el dato carácter en una anchura igual a la longitud de la variable carácter.
- Si w aparece, el descriptor A lee/escribe el dato carácter en una anchura fija a w.
    - Si w>longitud de la variable carácter:
        - para salida, la cadena se ajusta a la derecha del campo y,
        - para entrada, el dato del fragmento derecho del campo se lee en la variable carácter.
    - Si w<longitud de la variable carácter:
        - para salida, sólo se escriben los primeros w caracteres de la cadena y,

- para entrada, sólo los primeros w caracteres de la cadena se ajustan a la izquierda de la variable carácter y el resto se llena con blancos.

- Ejemplos de salida:

DESCRIPTOR	VALOR INTERNO	LONGITUD DE LA VARIABLE CARACTER	SALIDA
A	ABCDEF	6	ABCDEF
A8	ABCDEF	6	••ABCDEF
A4	ABCDEF	6	ABCD

Tabla 7.12: Formatos de escritura de caracteres

- Ejemplos de entrada:

DESCRIPTOR	CAMPO ENTRADA	LONGITUD DE LA VARIABL CARACTER	VALOR LEÍDO
A	ABCDEFGH	6	ABCDEF
A	ABCDEFGH	8	ABCDEFGH
A8	ABCDEFGH	4	EFGH
A4	ABCDEFGH	6	ABCD••

Tabla 7.13: Formatos de lectura de caracteres

7.4.7 Descriptores X, T de posición horizontal y / de posición vertical

- Los descriptores X y T se usan para controlar el espacio horizontal y el descriptor slash / para controlar el espacio vertical. La sintaxis general de cada uno de ellos es:

nX

- Para salida: suele emplearse para espaciar los datos. El descriptor nX salta n espacios en la línea actual.
- Para entrada: puede emplearse para saltar por encima de campos de entrada que no se quieren leer en la línea actual.

Tc

- Salta directamente a la columna número c de la línea actual. Funciona como un tabulador más general, pues puede saltar hacia derecha o izquierda.
    - Para salida: suele emplearse para espaciar los datos.

- Para entrada: puede emplearse para saltar por encima de campos de entrada que no se quieren leer o para leer varias veces unos datos.

/[/][…]

- Este es un descriptor especial que no es necesario separarlo de los demás descriptores por comas, si bien pueden usarse.
    - Para salida: un slash envía la línea actual a salida y empieza una nueva. Así, una sentencia WRITE puede escribir los valores de salida separados en dos líneas. Si aparecen varios slashes juntos, se saltarán varias líneas.
    - Para entrada, un slash ignora la línea actual y comienza a procesar la siguiente línea.
- Ejemplo. Sean las declaraciones:

INTEGER:: numero1=345, numero2=678

REAL:: a=7.5, b=0.182

PRINT '(1X,T30,A)','RESULTADOS'

PRINT '(1X,I3,2X,I3)', numero1, numero2

PRINT '(1X, 2I4, F6.3/,1X,F6.3)', numero1, numero2, a, b

Las salidas generadas son:

••••••••••••••••••••••••••••••RESULTADOS

•345••678

••345•678•7.500

••0.182

- Ejemplo. Sean las declaraciones:

INTEGER:: a,b,c,d

READ (*,30) a,b,c,d

30 FORMAT (2I2,//, 2I2)

Si los datos de entrada son:

•1•2•3

•4•5•6

•7•8•9

Se leen a, b, c, d con los valores 1, 2, 7 y 8, respectivamente.

7.4.8 Repetición de grupos de descriptores de formato

- Para repetir un grupo de descriptores de formato hay que encerrar tal grupo entre paréntesis y colocar un factor de repetición a la izquierda del mismo.
- Ejemplo:

30 FORMAT (1X, I4, F9.2, I4, F9.2, I4)

30 FORMAT (1X, I4, 2(F9.2, I4))

7.5 Procesamiento de archivos

- Las aplicaciones que manejan conjuntos de datos muy grandes, es conveniente que almacenen los datos en algún archivo en disco o algún otro dispositivo de memoria auxiliar.
- Antes de que Fortran pueda usar un archivo, debe abrirlo, asignándole una unidad. La sintaxis general para abrir un archivo es:

OPEN (lista_open)

- donde *lista_open* puede incluir varias cláusulas separadas por comas, colocadas en cualquier orden, de las cuales se estudian a continuación las más importantes:
    - [UNIT=]unidad es un número entero comprendido entre 1 y 99 que identifica al archivo.
    - FILE = archivo es una expresión carácter que indica el nombre del archivo que se quiere abrir.
    - [STATUS = estado_del_archivo] es una de las siguientes constantes carácter: 'OLD', 'NEW', 'REPLACE', 'SCRATCH' o 'UNKNOWN' (opción por defecto).
        - La opción 'SCRATCH' crea un archivo temporal que se destruye automáticamente cuando se cierra el archivo o cuando acaba la ejecución del programa. Se suele usar para guardar resultados intermedios durante la ejecución de un programa. Estos archivos no pueden tener nombre.
        - La opción 'UNKNOWN' implica, si existe el archivo antes de ejecutar el programa, que lo reemplaza, y si no existe, lo crea en tiempo de ejecución y lo abre.
    - [ACTION = accion] es una de las siguientes constantes carácter: 'READ', 'WRITE' o 'READWRITE' (opción por defecto).
    - [IOSTAT = error_de_apertura] es una variable entera que almacena el estado de la operación de apertura de archivo. Aunque es una cláusula opcional, se aconseja usarla para evitar abortar un programa cuando se produce un error de apertura. Si el valor de la variable es
        - cero, significa éxito en la apertura del archivo,
        - positivo, significa que se ha producido un error al abrir el archivo.
    - [ACCESS = acceso] es una de las siguientes constantes carácter: 'SEQUENTIAL'(opción por defecto) o 'DIRECT'.

Los archivos permiten acceso directo, es decir, saltar de una línea (también llamada registro) a cualquier otra, independientemente de su situación en el archivo. Sin embargo, por razones históricas, la técnica de acceso por defecto en Fortran es secuencial, es decir, que el acceso a los registros se realiza en orden consecutivo, desde el primer registro hasta el último.

- [POSITION = posicion] es una de las siguientes constantes carácter: 'REWIND', 'ASIS' (opción por defecto) o 'APPEND'. Si la posición es:
  - 'REWIND', el puntero de archivo se coloca en el primer registro.
  - 'ASIS', el puntero de archivo se coloca en un registro dependiente del procesador.
  - 'APPEND', el puntero de archivo se coloca después del último registro justo antes de la marca de fin de archivo.

- Ejemplo. Apertura de un archivo para lectura:

```
INTEGER:: error_de_apertura
OPEN(UNIT=12,FILE='datos',STATUS='OLD',ACTION='READ',&
IOSTAT=error_de_apertura)
```

- Ejemplo. Apertura de un archivo para escritura:

```
INTEGER:: error_de_apertura
OPEN(UNIT=9,FILE='resultado',STATUS='NEW',ACTION='WRITE
',&IOSTAT=error_de_apertura)
```

- La asociación unidad-archivo que estableció OPEN se termina con la sentencia:

```
CLOSE (lista_close)
```

- donde *lista_close* puede incluir varias cláusulas separadas por comas, de las cuales sólo el número de unidad es obligatoria:
  - [UNIT=]unidad

- Ejemplo. Cerrar archivo identificado por la unidad 12.

```
CLOSE (UNIT = 12)
```

7.6 Posición en un archivo

- Fortran proporciona dos sentencias para ayudar a moverse en un archivo secuencial. La sintaxis general es:

```
REWIND unidad
```

- para reposicionar un archivo al principio y

```
BACKSPACE unidad
```

- para reposicionar un archivo al principio de la línea o registro anterior.
- Si el archivo está en su posición inicial, las dos sentencias anteriores no tienen efecto.
- Ejemplos:

REWIND 10

BACKSPACE 30

7.7 Salida por archivo

- La sentencia WRITE permite escribir datos en cualquier dispositivo de salida, como los archivos. La sintaxis general de salida por archivo es:

WRITE ([UNIT =] unidad, formato[,IOSTAT = error_de_escritura][,...]) lista_de_variables

- Aunque esta sentencia puede incluir varias cláusulas, a continuación se estudian las tres más importantes, de las cuales sólo las dos primeras son obligatorias.
    - *unidad* es un número que identifica el dispositivo en el que se va a efectuar la salida de datos.
    - *formato* indica los formatos con que se van a escribir las variables de la lista en el dispositivo de salida. Puede ser, como ya se ha explicado en una sección anterior:
        - *, salida dirigida por lista.
        - Una expresión carácter, variable o constante, que contiene los descriptores de formatos de la lista.
        - La etiqueta de una sentencia FORMAT, es decir, un entero entre 1 y 99999. En este caso, debe existir una sentencia de especificación:

etiqueta FORMAT (lista de descriptores de formato).

    - *error_de_escritura*, indica el éxito o no de la operación de escritura. Aunque esta cláusula es opcional, se aconseja usarla para evitar abortar un programa cuando se produce un error de escritura. Si el valor de esa variable entera es:
        - cero, significa éxito en la operación de escritura,
        - positivo, significa que se ha producido un error en la escritura.
- Por lo tanto, la sentencia WRITE estándar toma los datos de la lista de variables, los convierte de acuerdo con los descriptores de formato especificados y, si no hay errores, los escribe en el archivo asociado a la unidad especificada.
- Ejemplos de escritura de un archivo:

```
INTEGER:: error_de_escritura
...
WRITE (UNIT=16,*, IOSTAT= error_de_escritura) X,Y
WRITE (20,'3F8.3') X,Y,Z
```

7.8 Entrada por archivo

- La sentencia READ estándar permite leer datos de cualquier dispositivo de entrada, como los archivos. La sintaxis general de entrada por archivo es:

```
READ ([UNIT =] unidad, formato [,IOSTAT = error_de_lectura][,...])
lista_de_variables
```

- Aunque esta sentencia puede incluir varias cláusulas, a continuación se estudian las tres más importantes, de las cuales sólo las dos primeras son obligatorias.
    - *unidad* es un número que identifica el dispositivo desde donde se va a efectuar la entrada de datos.
    - *formato* indica los formatos con que se van a leer las variables de la lista en el dispositivo de entrada. Puede ser, como ya se ha explicado en una sección anterior:
        - *, entrada dirigida por lista.
        - Una expresión carácter, variable o constante, que contiene los descriptores de formatos de la lista.
        - La etiqueta de una sentencia FORMAT, es decir, un entero entre 1 y 99999. En este caso, debe existir una sentencia de especificación:

```
etiqueta FORMAT (lista de descriptores de formato).
```

    - *error_de_lectura*, indica el éxito o no de la operación de lectura. Aunque esta cláusula es opcional, se aconseja usarla para evitar abortar un programa cuando ocurre un error de lectura o cuando se intenta leer más allá del fin de archivo. Si el valor de esa variable entera es:
        - cero, significa éxito en la operación de lectura,
        - positivo, significa que se ha producido un error en la lectura,
        - negativo, significa fin de archivo.
- Por lo tanto, la sentencia READ estándar lee los datos de un archivo asociado a una unidad, convierte sus formatos de acuerdo con los descriptores de formato y, si no hay errores, almacena esos datos fomateados en la lista de variables dada.
- Ejemplo de lectura de un archivo:

```
INTEGER:: error_de_lectura
```

```
…
READ (15, *, IOSTAT = error_de_lectura) a,b,c
```

- Ejemplo de procesamiento de archivos.

```
PROGRAM temperaturas
! Lee una serie de temperaturas de un archivo y
! calcula el valor medio
IMPLICIT NONE
CHARACTER (LEN=20):: archivo
INTEGER:: cuenta, error_de_apertura, error_de_lectura
REAL:: temperatura, suma, media
! Abrir archivo como unidad 15
WRITE( *, *) ‘     Dame nombre del archivo:’
READ (*, *) archivo
OPEN(15, FILE= archivo, STATUS=‘OLD’, ACTION=’READ’,&
IOSTAT= error_de_apertura)
IF (error_de_apertura > 0) STOP ‘ error al abrir el archivo ‘
100 FORMAT (1X, F4.1)
cuenta=0
suma=0
DO
  READ (15, 100, IOSTAT = error_de_lectura) temperatura
  IF (error_de_lectura >0 ) THEN
    WRITE(*,*) ‘error de lectura’
    EXIT
  ELSE IF (error_de_lectura <0 ) THEN
!    WRITE(*,*) ‘fin del archivo’
    EXIT
  ELSE
  suma = suma + temperatura
  cuenta = cuenta + 1
  ENDIF
END DO
! Calcular temperatura media
media = suma/ REAL(cuenta)
WRITE (*, *) ‘ temperatura media: ‘
```

```
WRITE (*, *) media
CLOSE (15)
END PROGRAM temperaturas
```

EJERCICIOS RESUELTOS

Objetivos:

Aprender a leer y escribir con formatos específicos los diferentes tipos de variables Fortran estudiados.

Además, se aprende a manejar archivos para leer información de entrada requerida para la ejecución de un programa y/o escribir información de salida, generada en la ejecución del mismo.

1. Escribe una tabla de raíces cuadradas y cúbicas de todos los números naturales desde 1 hasta 100.

```
PROGRAM cap7_1
IMPLICIT NONE
INTEGER:: n
WRITE(*,8) 'NUMERO, RAIZ CUADRADA, RAIZ CUBICA'
8 FORMAT (1X, A)
WRITE (*,10) (n,SQRT(REAL(n)),REAL(n)**(1.0/3.0),n=1,100)
10  FORMAT (1X, I5, 4X, F8.3, 8X, F8.3)
END PROGRAM cap7_1
```

2. Escribir algunas variables enteras, reales y carácter con distintos formatos.

```
PROGRAM cap7_2

IMPLICIT NONE
INTEGER :: m=-123,n=9877,l=889,i=77
REAL:: r=89.126, s=14.532
CHARACTER (LEN=6):: pal='abcdef'

WRITE(*,100) 'Enteros',m,n,l,i
100  FORMAT(1X/1X,T10,A/,1X,I4,1X,I4,1X,I3,1X,I1)

WRITE(*,150) 'Reales','se redondea',r,'se rellena con ceros',r,'no cabe',r
150 FORMAT(1X/1X,T10,A/,1X,A,1X,F5.2/,1X,A,1X,F7.4/,1X,A,1X,F4.2)

WRITE(*,200) 'formato real',s,'formato exponencial',s,'formato cientifico',s
200  FORMAT(1X/1X,A,1X,F5.2/,1X,A,1X,E9.2/,1X,A,1X,ES9.2)

WRITE(*,250) 'Caracteres','w=6=LEN(pal)',pal,&
'w=8. La variable se ajusta a la derecha del campo',pal,&
'w=4. Se muestran los 4 primeros caracteres de la variable',pal
250  FORMAT(1X/,T10,A/,1X,A,1X,A/,1X,A,1X,A8/,1X,A,1X,A4)

WRITE(*,300)'repeticion de descriptores',m,n,l,i,r,s,pal
```

```
300 FORMAT(1X/,1X,A/,1X,4I5,2F6.2,A7)
END PROGRAM cap7_2
```

3. Leer de un archivo con nombre "cap7_3in.txt" los nombres y las notas de todos los alumnos de una clase. Crear en la misma carpeta un archivo con nombre "cap7_3out.txt" y escribir en él los nombres de los alumnos y su clase según sus notas, de acuerdo con la siguiente clasificación:

Clase	Nota
1	[8.5,10]
2	[5,8.5)
3	[0,5)

```
PROGRAM cap7_3
IMPLICIT NONE
CHARACTER (LEN=18) :: nombre
REAL :: nota
INTEGER                                                              ::
clase,error_de_apertura,error_de_lectura,error_de_escritura


OPEN (60,FILE='cap7_3in.txt',STATUS='OLD',ACTION='READ',&
IOSTAT=error_de_apertura)
IF (error_de_apertura>0) STOP 'cap7_3in.txt NO ABIERTO'
OPEN (70,FILE='cap7_3out.txt',STATUS='NEW',ACTION='WRITE',&
IOSTAT=error_de_apertura)
IF (error_de_apertura>0) STOP 'cap7_3out.txt NO ABIERTO'
WRITE(70,9,IOSTAT=error_de_escritura) 'ALUMNO','CLASE'
WRITE(70,'(1X,A6,T20,A5)')'======','====='
IF (error_de_escritura>0) STOP 'error de escritura'
9 FORMAT (1X,A6,T20,A5)
DO
  READ(60,*,IOSTAT=error_de_lectura) nombre,nota
  externo:IF (error_de_lectura >0 ) THEN
    WRITE(*,*) 'error de lectura'
    EXIT
  ELSE IF (error_de_lectura <0 ) THEN
```

```
   WRITE(*,*) 'fin del archivo'
   EXIT
  ELSE
!   WRITE(*,*) nombre,nota
   interno:IF (nota < 5.0) THEN
     clase=3
   ELSE IF (nota < 8.5) THEN
     clase=2
   ELSE
     clase=1
   END IF interno
   WRITE (70,10,IOSTAT=error_de_escritura) nombre,clase
10 FORMAT(1X,A,I2)
   IF (error_de_escritura>0) STOP 'error de escritura'
  END IF externo
END DO
END PROGRAM cap7_3
```

– El archivo *cap7_3in.txt* debe existir en el mismo directorio donde se guarda este programa, antes de ejecutarlo.

– El archivo *cap7_3out.txt* se crea como resultado de la ejecución del programa anterior en la misma ubicación.

– Un ejemplo de archivo de entrada es:

'Raul Cagigal' 7.3

'Sara Mayoral' 2.6

'Guillermo Fuentes' 8.9

'Laura Cayon' 5.5

'Alvaro Torres' 9.7

'Gregorio Lanza' 4.9

'Ana Cuadra' 3.4

'Maria Aguirre' 4.7

'Lorenzo San Miguel' 6.5

'Jose Luis Casado' 6.2

4. Obtener el nombre y la nota del mejor y peor alumno de clase en Informática. Dar la media de la clase y la cantidad de suficientes.

– El nombre de los alumnos y sus notas están en un archivo de datos con nombre “cap7_4in.txt”.

– La media de la clase y la cantidad de suficientes han de calcularse a través de sendas funciones.

– La escritura de los resultados ha de volcarse a un archivo de salida, con nombre “cap7_4out.txt”.

– Usar formatos para escribir en el archivo de salida. Dedicar una cifra decimal para los números reales involucrados.

```
PROGRAM cap7_4
IMPLICIT NONE
CHARACTER (LEN=15), DIMENSION (5) :: nomb
CHARACTER (LEN=15)::nombm,nombp
REAL, DIMENSION(5) :: nota
REAL:: mejor,peor,mediaclase,media
INTEGER:: i,n,error_de_apertura,error_de_lectura,suficientes,num

OPEN (10,FILE='cap7_4in.txt',STATUS='OLD',ACTION='READ',&
IOSTAT=error_de_apertura)
IF (error_de_apertura>0) STOP 'cap7_4in.txt NO ABIERTO'

!..LECTURA DE DATOS.
i=1
DO
  READ(10,*,IOSTAT=error_de_lectura) nomb(i),nota(i)
  externo:IF (error_de_lectura >0 ) THEN
    WRITE(*,*) 'error de lectura'
    EXIT
  ELSE IF (error_de_lectura <0 ) THEN
!    WRITE(*,*) 'fin del archivo'
    n=i-1
    EXIT
  ENDIF
  i=i+1
END DO
!..CALCULO DEL PEOR DE LA CLASE..
peor=nota(1)
nombp=nomb(1)
DO i=2,n
```

```
  IF (nota(i) < peor) THEN
    peor=nota(i)
    nombp=nomb(i)
  END IF
END DO
!..CALCULO DEL MEJOR DE LA CLASE..
mejor=nota(1)
nombm=nomb(1)
DO i=2,n
  IF (nota(i) > mejor) THEN
    mejor=nota(i)
    nombm=nomb(i)
  END IF
END DO
num=suficientes(nota,n)
media=mediaclase(nota,n)

!ESCRITURA DE RESULTADOS EN UN ARCHIVO

OPEN (20,FILE='cap7_4out.txt',STATUS='NEW',ACTION='WRITE',&
IOSTAT=error_de_apertura)
IF (error_de_apertura>0) STOP 'cap7_4out.txt NO ABIERTO'
WRITE(20,'(1X,A,I3,A)') 'HAY',num,' ALUMNOS CON SUFICIENTE'
WRITE(20,'(1X,A,F4.1)') 'LA NOTA MEDIA DE LA CLASE ES:',media
WRITE(20,'(1X,3A,F4.1)') 'El peor alumno es: ',nombp,'y su nota
es:',peor
WRITE(20,'(1X,3A,F4.1)') 'El mejor alumno es: ',nombm, &
'y su nota es:',mejor
CLOSE (10)
CLOSE (20)
END PROGRAM cap7_4

!..ALUMNOS QUE HAN SACADO SUFICIENTE..

INTEGER FUNCTION suficientes(nota,n)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n),INTENT(IN) :: nota
INTEGER :: contador,i
```

```
contador=0
DO i=1,n
  IF (nota(i) >= 5.AND.nota(i) < 6) THEN
    contador=contador+1
  END IF
END DO
suficientes=contador
END FUNCTION suficientes

!..CALCULO DE LA MEDIA DE LA CLASE..

REAL FUNCTION mediaclase(nota,n)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n), INTENT(IN) :: nota
INTEGER:: i
mediaclase=0
DO  i=1,n
  mediaclase=mediaclase+nota(i)
END DO
mediaclase=mediaclase/n
END FUNCTION mediaclase
```

– Un ejemplo de archivo de entrada es:

'Santiago Lopez' 8.75
'Elisa Fernandez' 2.5
'Jose Garcia' 5.25
'Maria Rodriguez' 6.8
'Luis Martin' 5.9

EJERCICIOS PROPUESTOS

1) Programa que escriba en un archivo con nombre cap7_1p_out.txt los 100 primeros números naturales, generados por el propio programa.

2) Programa que abra el archivo del ejercicio anterior, calcule el cuadrado de cada número y lo escriba en un archivo con nombre cap7_2p_out.txt. ¿Cómo se escribiría todo en el mismo archivo? Para comodidad del usuario, copiar cap7_1p_out.txt en cap7_2p_inout.txt antes de ejecutar el programa.

3) Programa que escriba las temperaturas de los días de una semana en un archivo con nombre cap7_3p_out.txt.

4) Programa que lea las temperaturas de los días de la semana del ejercicio anterior y escriba en otro archivo con nombre cap7_4p_out.txt:

– Temperatura media = valor con dos cifras decimales.

– Temperatura mínima = valor y nombre del día de la semana correspondiente.

– Temperatura máxima = valor y nombre del día de la semana correspondiente.

¿Cómo se añadiría la información anterior en el archivo de entrada?

5) Programa que lea las temperaturas de los días de la semana del ejercicio anterior y escriba en otro archivo con nombre cap7_5p_out.txt las temperaturas ordenadas de menor a mayor, con su valor y nombre del día de la semana correspondiente.

6) Una empresa dispone de una base de datos de sus trabajadores. Se dispone de la siguiente información para este mes:

Nombre completo, Sueldo (Euros), Pagado (SI/NO), Antigüed (años).

El archivo que recoge esta información se llama cap7_6p_in.txt.

El aspecto del archivo es el siguiente:

‘Ana García’ 1200.95 ‘SI’ 5

‘Roberto Iglesias’ 1800.59 ‘NO’ 10

…

Se desea saber:

– ¿Cuál es la antigüedad de cualquier empleado en la empresa?

– ¿A cuánto asciende la cantidad pagada a todos los trabajadores este mes?

– ¿Cuántos trabajadores no han sido pagados aún este mes? Escribe sus nombres en un archivo que se llame cap7_6p_out.txt junto con sus sueldos. (La lista debe de estar ordenada alfabéticamente).

Escribe un programa Fortran que permita responder a todo esto, mostrando los resultados de los dos primeros apartados por monitor. Usa técnicas de programación modular en la elaboración del programa.

BIBLIOGRAFÍA

1) Stephen J. Chapman. *Fortran 90/95 for Scientists and Enginners*. 2nd Edition, International Edition, McGraw-Hill, 2004.

2) Michael Metcalf, John Reid, Malcolm Cohen. *Fortran 95/2003 explained*. Oxford University Press, 2005.

3) L.R. Nyhoff, S.C. Leestma. *Introduction to Fortran 90 for Engineers and Scientists*. Prentice Hall, 1997.

4) Elliot B. Koffman, Frank L. Friedman. *FORTRAN with engineering applications*. 5th Edition, Addison-Wesley, 1997.

5) F. García Merayo, V. Martín Ayuso, S. Boceta Martínez y E. Salete Casino. *Problemas resueltos de programación en Fortran 95*. Thomson, 2005.

6) F. García Merayo. *Lenguaje de programación Fortran 90*. Paraninfo, 1999.

7) Sebastián Ventura Soto, José Luis Cruz Soto, Cristóbal Romero Morales. *Curso básico de Fortran 90*. Universidad de Córdoba, 2000.

8) L.A.M. Quintales, L. Alonso Romero. *Programación en Fortran 90*. Curso en Internet. Dpto. Informática y Automática, Universidad de Salamanca, 2000.

9) E. Alcalde, M. García. *Metodología de la programación. Aplicaciones en Basic, Pascal, Cobol*. Serie: Informática de gestión. McGraw-Hill, 1989.

10) Clifford A. Pickover. *El prodigio de los números. Desafíos, paradojas y curiosidades matemáticas*. Ciencia Ma Non Troppo, 2002.