NOTAS ELEMENTALES SOBRE PROGRAMACION EN FORTRAN

JUSTO R. PEREZ CRUZ

Departamento de Física Fundamental y Experimental Electrónica y Sistemas

Facultad de Física

Universidad de La Laguna.

NOTAS ELEMENTALES SOBRE PROGRAMACION EN FORTRAN

1.-INTRODUCCION

El FORTRAN fue el primer lenguaje de programación desarrollado. Su nombre (FORmula TRANslation) indica claramente que su filosofía es proveer al ordenador de un traductor para realizar calculos matemáticos. El uso del FORTRAN se ha extendido ampliamente en la comunidad científica, y a pesar de la aparición de otros lenguajes, más versátiles y que permiten una mayor facilidad en la programación, el FORTRAN a través de sus sucesivas versiones continúa siendo hoy día ampliamente utilizado.

Una de las características del FORTRAN es la compatibilidad de las distintas versiones con las anteriores, por tanto desarrollaremos en estas notas una serie de instrucciones básicas centradas fundamentalmente en el FORTRAN 77 con el objeto de tener una amplia compatibilidad con los distintos compiladores utilizados comúnmente en los ordenadores personales.

Para comenzar diremos que un programa de ordenador consta fundamentalmente de una serie de órdenes escritas en un lenguaje clave, que permiten a la máquina leer un conjunto de datos, operar con ellos y devolver al usuario los resultados requeridos.

A cada una de las órdenes escritas en un lenguaje las denominaremos SENTENCIAS.

La labor de realizar un programa tiene tres etapas:

- La edición o escritura de las órdenes o sentencias de que consta el programa, que se realiza habitualmente con un editor de textos.
- La compilación, es decir la transformación de las órdenes escritas en texto a código máquina.
- La ejecución o utilización del programa.

Nos centraremos en estas notas en la edición del programa. La compilación y la ejecución dependen del sistema operativo y de la versión del compilador FORTRAN instalado en la máquina que se esté usando.

A la hora de confeccionar un programa debemos plantearnos tres cuestiones

- ¿Cómo la máquina puede recibir información?
- ¿Cómo la procesa?
- ¿Cómo nos comunica los resultados?

Veamos las herramientas que permiten desarrollar este esquema en su forma más sencilla:

2.-SENTENCIAS DE ENTRADA SALIDA (I/O)

2.1.-READ

La sentencia que permite al ordenador leer la información que queremos suministrarle es la sentencia

READ

Por ejemplo, para leer dos números de la pantalla podemos poner

READ (*,*) A,B

donde hemos indicado que los números leídos van a ser almacenados en las variables A y B respectivamente.

Para comprender mejor el sistema de almacenamiento, así como la forma de operar podemos hacernos el esquema mental de que el ordenador posee un conjunto de "cajas", en cada una de las cuales almacena un dato, poniéndole en la tapa una etiqueta para su identificación (los nombres de las variables).

En el ejemplo anterior A,B, serían las etiquetas de identificación de los lugares donde han sido guardados los datos.

Por otra parte, el ordenador interacciona con un conjunto de unidades lógicas (pantalla, ficheros, etc.) a cada una de las cuales identifica con un número, lo que le permite leer o escribir en cada una de ellas.

Así la sentencia READ puede ponerse como

READ(n1,*)

donde n1 debe ser un número que indica la unidad lógica de la cual debe ser leída la información.

Por defecto, la pantalla es la unidad * con lo que

READ(*,*) A,B

indica al ordenador que debe leer las variables A y B de la pantalla.

En general, la sentencia READ tiene la forma

READ(n1,n2)

donde n1 indica de donde deben ser leídos los datos y n2 está asociado a una sentencia FORMAT que indica como deben ser leídos y que veremos más adelante.

2.2.-WRITE

La sentencia que permite al ordenador escribir la información que ha procesado es la sentencia

WRITE

El funcionamiento de la sentencia WRITE es similar al de la sentencia READ estudiada anteriormente. Así para escribir las variables A y B leídas anteriormente en la pantalla pondremos

WRITE(*,*) A,B

Donde el * denota el lugar de escritura que la máquina toma por defecto (la pantalla) y el segundo que escriba los datos tal como los tiene almacenados.

Para escribir en una unidad distinta pondremos

WRITE(n1,*)

La forma general de la sentencia WRITE es

WRITE(n1,n2)

donde n1 indica la unidad en que deben ser escritos y n2 la sentencia FORMAT que indica con que formato deben ser escritos los datos (número de decimales, posición en pantalla, separación, etc.)

3.-SENTENCIAS ARIMETICAS

Para operar con dos variables, (por ejemplo sumar las dos leídas anteriormente) pondremos

C=A+B

Hay que destacar que el significado del signo = no tiene en FORTRAN el mismo sentido que en al arimética normal. Aquí tiene el sentido de colocar el resultado de la operación que se realiza a la derecha de la variable indicada a la izquierda.

Por lo tanto es absolutamente correcto poner

A=A+B

lo cual tendría el sentido siguiente:

"Suma los números contenidos en las variables A y B y coloca el resultado en la variable A".

En este caso el contenido anterior de la variable A desaparece siendo "machacado" por la nueva incorporación.

Análogamente pueden realizarse las siguientes operaciones

Diferencia	A-B
Producto	A*B
Cociente	A/B
Potencia	A**B

así como funciones trigonométricas, logarítmicas, exponenciales y algunas de otro tipo como valor absoluto, parte entera, máximo entre varios números, etc como por ejemplo

Seno	**SIN**(X)
Coseno	**COS**(X)
Tangente	**TAN**(X)
Logaritmo	**LOG**(X)
Raiz Cuadrada	**SQRT**(X)
Exponencial	**EXP**(X)
Valor absoluto	**ABS**(X)
Parte entera	**INT**(X)

.....

las restantes deben ser consultadas en la bibliografía.

Al combinar varias de estas operaciones, hay establecido un orden de prioridad en el sentido de que se ejecutan primero las operaciones encerradas entre paréntesis, luego las funciones, la potenciación, el producto o el cociente y la suma o diferencia.

Para operaciones como producto o cociente y suma o diferencia de igual prioridad se ejecutan ordenadas de izquierda a derecha.

Así por ejemplo

```
2.*X+Y**5
```

representa la magnitud $2x+y^5$

mientras que la expresión $2(x+y)^5$ debe ponerse como

```
2.*(X+Y)**5.
```

4.-ESCRITURA Y FINAL DEL PROGRAMA

Los programas en FORTRAN deben ser escritos a partir de la columna 7 dejando las cinco primeras para colocar los números de sentencia cuyo cometido se verá más adelante y la columna 6 para poder partir una línea en caso de que sea muy larga.

Para comenzar la escritura del programa puede utilizarse (no es obligatorio) la sentencia

PROGRAM

Que indica inicio del programa y a su vez su nombre

PROGRAM *el nombre*

y para terminar debe hacerse con las sentencias STOP, que indica el final de la ejecución y END el final físico del programa.

```
STOP
END
```

Un ejemplo

Supongamos un programa simple en el que queremos que el ordenador lea dos números, los sume y nos indique la suma, podemos poner:

```
PROGRAM SUMA
READ(*,*) A,B
C=A+B
WRITE(*,*)C
STOP
END
```

Nótese que cada una de las sentencias están escritas a partir de la columna 7. (no es estrictamente necesario comenzar en la 7 puede utilizarse cualquier otra columna posterior, como por ejemplo la que se obtenga con el tabulador del teclado.)

La comunicación interactiva y la lectura del listado del programa puede facilitarse si en la sentencia WRITE colocamos un texto entre comillas simples, este aparecerá en la pantalla, así podríamos poner:

```
PROGRAM SUMA
WRITE(*,*)'ESCRIBE DOS NUMEROS'
READ(*,*) A,B
C=A+B
WRITE(*,*)'LA SUMA DE ',A,' Y ',C, ' ES ',C
STOP
END
```

Al ejecutar el programa aparecerá en pantalla:

```
ESCRIBE DOS NUMEROS

45  33        (Escribimos los números)

LA SUMA DE 45 Y 33 ES 78
```

5.-COMENTARIOS

En el listado del programa pueden incluirse textos aclaratorios sobre la forma de confección del programa o por cualquier otra circunstancia para lo cual basta colocar una C en la primera columna añadiendo el texto a continuación. Así en el ejemplo anterior

```
C----ESTE ES UN PROGRAMA DE SUMA DE DOS NUMEROS
      PROGRAM SUMA
C----PIDE QUE SE DEN DOS NUMEROS
      WRITE(*,*) 'ESCRIBE DOS NUMEROS'
      READ(*,*) A,B
      C=A+B
C----AHORA ESCRIBE LOS NUMEROS Y SU SUMA
      WRITE(*,*)'LA SUMA DE ', A,' Y ', C, ' ES ',C
      STOP
      END
```

Hay que notar que estos textos o comentarios no tienen ninguna misión a la hora de ejecutar el programa. Son meramente aclaratorios a la hora de obtener un listado del mismo.

6.-TIPOS DE VARIABLES

Los datos almacenados en cada variable no tienen porque ser de la misma naturaleza, pueden ser reales, enteros, complejos o bien necesitan de un número elevado de cifras decimales. Asimismo para cada tipo de variables el ordenador no reserva el mismo espacio de memoria, ni opera con ellas a la misma velocidad. Por lo tanto debe especificarse de que tipo son las variables que se utilizan en el mismo.

Las sentencias que permiten esto son:

REAL

Indica que las variables que se indican a continuación son variables reales tomando para ellas 7 cifras significativas. Así por ejemplo al poner:

REAL X,Y,NUMER,DATO

indicamos que las variables X,Y,NUMER,DATO son reales con un número máximo de 7 cifras significativas.

Análogamente se procede con las sentencias:

INTEGER para variables enteras

COMPLEX para variables complejas

DOUBLE PRECISION para variables reales en doble precisión es decir con 13 cifras significativas.

Todas estas sentencias deben ser escritas al inicio del programa.

Si las únicas variables utilizadas son reales y enteras, puede prescindirse de estas sentencias mediante una declaración implícita, es decir teniendo en cuenta la primera letra de la variable correspondiente.

Así si la variable comienza por las letras:

I,J,K,L,M,N

el compilador asume por defecto que son enteras, siendo reales el resto, excepto que se haya indicado lo contrario con alguna de las sentencias especificadas anteriormente.

Hay que tener precaución al asignar valores reales a variables enteras, ya que la máquina trunca la parte decimal, asimismo hay que señalar que las operaciones entre variables enteras se convierte en un número entero, lo cual puede dar lugar a errores difíciles de detectar.

Así por ejemplo al poner

IX=0.5

Guarda en la variable IX el valor 0

Y al poner

A=1/2

guarda en la variable A el valor 0 ya que el resultado de la operación entre los números enteros 1 y 2 es 0.5 que al convertirlo en entero se transforma en 0

Por este motivo es aconsejable el uso del punto decimal y de variables reales en aquellas operaciones que lleven cocientes, como por ejemplo

A=1./2.

7.-SENTENCIAS IF Y GOTO

El lenguaje FORTRAN ejecuta las sentencias en el orden en el que están escritas. Sin embargo, hay varias formas de romper este orden, bien continuando la ejecución en alguna sentencia posterior o anterior, o bien eligiendo entre ejecutar un grupo de sentencias u otro, según el valor que tomen una o varias variables.

7.1.-IF

Comenzaremos analizando esta segunda alternativa, la cual es permitida utilizando la sentencia IF combinada con la sentencia ELSE. Tiene la forma siguiente:

IF(expresión lógica) **THEN**

(Bloque 1 de sentencias)

ELSE

(Bloque 2 de sentencias)

END IF

Si la expresión lógica es verdadera, ejecuta el bloque de sentencias 1 y si es falsa el bloque de sentencias 2.

Cuando la decisión a tomar radica entre ejecutar un bloque de sentencias o no, no es necesaria la sentencia ELSE, con lo que el conjunto toma la forma:

IF(expresión lógica) **THEN**

(Bloque de sentencias)

END IF

De forma que si la expresión lógica es verdadera ejecuta el bloque de sentencias y si es falsa no.

Las expresiones lógicas más usadas son de tipo comparativo, entre dos variables y combinaciones de las mismas.

Así para indicar A<B expresamos

A.**LT**.B indicando **LT** less than (menor que)

Asimismo se tiene

LE	(less or equal)	(menor o igual)
EQ	(equal)	(igual)
GT	(greater than)	(mayor que)
GE	(greater or equal)	(mayor o igual)
NE	(non equal)	(distinto)

Para combinar estas expresiones pueden utilizarse los conectores

AND	y lógico
OR	o lógico
NOT	negación lógica

Así por ejemplo

A.**LT**.B.**OR**.C.**GT**.D (A<B ó C>D)

NOT.(A.**EQ**.B) (No A=B)

7.2 GO TO

Para continuar la ejecución del programa en otra sentencia que no sea la siguente en el orden de escritura del programa puede utilizarse la sentencia

GO TO

la cual tiene la forma

GOTO n1

donde n1 es un número entero que indica el número de sentencia en la cual se quiere continuar la ejecución, el cual debe ser escrito como ya hemos señalado en las primeras 5 columnas de la sentencia correspondiente.

Es de señalar que estos números no tienen por que aparecer ordenados en el listado del programa, ni corresponderse con el número de orden de las sentencias en el mismo.

Un Ejemplo

Como ejemplo ilustrativo de la utilización de estas sentencias consideraremos un programa para resolver ecuaciones de segundo grado:

```
C---RESOLUCION DE ECUACIONES DE SEGUNDO GRADO
      PROGRAM ECUACIONES2
30    WRITE(*,*)' ESCRIBE LOS COEFICIENTES A,B,C'
      READ(*,*) A,B,C
C---CALCULO DEL DISCRIMINANTE DE LA ECUACION
      DISC=B*B-4.*A*C
C---DECISION, RAICES REALES     O COMPLEJAS
      IF(DISC.LT.0.)THEN
            WRITE(*,*)' RAICES COMPLEJAS '
      ELSE
```

```
                WRITE(*,*)' RAICES REALES'
                X1=(-B+SQRT(DISC))/(2.*A)
                X2=(-B-SQRT(DISC))/(2*A)
                WRITE(*,*)'X1= ',X1,' X2= ',X2
            END IF
C---DECISION TERMINAR EL PROGRAMA O RESOLVER OTRA ECUACION
            WRITE(*,*)' PARA TERMINAR PON UN CERO, OTRA ECUACION PON
     &      UN UNO'
            READ(*,*)K
            IF(K.EQ.1)THEN
                GO TO 30
            END IF
            STOP
            END
```

Observese que en el último bucle si la variable leída K es un 1 el programa continúa su ejecución en la sentencia con el número 30 al principio del programa.

Nótese que en la sentencia

```
    WRITE(*,*)' PARA TERMINAR PON UN CERO, OTRA ECUACION PON
&   UN UNO'
```

se ha continuado en la línea siguiente colocando el símbolo & en la columna 6 (cualquier otro símbolo es igualmente válido). Aunque la longitud de la línea depende del compilador muchos de ellos truncan la línea en la columna 72 por lo que no es recomendable escribir líneas demasiado largas que puedan rebasar esta columna.

El último bucle IF puede abreviarse de la forma

```
IF(K.EQ.1) GOTO 30
```

lo cual puede hacerse siempre que el IF decida la ejecución o no de una sola sentencia, por ejemplo:

```
IF(X1.LT.X2) XMAYOR=X2
```

La sentencia IF puede combinarse con la sentencia ELSE para tomar decisiones múltiples en el sentido siguiente:

```
IF(expresión lógica 1) THEN
    (bloque de sentencias 1)
ELSE IF(expresión lógica 2) THEN
    (bloque de sentencias 2)
ELSE IF (expresión lógica 3) THEN
    (bloque de sentencias 3)
ELSE
    (bloque de sentencias 4)
END IF
```

Si la expresión lógica 1 es verdadera ejecuta el bloque de sentencias 1, si ésta es falsa pero es cierta la expresión lógica 2 ejecuta el bloque 2 y así sucesivamente y si todas son falsas ejecuta el bloque de sentencias 4 . Este secuencia puede prolongarse si se considera necesario.

8.-SENTENCIA DO

El cálculo de expresiones del tipo

$$\sum_{i=1}^{n} i(i+1)$$

o de constitución similar, y el trabajo con vectores o matrices, requiere la confección de algoritmos en los que una serie de operaciones se repiten sucesivamente mientras una variable se modifica en una cantidad constante.

La sentencia DO permite la creación de un algoritmo de este tipo de una forma sencilla y simplificada.

Esta sentencia puede escribirse de varias maneras, la forma más clásica es la siguiente

```
DO X=X1,X2,X3
        (bloque de sentencias)
END DO
```

El bloque de sentencias es ejecutado sucesivamente desde que la variable X (que puede ser entera o real) toma el valor X1 hasta el valor X2, incrementándose a cada paso en X3 (que puede ser positivo o negativo). En el caso de que X3 tome el valor 1 no es necesario ponerlo.

Así el cálculo de la expresión tomada com ejemplo podemos llevarlo a cabo de la forma siguiente:

```
C-----CALCULO DE LA SUMA DE LOS  TERMINOS DE UNA SUCESION
      PROGRAM SUMATORIO
      SUMA=0.
            DO N=1,20
                  SUMA =SUMA+N*(N+1)
            END DO
      WRITE(*;*) 'LAS SUMA ES ',SUMA
      STOP
      END
```

9.- ARRAYS

Análogamente al cálculo que planteabamos anteriormente podemos pretender calcular

$$\sum_{i=1}^{n} a_i a_{i+1}$$

Donde los términos a sumar son los correspondientes de una determinada sucesión o los componentes de un vector o matriz.

Este cálculo y similares puede simplificarse notablemente haciendo uso del tratamiento del FORTRAN para las variables subindicadas o ARRAYS.

Para utilizar una variable subindicada o array basta declararla como tal y referirnos a ellas de la forma

X(I) para un solo subíndice

X(I,J) para dos subíndices y así sucesivamente hasta un máximo de 7

Para declarar una variable como subindicada (array) es necesario conocer el número de subíndices y el valor máximo que toma cada uno de ellos. La forma más conveniente es hacerlo utilizando las sentencias REAL, INTEGER, DOUBLE PRECISION, COMPLEX, según que la variable sea de un tipo u otro, especificando entre paréntesis los valores máximos de los subíndices. Por ejemplo

```
REAL A(20),X(10,15)
DOUBLE PRECISION PAR(30,10,5),VEC(10)
```

indican que la variable A tiene un subíndice que puede tomar un valor máximo de 20, la variable X tiene dos de valor máximo 10 y 15 siendo ambas variables reales. Análogamente para las de DOUBLE PRECISION.

Un ejemplo

Como ejemplo ilustrativo del uso de estas sentencias consideraremos un programa para calcular el producto de dos matrices A de 3x2 y B de 2x4.

```
C---PROGRAMA PARA CALCULAR EL PRODUCTO DE MATRICES
C---DEFINICION DE LOS ARRAYS
      PROGRAM PRODUCTO
      REAL A(3,2),B(2,4),C(3,4)
C---LECTURA DE MATRICES
      DO  I=1,3
          DO J=1,2
          WRITE(*,*) 'ESCRIBE EL ELEMENTO A ',I,J
          READ(*,*)A(I,J)
          END DO
      END DO
      DO I=1,3
          DO J=1,4
          WRITE(*,*) 'ESCRIBE EL ELEMENTO B ',I,J
          READ(*,*) B(I,J)
```

```
            END DO
        END DO
C----CALCULO DEL PRODUCTO DE LAS MATRICES
        DO  I=1,3
            DO  J=1,4
                SUM=0
                DO  K=1,2
                SUM=SUM+A(I,K)*B(K,J)
                C(I,J)=SUM
                END DO
            END DO
C---ESCRITURA DE LA MATRIZ PRODUCTO C
        WRITE(*,*) ((C(I,J),I=1,3),J=1,4)
        STOP
        END
```

Obsérvese en este ejemplo la estructura anidada de los bucles DO, es decir en el bloque de sentencias que se repite con una sentencia DO, existe otro bucle que se repite completo a cada paso del más externo.

La sentencia WRITE en la escritura de la matriz presenta una simplificación propia de la escritura de un array, la expresión

```
WRITE(*,*) ((C(I,J),I=1,3),J=1,4)
```

es equivalente a

```
DO J=1,4
    DO  I=1,3
    WRITE(*,*) C(I,J)
END DO
END DO
```

10.-SUBPROGRAMAS. FUNCTIONS, SUBROUTINES.

Es frecuente que en diferentes etapas de un programa, distintas variables repitan exactamente las mismas operaciones, o bien que una parte de un programa sea un cálculo típico que ya ha sido previamente programado en otro. Aunque estas etapas se pueden realizar por separado cada vez que se necesite, el programa puede simplificarse notablemente utilizando lo que llamamos subprogramas. Es decir, cada vez que se precise realizar las operaciones citadas se transmite el control a un programa accesorio que las ejecuta, con lo cual sólo habrá que escribirlas una sola vez.

Puede suceder que el resultado de las operaciones citadas sea simple (un solo valor) o bien múltiple (distintos valores y/o operaciones de lectura o escritura).

En el primero de los casos el cálculo tiene el sentido de una función de un solo valor y el FORTRAN nos permite programarla como tal utilizando lo que llamamos un subprograma FUNCTION.

10.1.-FUNCTION

Es un programa escrito aparte del programa principal que tiene el siguiente esquema

FUNCTION nombre (variable1,variable2,...)

(sentencias que definen el valor de la función)

RETURN

END

Es de tener en cuenta que entre las sentencias que definen el valor de la función ésta debe aparecer de la forma

NOMBRE=operación o variables que la definen

Para comprender mejor como se confecciona comentaremos el siguiente ejemplo:

Ejemplo

Supongamos un programa que calcula el número **e** mediante su desarrollo en serie

$$e = \sum_{n=1}^{\infty} \frac{1}{n!}$$

A cada paso es necesario calcular el factorial lo que haremos mediante un subprograma FUNCTION.

```
C---SUBPROGRAMA CALCULO DEL FACTORIAL
      FUNCTION FAC(X)
      FAC=1
      IF(X.GT.1)THEN
            DO  I=2,N
                  FAC=FAC*I
            END DO
      END IF
      RETURN
      END
C----PROGRAMA CALCULO DEL NUMERO E CON SUBPROGRAMA
      PROGRAM NUMEROE
      REAL J,IMAX
      WRITE(*,*)'CUANTOS SUMANDOS TOMAMOS PARA LA SERIE?'
      READ(*,*)IMAX
      E=1.
      DO J=1,IMAX
            E=E+1./FAC(J)
      END DO
      WRITE(*,*) 'VALOR CALCULADO DE E= ',E
      STOP
      END
```

Observemos que en la primera línea del subprograma se coloca FUNCTION y el nombre del mismo seguido entre paréntesis de las variables que van a utilizarse (que pueden ser varias).

```
FUNCTION FAC(X)
```

Al calcular el valor que toma la función se trata el nombre como si fuera una variable

```
FAC=1

.....

FAC=FAC*I
```

Al utilizar el valor calculado por el subprograma dentro del programa principal se utiliza el subprograma como si fuera una función

```
E=E+1./FAC(J)
```

El nombre de las variables que utiliza el subprograma no tiene por que coincidir con las del subprograma, aunque si lo deben hacer en orden, número y tipo (entero, real,...)

Hay que tener en cuenta que en el nombre de la FUCTION también se sigue el convenio de la primera letra para el tipo de las variables que en este caso corresponde al tipo de función (entera, real, etc.)

En el ejemplo anterior, aunque el factorial de un número entero siempre es entero, hemos conservado FAC como real debido a que puede almacenar un número mayor que si fuese entera.

Este criterio puede cambiarse con las sentencias REAL, INTEGER, etc. para lo cual debemos poner por ejemplo:

```
INTEGER FUNCTION FAC(N)
```

10.2.-SENTENCIA DEFINICION DE FUNCION

Cuando el subprograma FUNCTION puede expresarse en una sóla sentenciaÉeste se puede escribir en el interior del programa principal antes de la primera sentencia ejecutable, de la forma

```
Nombre(variables)=operación
```

y ser utilizado en el interior de un programa de forma análoga a una FUCTION.

A este tipo de procedimiento lo denominamos **SENTENCIA DEFINICION DE FUNCION**

Por ejemplo definiendo al inicio del programa

```
DIST(A,B)=SQRT(A*A+B*B)
```

puede utilizarse dentro del mismo como por ejemplo

```
IF(DIST(A,B).LT.1) WRITE(*,*)'PUNTO DENTRO DEL CIRCULO UNIDAD'
```

10.3.-SUBROUTINE

Cuando precisamos un subprograma en el que los resultados son varios, o ejecuta otras operaciones como escribir o leer, debemos utilizar lo que llamamos una

SUBROUTINE.

Esta tiene la forma

SUBROUTINE nombre (lista de variables)

(Bloque de sentencias)

RETURN

END

En este caso el nombre sólo tiene la misión de identificarla desde el programa principal. Para ello, cuando se precise realizar el cálculo que efectúe la subroutine se utiliza la sentencia

CALL

que tiene la forma

CALL nombre (lista de variables)

Esta lista de variables debe coincidir en tipo y número con la declarada al comienzo de la subroutine (argumentos de la subroutine) y consta de las variables que va a utilizar el subprograma como datos para sus cálculos y las variables donde deben almacenarse sus resultados.

Hay que tener en cuenta que en el momento de llamar a la subroutine (ejecutar la sentencia CALL) los argumentos de ésta toman los valores asignados desde el programa principal a los argumentos de la sentencia CALL , en este momento se ejecutan las sentencias especificadas en la subrutine y al finalizar ésta, los argumentos de la sentencia CALL , toman en el programa principal los valores resultado de la ejecución de la subroutine.

Para ilustrar su utilización analicemos el siguiente ejemplo.

Ejemplo

Confeccionaremos un programa para leer 10 grupos de 5 datos, escribirlos ordenados, seleccionar el mayor de cada grupo y escribir los máximos ordenados a su vez, seleccionando el máximo de todos.

```
C-----PROGRAMA DE LECTURA Y ORDENACION, EJEMPLO DE SUBROUTINE
C-----SUBROUTINE QUE ORDENA UN CONJUNTO DE N DATOS POR EL
C-----METODO DE LA BURBUJA
      SUBROUTINE ORDEN(X,SUP,N)
      REAL X(N)
      DO  I=N,2,-1
      DO J=1,I-1
      IF(X(J).GT.X(J+1))THEN
            PASO=X(J+1)
```

```
            X(J+1)=X(J)
            X(J)=PASO
        END IF
        END DO
        END DO
        SUP=X(N)
        RETURN
        END
C*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C----PROGRAMA PRINCIPAL
        PROGRAM TABLAS
        REAL A(5),B(10),MAYOR,MAXIM
C----COMIENZO DEL BUCLE QUE PASA DE GRUPO A GRUPO
        DO  J=1,10
C----LECTURA DE CADA GRUPO
        DO K=1,5
            WRITE(*,*)'ESCRIBE EL ELEMENTO ',K,'DEL GRUPO ',J
        READ(*,*) A(K)
        END DO
C----ORDENACION DE CADA GRUPO
        INDA =5
        CALL ORDEN(A,MAYOR,INDA)
C----ESCRITURA DE CADA GRUPO ORDENADO Y ALMANCENAMIENTO
C----DEL MAYOR EN B
        B(J)=MAYOR
        WRITE(*,*)(A(J),J=1,5)
C----FIN DEL BUCLE
        END DO
C----ORDENAMIENTO DE LOS MAYORES DE CADA GRUPO
        INDB=10
        CALL ORDEN(B,MAXIM,INDB)
        WRITE(*,*)'LOS MAYORES DE CADA GRUPO ORDENADOS SON'
        WRITE(*,*)(B(J),J=1,10)
        WRITE(*,*)'EL MAS GRANDE DE TODOS ES ',MAXIM
C-----FIN DEL PROGRAMA
        STOP
        END
```

Es conveniente a la hora de escribir una subroutine dar una explicación detallada de los argumentos ya que son su única interacción con el programa principal, salvo otras estrategias puntuales como son la utilización de sentencias COMMON, que no trataremos aquí. Así por ejemplo en el caso anterior.

SUBROUTINE ORDEN(X,SUP,N)

- Proposito: Ordena los elementos de un conjunto y calcula su máximo.
- Argumentos:
- X, Array real de dimension N. En entrada (al llamar a la suborutine) debe contener los elementos a ordenar y en salida (después de la ejecución de la sentencia CALL) contiene los elementos ordenados.
- SUP , variable real. En salida contiene el máximo del conjunto a ordenar.
- N Variable entera. En entrada contendrá el número de elementos a ordenar (dimensión del array X) , no es alterado en la salida.

Vemos que los argumentos utilizados en las llamadas al subprograma

CALL ORDEN(A,MAYOR,INDA)

CALL ORDEN(B,MAXIM,INDB)

verifican estos condicionantes.

Es de notar que en la subroutine orden se ha utilizado una dimensión variable N para el array X, esto sólo puede hacerse en una subroutine y no en el programa principal. Podríamos haber utilizado una dimensión fija para la variable X en este caso debe coincidir con la dimensión del correspondiente argumento en el programa principal.

11.-ESCRITURA Y LECTURA EN FICHEROS

Cuando la lectura o escritura no tienen lugar en pantalla se utilizan los ficheros. El programa interpreta una serie de unidades lógicas, (de las cuales la pantalla se toma por defecto *) cada una de las cuales identificada con un número sobre las que se escribe o de las cuales lee.

Para utilizar un fichero es necesario abrirlo previamente, darle un nombre e identificarlo con un número.

Esto se realiza con la sentencia

OPEN

que tiene la forma:

OPEN(UNIT=*u*,FILE='*nombre*',STATUS='*status*')

u es un número que le asignamos para la identificación del fichero

'*nombre*' es una variable alfanumérica que indica la identificación del fichero (su nombre)

'*status*' es una variable alfanumérica (character) que puede ser 'OLD' si el fichero ya existe o 'NEW' si necesita ser creado o 'UNKNOWN' que toma el fichero ya existente si lo encuentra y si no abre uno nuevo. Si el campo 'STATUS' no se incorpora al programa toma por defecto 'UNKNOWN'

Así por ejemplo

OPEN(**UNIT=**10,**FILE=**'DATOS.DAT',**STATUS=**'NEW')

abre el fichero DATOS.DAT y le asigna la unidad lógica 10.

En el caso de que el fichero se encuentre en otro directorio o unidad de disco de la que se esté trabajando debe ser indicado en el nombre así por ejemplo

OPEN(UNIT=10,FILE='A:\TRABAJOS\DATOS.DAT',STATUS='NEW')

abre el fichero DATOS.DAT en el subdirectorio TRABAJOS de la unidad A.

Para cerrar un fichero se utiliza la sentencia

CLOSE(UNIT=u)

donde u indica el número de la unidad lógica. Esta sentencia es opcional, ya que la sentencia END cierra todos los ficheros abiertos.

Para escribir o leer en un fichero, basta con indicarlo en el primer índice de las sentencias READ o WRITE correspondiente.

Así por ejemplo,

READ(10,*)A,B

Lee las variables A,B del fichero al que se le haya asignado la unidad 10.

WRITE(15,*)C,X

Escribe las variables C,X en el fichero al que se le haya asignado la unidad 15.

12.-FORMATOS

Hasta la fecha, hemos utilizado las sentencias READ y WRITE con formato libre, es decir, el ordenador lee los datos como los encuentra en la pantalla o el fichero y los escribe tal como los tiene en su memoria interna.

A veces es interesante fijar el tipo de las variables a escribir y el número de decimales correspondiente, su localización dentro del fichero, así como insertar mensajes para hacer tablas, etc.

Para este fin se utiliza la sentencia

FORMAT

que tiene la forma

n1 **FORMAT**(d1,d2,d3,..,dn)

donde n1 es el número de sentencia que permite identificarla con la sentencia READ o WRITE correspondiente, y d1,...,dn son símbolos alfanuméricos que nos permiten establecer la naturaleza y tamaño de la variable correspondiente en orden con las que aparecen en la sentencia READ o WRITE asociadas.

De estos símbolos, que llamaremos descriptores, los más usados son los siguentes:

In donde n es un número entero, indica que la variable correspondiente es entera y con un número máximo de n dígitos (incluido el signo si es negativa)

Por ejemplo I3 indica que es un número entero para el que se reservan tres espacios, es decir contenido entre -99 y 999

Fn1.n2 indica que la variable es real, siendo n1 el número total de caracteres que ocupa, incluido el signo si es negativo y el punto decimal, y n2 es el número de cifras decimales.

Por ejemplo F9.3 indica que es un número para el que se reservan 9 espacios y que se escribe con 3 cifras decimales es decir de -9999.999 a 99999.999

En1.n2 para variables escritas en forma exponencial, donde n1 es el numero de espacios ocupados por la variable incluidos el signo el punto decimal la letra E el signo del exponente de 10 y n2 indica el número de cifras decimales.

Así la escritura de

$6.7x10^{-43}$

En la forma -6.7E-43 requiere un formato E8.1 .

Dn1.n2 para variables en doble precisión teniendo n1 y n2 el mismo significado que para el descriptor D.

An para escribir variables character (alfanuméricas), siendo n el número total de de caracteres de la misma.

Hay otra serie de descriptores que no se asocian a ninguna variable y que tienen diferentes cometidos, así por ejemplo:

nX inserta n espacios entre la variable escrita (leída) anteriormente y la próxima.

Tn continúa escribiendo (leyendo) a continuación de la columna n

TRn salta a la derecha n columnas a partir de la actual.

TLn salta a la izquierda n columnas a partir de la actual.

'texto' las comillas sirven para escribir el texto que encierran.

/ la barra inclinada tiene la misión de continuar la escritura(lectura) en la línea siguiente.

Con la excepción de los formatos T y / todos los anteriores pueden repetirse colocando antes de la especificación un número entero que actúa como multiplicador, así

```
10      FORMAT(' ',F5.2,F5.2,F5.2)
```

es equivalente a

```
10      FORMAT(' ',3F5.2)
```

También podemos colocar un factor multiplicativo delante de una serie de especificaciones entre paréntesis, así

```
10      FORMAT(' ',F5.2,2X,F5.2,2X,F5.2,2X)
```

es equivalente a

```
10      FORMAT(' ',3(F5.2,2X))
```

Una forma abreviada de combinar la sentencia READ y WRITE con la sentencia FORMAT es sustituir el segundo carácter en estas por el formato directamente entre comillas y paréntesis.

Así

```
        WRITE(*,11)B,IB

11      FORMAT(' ',25X,F5.2,/,27X,I2)
```

Puede sustituirse por

```
12      WRITE(*,(' ',25X,F5.2,/,27X,I2))B,IB
```

13.-OTRAS SENTENCIAS

En el lenguaje FORTRAN existen otra serie de sentencias así como distintas opciones en las sentencias ya estudiadas, que dejamos para su consulta en la bibliografía especializada. Algunas sentencias u opciones que pudieran ser de utilidad son las siguientes:

13.1.-PAUSE

La sentencia **PAUSE** detiene momentáneamente la ejecución del programa, que puede ser continuada dando a la tecla ENTER o cualquier otra del teclado (depende del compilador) resulta a veces de utilidad para ir siguiendo una salida de datos que se hace demasiado larga y llena la pantalla.

13.2.-LECTURA CON FIN DE FICHERO

A veces en la lectura de un fichero, no se conoce previamente su tamaño, por lo que es necesario enviar a leer hasta que este se termine. Esto se puede hacer con una opción del READ de la forma

```
        READ(nf,*,END=n1)lista de variables
```

Lee de la unidad nf una lista de variables hasta que el fichero se agota y entonces continua en la sentencia n1. Así por ejemplo

```
READ(11,*,END=15) X,Y
```

lee del fichero al que se le ha asignado la unidad 11 dos variables en columna X,Y y cuando este se acaba continua en la sentencia con la etiqueta 15.

13.3.-FUNCIONES Y CIFRAS EN DOBLE PRECISION

Cuando se trabaja en doble precisión hay que tener en cuenta que es necesario definir las variables numéricas como tales así como utilizar las funciones específicas para doble precisión, esto debe ser consultado en la tabla correspondiente aunque el procedimiento común es añadir una D al principio de su nombre. Así por ejemplo

```
DOUBLE PRECISION X,Y

X=1.D00

Y=DSIN(X)
```

Sería el procedimiento correcto para especificar el valor numérico 1 a X y calcular su seno si X es una variable de doble precisión.

14.-CONSIDERACIONES FINALES.

Estas notas son sólo una primera aproximación al lenguaje FORTRAN, el lenguaje de programación más extendido entre la comunidad científica. La base está tomada del Fortran 77 compatible con las versiones posteriores Fortan 90, etc. .Hemos tratado de describirlas en un lenguaje sencillo y de uso fácil para simplificar la tarea de hacer los primeros programas. Sin embargo esto hace que no estén exploradas todas las posibilidades del lenguaje y no se hayan mencionado un numeroso grupo de sentencias y posibilidades, por lo cual es imprescindible tras pasar una primera etapa consultar un texto o manual de FORTRAN más especializado y explorar los recursos que podemos encontrar en el manual de instalación del compilador con el que se esté trabajando.

15.-BIBLIOGRAFIA.

En la biblioteca de la Facultad de Física puede encontrarse una amplia bibliografía sobre lenguajes de programación, en particular sobre fortran, sin embargo una forma más cómoda puede ser la utilización de internet.Una colección de manuales de fortran puede encontrarse en

[1] http://www.todoprogramas.com/manuales/programacion/fortran/