COMPUTACIÓN 2003 – ASTRONOMIA

PRACTICA 6

1. FORTRAN permite que un *procedimiento* (función o subrutina) sea utilizado como argumento de otro procedimiento. Esta característica es útil para llevar a cabo el mismo conjunto de operaciones sobre diferentes funciones.

   Por ejemplo, considérese el problema de tabular una función. Sea f una función definida sobre un intervalo cerrado $[a, b]$. La *tabla* de f sobre una red de puntos $\{x_i\}_{i=0}^{n}$ en $[a, b]$ es el vector $(n + 1)-$dimensional

   $$\text{tab}(f) = (f(x_0), f(x_1), \ldots, f(x_n))$$

   Por simplicidad consideraremos una red *equispaciada* de puntos $x_i = x_0 + ih, i = 0, 1, \ldots, n$

   a) Escriba una subrutina TABLA que construya la tabla TABF de una función FUNC sobre una red equispaciada de N+1 puntos con extremos XMIN, XMAX.

   ```
   SUBROUTINE TABLA (FUNC,XMIN,XMAX,N,TABF)
   ...
   END
   ```

   b) La subrutina TABLA puede ser usada para obtener la tabla de una función simplemente dando el nombre de ella en el primer argumento de la llamada. El único requisito es que el nombre de cada función usada como argumento sea especificado en una sentencia INTRINSIC o EXTERNAL, según sea ésta una función implementada internamente en la biblioteca de funciones del FORTRAN o un procedimiento de función construida por nosotros.

   Construya un programa CURVAS que utilice la subrutina TABLA del punto anterior para obtener la tabla de funciones de *a)* $\sin(x)$, *b)* $\cos(x)$, *c)* $f(x) = \cos(x) + 0.5 \sin(\sqrt{x})$.

   Nota: El nombre de una función intrínseca utilizada como argumento debe ser el nombre específico y no el nombre genérico. Esta es la única circunstancia en la cual debería utilizar los nombres específicos de las funciones internas.

2. Un nombre de variable en un subprograma no tiene relación con una variable del mismo nombre en el programa que realiza la llamada. Las dos variables se asignan a posiciones diferentes de memoria y los cálculos que afectan a una no afectan a la otra. Así, las variables de igual nombre en un programa principal y un subprograma son totalmente independientes (aún cuando pueden asociarse mediante la lista ficticia de argumentos de la subrutina y la lista de argumentos de la llamada de la subrutina).

   Ahora bien, existen muchas casos en que es conveniente y deseable tener un nombre de variable en ambos programas (principal y subprograma) que se refiera a la misma posición de memoria sin hacer que aparezca en la lista de argumentos. Esto se logra con un área de almacenamiento especial llamada COMMON. Al utilizar el almacenamiento COMMON permitimos la comunicación entre unidades independientes del programa sin utilizar una lista de argumentos en la definición de un subrutina y en una proposición CALL de llamada a la subrutina.

   Existen dos formas de almacenamiento COMMON: en blanco y con nombre. Un bloque de almacenamiento COMMON en blanco se define con la sentencia:

COMMON *lista*

donde *lista* es una lista de nombres de variables o de nombres de arreglos comunes a más de una unidad de programa. Así, si en el programa principal declaramos

COMMON A,B,C(10)

y en el subprograma:

COMMON A,X,Y(10)

las variables A,B,C(10) en el programa principal hacen referencia a los mismos datos que A,X,Y(10) (en ese orden) en el subprograma.

Un bloque de almacenamiento COMMON con nombre permite definir distintos bloques de almacenamiento, cada uno de ellos identificados con un nombre. La forma de declaración consiste en presentar un nombre de bloque común encerrado entre diagonales antes de la lista de nombre de las variables que se declaran comunes:

COMMON /*nombre del bloque común*/ *lista*

Si no aparece un nombre entre las diagonales, las variables a continuación corresponden a un COMMON en blanco. En otras palabras, // es lo mismo que un COMMON en blanco. Veamos dos ejemplos:

COMMON /A/ X,Y,Z /B/ M,N,F(10)

El bloque COMMON con nombre A contiene X,Y,Z y el bloque COMMON con nombre B contiene M,N,F(10), siendo el último un arreglo unidimensional de diez componentes.

COMMON //R,S

las variables R y S están en COMMON en blanco.

Teniendo en cuenta lo anterior esquematice la ubicación de cada variable en cada zona común e indique que variables son conocidas para cada unidad del programa, siendo en el programa principal

COMMON XX,Y(3)
COMMON /ALFA1/ A1 /ALFA2/ A1,A2,A3

el subprograma 1,

COMMON Z,A1(3)
COMMON /ALFA1/ X

el subprograma 2,

COMMON X,A2(3)
COMMON /ALFA2/ U,V,W

3. Si se desea inicializar las variables y elementos de arreglos que se encuentran en almacenamiento COMMON con nombre utilizando la sentencia DATA, ésta debe colocarse en un *subprograma de bloque de datos* diseñado para tal fin. Tal subprograma se define mediante la sentencia

BLOCK DATA *nombre*

(donde *nombre* es un nombre simbólico optativo) seguido por la declaración COMMON del almacenamiento con nombre que quiere inicializarse, la inicialización con la sentencia DATA y cualquier otra sentencia que no sea ejecutable. Como todo subprograma debe finalizar con la sentencia END.

Por ejemplo, si la unidad de programa utilizando el COMMON con nombre es

```
SUBROUTINE CALC
COMMON /BD/ALFA(50),BETA,A
...
RETURN
END
```

un subprograma bloque de datos adecuado para inicializar el COMMON con nombre es

```
BLOCK DATA
COMMON /BD/ALFA(50),BETA,A
DATA ALFA/50*0/BETA/1.0/
END
```

La razón para la existencia de un subprograma bloque de datos es que el COMMON con nombre puede ser utilizado por varias unidades de programa, por lo que no es posible confiar en que una de las unidades del programa realice la inicialización de los datos. El subprograma bloque de datos define la inicialización globalmente para todo el conjunto de programas y subprogramas.

Teniendo en cuenta lo anterior, defina un subprograma BLOCK DATA que inicialice los valores pedidos para los siguientes COMMON con nombres:

a) Siendo

```
COMMON /BA1/ A(10),I,J/BA2/B(5)
```

inicialice a cero todos los componentes de los arreglos A y B.

b) Siendo

```
LOGICAL A,B
COMMON /BA/A,B,C,J,K
```

inicialice A = .TRUE., B=.FALSE, J=1, K=1.

4. Supongamos que tenemos un programa grande que consta de varias subrutinas compartiendo bloques COMMON entre ellas y el programa principal. Así en cada una de las unidades del programa tenemos declaraciones COMMON idénticas (si utilizamos los mismos nombres para las variables). Supongamos ahora que queremos eliminar o agregar una o más variables en la lista. Para hacer esto debemos localizar en el archivo fuente cada una de las lineas que declaran al COMMON en las distintas unidades y modificarla en forma apropiada. Si el número de subrutinas o el número de variables es grande, este procedimiento es penoso y con frecuencia conduce a errores.

   Para garantizar que todas las unidades de programa se suplirán con las mismas definiciones y declaraciones COMMON, reúna todas ellas en un archivo separado, digamos `common.h`, y utilice la sentencia

INCLUDE 'common.h'

en cada subprograma que lo necesite. Con esto solo tendrá que modificar las sentencias en un único lugar, evitando errores y complicaciones.

5. Las variables de un subprograma (subrutina o función) que *no* se encuentran en un COMMON o en la lista de argumentos pierden sus valores cuando el control regresa al programa que realizó la llamada. La sentencia SAVE permite conservar los valores del subprograma entre sucesivas llamadas del subprograma.

   Utilizando el hecho de que

   $$n! = n(n-1)!$$

   construya una función que permita calcular los factoriales de los *sucesivos* números naturales en forma eficiente. Utilice la sentencia DATA para inicializar el o los valores apropiados y la sentencia SAVE para reternerlos.