9.1.- PROGRAMACION MODULAR:

Por regla general un programa de ordenador de calculo numérico tiene por objeto resolver un problema matemático que de manera manual nos llevaría excesivo tiempo de calculo Cuando se desarrolla un código de estas características, es muy común encontrarse con grupos de operaciones que han de ser realizadas a lo largo del programa varias veces. Supongamos que estamos trabajando con matrices y que precisamos realizar su suma varias veces. Este hecho nos llevaría a repetir reiteradas veces un conjunto de sentencias encargadas de sumar matrices en diferentes puntos del código. Para evitar este trabajo de repetición FORTRAN permite establecer ese conjunto de sentencias de manera única y utilizarlas cuando sea necesario sin precisar su escritura reiterada, para ello será preciso emplear las FUNCIONES o las SUBRUTINAS dependiendo de los requerimientos operacionales. Ambas sentencias serán descritas en este capítulo.

Esta metodología de programación se denomina modular y nos obliga a adquirir de manera más intensa una conducta de programación indispensable para que el trabajo de programación sea eficiente. Esta conducta se basa en los siguientes pasos:

Paso 1. *Antes de iniciar la codificación en el ordenador es preciso analizar y definir previamente el problema a resolver los mas claramente posible.*

Paso 2. *Elaborar un diseño completo de la estructura prevista para la resolución del problema, poniendo especial interés en la posibilidad de modularizar la estructura general.*

Paso 3. *Establecimiento de los requerimientos (variables, constantes, dimensiones, etc.) y de la estructura (lazos, bifurcaciones, funciones, subrutinas, etc.) del programa.*

Paso 4. *Comenzar la codificación en el ordenador siempre y cuando los pasos anteriores estén claros.*

De este modo la tarea de programación se verá respalda por un estudio previo que va a garantizar el ahorro de trabajo y tiempo finales.

9.2.- FUNCIONES

Una de las manera que FORTRAN permite establecer una estructura modular es mediante el uso de las funciones. Este tipo de elementos son módulos separados del programa principal y cuando son llamados devuelven el resultado al programa principal. Para que la función al ser llamada se ejecute correctamente es preciso escribir su nombre desde el programa principal, junto con los argumentos que precise para su resolución. El nombre de la función cuando ésta es llamada debe formar parte de una expresión como operando en el programa principal.

En FORTRAN existen tres tipo de funciones:

1. FUNCIONES INTRINSECAS: Este tipo de funciones ya fueron descritas en el capitulo 3, en él se mostró su utilización y características. Las funciones intrínsecas están definidas en FORTRAN y no permiten su modificación.

2. FUNCIONES SENTENCIA: Este tipo de funciones que a continuación se describirán, permiten establecer su estructura operacional y se ubican como su propio nombre indica en una sentencia del programa principal.

3. FUNCIONES EXTERNAS: De similares características a las anteriores pero con muchas mas posibilidades, se ubican aparte del programa principal

9.2.1.- FUNCIONES SENTENCIA

Para entender la utilidad de este tipo de funciones es preciso analizar el siguiente conjunto de sentencias FORTRAN:

```
R1=5.6*S + A/2.5
.................
R2=5.6*F + B/2.5
.................
R3=5.6*G + C/2.5
R4=5.6*H + D/2.5
```

Puede observarse que se trata de una misma operación repetida en la que intervienen dos constantes 5.6 y 2.5 y en la que van cambiando determinadas variables.

Matemáticamente la expresión general de la función anterior puede ser expresada de la siguiente manera:

$$F(X,Y)=5.6*X + Y/2.5$$

Según lo anterior la expresión anterior puede ser expresada como una única sentencia FORTRAN y su resultado es un único valor, todo ello nos permite establecer una FUNCION SENTENCIA FORTRAN, la cual se expresaría de la siguiente manera:

```
CALCU(X,Y)=5.6*X+Y/2.5
```

Con la inclusión de la sentencia anterior quedaría definida la función sentencia por lo que a continuación podría ser llamada en cualquier parte del programa de la siguiente manera:

```
R1=CALCU(S,A)
..............
R1=CALCU(F,B)
..............
..............
R1=CALCU(G,C)
R1=CALCU(H,D)
```

Es importante decir que en cada llamada a CALCU los argumentos X,Y son sustituidos por los correspondientes a la llamada, es decir para la primera llamada X será sustituida por S e Y por A para realizar los cálculos. A los argumentos que interviene en la definición de la función como son en este caso X e Y se les denomina ***argumentos ficticios*** y son variables independientes de las demás que aparecen en el programa, tienen un carácter local. Por ello podrán existir otras variables con el mismo nombre en el programa principal sin peligro de conflicto. Por otra parte, a los argumentos que son pasados a la función como S, A, F, B, etc. se les denomina ***argumentos verdaderos***. En este punto es preciso resaltar que la lista de argumentos ficticios contenidos en la declaración de la función y verdaderos contenidos en cada una de las llamadas se deberán corresponder en numero, orden y tipo.

De una manera general las FUNCIONES SENTENCIAS tienen la siguiente configuración:

SENTENCIA	DESCRIPCION
NOMBRE(V1, V2,.....VN) = EXPRESION	**NOMBRE**: Será el nombre simbólico mediante el cual realizaremos la llamada a esta función durante el programa. **V1,.....VN**: Nombre de las variables que intervienen el la función, son los argumentos ficticios. **EXPRESIÓN**: Será una expresión FORTRAN aritmética, lógica o carácter y no deberá poseer ninguna variable tipo array como por ejemplo A(1), B(2,3).

	La sentencia en la que se define una FUNCION SENTENCIA debe ir ubicada inmediatamente después del bloque correspondiente a la declaración de variables y siempre antes de la primera sentencia ejecutable.

A continuación se muestra el ejemplo anterior implementado en un programa completo:

```
program ejemplo1

real a,b,d,calcu
calcu(x,y)=5.6*x+y/2.5

read(*,*)a,b
d=calcu(a,b)
write(*,*)d
end
```

Obsérvese la ubicación de la sentencia correspondiente a la FUNCION SENTENCIA así como su declaración previa como función real.

EJERCICIO 9.1

- Copiar el programa anterior y ejecutarle. Una vez comprobado su correcto funcionamiento modificar la expresión de la función sentencia para que calcule lo siguiente:

  calcu(x,y,z)=(5.6*X+Y/2.5)*sen(z)

EJERCICIO 9.2

- Crear un programa que implemente mediante FUNCIONES TIPO SENTENCIA las siguientes exprexiones:

  F(x,y)=2*X/3*Y
  G(t,z)=t**2 + z

  Siendo z = F(x,y)

- Las variables de entrada y salida se leerán a través de pantalla.

9.2.2.- FUNCIONES EXTERNAS

Este tipo de funciones se utilizan cuando las expresiones que deseamos incluir en un modulo de programa de tipo función requieren mas de una sentencia y cuando los valores que deseamos recibir de la sentencia función pueden ser mas de uno. A diferencia de las funciones sentencia las externas no van incluidas en el programa principal sino que deben ser compiladas con independencia del programa principal y sus variables son locales por lo que no entraran en conflicto con variables del mismo nombre incluidas en el programa principal.

Para entender su funcionamiento podemos ver un ejemplo en el que se utiliza una función externa para calcular la suma de dos números introducidos por pantalla:

```
      program ejmplo2
      real a,b,d
      read(*,*)a,b
      d=suma(a,b)
      write(*,*)d
      end

      function suma(a,b)
      suma=a+b
      return
      end
```

EJERCICIO 9.3

- Copiar el programa anterior y ejecutarle.

El ejemplo anterior constituye un programa en el que existen dos módulos uno el programa principal denominado `ejemplo2` y otro la FUNCION EXTERNA de nombre `suma.` Este último módulo correspondiente a la suma podría estar en otro fichero independiente con extensión *.FOR de tal modo que habría sido preciso compilar por un lado el programa principal y por otro la función externa y proceder después a su lincado, finalizando con la creación de un único ejecutable. Este división por ficheros como se ha podido comprobar no es obligatoria y únicamente se emplea para programas con un grado de modularidad elevado.

De una manera general las FUNCIONES EXTERNAS tienen la siguiente configuración:

SENTENCIA	DESCRIPCION
FUNCTION NOMBRE(V1, V2,.....VN) **NOMBRE = valor** **V1= valor** (OPCIONAL) **RETURN** **END**	**NOMBRE**: Será el nombre simbólico mediante el cual realizaremos la llamada a esta función en el programa principal. **V1,.....VN**: Nombre de las variables que intervienen el la función, son los argumentos ficticios. El nombre simbólico asignado a la función será una variable de tipo global y por lo tanto será conocida por todos los programas que forman parte de la estructura modular diseñada. Por otro lado dentro de la función externa deberá aparecer la sentencia: **NOMBRE = valor** en la que se hará la asignación del valor correspondiente. La función externa devolverá al programa principal la variable NOMBRE con el valor asignado, pero también puede devolver a través de los argumentos ficticios V1.....VN valores asignados a estos argumentos durante la ejecución de la función externa. Este tipo de asignación **V1= valor** es opcional. En resumen, una FUNCION EXTERNA puede emplearse para devolver un único valor como en el caso de las funciones sentencia o para devolver un numero determinado de valores en forma de argumentos de salida a través de los argumentos V1.....VN. Por último, recalcar que la función externa deberá incluir al menos una sentencia RETURN y la sentencia END.

En el caso de que la función posea bifurcaciones operacionales podremos incluir una sentencia RETURN en cada

bifurcación con el fin de volver al programa principal. A continuación se muestra un ejemplo:

```
      program ejemplo3

      real a,b,d

      read(*,*)a,b
      d=suma(a,b)
      write(*,*)d,a,b
      end

      function suma(a,b)
      suma=a+b
      if (suma.lt.0) then
      a=10
      b=10
      return
      else
      a=0
      b=0
      return
      endif
      end
```

Obsérvese detenidamente la inclusión de dos sentencia RETURN una en cada bifurcación y obsérvese también que la función suma no solo devuelve el valor de la suma de a y b, sino que además dependiendo del resultado de a+b, asigna nuevos valores a las argumentos a y b que serán devueltos al programa principal.

EJERCICIO 9.4

- Copiar el programa anterior y ejecutarle. Obsérvese detenidamente la salida de las variables d,a,c para valores de a+b positivos y negativos.

EJERCICIO 9.5

- Desarrollar un programa que mediante una función externa calcule la integral de una función $f(x)=3x^2+4$ empleando el método de los trapecios:

$$\int_a^b f(x)dx = \frac{h}{2}(f_0 + 2f_1 + 2f_2 + \ldots\ldots\ldots\ldots + 2f_{n-1} + f_n)$$

$$h = \frac{(b-a)}{n} \quad , \quad f_0 = f(a) \quad , \quad f_1 = f(a+h) \quad , \quad f_i = f(a+h)$$

9.3.- SUBRUTINAS

Los dos tipos de funciones presentados anteriormente son válidos para determinadas situaciones en las que es conveniente realizar una estructura modular del código, sin embargo presentan limitaciones si las necesidades y complejidad de las operaciones aumenta. Por ejemplo si precisamos un modulo que calcule el producto entre dos matrices de NxN elementos precisaremos de un subprograma capaz de devolver como salida los NxN elementos correspondientes a la matriz resultante. Estos requerimientos no pueden ser satisfechos por las funciones estudiadas pero sí por las SUBRUTINAS FORTRAN.

Una SUBRUTINA FORTRAN es en realidad un modulo o subprograma completamente autónomo con las siguientes características:

- Se compila independientemente, puede estar ubicada en un fichero independiente de extensión *.FOR, o bien al igual que las funciones externas pueden estar ubicadas a continuación de la sentencia END del programa principal, con lo que únicamente se compilará un único fichero.
- Su estructura interna puede presentar varias sentencias al igual que el programa principal.
- Puede recibir y devolver al programa principal varios datos, uno solo o ninguno siendo estos de todo tipo.
- Incluye las sentencia RETURN y END, al igual que las funciones externas.

El modulo correspondiente a una SUBRUTINA tiene la siguiente configuración:

SENTENCIA	DESCRIPCION
SUBROUTINE NOMBRE(V1,V2,.....VN) **.....** **.....** **.....** **RETURN** **END**	**NOMBRE**: Será el nombre simbólico mediante el cual realizaremos la llamada a la SUBRUTINA en el programa principal. A diferencia de las funciones a NOMBRE no se le asigna ningún valor y únicamente sirve para identificar la subrutina **V1,.....VN**: Nombre de las variables que intervienen el la subrutina, son los argumentos ficticios. Por regla general, parte de estos argumentos serán empleados como argumentos de entrada (DATOS) y parte de ellos como argumentos de salida (RESULTADOS). Por último, recalcar que la subrutina deberá incluir al menos una sentencia RETURN y la sentencia END.

Resaltar que en la sentencia se escribe SUBROUTINE y no SUBRUTINA.

Por otro lado el programa principal incluirá la llamada correspondiente a la subrutina, esta llamada tiene la siguiente configuración:

SENTENCIA	DESCRIPCION
CALL NOMBRE((V1',V2',.....VN')	**NOMBRE**: Será el nombre simbólico con el que hemos denominado a la subrutina. **V1',.....VN'**: Nombre de las variables que intervienen el la subrutina, son los argumentos verdaderos. Por regla general, parte de estos argumentos serán empleados como argumentos de entrada (DATOS) y parte de ellos como argumentos de salida (RESULTADOS). En este punto es preciso resaltar que la lista de argumentos ficticios y verdaderos se deberán corresponder en numero, orden y tipo. V1 ← *mismo tipo* → V1' V2 ← *mismo tipo* → V2' ..... VN ← *mismo tipo* → VN' Al igual que con las funciones pueden existir varias llamadas a los largo del programa principal.

A continuación se mostraran dos ejemplos de cómo emplear SUBRUTINAS para establecer una programación modular:

En primero lugar calcularemos el volumen de las esferas de radios 1,2,3,4,5,6,7,8,9 y 10 metros.

```
      program esferas

      real vol
      integer i

      do i=1,10
      call volumen(i,vol)
      write(*,*)'radio=',i,'volumen=',vol
      enddo
      end

      subroutine volumen(i,vol)
      real vol,pi
      integer i
      parameter (pi=3.14)
      vol=(4.0/3.0)*pi*i**3.0
      return
      end
```

EJERCICIO 9.6

- Copiar el programa anterior y ejecutarle.

Del ejemplo anterior es preciso observar lo siguiente:

- Las variables que utiliza la subrutina hay que declararlas todas, al igual que en el programa principal.
- Se ha optado por mantener el mismo nombre de los argumentos ficticios y verdaderos, sin embargo esto nos es necesario, por lo que podríamos haber optado por emplear en la subrutina otro nombres de variables, manteniendo siempre el mismo tipo de variable. Es decir la subrutina podría haberse escrito de la siguiente manera:

```
subroutine volumen(j,esp)
real espa,pi
integer j
parameter (pi=3.14)
esp=(4.0/3.0)*pi*j**3.0
return
end
```

- Son diez las llamadas que realizan a la subrutina, en cada una de ellas los argumentos van actualizándose.

Ahora se mostrara la cualidad de las subrutinas de utilizar como argumentos ficticios arrays lo que supone una gran ventaja a al hora de diseñar un modulo encargado de procesar determinada información. Para ello se mostrara un ejemplo de cómo sumar matrices empleando un modulo tipo subrutina.

```
      program matsum1

      integer i,j,n
      real a(5,5),b(5,5),c(5,5)

      write(*,*)'dame el orden de las matriz cuadradas'
      read(*,*)n
      do i=1,n
       do j=1,n
        write(*,*)'matriz A elemento',i,j
```

```
        read(*,*)a(i,j)
       enddo
      enddo
      do i=1,n
       do j=1,n
        write(*,*)'matriz B elemento',i,j
        read(*,*)b(i,j)
       enddo
      enddo

      call suma(a,b,c,n)

      do i=1,n
       do j=1,n
        write(*,*)'matriz C elemento',i,j,c(i,j)
       enddo
      enddo
      end

      subroutine suma(mat1,mat2,mat3,n)

      integer n,i,j
      real mat1(5,5),mat2(5,5),mat3(5,5)

      do i=1,n
       do j=1,n
        mat3(i,j)=mat1(i,j)+mat2(i,j)
       enddo
      enddo
      return
      end
```

Del ejemplo anterior es preciso observar lo siguiente:

- Cuando se desea incluir un array completo en los argumentos ficticios y verdaderos únicamente se define su nombre sin incluir sus dimensiones, la subrutina conocerá sus dimensiones puesto que son variables que están declaradas en la propia subrutina.

- En caso de querer enviar un único elemento del array la llamada y la subrutina deberían ser como se muestra a continuación:

```
call suma(a(1,2),b,c,n)

subroutine suma(elem,mat2,mat3,n)
real elem,mat2(5,5),mat3(5,5)
```

  Tener presente que aquí únicamente se envía el elemento (1,2) de a y que la subrutina lo debe recibir como un simple real

- Obsérvese que ha sido preciso incluir en los argumentos la variable n para conocer que longitudes han de tener los lazos i y j que realizan la suma en la subrutina. Por otro lado es evidente es que n no deberá sobrepasar el valor de 5 puesto que las matrices tienen dimensión 5X5.

El ejemplo anterior es una opción muy correcta de enfocar el proceso de suma de matrices mediante subrutina, sin embargo puede ser mejorado sensiblemente mediante una configuración que evita errores de dimensionamiento de matrices. El siguiente ejemplo muestra esta mejora.

```
      program matsum2
```

```
      integer maxn,i,j
      parameter (maxn=5)
      real a(maxn,maxn),b(maxn,maxn),c(maxn,maxn)

      write(*,*)'dame el orden de las matriz cuadradas <',maxn
      read(*,*)n
      do i=1,n
       do j=1,n
        write(*,*)'matriz A elemento',i,j
        read(*,*)a(i,j)
       enddo
      enddo
      do i=1,n
       do j=1,n
        write(*,*)'matriz B elemento',i,j
        read(*,*)b(i,j)
       enddo
      enddo

      call suma(a,b,c,n,maxn)

      do i=1,n
       do j=1,n
        write(*,*)'matriz C elemento',i,j,c(i,j)
       enddo
      enddo
      end

      subroutine suma(mat1,mat2,mat3,n,maxn)

      integer n,maxn,i,j
      real mat1(maxn,maxn),mat2(maxn,maxn),mat3(maxn,,maxn)

      do i=1,n
       do j=1,n
        mat3(i,j)=mat1(i,j)+mat2(i,j)
       enddo
      enddo
      return
      end
```

EJERCICIO 9.7

- Copiar el programa anterior y ejecutarle.

Del ejemplo anterior es preciso observar lo siguiente:

- Ahora las dimensiones de todas las matrices están controladas por un parámetro incluido en la sentencia PARAMETER el cual fija el tamaño máximo de las matrices.
- MAXN ahora es un valor fijo que el programador establece en el programa principal.
- MAXN se envía a la subrutina como argumento para así controlar el dimensionamiento de las matrices MAT1, MAT2 y MAT3.
- Si el programador cambia el valor de MAXN en el programa principal en la sentencia PARAMETER automáticamente redimensionará todas las matrices dependientes de este parámetro, esto supone un ahorro de tiempo muy importante.

EJERCICIO 9.8

- Modificar el ejemplo 9.5 para que el cálculo se realice mediante una subrutina en vez de con una función externa.

EJERCICIO 9.9

- Desarrollar un programa que mediante una subrutina calcule la intersección entre dos rectas en R^2. Se recomienda incluir varias sentencias RETURN dependientes de la existencia de solución.