Capítulo 1

Estructuras de Control - DO

1.1. Sentencia DO

La sentencia **DO** permite la repetición de un cálculo modificando uno de sus parámetros en forma controlada. Para realizar la tarea se establece un valor de inicio, un valor final y un paso que delimita los valores en los cuales el parámetro tomará valores. Pero mejor, veámoslo con un ejemplo: quiero calcular la suma de la siguiente serie:

$$S = \sum_{i=1}^{N} 1/i^2$$

donde **N** podría ser incluso número muy grande. Como vemos el término $1/i^2$ se calcula repetidamente al hacer las cuentas. Si quisiera hacer un programa tendría que repetir una y otra vez este término cambiando el valor de **i**, lo cual sería muy un trabajo muy arduo además de tedioso. Sin embargo, la fórmula que define la serie es compacta y para la variable **i** se establece que toma todos los valores desde **i = 0** hasta **i = N**
Lo que se hace con la sentencia **DO** es escribir la fórmula en una manera muy parecida a la notación matemática usual, es decir la que se utiliza para describir la sumatoria como en este caso.

Un programa que haga este cálculo se podría escribir con mucha simplicidad y quedaría:

```
C Programa para realizar el calculo de la suma de la Serie finita i**2

      Program Suma

      write(*,*) '¿Cuantos términos quiero sumar?'
      read(*,*) N

      suma=0.
      DO i=1, N
        x = i   ! i es entero. No quiero que las operaciones se realicen en números enteros[1]
        suma = suma + 1/ x**2
      ENDDO
```

[1] Se podría modificar esta sentencia y escribirla como **suma = suma +1 /float(i)**2**. la orden float convierte a flotante el número que está en **i** y evita el problema. Otra manera es convertir el 1 en 1. (el punto decimal lo convierte en número real y todas las operaciones que se realicen considerarán que los números son reales).

```
write(*,*) 'La suma de la serie es =', suma
end
```

En este programa, vemos que se ingresa el valor de **N**, que es el número que indica la cantidad de términos de la serie que se van a sumar. Luego se asigna el valor 0 en la variable **suma**[2]. Esta variable irá acumulando la suma parcial de los términos calculados. Luego se ejecuta la orden **DO** que actúa sobre la variable **i** ¿Cómo lo hace?

La variable **i** va a tomar primero el valor 1, porque en el **DO** se indica que es es el comienzo, y su último valor será el valor de **N**. Como no se indica el paso este será 1. Entonces **i** comenzará valiendo 1 y se realizará el cálculo hasta el **ENDDO**, luego con **i**=2 y se calculará de nuevo, luego las operaciones se repetirán con **i**=3, así hasta lleguemos a que **i** tome el valor **N**. En el próximo ciclo **i** valdrá **N+1**, entonces al haber pasado el valor límite ya no continuará el cálculo y continuará ejecutando las sentencias después del **ENDDO**. Que para nuestro caso en particular es escribir en pantalla el resultado de la suma de la serie.

Hay varias cosas para señalar, la primera es que valor final de **i** al terminar será **N+1**, ya que el sistema sumará un 1 a **i** y descubrirá que ya se pasó del valor límite que es **N**, por lo cual no continuará el cálculo.

La segunda, es que el tiempo que la computadora tardará en hacer el cálculo es lineal con **N** para este caso en particular. Para un **N** más grande, más tarda el programa. Si determino el tiempo que tarda para N = 100 con este tiempo podré estimar el tiempo que tardará para cualquier otro valor de N. Ya que: **tiempo** $\propto$ **N**

Y la tercera, es que el programa fue escrito a un nivel abstracción en el cual sólo hay que indicar cuantos términos de la serie quiero y el sistema los calcula sin modificar el código. Es decir, mi programa sólo depende de ingresar el **N** y da lo mismo si la serie tiene pocos términos o muchos, no hay que modificar el programa ni re-compilarlo. Como punto importante a señalar, es que muy fácilmente puedo construir un programa cuya ejecución supere cualquier tiempo razonable para que la computadora lo finalice, o peor no termine nunca.

1.1.1. Formalidad y usos de la sentencia DO

La sentencia **DO** se escribe:

DO variable=inicio, final, paso

En inicio, final y paso podemos poner un número, una variable (se utiliza el valor que se guarda en esa variable, como en el ejemplo anterior) o una fórmula (la cual se calculará y el resultado se usará como el valor en cuestión). En general se considera, que se puede poner en el inicio, final y paso expresiones matemáticas. Un valor constante es la expresión matemática más simple. Si no ponemos el paso, este se considerará 1, este es el valor por default[3]
El paso puede ser negativo, en cuyo caso el inicio debe ser una número mayor al numero final y se hará un cálculo con números que van decreciendo. Inicio, final y paso pueden ser números reales,

[2] La mayoría de los compiladores construyen el código ejecutable de tal manera que haya un cero inicialmente en todas sus variables, por alguna razón misteriosa el gfortran no lo hace y debemos asignar un 0 a esta variable, ya que podría haber un valor en ella producto de lo que ha quedado en memoria de un programa anterior.

[3] Default es una palabra anglosajona que se usa en computación para indicar valores que ya han sido predeterminados en el sistema. En este caso si yo no pongo paso por default el paso del DO es paso=1.

pero hacer esto es desaconsejado porque por pérdida de decimales podría en algún caso que se realice un loop de menos o de más de lo que se pensó hacer. Por ejemplo, se programa el final con el número real 4.0 pero el calculo da que la variable del DO en vez de 4.0 da 3.9999999 entonces se repetiría un loop de más que el programador nunca quizo hacer y quizás le arruine el cálculo.
La variable que es el parámetro de un **DO** no puede ser modificada por ningún cálculo dentro del propio **DO**, si puede una vez que el **DO** ha finalizado. Sólo son posibles las modificaciones indicadas en la sentencia **DO** a través de la definición (inicio, final y paso) que se le da a la variable. Intentar cambiar el valor de este variable será indicado como error y en la mayoría de los casos por el propio compilador.

Puede existir un o varios **DO**s dentro de otro, pero sobre una variable diferente, ejemplo:

```
DO i=1, N
  DO j=1, N
  Varias sentencias con cálculo
  ENDDO
ENDDO
```

Una sentencia equivalente al **DO** existe en todos los lenguajes de computación, muchas veces con otro nombre, pero su uso es similar. Una cantidad importante de lenguajes la escriben como **for** y con parámetros similares al Fortran. En algunos lenguajes el paso no solamente puede ser aditivo, sino que ademas hay opciones para que sea geométrico (multiplicativo), que siga una ley de potencias o que sea logarítmico.

Dato importante: toda fórmula matemática a calcular del tipo sumatoria, productoria, operaciones con subíndices como por ejemplo cálculos con matrices, claramente es un **DO** obligado al programarla en Fortran.

Por ejemplo, el segmento de un programa que calcule el factorial de un número N y guardarlo como resultado en una variable llamada F sería:

```
⋮
  F=1.
  DO i=1,N
    F = F * i
  ENDDO
⋮
```

Donde ahora estamos usando a la variable F para primero cargarle un 1. (el elemento neutro del producto), luego los resultados parciales y por último quedaría el factorial del número.

1.1.2. Ejemplos del uso del DO

Calcular la tabla de numérica que se produce de la siguiente fórmula:

F = 2n + m y **n** toma valores en el rango **n**=2,4,6,8...20 y **m** los toma tal **m**=1,2,3,4,...,**n**
Vemos que n toma los pares hasta el 20 (de esta información se deduce el inicio, final y el paso de la secuencia), mientras que **m** comienza en 1, tiene paso 1, pero finaliza en **n**. Con esta información se debería hacer:

```
Program Tabla
integer F
write(*,*) ' N M F'
DO n=2,20,2
  DO m=1,n
    F= 2*n+m
    write(*,*) n,m,F
 ENDDO
 ENDDO
END
```

Como se puede apreciar en este ejemplo, el **DO** más externo controla la variable **n**, la cual es parte de la definición del rango de números del **DO** mas interno (el de **m**). Por ello, cuando crece **n**, crece también la cantidad de loops que el **DO** mas interno está obligado a realizar. Esto es un ejemplo de la importante variedad de situaciones que se pueden programar con esta sentencia.

1.1.3. El problema de la pérdida de decimales

El problema de la pérdida de decimales debido a que los números no tienen infinitos decimales en su representación binaria en la computadora lo hemos comentado pero no hemos visto ejemplos donde esta situación nos pueda perjudicar. Este problema puede tener un efecto negativo en cálculos largos o que se realicen sobre programas donde sus algoritmos propagan inadecuadamente los errores (por ejemplo, sistemas donde las perturbaciones crecen en magnitud a medida que se realizan más operaciones matemáticas).
El programa que calcula la serie que discutimos al principio de este capítulo puede servirnos para visualizar el efecto que se produce al perder constantemente los decimales menos significativos en cada operación matemática que se hace. En un principio, esta pérdida puede aparentar ser una pérdida muy menor, su acumulación como un error de cálculo sistemático puede afectar los resultados finales. Si bien, también hay que considerar que en la mayoría de los cálculos este efecto no suele ocurrir, pero no por eso hay que dejar de estar conscientes de su existencia. Ya que cuando ocurre podemos estar en el caso de realizar un cálculo muy complejo o largo y por lo tanto obtener resultados incorrectos al final de este.
El programa anterior calcula la sumatoria de la serie con término $1/i^2$ y al tener en el denominador un término cuadrático este provoca que al crecer el valor de i los términos de la serie sean números cada vez más pequeños. Vamos a aprovechar esta situación para visualizar el problema. Como la serie no es mas que una suma, es equivalente calcularla de dos maneras: sumándola desde el principio (desde i = 1, hasta N, con paso 1) o haciéndola desde el final (empieza en i = N, con paso -1, y termina cuando i = 1 como valor final).
En Fortran podemos hacer ambos cálculos con sólo cambiar sentencia DO del programa que vimos como ejemplo anteriormente. Es decir podríamos calcular usando:
DO i=1,N o **DO i=N,1,-1** y ambos métodos deberían dar el mismo resultado. Pero además, hay que recordar que la serie converge al infinito:

$$S = \sum_{i=1}^{\infty} 1/i^2 = \pi^2/6 \sim 1.64493406684822643$$

Con lo cual tenemos el valor al cual converge la serie en el infinito y por lo tanto una referencia con la cual comparar los números obtenidos con distintos N. Con la ventaja de que usaremos N grandes y entonces los resultados deberían parecerse a este número. Con este valor puedo estimar la precisión del resultado que estoy obteniendo y al mismo tiempo comparar este resultado contra los

dos métodos de cálculo. La idea es que al sumar más términos de la serie, veamos si los errores en los cálculos aumentan por tener una cantidad finita de decimales o no, ya que podemos contrastar el resultado contra la suma exacta. También podremos ver si hay diferencia entre ambos métodos: la suma creciente y la suma decreciente.
Veamos en la tabla 1.1 los resultados de las corridas del programa para distintos valores de N en ambos cálculos, es decir con *i* creciente hasta *N* y con *i* decreciendo desde *N*.

N	Resultado *i* creciendo	Error	Resultado *i* decreciendo	Error
100	1.63498402	9.95016098E-03	1.63498390	9.95028019E-03
1000	1.64393485	9.99331474E-04	1.64393449	9.99689102E-04
10,000	1.64472532	2.08854675E-04	1.64483404	1.00135803E-04
100,000	1.64472532	2.08854675E-04	1.64492404	1.01327896E-05
1,000,000	1.64472532	2.08854675E-04	1.64493299	1.19209290E-06
10,000,000	1.64472532	2.08854675E-04	1.64493394	2.38418579E-07
100,000,000	1.64472532	2.08854675E-04	1.64493406	1.19209290E-07

Tabla 1.1. Cálculo de la serie anterior con i creciendo desde *i=1* hasta *N*, y al revés, decreciendo desde *N* hasta 1. La columna de error, no es exactamente el error formal si no la diferencia contra el resultado que debería dar la serie infinita y lo que se calculó con una serie con una cantidad finita de términos. Es decir, es una referencia para verificar la variación de las últimas cifras decimales e identificar cuales son los correctos. Nótese que el cálculo de la serie en la que decrece la variable *i* obtuvo resultados mas precisos.

Para pensar:
¿Por qué la suma decreciente da mejor resultado?
¿Cuál sería la manera de mejorar este cálculo, con el fin de disminuir este efecto?