3 ESTRUCTURAS DE CONTROL REPETITIVAS. BUCLES

3.1 Estructuras de repetición

- Una estructura de repetición, también llamada lazo o bucle, hace posible la ejecución repetida de secciones específicas de código.
- Hay dos tipos básicos de estructuras de repetición, cuya diferencia principal radica en cómo se controlan las mismas:
    - Repetición controlada por contador o bucle DO iterativo. Con esta estructura, un bloque de sentencias se ejecuta una vez para cada uno de los valores que va tomando un contador. Se ejecuta un número específico de veces, siendo el número de repeticiones conocido antes de que empiece la ejecución de tal bucle.
    - Repetición controlada por expresión lógica o bucle WHILE. En este caso, un bloque de sentencias se ejecuta un número indefinido de veces, hasta que se satisface alguna condición establecida por el usuario, lo cual desde el punto de vista de la programación, equivale a que una cierta expresión lógica tome el valor .TRUE..

3.2 Repetición controlada por contador o bucle DO iterativo

- Esta estructura ejecuta un bloque de sentencias un número específico de veces. Se construye de la siguiente manera:

```
DO indice = inicio, fin [, paso]
  sentencia_1
  [sentencia_2]   Cuerpo
  ...
END DO
```

- *Índice* es una variable entera que se usa como contador del bucle.
- *Inicio, fin y paso* son cantidades enteras y constituyen los parámetros del *índice* del bucle. Controlan los valores de la variable *índice* durante la ejecución. Pueden ser constantes, variables o expresiones enteras.
- Las sentencias entre la sentencia DO y la sentencia END DO se llaman *cuerpo* del bucle. Para mejorar la lectura del código es recomendable endentar el cuerpo del bucle con dos o más espacios.
- *Inicio* marca el valor inicial que toma *índice* cuando comienza la ejecución del bucle DO.
- *Fin* marca el valor final que puede tomar *índice*.

- *Paso* indica el incremento con que *índice* es modificado después de cada ejecución del bucle DO. Es un parámetro opcional, cuando no aparece, su valor por defecto es 1. Puede ser positivo o negativo.
- Cuando se ejecuta una sentencia DO, tienen lugar la secuencia de operaciones siguiente:
    - Si los parámetros del índice del bucle son expresiones enteras se calculan sus valores antes del comienzo de la ejecución del bucle.
    - Se asigna el valor *inicio* a la variable de control *índice*. Si la condición *índice* * *paso* ≤ *fin* * *paso*, el programa ejecuta las sentencias del cuerpo del bucle.
    - Después, se recalcula el valor de *índice* como*: índice* = *índice* + *paso*. Si aún se cumple la condición anterior *índice* * *paso* ≤ *fin* * *paso*, el programa ejecuta otra vez las sentencias del cuerpo del bucle.
    - Se repite el punto anterior hasta que deja de cumplirse la condición de desigualdad dada, en cuyo caso la ejecución salta a la primera sentencia siguiente a la sentencia END DO.
    - El número de iteraciones que se llevan a cabo en el bucle DO se puede calcular a partir de la siguiente ecuación:

$$n^{o}\,iteraciones = \frac{fin - inicio + paso}{paso}$$ **Ecuación 3-1**

- Precauciones:
    - No está permitido modificar el valor de *índice* en el cuerpo del bucle. La mayoría de los compiladores Fortran suelen reconocer este problema y generan un error en tiempo de compilación si se intenta modificar su valor. Caso de que el compilador no detectara este problema, se podría dar el caso de un bucle infinito.
    - La única manera de parar la ejecución de un programa cuando éste contiene un bucle infinito es *matar* el programa pulsando control +C.
    - En muchos computadores, el valor de *índice* después de la finalización normal del bucle DO, está definido por el siguiente valor asignado como resultado del incremento. Esto es lo que ocurre en el compilador de la Salford FTN95. Sin embargo, su valor es indefinido oficialmente en el Fortran estándar, de modo que algunos compiladores pueden producir resultados diferentes. Por lo tanto, no conviene nunca usar el valor del índice de un bucle en las siguientes sentencias del programa Fortran.
    - Si *paso* toma el valor 0 se puede dar un bucle infinito.
    - Puede ocurrir que el cuerpo de un bucle no se ejecute nunca, si *inicio* * *paso* > *fin* * *paso*.

- Ejemplos de bucles DO iterativos

```
INTEGER :: i
DO i=1,5
  sentencia
END DO
WRITE (*,*) i
```

Aplicando la ecuación 3-1, se obtiene que sentencia se ejecuta 5 veces. La variable índice i toma los valores 1, 2, 3, 4 y 5. A continuación, el control vuelve a la sentencia DO, i toma el valor 6, pero 6*1 > 5*1 y se salta a la sentencia WRITE, escribiéndose 6 por monitor.

```
INTEGER :: i
DO i=5,0,-2
  sentencia
END DO
```

Aplicando la ecuación 3-1, se obtiene que sentencia se ejecuta 3 veces, para los valores de i: 5, 3, y 1. A continuación, el control vuelve a la sentencia DO, i toma el valor -1, pero (-1)*(-2) > 0*(-2) y se salta a la sentencia siguiente a END DO.

- Escribir un mensaje por monitor 10 veces:

```
DO i=1, 10
  WRITE (*,*) 'HOLA'
END DO
```

En este caso, el índice del bucle se usa únicamente para indicarnos cuántas veces se ejecuta el cuerpo.

- Sumar los pares de 2 a 10 sin leer datos.

```
acum=0
DO k = 2, 10,2
  acum=acum+k
END DO
WRITE (*,*) acum
```

En este caso, el índice del bucle interviene en el cuerpo del mismo.

3.3 Repetición controlada por expresión lógica o bucle WHILE

- En un bucle WHILE, un bloque de sentencias se ejecuta un número indefinido de veces, hasta que se satisface alguna condición

establecida por el usuario, o dicho de otro modo, hasta que una cierta expresión lógica toma el valor .TRUE..

- La sintaxis general de esta estructura es:

```
DO
  sentencia_1
  [sentencia_2]
  …
  IF (expresión_lógica) EXIT
  sentencia_3
  [sentencia_4]
  …
END DO
```

- El bloque de sentencias entre la sentencia DO y la sentencia END DO se ejecuta indefinidamente mientras *expresión_lógica* es falsa. Cuando *expresión_lógica* se hace cierta; entonces, se ejecuta la palabra reservada EXIT cuyo efecto es transferir el control fuera del bucle DO, a la primera sentencia siguiente a END DO.
- Un bucle WHILE puede contener varias sentencias EXIT, generalmente, formando parte una sentencia IF o bloque IF. Sin embargo, no conviene usar más de uno por bucle WHILE, en aras de construir programas bien estructurados. De esta manera, cada bucle WHILE tendrá un único punto de entrada, la sentencia DO, y un único punto de salida, la sentencia EXIT.
- Precauciones: si *expresión lógica* nunca se hace cierta, estaremos en un bucle infinito.
- Ejemplos de bucles WHILE:
    - Se presentan los dos ejemplos vistos en la sección anterior para comparar mejor las estructuras.
    - Escribir un mensaje por monitor 10 veces:

```
i=1
DO
  IF (i>10) EXIT
  WRITE (*,*) 'HOLA'
  i=i+1
END DO
```

    - Sumar los pares de 2 a 10 sin leer datos.

```
i=2
acum=0
DO
```

```
  IF (i>10) EXIT
  acum=acum+i
  i=i+2
END DO
WRITE (*,*) acum
```

- Leer por teclado un número entre 0 y 10, ambos límites incluidos. El usuario puede equivocarse, en cuyo caso el programa le dará tantas posibilidades como necesite (indefinidas veces) hasta conseguir que el número introducido esté en el rango dado. Entonces, el programa muestra cuantos intentos ha usado.

```
INTEGER :: num,cont=0
DO
  WRITE (*,*)
  WRITE (*,*) "Dame un numero de 0 a 10"
  READ (*,*) num
  cont=cont+1
  IF (num>=0.AND.num<=10) EXIT
  WRITE (*,*) "Has tecleado",num
  WRITE (*,*) "El numero debe estar entre 0 y 10"
  WRITE (*,*) "Vuelve a intentarlo"
END DO
WRITE (*,*) "*****ENHORABUENA*****"
WRITE (*,*) "lo conseguistes en",cont,"veces"
```

3.4 Bucle DO WHILE

- En Fortran 90/95 hay otra forma alternativa de bucle WHILE, el llamado bucle DO WHILE.
- Su sintaxis general es:

```
DO WHILE (expresión_lógica)
  sentencia_1
  [sentencia_2]          Cuerpo
  ...
END DO
```

- Esta estructura funciona de la siguiente manera:
  - si *expresión_lógica* es cierta se ejecuta el cuerpo del bucle y el control vuelve a la sentencia DO WHILE.

  - si *expresión_lógica* es cierta aún, se ejecuta otra vez el cuerpo del bucle y el control vuelve a la sentencia DO WHILE.
  - El proceso anterior se repite hasta que *expresión*_lógica se hace falsa, en cuyo caso el control pasa a ejecutar la primera sentencia siguiente a END DO.
  - El bucle DO WHILE es un caso especial del bucle WHILE, en el que el testeo de salida ocurre siempre en el cabecero del bucle.
- Ejemplos de bucles DO WHILE:
  - Nuevamente, se presentan los ejemplos vistos en la sección anterior para comparar mejor las diferentes estructuras.
  - Escribir un mensaje por monitor 10 veces:

```
i=1
DO WHILE (i<=10)
  WRITE (*,*) 'HOLA'
  i=i+1
END DO
```

  - Sumar los pares de 2 a 10 sin leer datos.

```
i=2
acum=0
DO WHILE (i<=10)
  acum=acum+i
  i=i+2
END DO
WRITE (*,*) acum
```

  - Leer por teclado un número entre 0 y 10, ambos límites incluidos. Cerciorarse de que el número está en el rango dado sin limitar el número de intentos. Mostrar cuantos intentos ha necesitado el usuario.

```
INTEGER :: num,cont=1
WRITE (*,*) "Dame un numero de 0 a 10"
READ (*,*) num
DO WHILE (num<0.OR.num>10)
  WRITE (*,*) "Has tecleado",num
  WRITE (*,*) "El numero debe estar entre 0 y 10"
  WRITE (*,*) "Vuelve a intentarlo"
  WRITE (*,*) "Dame un numero de 0 a 10"
```

```
  READ (*,*) num
  cont=cont+1
END DO
WRITE (*,*) “*****ENHORABUENA*****”
WRITE (*,*) “lo conseguistes en”,cont,”veces”
```

- En la sección siguiente, se verá que es posible saltar algunas sentencias del cuerpo de un bucle en una iteración y pasar a ejecutar la siguiente iteración, en cualquier momento mientras el bucle se está ejecutando.

3.5 Sentencias EXIT y CYCLE

- En las secciones anteriores se ha visto que la sentencia EXIT produce la salida inmediata de un bucle.
- Otra sentencia que permite controlar la ejecución de un bucle es la sentencia CYCLE.
- CYCLE detiene la ejecución de la iteración actual y devuelve el control a la cabecera del bucle continuando su ejecución en la iteración siguiente.
- Ambas sentencias pueden usarse tanto en bucles WHILE como en bucles iterativos.
- Ejemplo de la sentencia CYCLE.
    - Si en el código Fortran anterior construido con bucle DO queremos mostrar los tres últimos mensajes de error sólo a partir del tercer intento:

```
INTEGER :: num,cont=0
DO
  WRITE (*,*)
  WRITE (*,*) "Dame un numero de 0 a 10"
  READ (*,*) num
  cont=cont+1
  IF (num>=0.AND.num<=10) EXIT
  IF (cont<3) CYCLE
  WRITE (*,*) "Has tecleado",num
  WRITE (*,*) "El numero debe estar entre 0 y 10"
  WRITE (*,*) "Vuelve a intentarlo"
END DO
WRITE (*,*) "*****ENHORABUENA*****"
WRITE (*,*) "lo conseguistes en",cont,"veces"
```

3.6 Bucles con nombre

- Es posible asignar un nombre a un bucle.
- La sintaxis general de un bucle WHILE con un nombre añadido es:

```
[nombre:] DO
  sentencia_1
  [sentencia_2]
  ...
[  IF (expresión_lógica) CYCLE [nombre]]
  [sentencia_3]
  ...
  IF (expresión_lógica) EXIT [nombre]
  [sentencia_4]
  ...
END DO [nombre]
```

- La sintaxis general de un bucle iterativo con un nombre añadido es:

```
[nombre:] DO indice = inicio, fin [, paso]
  sentencia_1
  [sentencia_2]      Cuerpo
  ...
[  IF (expresión_lógica) CYCLE [nombre]]
  [sentencia_4]
  ...
END DO [nombre]
```

- En ambas estructuras, el nombre del bucle debe ser un identificador válido.
- Es opcional poner nombre a un bucle, pero si se pone, éste debe repetirse en la sentencia END DO.
- Es opcional poner nombre a las sentencias CYCLE y EXIT, pero si se pone, éste debe ser el mismo que el de la sentencia DO.
- La utilidad de los nombres en los bucles aumenta a medida que lo hace el tamaño de los mismos. Por un lado, ayuda al usuario a identificar rápidamente las sentencias que pertenecen un bucle complicado y extenso y, por otro, ayuda al compilador a afinar la localización de las sentencias de error.

3.7 Bucles anidados

- Un bucle puede estar contenido completamente dentro de otro bucle. En este caso, se dice que ambos bucles están *anidados*.
- Cuando dos bucles están anidados, el bucle interno se ejecuta completamente para cada iteración del bucle externo. Es decir, el índice del bucle interno toma todos los valores posibles permitidos por su condición de desigualdad para cada uno de los valores posibles del índice del bucle externo permitidos por su condición de desigualdad.
- Los bucles anidados se cierran en orden inverso a cómo se abren. Así, cuando el compilador encuentra una sentencia END DO, asocia esa sentencia con el bucle más interno abierto.
- Es conveniente asignar nombres a todos los bucles anidados para que sean más fáciles de comprender y de localizar los errores de compilación.
- Los índices de bucles anidados deben tener identificadores distintos.
- Si aparecen sentencias CYCLE o EXIT sin nombre en bucles anidados, éstas se asocian por defecto con el bucle más interno abierto. Para evitar asociaciones automáticas no deseadas, es conveniente escribir las sentencias CYCLE o EXIT con nombre, que será el mismo que el dedicado al cabecero y fin del bucle involucrado.
- La sintaxis general de dos bucles anidados WHILE con nombre añadido es:

```
[externo:] DO
  bloque_de_sentencias1
  [IF (expresión_lógica) CYCLE [externo]]
  bloque_de_sentencias2
  IF (expresión_lógica) EXIT [externo]
  bloque_de_sentencias3
[interno:] DO
        bloque_de_sentencias4
  [IF (expresión_lógica) CYCLE [interno]]
  bloque_de_sentencias5
  IF (expresión_lógica) EXIT [interno]
  bloque_de_sentencias6
END DO [interno]
  bloque_de_sentencias7
END DO [externo]
```

- No es necesario tener bloques de sentencias en todos los puntos marcados del bucle. Dependiendo del caso, como mínimo debe haber alguna sentencia o bloque de ellas a elegir entre los bloques llamados 1, 2 y 3, y, lo mismo, con los bloques 4, 5 y 6.
- Ejemplo de dos bucles DO anidados:
    - Calcular el factorial de los números 3 al 6 y mostrar los resultados por monitor

```
numero: DO i=3,6
  fact=1
  factorial_de_numero: DO j=1,i
     fact=fact*j
  END DO factorial_de_numero
  WRITE (*,*) 'FACTORIAL DE ',i,'=',fact
END DO numero
```

- Para cada valor de i, j toma todos los valores desde 1 hasta esa i, multiplicándolos entre sí para calcular el factorial de ese número i.

3.8 Bucles anidados dentro de estructuras IF y viceversa

- Es posible anidar bucles dentro de estructuras IF o tener estructuras IF dentro de bucles.
- Si un bucle se anida dentro de una estructura IF, el bucle debe permanecer completamente dentro de un único bloque de código de la estructura IF.
- Ejemplo:

```
externo: IF (x<y) THEN
.....
  interno: DO i=1,5


..END DO interno
.....
ELSE
  .....
END IF externo
```

EJERCICIOS RESUELTOS

Objetivos:

Aprender a usar estructuras de control repetitivas o bucles para abreviar el código de los programas Fortran.

Se muestran ejemplos de bucles DO controlados con contador y bucles controlados con expresión lógica. En primer lugar, aparecen estas estructuras simples en los programas y, a continuación, anidadas.

1. Mostrar un mensaje por pantalla, por ejemplo, HOLA A TODOS, cien veces.

```
PROGRAM cap3_1
IMPLICIT NONE
INTEGER :: i

DO  i=1,100
 WRITE (*,*) 'HOLA A TODOS'
END DO
END PROGRAM cap3_1
```

– El usuario que desconoce la existencia de los bucles en programación, podría pensar, en principio, en construir el programa copiando cien veces la instrucción: WRITE (*,*) 'HOLA A TODOS'. Sin embargo, intuitivamente el usuario comprende que debe existir algún procedimiento en Fortran que abrevie significativamente esta tarea, es decir, los bucles.

– En este ejercicio, el índice del bucle i se usa únicamente para indicarnos cuántas veces se ejecuta el cuerpo del bucle.

– Cambia los valores de los parámetros del índice del bucle sin modificar el resultado de la ejecución del programa. ¿Cuántas posibilidades hay?

2. Sumar todos los números naturales desde el 1 hasta el 100, ambos incluidos.

```
PROGRAM cap3_2
IMPLICIT NONE
INTEGER :: suma=0, i

DO i=1,100
 suma=suma+i
!  WRITE(*,*) 'i,suma',i,suma
END DO
WRITE(*,*) 'EL RESULTADO ES',suma
END PROGRAM cap3_2
```

– En este caso, el índice del bucle interviene en el cuerpo del mismo.

– Activa la sentencia comentada del programa y estudia su significado.

– ¿Qué resultado se obtiene si cambiamos el cabecero del bucle por la instrucción DO i=100,1,-1

– Utiliza el depurador del entorno Fortran para comprender mejor la ejecución del bucle.

3. Sumar todos los números naturales pares desde el 10 hasta el 100, ambos incluidos.

```
PROGRAM cap3_3
IMPLICIT NONE
INTEGER :: sumapar=0, i
DO i=100,10,-2
  sumapar=sumapar+i
  WRITE(*,*) 'i,sumapar',i,sumapar
END DO
WRITE(*,*) 'EL RESULTADO ES',sumapar
END PROGRAM cap3_3
```

4. Calcular el valor de π aplicando la fórmula: π = 4* (1 – 1 / 3 +1 / 5 – 1/7) incluyendo hasta el término1 / 99.

```
PROGRAM cap3_4
IMPLICIT NONE
INTEGER :: signo,i
REAL :: pic
pic=0
signo=1
DO  i=1,99,2
  pic=pic+1.0/i*signo
  signo=-signo
END DO
WRITE(*,*) 'EL RESULTADO ES',4*pic
END PROGRAM cap3_4
```

– El índice del bucle se usa para obtener los denominadores de la serie alternada. En cada pasada se cambia el signo del sumando en la variable signo.

– Aunque el ejercicio puede hacerse, como vemos, con un único bucle, repite el ejercicio usando dos, uno para sumar los positivos de la serie y otro para sumar los negativos de la misma. Resta ambos resultados y multiplica por cuatro; debes obtener el mismo resultado. De esta manera, el código del programa aumenta en relación con el anterior, sin embargo es más legible, y, por lo tanto, más sencillo de comprender.

5. Calcular la media de un conjunto de números. El programa recoge tantos datos como el usuario quiera.

```
PROGRAM cap3_5

INTEGER :: cont=0
REAL :: num,acum=0
CHARACTER (LEN=2) :: seguir

DO
  WRITE(*,*) 'DAME UN NUMERO'
  READ (*,*) num
  cont=cont+1
  acum=acum+num
  WRITE(*,*) 'OTRO NUMERO?'
  WRITE(*,*) 'TECLEA SI O NO EN MAYUSCULAS Y ENTRE APOSTROFES'
  READ (*,*) seguir
  IF (seguir /= 'SI') EXIT
END DO
WRITE(*,*) 'LA MEDIA ES:',acum/cont
END PROGRAM cap3_5
```

– Mientras el valor de la variable seguir sea ‘SI’, la expresión lógica dada es FALSA y se ejecuta el cuerpo del bucle.

– En el momento en que el valor de la variable seguir es distinto de ‘SI’, la expresión lógica se hace cierta, se ejecuta la instrucción EXIT y, por lo tanto, el control pasa a ejecutar la instrucción WRITE(*,*) ‘La media es’,acum./cont

– Recuerda que Fortran no distingue las letras minúsculas de las mayúsculas en todos los contextos excepto en constantes de tipo carácter.

– Si el número de elementos de la lista es conocido a priori, ¿Cómo cambia el código del programa? Sustituye el bucle WHILE por uno iterativo.

6. Calcular la media de un conjunto de números. El programa recoge tantos datos como el usuario quiera calculando tantas medias como él quiera.

```
PROGRAM cap3_6
IMPLICIT NONE
INTEGER :: cont
REAL :: num,acum
CHARACTER (LEN=2) :: seguir,mas_medias

externo: DO
 cont=0
 acum=0
 interno: DO
  WRITE(*,*) 'DAME UN NUMERO'
  READ (*,*) num
  cont=cont+1
  acum=acum+num
  WRITE(*,*) 'OTRO NUMERO?'
  WRITE(*,*) 'TECLEA SI O NO EN MAYUSCULAS Y ENTRE APOSTROFES'
  READ (*,*) seguir
  IF (seguir /= 'SI') EXIT interno
 END DO interno
 WRITE(*,*) 'LA MEDIA ES:',acum/cont
 WRITE(*,*) 'OTRA MEDIA?'
 WRITE(*,*) 'TECLEA SI O NO EN MAYUSCULAS Y ENTRE APOSTROFES'
 READ (*,*) mas_medias
 IF (mas_medias /= 'SI') EXIT externo
END DO externo
END PROGRAM cap3_6
```

– Necesitamos dos bucles anidados para realizar el ejercicio. Destacar que la inicialización de las variables cont y acum a cero debe

hacerse antes de entrar al bucle interno WHILE, para que cada vez que se calcule otra media se vuelvan a poner a cero sus valores.

– Si el número de elementos de la lista y el número de listas es conocido a priori, ¿Cómo cambia el código del programa? Sustituye los bucles WHILE por dos iterativos.

7. Calcula y escribe por pantalla la cantidad de números positivos que hay en una lista dada de N elementos. El proceso se repite todas las veces que el usuario quiera.

```
PROGRAM cap3_7

CHARACTER (LEN=1) :: seguir
INTEGER :: pos
REAL :: num

externo: DO
 WRITE(*,*) '"DAME EL NUMERO DE ELEMENTOS DE LA LISTA"'
 READ (*,*) n
 pos=0 !IMPORTANTE, INICIALIZAR PARA CADA LISTA
 interno: DO  i=1,n
  WRITE(*,*) 'INTRODUCE NUMERO',i
  READ (*,*) num
  IF (num > 0) THEN
    pos=pos+1
  END IF
 END DO interno
 WRITE(*,*) 'CANTIDAD DE POSITIVOS DE LOS',n,' LEIDOS ES:',pos
 WRITE(*,*) 'QUIERES SEGUIR CON OTRA LISTA (S/N)?'
 READ (*,*) seguir
 IF (seguir /= 'S') EXIT externo
END DO externo
END PROGRAM cap3_7
```

– Se utilizan dos bucles anidados: el bucle externo controla si seguimos con otra lista o no, mientras el interno opera con una lista determinada, controlando que se lea cada número de la misma, se evalúe si es positivo y, si es cierto, se incremente el valor de un contador una unidad.

– Notar que declarando una única variable num se consigue realizar el ejercicio lo que supone un aprovechamiento eficiente de la memoria.

– Completa el ejercicio contando la cantidad de positivos, negativos y ceros que hay en cada lista.

– Repite el ejercicio sustituyendo el bucle externo WHILE por un bucle DO WHILE.

8. Leer números por teclado hasta introducir uno negativo.

```
PROGRAM cap3_8
IMPLICIT NONE
LOGICAL :: positiv
REAL :: num
DO
  WRITE(*,*) 'DAME UN NUMERO POSITIVO'
  READ (*,*) num
  IF (num > 0) THEN
    positiv=.TRUE.
  ELSE
    positiv=.FALSE.
  END IF
    IF (.NOT.positiv) EXIT
END DO
WRITE(*,*) num,' NO ES POSITIVO'
END PROGRAM cap3_8
```

– Ejemplo de estructura IF dentro de un bucle WHILE.

– Este ejercicio utiliza una variable lógica, llamada positiv, para evaluar la expresión lógica del bucle WHILE. Mientras se lean números positivos, su valor será CIERTO y se ejecuta el cuerpo del bucle. En el momento en que se lee un número negativo, positiv toma el valor FALSO, la expresión lógica se hace cierta, se ejecuta la instrucción EXIT y, por tanto, el control del bucle pasa a ejecutar la sentencia WRITE, escribiendo por monitor que el número dado no es positivo.

9. Calcular el factorial de un número natural (que se pedirá por teclado) usando un bucle. Escribir los resultados parciales.

```
PROGRAM cap3_9
IMPLICIT NONE
```

```
INTEGER :: num,fact,i
WRITE(*,*) 'DAME UN NUMERO'
READ (*,*) num
IF (num<0) THEN
  WRITE(*,*) 'NO EXISTE EL FACTORIAL DE UN NEGATIVO'
ELSE IF (num==0) THEN
  WRITE(*,*) 'EL FACTORIAL DE CERO ES UNO'
ELSE
  fact=1
  interno: DO i=num,1,-1
     fact=fact*i
     WRITE(*,*) 'RESULTADO PARCIAL',fact
  END DO interno
  WRITE(*,*) 'EL FACTORIAL DE',num,' ES ',fact
END IF
END PROGRAM cap3_9
```

– Ejemplo de bucle DO iterativo dentro de una estructura IF.

– Si el número introducido es negativo se escribe un mensaje avisando de que no existe su factorial, sino, si el número es el cero, su factorial es uno y sino se calcula el factorial del número usando un bucle.

EJERCICIOS PROPUESTOS

1) Programa que sume y muestre por pantalla todos los números naturales del 1 hasta el 5, ambos incluidos. Lo mismo pero de 1 a 50; lo mismo pero de 1 a 500.

2) Programa que sume el número 5 y sus múltiplos hasta el 100 inclusive y muestre el resultado por pantalla.

3) Realizar un programa que calcule y muestre la suma de los múltiplos de 5 comprendidos entre dos valores A y B. El programa no permitirá introducir valores negativos para A y B y verificará que A es menor que B. Si A es mayor que B, intercambiará sus valores.

4) Programa que calcule y muestre e^x utilizando los cuatro primeros términos de la serie: $e^x = 1 + x /1! + x^2/2! + x^3/3!$. El valor "exacto" se puede obtener con la función intrínseca EXP(argumento).

5) Programa que lea un número natural y diga si es o no es *triangular*. A saber: un número N es triangular si, y solamente si, es la suma de los primeros M números naturales, para algún valor de M. Ejemplo: 6 es triangular pues 6 = 1 + 2 + 3. Una forma de obtener los números triangulares es aplicando la fórmula: $\frac{n(n+1)}{2} \forall n \in N$.

6) Programa que encuentre un número natural n y otro m tal que se cumpla: $1^2 + 2^2 + 3^2 + 4^2 + \ldots + m^2 = n^2$. Solución: $1^2 + 2^2 + 3^2 + 4^2 + \ldots + 24^2 = 70^2$.

7) Programa que muestre la serie de Fibonacci 0, 1, 1, 2, 3, 5, 8, 13, 21,...Los primeros términos son 0 y 1, los siguientes suma de los dos anteriores.

8) Programa que verifique si un número es reproductor de Fibonacci. A saber: un número n se dice reproductor de Fibonacci si es capaz de reproducirse a sí mismo en una secuencia generada con los m dígitos del número en cuestión y continuando en la serie con un número que es la suma de los m términos precedentes. Ejemplo: 47 es un número reproductor de Fibonacci pues la serie: 4, 7, 11, 18, 29, 47,... contiene el 47.

9) Retoma el ejercicio anterior y explora todos los números reproductores de Fibonacci de 2 y de 3 dígitos. Soluciones para dos dígitos: 14, 19, 28, 47, 61, 75. Soluciones para tres dígitos: 197, 742.

10) Programa que pida por teclado la nota de examen, mientras sea suspenso.

11) Programa que pida la estatura (en metros) y sexo (V/M) de un número indeterminado de personas (mientras el operador quiera). Posteriormente escribirá la estatura media de los varones y la estatura media de las mujeres.

12) Escribir un programa que calcule los centros numéricos entre 1 y n. Un centro numérico es un número que separa una lista de números enteros (comenzando en 1) en dos grupos de números, cuyas sumas son iguales. El primer centro numérico es el 6, el cual separa la lista (1 a 8) en los grupos: (1, 2, 3, 4, 5) y (7, 8) cuyas sumas son ambas iguales a 15. El segundo centro numérico es el 35, el cual separa la lista (1 a 49) en los grupos: (1 a 34) y (36 a 49) cuyas sumas son ambas iguales a 595.

13) Programa que calcule el producto n! (n-1)!...3!*2!*1!

14) Programa que pida un número por teclado y diga si es primo o no, mostrando todos sus divisores.