



**APUNTES DE FORTRAN**

**PROGRAMACIÓN AVANZADA**

**(2003-1)**

This file was generated with the demo version of the PDFconverter

El logo de PDFconverter, que muestra un rostro humano con los ojos cerrados y una sonrisa, formado por líneas azules.

## ÍNDICE

<b>TEMA</b>	<b>PAG.</b>
Tipos de datos.....	3
Tipos de operadores .....	4
Editor Lahey (Fortran) .....	6
Estructuras básicas .....	7
Subprogramas .....	8
Funciones .....	8
Subrutinas .....	9
Estructuras de decisión .....	11
If .....	11
If – then – else .....	12
Anidación de estructuras .....	12
Select case .....	13
Estructuras de repetición .....	14
Do .....	14
Do (If) .....	15
Do – While .....	16

## Tipos de Datos

En general, los tipos de datos que puede manejar Fortran son:

1. **Constantes.** Este término designa un valor específico y determinado que se define al hacer un programa y que no cambia a lo largo del mismo
2. **Variables.** El concepto de variable coincide con el concepto habitual que se tiene de ella; es un nombre simbólico con el que se designa o hace referencia a un dato que puede tomar valores diversos.

Tanto las constantes como las variables pueden ser de cuatro tipos principalmente:

- **Entera** (INTEGER). Una constante entera es una sucesión de dígitos precedidos o no del signo positivo (+) o negativo (-) y sin punto decimal. El límite en la cantidad para datos enteros es de -32768 al 32767 para máquinas con memoria de 16 bits y de -2147433647 al 2147433647 para máquinas las cuales la memoria guarde hasta 32 bits.
- **Reales** (REAL). En cualquier caso, una constante real en FORTRAN equivale a una cantidad formada por una parte entera y una fraccionaria (con punto decimal). Se define como exponente real, el carácter alfabético E seguido por un signo + o - y por una constante formada por dos dígitos como máximo (Ej: E25, 1.23E-3, -5E-02)
- **Lógicas** (LOGICAL). Un dato lógico únicamente posee dos valores: cierto (.TRUE.) o falso (.FALSE.) y se guardan en memoria mediante códigos binarios especiales. Los puntos que preceden y siguen a estos valores son indispensables para su uso dentro del lenguaje.
- **Caracteres** (CHARACTER). Es un conjunto de caracteres válidos y su longitud es el número total de caracteres que contiene. Se define una constante de este tipo precedidos y seguidos por un apóstrofe ('), cuyo carácter denomina delimitador.
- **Doble precisión** (REAL\*8). Se denomina exponente doble precisión en FORTRAN, al carácter alfabético D seguido opcionalmente del signo + o - y finalizando por una constante entera (Ej: D01, 14.2D+3, 12D-02). La diferencia entre este tipo de valor con el de real es que para este caso el compilador reserva más memoria para la doble precisión, con lo cual el número de dígitos es mayor.
- **Complejos** (COMPLEX). Son los datos que constan de dos partes, una entera y una imaginaria. Por ejemplo para asignar valor a una variable  $C = 2 + 3i$ , después de declarar C como compleja se asigna  $C = (2,3)$ .
- **Parametrizadas** (REAL(kind = #)). Se denominan así a las variables que pueden ser declaradas con una precisión definida por el usuario, la función utilizada es kind y los valores disponibles son 4, 8 y 16 donde 4 corresponde a simple precisión, 8 a doble precisión y 16 a cuádruple precisión.

Los nombres para las constantes y variables se forman utilizando un conjunto de caracteres denominados IDENTIFICADORES, los cuales hay que seguir varias reglas:

- No pueden empezar por un carácter numérico
- Todas aquellas variables cuyo nombre comience con los caracteres alfabéticos I, J, K, L, M o N se dice que queda implícitamente definida como variable entera.

- Todas aquellas variables cuyo primer carácter alfabético sea una letra distinta de las mencionadas anteriormente, se dice que son variables reales definidas de forma implícita.
- No deben rebasar los 31 caracteres.
- Su nombre debe ser significativo al uso que se le de dentro del código.

La segunda y tercera regla se aplica para cuando no es empleado el comando `Implicit none` al inicio del código.

Para la declaración de variables se sigue el siguiente formato:

<b>INTEGER</b> contador	Variable entera denominada contador
<b>REAL</b> numero	Variable real denominada numero
<b>LOGICAL</b> salir	Variable lógica denominada salir
<b>CHARACTER*6</b> mensaje	Variable de 6 caracteres denominada mensaje
<b>REAL*8</b> suma	Variable doble precisión denominada suma

Para la declaración de constantes, se realiza mediante la orden **PARAMETER**, de la siguiente forma:

<b>REAL, PARAMETER</b> (radio=6.5)	Constante real de valor 6.5
------------------------------------	-----------------------------

Otro tipo de datos es el concepto denominado lista (vector) o tabla (matriz), si tiene más de una dimensión. Por ejemplo, la temperatura en 5 ciudades diferentes, o la asociación de la temperatura con la humedad de las mismas ciudades. Las listas y tablas son estructuras de datos que se designan con el nombre genérico de conjunto de datos (en inglés: array) y que se declaran mediante la instrucción **DIMENSION** de la siguiente forma:

<b>DIMENSION</b> B(3,4)	Matriz de datos reales con 3 filas y 4 columnas
-------------------------	---

## Tipos de Operadores

Los operadores a los que Fortran da soporte se dividen en tres grupos principalmente: aritméticos, de comparación y lógicos. Los operadores aritméticos a los que Fortran da soporte en su orden de ejecución son:

Orden	Descripción	Operadores
1	Paréntesis	()
2	Exponenciación	**
3	Mult , Div	* , /
4	Suma , Resta	+ , -

Los operadores de comparación son:

Operador	Descripción	Ejemplo	Resultado
<code>==</code> , <code>.EQ.</code>	Igual que	<code>5 == 7</code>	Falso
<code>/=</code> , <code>.NE.</code>	Diferente a	<code>8 /= 8</code>	Falso
<code>&lt;</code> , <code>.LT.</code>	Menor que	<code>7 &lt; 5</code>	Falso
<code>&lt;=</code> , <code>.LE.</code>	Menor o igual que	<code>7 &lt;= 11</code>	Verdadero
<code>&gt;</code> , <code>.GT.</code>	Mayor que	<code>8 &gt; 2</code>	Verdadero
<code>&gt;=</code> , <code>.GE.</code>	Mayor o igual que	<code>10 &gt;= 10</code>	Verdadero

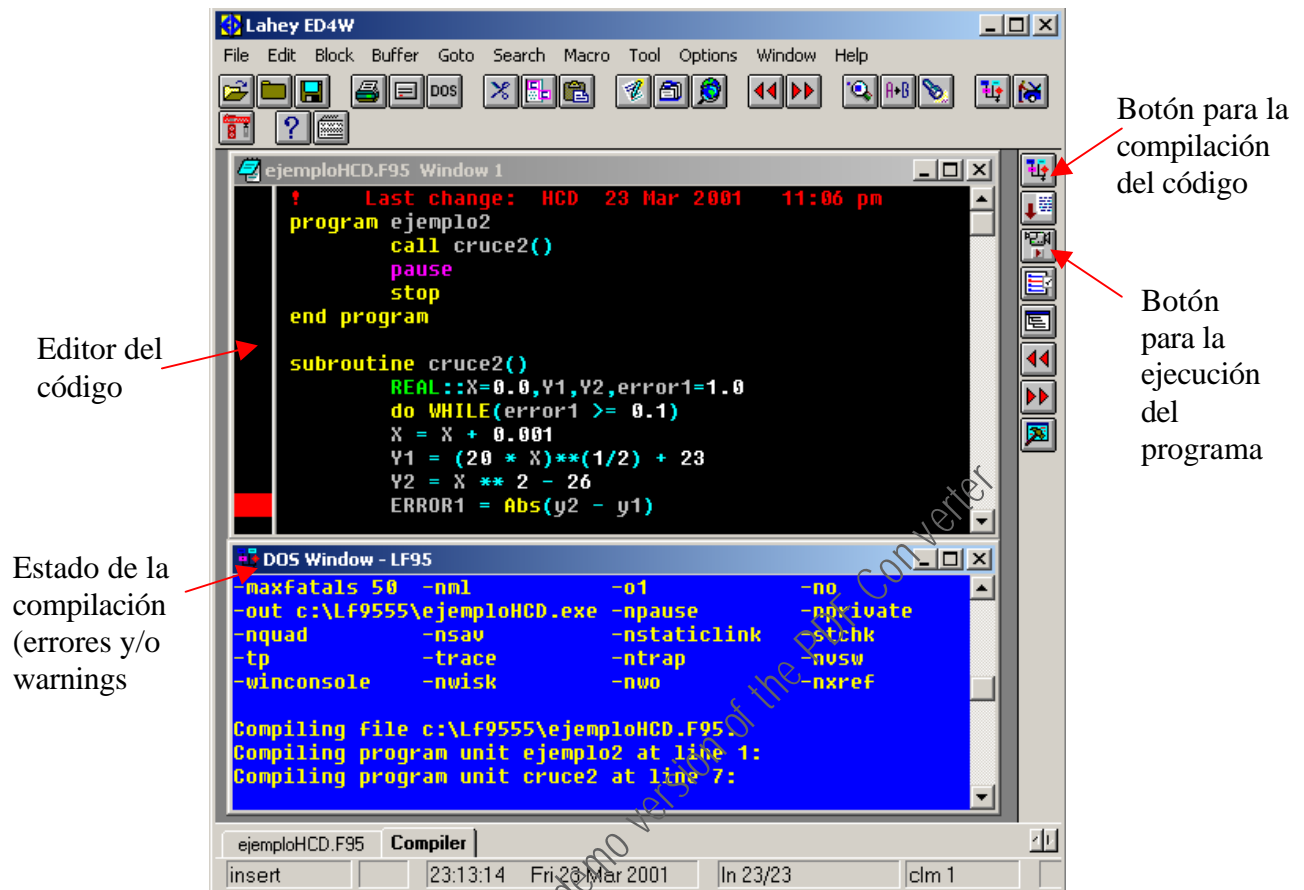
Los operadores lógicos son:

Operador	Descripción	Ejemplo	Resultado
<code>.NOT.</code>	Niega un resultado	<code>.NOT.(3 == 3)</code>	Falso
<code>.AND.</code>	Ambos lados deben ser verdaderos	<code>(2 &lt; 3).AND.(4 &lt; 5)</code>	Verdadero
<code>.OR.</code>	Un lado o ambos son verdaderos	<code>(2 &lt; 3).OR.(5 &lt; 4)</code>	Verdadero
<code>.EQV.</code>	Ambos lados deben ser falsos o verdaderos	<code>(7 &lt; 2).EQV.(10 &lt; 3)</code>	Verdadero
<code>.NEQV.</code>	Ambos lados deben ser diferentes, ya sea falso y verdadero o verdadero y falso	<code>(5 &gt; 1).NEQV.(3 &gt; 2)</code>	Verdadero

El orden de ejecución se refiere a la jerarquía que tiene un operador sobre otro, es decir, en una línea de código donde aparecen ambos el operador de mayor orden se ejecutará primero que el operador de orden menor. La tabla completa del orden de ejecución de los operadores es:

Orden	Operador
1	Paréntesis
2	<code>^</code>
3	<code>*</code> , <code>/</code>
5	<code>+</code> , <code>-</code>
6	Operadores de comparación
7	Operadores Lógicos

## Editor Lahey (Fortran)



Los archivos creados al compilar un código en Fortran son cinco, con extensiones .f95, .map, .obj, .bak, .exe. Al guardar un programa se debe especificar además del nombre su extensión (Ej: ejemplo.f95), el nombre del programa debe ser corto ya que la compilación se efectúa en ambiente MS-DOS. Es posible realizar la compilación directamente desde MS-DOS desde el directorio donde se haya guardado el código por medio del comando LF95 como se muestra a continuación:

```
C:\>Fortran\Curso\LF95 ejemplo.f95
```

Una vez efectuada la compilación se crea un archivo ejecutable con el mismo nombre, de manera que pueda ser ejecutado desde el mismo lugar:

```
C:\>Fortran\Curso\ejemplo.exe
```

## Estructuras básicas

Para iniciar un programa en Fortran se escribe program al inicio del editor, seguido de un nombre relativo al código que se vaya a ejecutar, se consideran las mismas reglas para los identificadores. En seguida se realiza la declaración de las variables.

Para la lectura y escritura de datos en pantalla se utilizan los comandos READ para la lectura, WRITE o PRINT para la escritura. Ejemplo:

Program primero

```
INTEGER::uno, dos, suma
```

! los parámetros dentro del paréntesis para el comando Write y Read se denominan formateadores y son necesarios para establecer el formato de entrada o salida de datos y la unidad de la que serán leídos.

```
WRITE(6,*)'Dame el primer numero ='
```

```
READ(5,*)uno
```

```
WRITE(6,*)'Dame el segundo numero ='
```

```
READ(5,*)dos
```

```
suma=uno+dos
```

```
PRINT *, 'La suma es =',suma
```

End program

Para Fortran es indiferente el uso de mayúsculas o minúsculas para la escritura de comandos y funciones, en ocasiones el mismo compilador lo cambiará de acuerdo a su configuración. Fortran no es sensible al uso de mayúsculas o minúsculas incluso en los nombres de las variables, esto es, una variable declarada como VAR1, puede ser empleada a lo largo del código como Var1 o como var1 sin ningún problema.

El signo ! se utiliza para realizar comentarios dentro del código, es decir lo que se escriba a continuación de la exclamación no será en tomado en cuenta para la compilación.

El punto y coma (;) se utiliza para escribir varios comandos seguidos en una misma líneas de código, de esta manera se evita tener que cambiar de renglón para cada comando.

El signo & es empleado para separar en dos renglones un mismo comando, no importa si la truncación parte en dos el nombre del comando, la función o su argumento.

### Ejercicios propuestos

Declarar variables de diferente tipo y asignar valores a cada una para luego desplegarlos en pantalla.

Realizar operaciones sencillas (suma. Resta. Multiplicación y división) entre las variables creadas en el ejercicio anterior.

Introducir los signos de “!”, “;” y “&” para observar su comportamiento.

## Subprogramas

Cuando se manejan códigos de programación muy grandes suele hacerse tedioso o muy complicado su seguimiento, la única forma para manejar códigos tan grandes es usar una aproximación modular y dividir el programa en muchas unidades independientes pequeñas llamadas subprogramas.

Un subprograma es una pequeña pieza de código que resuelve un sub-problema bien definido. El mismo subprograma puede ser llamado varias veces con distintas entradas de datos.

En Fortran se tienen dos tipos diferentes de subprogramas, conocidas como funciones y subrutinas.

### Funciones

Las funciones en Fortran son bastante similares a las funciones matemáticas: ambas toman un conjunto de variables de entrada (parámetros) y regresan un valor de algún tipo. El mismo lenguaje tiene un cierto número de funciones incorporadas como por ejemplo: abs (valor absoluto), cos (coseno), tan (tangente), sqrt (raíz cuadrada), estas funciones se clasifican en cuatro:

- 1) Funciones de indagación (inquiry functions).- son aquellas que devuelven información sobre algo, sus resultados dependen más bien de las propiedades del principal argumento que del valor de este.
- 2) Funciones elementales.- muchas funciones han sido definidas para argumentos escalares, aunque es posible emplearlas también para arreglos de modo que la función es aplicada elemento por elemento al arreglo y el resultado es también un arreglo.
- 3) Funciones de transformacionales.- una función de este tipo transforma un arreglo en un resultado escalar o en otro arreglo, en vez de aplicar la operación elemento por elemento.
- 4) Subrutinas.- el nombre de una subrutina intrínseca no puede ser empleado como un argumento actual.

La forma general de una función definida por el usuario es:

```
tipo function nombre (lista_de parámetros)
    declaraciones
    sentencias
    return
end
```

Para ejemplificar cómo se desarrolla una función definida por el usuario se supone el siguiente problema: un meteorólogo ha estudiado los niveles de precipitación en el área de una bahía y ha obtenido un modelo (función)  $ll(m,t)$  donde  $ll$  es la cantidad de lluvia,  $m$  es el mes, y  $t$  es un parámetro escalar que depende de la localidad. Dada la fórmula para  $ll$  y el valor de  $t$ , calcular la precipitación anual

La forma obvia de resolver el problema es escribir un ciclo que corra sobre todos los meses y sume los valores de  $ll$ . Como el cálculo del valor de  $ll$  es un subproblema independiente, es conveniente implementarlo como una función. El siguiente programa principal puede ser usado:

```

program lluvia
real:: t, suma
integer:: m

write (6,*) 't ='
read (5,*) t
suma = 0.0
do m = 1, 12
  suma = suma + ll(m, t)
end do
write (6,*) 'La precipitación Anual es ', suma, 'pulgadas'

stop
end program

```

Se observa la presencia de la función  $ll(m,t)$ , su implementación es la siguiente:

```

real function ll(m,t)
integer:: m
real:: t
ll = 0.1*t * (m**2 + 14*m + 46)
if (ll .LT. 0) then ll = 0.0
return
end

```

Las diferencias entre una función y el programa principal son:

- Las funciones tienen un tipo. El tipo debe coincidir con el tipo de la variable que recibirá el valor.
- El valor que devolverá la función, deberá ser asignado en una variable que tenga el mismo nombre que la función.
- Las funciones son terminadas con la sentencia return en vez de la sentencia stop.

La función es llamada simplemente usando el nombre de la función y haciendo una lista de argumentos entre paréntesis.

### Subrutinas

La limitante de las funciones es que solamente devuelven un valor, en ocasiones se hace necesario regresar dos o más valores y en ocasiones ninguno. Para este propósito se usa la construcción subrutina. La sintaxis general es la siguiente:

```

subroutine nombre (lista_de_parámetros)
  declaraciones
  sentencias
  return

```

end

Observar que las subrutinas no tienen tipo y por consecuencia no pueden hacerse asignación al momento de llamar al procedimiento. Por ejemplo, si se desea intercambiar dos valores enteros por medio de una subrutina:

```
program intercambio
integer:: a,b
a = 5
b = 7
call iswap(a,b)
write (6,*) a ,b
end program

subroutine iswap (a, b)
integer:: a, b
integer:: tmp

tmp = a
a = b
b = tmp

return
end subroutine
```

Se debe observar que hay dos bloques de declaración de variables en el código. Primero, se declaran los parámetros de entrada/ salida, es decir, las variables que son comunes al que llama y al que recibe la llamada. Después, se declaran las variables locales, esto es, las variables que serán sólo conocidas dentro del subprograma. Se pueden usar los mismos nombres de variables en diferentes subprogramas.

### Ejercicios propuestos

Buscar en los manuales de línea de Fortran las diferentes funciones que maneja y hacer una breve descripción de cada una.

Con lo investigado anteriormente realizar programas utilizando dichas funciones.

Realizar un programa que calcule el factorial de un número implementándolo como función y llamándolo desde una subrutina donde se pida el número y se imprima a pantalla el resultado.

## Estructuras de Decisión

La finalidad de este tipo de estructuras es la toma de decisiones, lo que nosotros comúnmente usamos para determinar si algo es bueno o malo, falso o verdadero, si o no, es o no es, etc. Con estas estructuras podremos responder a este tipo de situaciones a la hora de elaborar nuestro programa.

### IF

La primera y más básica de estas estructuras es el IF, y el formato que debe seguir es el siguiente:

```

If (Condición) Then
    Conjunto de instrucciones
EndIf

```

Lo que se debe de entender como:

```

Si la (Condición) se cumple Entonces
    Se realiza el Conjunto de instrucciones y
    Se cierra la estructura
Fin de la estructura

```

Ahora bien la condición puede estar constituida mediante operadores aritméticos, lógicos y no puede faltar uno de comparación.

Ejemplo:

Podemos hacer un programa que en cierto momento pida una contraseña, para que haga alguna operación restringida:

```

Program pruebaIf
  Implicit none
  CHARACTER(10)::nombre
  REAL:: B
  WRITE(6,*)"Escriba la contraseña"
  WRITE(*,*)
  READ(5,*)nombre
  If (nombre=="sapo") then
    WRITE(6,*)"Dame el valor"
    WRITE(*,*)
    READ(5,*)b
    B=(b**2)
    WRITE(6,*)"El resultado es:",b
  endif
stop

```

`end program pruebaIf`

Lo que hace es verificar si el nombre que se ingresa es “sapo”, si no es no hará nada, pero si por el contrario si es correcto entonces se podrá realizar una operación, que es elevar al cuadrado un número.

### IF –THEN – ELSE

El formato que debe seguir es el siguiente:

```
If (Condición) Then
    Conjunto de instrucciones
Else
    Conjunto de instrucciones
End If
```

Lo que se debe de entender como:

```
Si (Condición) Entonces
    Se realiza el Conjunto de instrucciones
    Si se cumple la condición y
    Se cierra la estructura

Else
    Se realiza el Conjunto de instrucciones
    Si no se cumple la condición y
    Se cierra la estructura

Fin de la estructura
```

Como ejemplo modifique el programa anterior para que en el caso de que el nombre sea incorrecto aparezca un letrero que diga “**La contraseña es incorrecta**”.

### ANIDACIÓN DE ESTRUCTURAS

Todas las estructuras que se han visto y que se verán más adelante pueden ser anidadas mientras se respete la regla del cierre de estructuras.

Esto es que no se puede cerrar una estructura de nivel superior sin antes haber cerrado una de nivel inferior que se encuentra alojada en él. Por ejemplo:

Incorrecto	Correcto
<pre>(1) If (condicion1) Then     (2) If (condicion2) Then         Hace algo     (1) End IF     (2) End If</pre>	<pre>(1) If (condicion1) Then     (2) If (condicion2) Then         Hace algo     (2) End IF     (1) End If</pre>

Ejemplo:

El código de un programa que asigna una letra de calificación según el número que haya obtenido el alumno.

```

Program pruebaIf
  Implicit none
  REAL::numero

  WRITE(6,*)"Dame el numero"
  WRITE(*,*)
  READ(5,*)numero
  If (numero .GT. 5) then
    if ((numero.EQ.6).OR.(numero.EQ.7)) then
      WRITE(6,*)"La calificacion es S"
    endif
    if ((numero.EQ.8).OR.(numero.EQ.9)) then
      WRITE(6,*)"La calificacion es B"
    endif
    if (numero.EQ.10) then
      WRITE(6,*)"La calificacion es MB"
    endif
  else
    WRITE(6,*)"La calificacion es NA"
  endif
  stop
end program pruebaIf

```

### SELECT CASE

La estructura del Select Case es mas conveniente para verificar varias condiciones que se basan en un común denominador. Su formato general se muestra a continuación:

```

Select Case Variable
  Case condicion1
    Instrucciones a realizar
  Case condicion2
    Instrucciones a realizar
  Case condicion3
    Instrucciones a realizar
  Case condicionN
    Instrucciones a realizar
  Case Default
    Instrucciones a realizar
End Select

```

En donde en las condiciones se puede trabajar con caracteres como “A” , o con rangos como (0 : 59) o inclusive verificar si se es positivo o negativo con (-:1) o (:1).

El último Case que es el Case Default se refiera a la condición que se realizará en caso de que la variable no haya sido ninguno de los casos anteriores.

Una observación importante es que en la estructura “Select Case” no podemos comprobar variables flotantes.

Como ejemplo: desarrolle el programa anterior pero mediante la aplicación de la estructura Select Case

### Ejercicios propuestos.

Un programa que tome 3 números y analice cuál es el mayor.

Un programa que tome 4 números y diga cual es el menor positivo, utilizando la estructura IF-ELSEIF.

Una agenda telefónica usando la estructura SELECT CASE, en donde el nombre y número telefónico de las personas sea referido por el apellido paterno.

## Estructuras de Repetición

Las estructuras de repetición son útiles ya que dan lugar a un lazo o bucle, esto es, permiten ejecutar un conjunto de sentencias cierto número de veces.

### DO

Para ejecutar repetidamente una serie de sentencias en bucle, tiene mucha utilidad en el lenguaje FORTRAN el uso de una sentencia especial: la sentencia DO. Su sintaxis es:

```
DO Var = expresion1, expresion2, [expresion3]
  Sentencias
END DO
```

Cuando se trata de una estructura tipo DO se requiere emplear una variable que sirva para asignar el número de veces que se repetirán las sentencias dentro de la estructura.

Var es una variable entera y expresion1, expresion2 y expresion3 son constantes o variables enteras, reales, o de doble precisión e indican un valor inicial, un valor final y un incremento respectivamente.

El valor de expresion1 debe ser mayor que el valor de expresion2, de otro modo el control del programa salta hasta las líneas que se encuentren después de Next; aunque si se asigna un valor negativo a expresion3 el valor de Variable sufrirá un decremento en lugar

de un incremento y las sentencias dentro de DO si se ejecutarán. La variable expresion3 puede omitirse si su valor es igual a 1, ya que indica el incremento que tendrá la variable para llegar de expresion1 a expresion2.

Ejemplo:

Obtener el promedio de las calificaciones de las materias de un alumno.

```

Program ejemploDo
  Implicit none
  INTEGER::i
  REAL::prom=0.0
  REAL, DIMENSION(10)::calif

  open(UNIT=1,FILE="C:\Calificaciones.txt",ACTION='READ')
  do i=1,10
    READ(1,*)calif(i)
  end do
  CLOSE(1)

  do i=1,10
    prom=prom+calif(i)
  end do
  prom=prom/10
  print *, 'promedio= ',prom
  stop
end program

```

### DO (IF)

La utilidad de este tipo de estructura radica en las veces que no podemos conocer el número de veces que necesitamos que un cierto número de sentencias se repitan, es decir, su repetición se basa en que se cumpla cierta condición; de este modo la estructura se ejecutará mientras o hasta que se cumpla la condición establecida. Al igual que en las estructuras de condición, el formato estará basado en operadores de comparación y en operadores lógicos si es necesario.

La forma general de un DO controlado por expresión lógica es:

```

DO
  Sentencias
  IF (condicion) EXIT
  Sentencias
END DO

```

Las sentencias se ejecutan repetidamente hasta que la condición tome un valor cierto. Se debe tener cuidado ya que si la condición nunca fuera cierta el bucle se repetiría indefinidamente. Se puede variar la forma de la estructura dependiendo si las sentencias se

escriben antes o después de IF. Si las sentencias se plantean solamente después de IF, primero se revisará la condición y si el resultado es cierto se realizarán las sentencias, de otro modo el control del programa continuará con las líneas de código localizadas después de END DO.

Si las sentencias se plantean solamente antes de IF la condición es revisada al final de la estructura, es decir, primero se ejecutan las sentencias y después se decide si se continua o no con el bucle. Por ejemplo si el programa requiere para plantear la condición que se realicen ciertas operaciones primero esta estructura resulta muy adecuada ya que el control del programa entrará a ella sin importar las condiciones de las variables involucradas, realizará las sentencias hasta encontrar IF, revisará la condición y en base al resultado regresará al lugar del Do o bien continuará con las líneas de código escritas después de END DO.

En lugar del uso de la sentencia EXIT es posible utilizar la sentencia CYCLE; la diferencia radica en que con EXIT la estructura se rompe, es decir salimos del ciclo DO, pero con la sentencia CYCLE no se termina el ciclo solamente pasamos a la siguiente iteración.

Ejemplo:

Encontrar la intersección de dos curvas que se cruzan.

```

program ejemplo2
  call cruce1()
  stop
end program

subroutine cruce1()
  REAL::X=0.0,Y1,Y2,error1
  do
    X = X + 0.001
    Y1 = (20 * X)**(1/2) + 23
    Y2 = X **2 - 26
    error = Abs(Y2 - Y1)
    if (error <= 0.1) THEN
      exit
    end if
  END do
  WRITE(6,*)'X =',X
end subroutine

```

### DO - WHILE

Su forma es comparable con colocar una estructura IF al principio del DO; las sentencias dentro de DO – WHILE se llevarán a cabo mientras la condición establecida sea cierta, de otro modo el control del programa continuará con las sentencias después de END DO.

Su forma general es:

```
DO WHILE (condicion)
    Sentencias
END DO
```

Ejemplo:

Encontrar la intersección de dos curvas que se cruzan.

```
program ejemplo2
    call cruce2()
    stop
end program

subroutine cruce2()
    REAL::X=0.0,Y1,Y2,error1=1.0
    do WHILE(error1 >= 0.1)
        X = X + 0.001
        Y1 = (20 * X)**(1/2) + 23
        Y2 = X ** 2 - 26
        error1 = Abs(Y2 - Y1)
    END do
    WRITE(6,*)'X =',X
end
```

### Ejercicios propuestos.

Escribir en un archivo una lista ordenada de 20 números.

Realizar un programa que sume y reste dos matrices, que pregunte al usuario después de cada operación si desea repetir el proceso o leer nuevos valores para las matrices.