CIDIAT

Curso de postgrado
en
Desarrollo de Recursos de Aguas y Tierras

APUNTES DE PROGRAMACION

por

Hervé J. JEGAT

Mérida, febero de 1997

CAPITULO I

INTRODUCCION A LA COMPUTACION

I. GENERALIDADES

I.1 El computador

El computador es un equipo electro-mecánico capaz de procesar información en forma rápida y confiable. En particular, el computador puede :

- ejecutar operaciones elementales.
- ejecutar una sola operación a la vez.
- ejecutar millones de operaciones por segundo.
- almacenar y manejar gran cantidad de información.

Toda operación a ser ejecutada por el computador debe formar parte de un programa, el cuál es un conjunto de instrucciones básicas a ser ejecutadas según un orden lógico preestablecido.

En general, el computador comprende una unidad central de procesamiento (CPU), y unidades periféricas de entrada y salida. El CPU comprende la memoria principal, la unidad de control y la unidad de procesamiento propiamente dicha.

Los periféricos de entrada suelen ser :

- Lectora de tarjetas perforadas (ya en desuso).
- Unidad de respaldo (casete, disquete, CD,...).
- Terminal de vídeo (cónsola).
- Unidad de disco (duro o flexible).
- Tabla de digitalización.

y los de salida :

- Terminal.
- Unidad de espaldo.
- Disco duro o disquete.
- Impresora.
- Graficador (plotter).

Los periféricos de entrada proporcionan información (datos y programas) en forma manejable por el computador. La memoria central almacena los datos y programas, la unidad de control controla el flujo de información entre la

memoria y la unidad de procesamiento, y activa los periféricos de acuerdo a las instrucciones recibidas. La unidad de procesamiento ejecuta las operaciones programadas, utilizando los datos almacenados en memoria central. Finalmente, los periféricos de salida proveen resultados en forma aceptable por el usuario.

I.2 El procesamiento.

La unidad de procesamiento trabaja en base a información en notación binaria, o sea 0 o 1 (pasa o no pasa corriente). El tamaño del microprocesador central, o " chip ", define la capacidad del computador. Hasta hace uncos años, los microcomputadores usaban un chip de 8 bits; luego fueron de 16 bits y ahora la mayoría son de 32 bits.

Aqui cabe definir algunos términos utilizados :

- un bit es la mínima información almacenada por el computador y corresponde a un dígito binario (0 o 1).
- Un byte es el conjunto de ocho bits
- Una palabra equivale a dos bytes.
- Kb. significa Kilobyte, o sea 1024 bytes (2^{10}).
- Mb. significa Megabyte, o sea un millón de bytes.

Si el computador trabaja en base a un chip de 16 bits, característico de los minicomputadores de hace unos años, el máximo valor entero que puede almacenar para direccionar una información, corresponde a 1111111111111111, o sea 2^{16} en notación decimal, lo que vale 65536 o en forma normalizada 64 Kb. Es una limitación importante, sin embargo se verá mas tarde como se puede obviar.

II. EL EQUIPO DE COMPUTACION

II.1 El sistema operativo.

Se llama sistema operativo al conjunto de programas que maneja los recursos del computador. Entonces, el sistema operativo puede ser considerado como un gerente quién :

- conoce siempre el estado de los recursos del equipo de computación.
- sabe quién , cuando y como usan esos recursos.
- distribuye esos recursos entre los usuarios.
- recupera los recursos cuando sea necesario.

Estos recursos, como se ha visto antes, vienen constituidos por el CPU, la memoria central y los periféricos.

Los sistemas operativos usados actualmente son el DOS con o sin el ambiente Windows, así como el OS-2, UNIX (o LINUX), esos últimos principalmente en estaciones de trabajo.

II.2 El manejo de archivos.

Un archivo es un conjunto de datos de información ;

- lluvias diarias de una cierta estación.
- listado de estudiantes.
- conjunto de instrucciones de un programa.

Un registro es un elemento de información de un archivo y puede ser :

- la lluvia de un día.
- el nombre de un estudiante.
- una sola instrucción (Fortran por ejemplo).

Un archivo puede estar ubicado en diferentes sitios, tales como en disco, en disquete,... . Cada uno de esos dispositivos contiene un directorio de archivos, el cuál contiene información acerca del mismo disco como también el listado de los nombres y ubicación de todos los archivos que ahí residen.

CAPITULO II

DESARROLLO DE PROGRAMAS EN FORTRAN

I. GENERALIDADES

En este capítulo, se describirá el proceso de desarrollo de programas en FORTRAN 77. Cuando se quiere desarrollar un programa para resolver un problema específico, hay que seguir un cierto procedimiento que asegure al usuario un programa final correcto y confiable. Por lo tanto, actualmente, el desarrollo de programas debe hacerse según una secuencia de pasos bien definidos. Las etapas consecutivas en este proceso son las siguientes :

1. Identificación del problema y definición de objetivos.

2. Selección del método de resolución.

3. Descripción del algoritmo y concepción del flujograma.

4. Codificación del programa.

5. Compilación del programa y corrección de los errores.

6. Carga del programa en memoria central.

7. Corrida del programa y corrección de los errores.

8. Documentación del programa.

Cada una de esas etapas tiene sus objetivos específicos, los cuales al final facilitan la obtención de un programa correcto y que cumpla con el objetivo inicial. A continuación, se detallan las diferentes etapas por separado, ilustrándolas con un ejemplo sencillo.

II. IDENTIFICACION DEL PROBLEMA Y DEFINICION DE OBJETIVOS.

A este nivel, se debe saber claramente cuál es el problema que se quiere resolver, es decir cuales son los datos básicos del mismo y que resultado se espera tener despues de resolverlo, para establecer con claridad cuál es el objetivo final que se persigue con el programa que se va a escribir.

Por ejemplo, se puede considerar el problema siguiente :

Cuál es la suma de los n primeros números enteros ?

En este caso, los datos del problema son el número n de enteros que se quieren sumar y esos mismos números. El resultado busquado es la suma de esos n números.

III. SELECCION DEL METODO DE RESOLUCION.

Ahora, el problema es : conociendo lo que se quiere, como obtenerlo. Se hace necesario seleccionar el método de resolución que mejor se adapte al problema, tomando en cuenta que el problema se va a resolver por computadora y el lenguaje de computación que se va a usar.

Para el ejemplo anterior, hay varias alternativas para sumar los n primeros enteros :

1. Hacer la suma de los números uno por uno en orden creciente.
2. Hacer la suma de los números uno por uno en orden decreciente.
3. Utilizar la fórmula : $S = n(n+1)/2$.

En este caso, se seleccionará la primera alternativa, o sea sumar los números en orden creciente.

IV. DESCRIPCION DEL ALGORITMO Y CONCEPCION DEL FLUJOGRAMA.

Cuando ya se conoce el método que se va a usar, entonces es necesario describir el algoritmo de resolución. Este consiste en una lista clara y sin ambigüedades de las operaciones que se deben realizar, en forma de instrucciones elementales. Luego, a partir del algoritmo, se realiza el flujograma o diagrama de flujo que es una representación gráfica del algoritmo elaborado. Esa representación gráfica se hace en base a una serie de figuras geométricas normalizadas, las cuales se presentan en la Figura V. El diagrama de flujo consiste entonces en describir el algoritmo por una secuencia de instrucciones incluidas dentro de esas figuras geométricas.

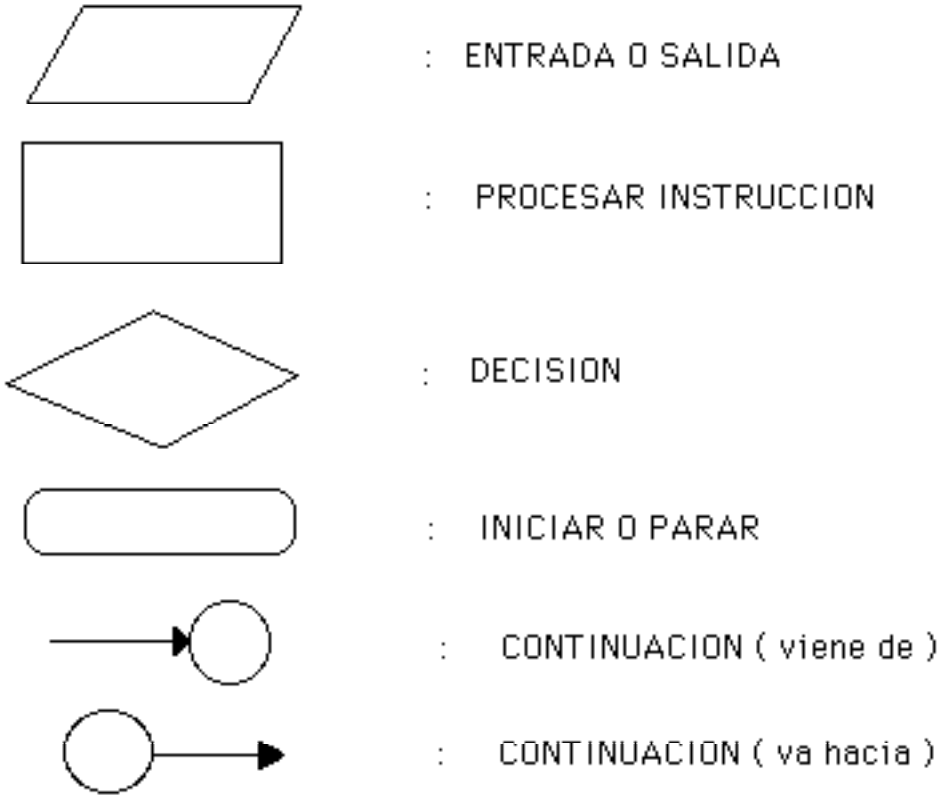


FIGURA V.

Para el ejemplo presentado, la descripción del algoritmo será la siguiente :

1. Leer el valor de n.
2. Inicializar la suma en cero.
3. Empezar con el número 1.
4. Si este número es inferior o igual a n, seguir.
   Sino, ir al paso 8
5. Sumar el número a la suma anterior.
6. Pasar al número siguiente.
7. Volver al paso 4.
8. Imprimir el valor de la suma.

Esa secuencia de instrucciones elementales describe los pasos que se deben seguir para realizar el calculo de la suma de los n primeros números enteros. Sin embargo, esa representación escrita del proceso de cálculo no es la mas conveniente y una representación gráfica del algoritmo puede resultar mucho mas cómoda para la codificación posterior del programa. Por lo tanto, siempre es muy útil hacer un diagrama de flujo antes de realizar la codificación del programa en el lenguaje seleccionado. Eso ha demostrado ser muy provechoso en el caso del Fortran.

Así, el algoritmo anterior puede ser representado por el flujograma siguiente, usando las figuras normalizadas (Figura VI). En cada una de esas figuras, aparecen las instrucciones del algoritmo tales como han sido descritas anteriormente.

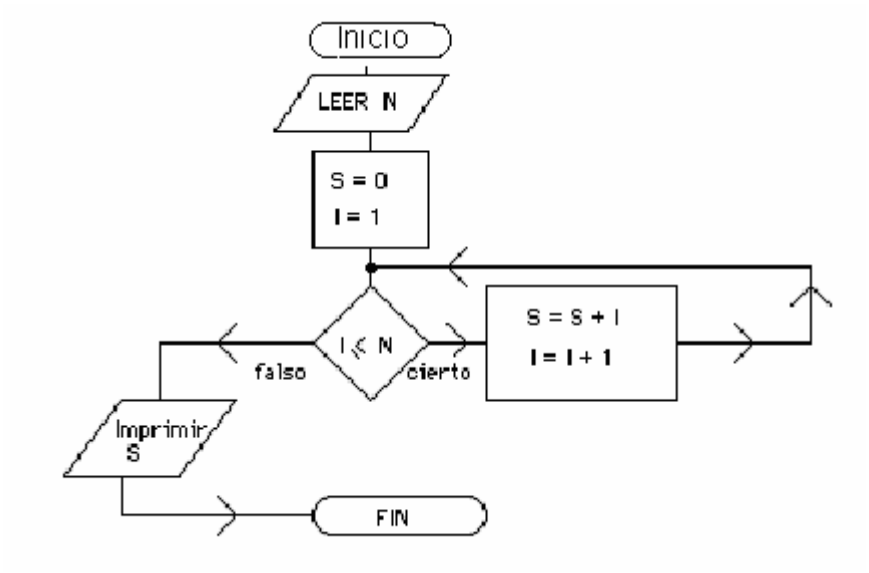


FIGURA VI.

En los últimos diez años, se han desarrolladas técnicas para mejorar la elaboración de programas. En particular, la técnica de programación estructurada ha resultado ser muy útil en la obtención de programas correctos y confiables. En la programación estructurada, las combinaciones de figuras permitidas en los diagramas de flujo son limitadas. Prácticamente, se permiten tres estructuras básicas. Esa técnica de programación estructurada se verá con mas detalles en el capítulo siguiente.

V. CODIFICACION DEL PROGRAMA.

La codificación del programa consiste en la traducción en el lenguaje seleccionado de las instrucciones descritas en el diagrama de flujo. En el presente caso, esta codificación se hace en Fortran 77. Teniendo un flujograma bien realizado y suficientemente detallado, la transcripción en Fortran de un programa no debe presentar muchas dificultades. De ahí la importancia del flujograma en el desarrollo de programas. Además, al usar la técnica de programación estructurada, la escritura del programa en Fortran se hace en el mismo orden de las instrucciones del diagrama de flujo.

También debe descatarse la importancia de intercalar comentarios en las instrucciones del programa. Resultarán de mucha utilidad en el momento de modificar el programa o de volver a usarlo después de algún tiempo.

Para el mismo ejemplo anterior, el programa codificado resultará ser :

```
      PROGRAM SUMA
      READ (1,*) N
      S=0
      I=1
10    IF ( I . LE . N ) THEN
      S=S+I
      I=I+1
      GO TO 10
      ENDIF
      WRITE (1,*) S
      END
```

En este ejemplo, se usan instrucciones Fortran que se detallarán en otros capítulos con el fin de ilustrar las diferentes etapas del desarrollo del programa. Sin embargo, esas instrucciones se entienden fácilmente y se puede ver que siguen el mismo orden del flujograma.

VI. COMPILACION, CARGA DEL PROGRAMA Y CORRECCION DE ERRORES.

La compilación del programa consiste en traducir en lenguaje máquina, el programa escrito en Fortran 77. Esa compilación se hace corriendo un programa (el compilador) que realiza esa traducción.

Al hacer la compilación se detectan los errores de sintaxis, es decir los errores cometidos en la codificación del programa al no respetar las reglas de escritura (caracteres equivocados, falta de paréntesis, falta de operador, ...). Todos esos errores son detectados por el compilador y se hace necesario corregirlos para seguir desarrollando el programa, o sea pasar a las etapas siguientes.

VII. CARGA, EJECUCION DEL PROGRAMA Y CORRECCIONES.

Una vez compilado el programa ha sido traducido al lenguaje de la máquina y existe como programa objeto. Para poder ser ejecutado, debe ser cargado en la memoria central con todas las librerías y subrutinas que pueda necesitar. Entonces el programa se encontrará en forma ejecutable en la unidad central de procesamiento, CPU, del computador. A este nivel, se pueden detectar los errores de lógica del programa, tales como :

- división por cero.
- logaritmo de un número negativo.
- errores de formatos de entrada o salida.
- arreglos mal dimensionados.
- etc ...

También se puede validar el programa corriéndolo para un caso cuya solución se conoce o haciendo chequeos manuales de partes del mismo cuando es posible. Así se tendrá la seguridad de que el programa escrito cumple con el objetivo inicial. Si no es el caso, entonces hay que reiniciar el procedimiento de desarrollo del programa desde el principio y chequear cada uno de lo pasos.

IX. DOCUMENTACION DEL PROGRAMA.

Teniendo ahora un programa correcto y confiable, es *indispensable* tener también una documentación precisa y completa del mismo. En efecto, cualquier usuario futuro de este programa no ha participado en la elaboración del mismo y, por lo tanto, no está enterado de como trabaja dicho programa. Igualmente, cualquier modificación o mejora que se quiera hacer al programa hace necesario conocer perfectamente el programa original. Esa documentación debe incluir lo siguiente :

Nombre y objetivo del programa.
Fecha de elaboración y autor (para cualquier información adicional).
Listado de variables y su significado.
Formato de introducción de los datos.
Formato de salida de los resultados.
Un ejemplo de corrida con sus respectivos datos de entrada y resultados.
Referencias bibliográficas.

CAPITULO III

PROGRAMACION ESTRUCTURADA

I. CONCEPTOS GENERALES.

A mediados de los años 60, la complejidad creciente de los programas así como su mayor tamaño, hizo que fueron necesarios equipos de programadores trabajando juntos para desarrollar programas importantes. También, el mantenimiento de los programas resultó ser una parte muy importante de las actividades de los centros de computación. Entonces, varios investigadores empezaron a trabajar en un estilo disciplinado y estructurado de programar. Las ideas mas importantes que manejaron entonces fueron las de evitar los "GOTO" incondicionales, de diseñar programas de "arriba hacia abajo", y de hacer diseños modulares. Basándose en eso, se desarrolló un conjunto de técnicas de programación que hoy en día se conocen como " Programación Estructurada ".

En este capítulo, se presentan los elementos mas importantes en el uso de la programación estructurada, insistiendo en la filosofía de esa técnica. Para tal fin, se usará también un ejemplo detallado de aplicación de esas técnicas.

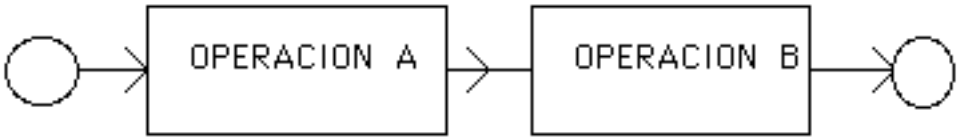
II. ESTRUTURAS BASICAS DE LA PROGRAMACION ESTRUCTURADA.

La idea básica de esa técnica es que cada módulo del programa debe tener un solo punto de entrada y un solo punto de salida. El tamaño de esos módulos depende del nivel de detalle, pero en cada nivel se debe respetar esa regla. Por lo tanto, existen tres módulos básicos que se emplean para programar. Esas tres estructuras son las siguientes :

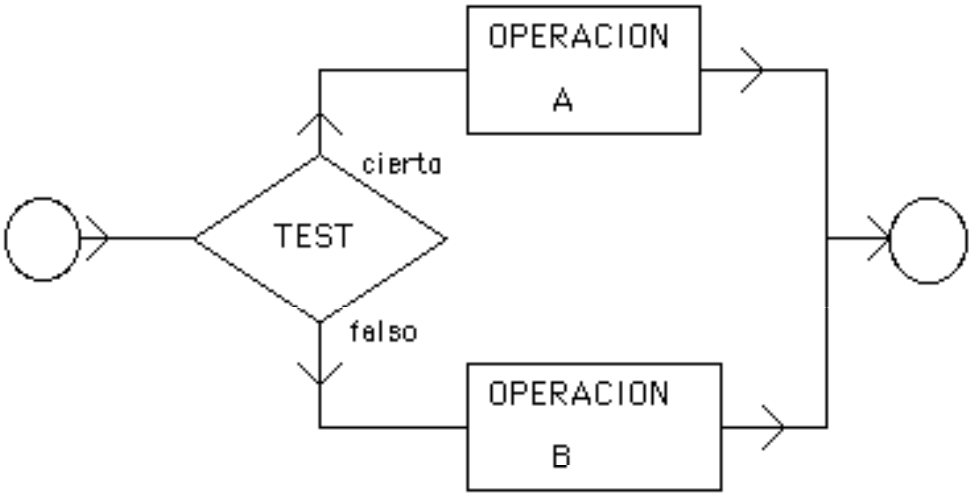
Secuencia simple
Selección o ramificación condicional
Repetición o iteración condicional

En la figura siguiente se presentan esas tres estructuras básicas.

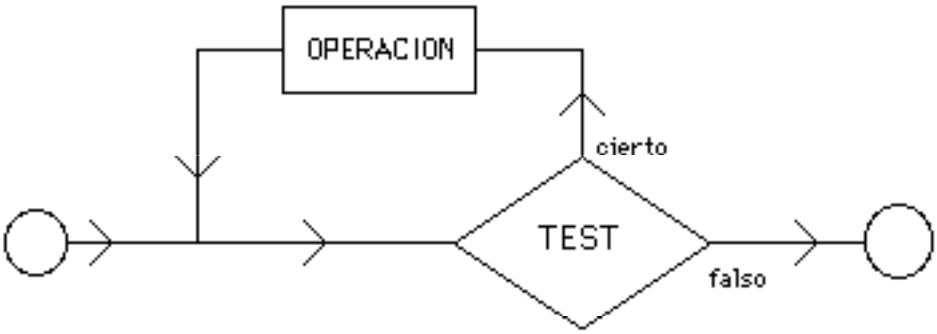
Secuencia



Selección



Iteración



La restricción a esos tres tipos de estructuras no es arbitraria, sino basada en consideraciones sobre la corrección de programas. En efecto, esas tres estructuras se caracterizan por tener una sola entrada y una sola salida, y todas ellas pueden ser reemplazadas por un solo bloque con una entrada y una salida. Además, cuando se codifican, pueden ser leídas de arriba hacia abajo en forma continua, sin tener que saltar de una línea a otra.

Cada una de esas estructuras corresponde a un tipo de operación diferente :

Secuencia de instrucciones
Estructura " IF-THEN-ELSE "
Estructura " DO-WHILE "

La estructura de secuencia consiste en una acción seguida por otra, o sea se realiza la operación A y luego la operación B.

La estructura de selección consiste en una prueba para una condición, seguida por dos rutas alternativas que el programa seguirá. Dependiendo del resultado de la prueba, el programa seguirá una u otra ruta. Luego de pasar por una de esas rutas, el programa regresa a un solo punto. Este tipo de operación puede identificarse como " IF ... THEN ... ELSE " es decir " SI ... ENTONCES ... DE LO CONTRARIO ". Eso quiere decir que si la condición de la prueba es verdadera, entonces se realizará la operación A; de lo contrario, se efectuará la operación B.

La estructura de repetición también puede llamarse de iteración, ya que la operación especificada se repite mientras se cumple la condición de la prueba. Lo que puede identificarse como una secuencia del tipo " DO WHILE " (Hacer Mientras). En la lógica de un programa, se prueba una condición, la cuál rige la ejecución de la operación que la sigue. Si esta condición es verdadera, se ejecuta la operación y se repite la prueba. Por lo tanto, esta operación se repetirá tantas veces como se cumpla la condición de prueba. Cuando esa condición ya no es verdadera, se termina el ciclo y el programa pasa a la instrucción siguiente.

La ventaja de usar solamente esas tres estructuras es la de realizar programas mas fáciles de entender y mantener. Es posible combinar esas tres estructuras para producir una secuencia mas compleja, pero sin perder nunca la simplicidad inherente a esas tres figuras. Ha sido demostrado que en base a esas tres estructuras, se puede expresar en Fortran 77 la lógica de cualquier programa.

En resumen, el empleo de esas figuras permite :

1. Simplificar los programas. Existe un solo punto de entrada a cada estructura asi como un solo punto de salida.

2. Leer un programa de arriba hacia abajo, haciendo mas visible la lógica del mismo, facilitando asi su corrección y modificación.

El lenguaje Fortran no fue diseñado inicialmente para programación estructurada. Sin embargo, con las modificaciones introducidas en el Fortran 77, se puede seguir esa filosofía en forma razonablemente buena. Una de las implicaciones del uso de esa técnica en Fortran, es la exclusión del " GOTO " incondicional, lo cuál se discutirá en los capítulos del Fortran.

III. EJEMPLO DE PROGRAMA ESTRUCTURADO.

En la figura siguiente, se presenta el diagrama de flujo de un programa, el cuál se va a analizar en base a lo anterior.

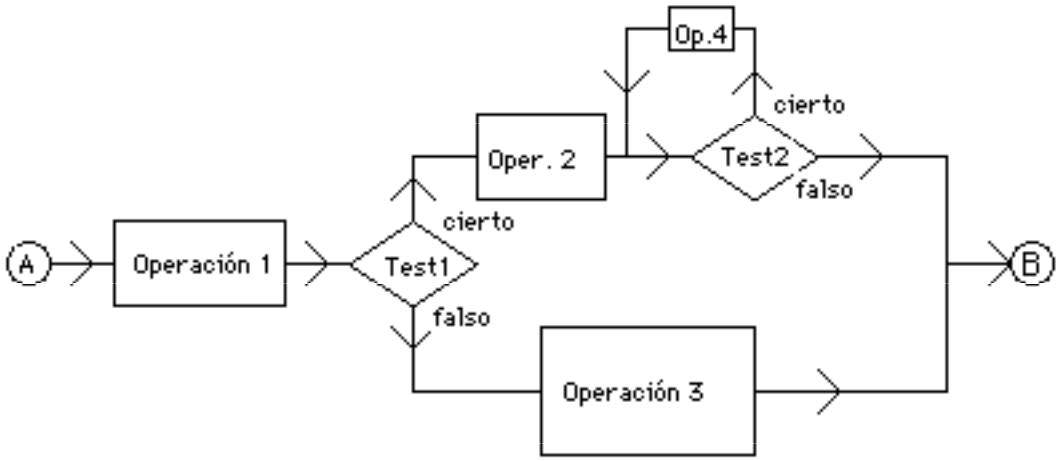


FIGURA VII

En este flujograma, se pueden identificar varias de las estructuras básicas vistas anteriormente. Por ejemplo, en la Figura siguiente, lo que aparece en el rectángulo representa una estructura tipo " DO-WHILE ".

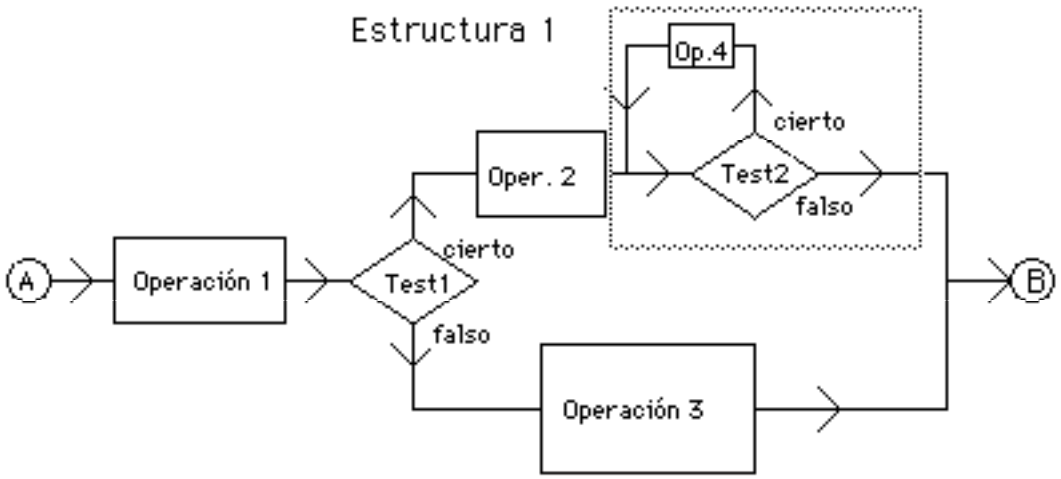


FIGURA VIII.

En este rectángulo, se hace una prueba sobre la condición Test2, la cuál manda a realizar la operación 4 mientras sigue verdadera. Cuando esta condición ya no se cumple, el programa sigue a la instrucción B. Esta secuencia puede entonces ser reemplazada por un rectángulo equivalente, tipo caja negra, que cumple la misma función, debido al hecho que ambas tienen una sola entrada y una sola salida. Ahora, se tiene lo que aparece en la Figura IX en la cuál, otra vez, se identifican las estructuras básicas de la programación estructurada.

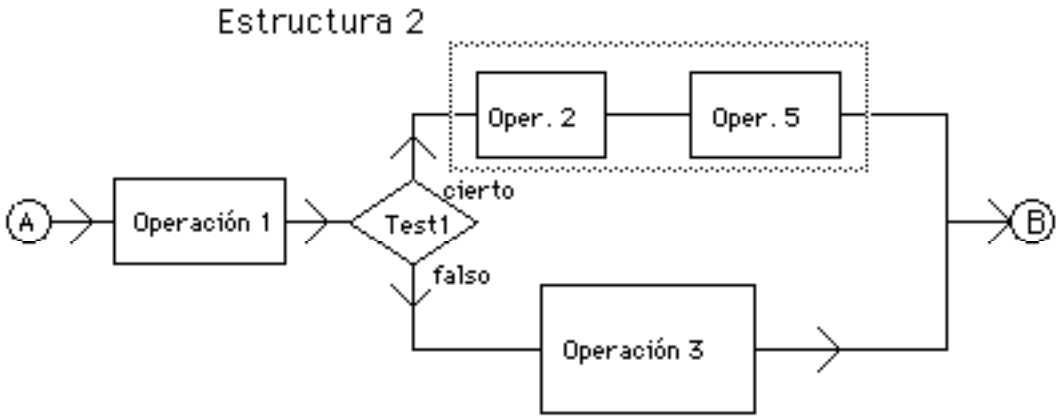


FIGURA IX.

Ahora en el rectángulo, aparece una secuencia simple, la cuál otra vez se va a reemplazar por una caja negra, con una entrada, una salida.

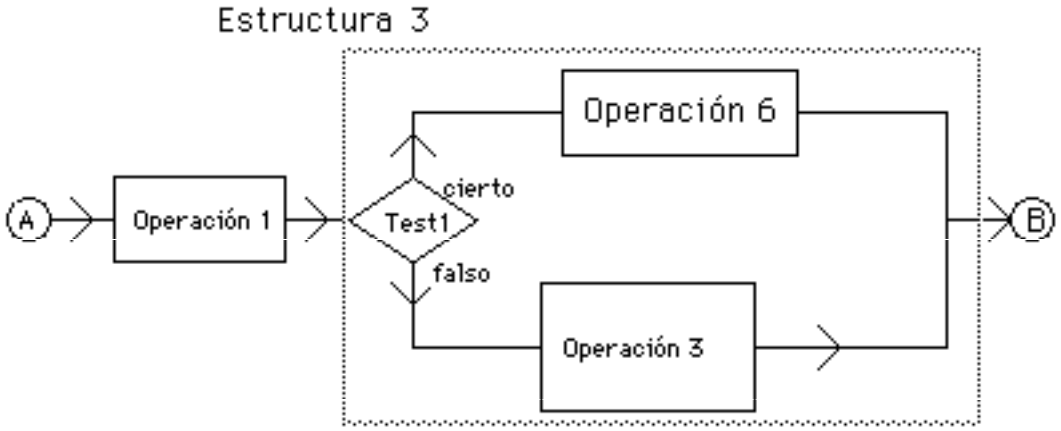


FIGURA X.

El rectángulo encierra ahora la estructura 3, la cuál es del tipo " IF-THEN-ELSE ", en que se chequea la condición Test1. Si es verdadera, se ejecuta la operación 6, sino se realiza la operación 3. Otra vez, la secuencia encerrada en el rectángulo puede ser reemplazada por una operación equivalente de una sola entrada, una sola salida, como aparece en la figura siguiente :

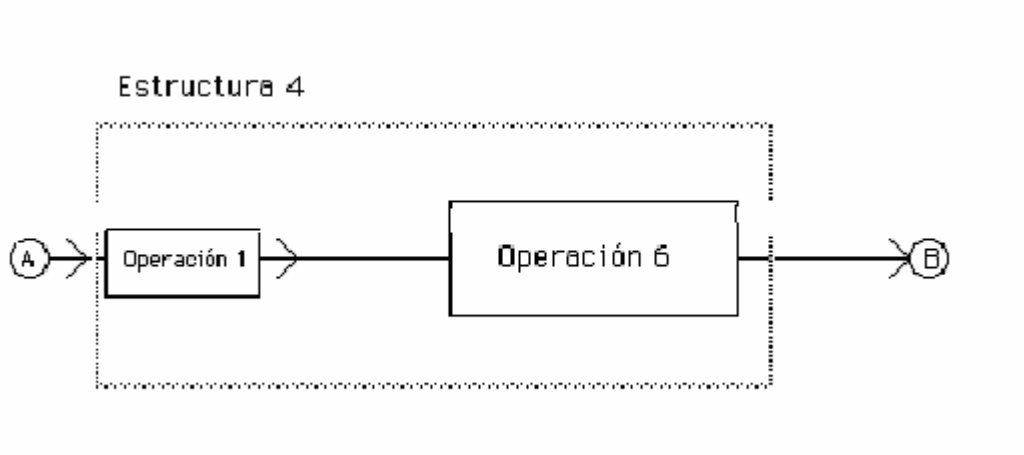


FIGURA XI

Ya se tiene una secuencia simple de operaciones, la cuál reemplaza todas las operaciones anteriores. Esa secuencia viene encerrada en un rectángulo y puede ser también reemplazada por otra operación, tal como aparece finalmente en la Figura XII.

Así, se puede ver en este caso sencillo, la aplicación de la programación estructurada, basándose siempre en las mismas tres estructuras básicas, secuencia simple, selección y repetición, respetando la condición de una sola entrada, una sola salida.

Adicionalmente, se puede ver que la codificación del programa presentado en el diagrama de flujo inicial, podrá hacerse siguiendo directamente el orden de las instrucciones en dicho diagrama, lo cuál también es característico de la técnica empleada.

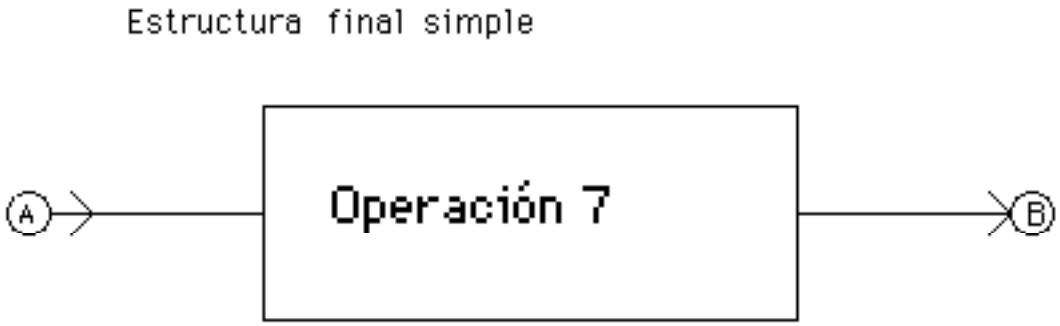


FIGURA XII.

IV. DISEÑO DEL PROGRAMA.

En el ejemplo anterior, se ha simplificado progresivamente el diagrama de flujo hasta llegar a una sola operación. La técnica de diseño de arriba hacia abajo consiste en la operación inversa, es decir ir detallando cada vez mas el flujograma.

Primero, se va entonces a usar un solo bloque de operaciones de una entrada, una salida tal como el de la Figura XII. Luego, en cada etapa del diseño, se refina el bloque de operaciones, empleando siempre estructuras de una sola entrada, una sola salida.

Este proceso de reemplazo o de refinamiento sucesivo producirá finalmente un diagrama de flujo similar al de la Figura VII. Lo que permitirá luego codificar sin problema el programa y también facilitará en el futuro la corrección y modificación del mismo.

En efecto, el uso de estructuras de una sola entrada, una sola salida, permite luego incorporar nuevas estructuras al programa sin introducir nuevas conexiones con el resto del flujograma. Como se ha mencionado antes, el uso del GOTO incondicional debe ser eliminado o por lo menos restringido al mínimo ya que puede introducir muchas posibilidades de ramificación en el programa, el cuál perdería entonces su característica mas importante.

Por ejemplo, en la Figura XIII, aparece un diagrama de flujo no-estructurado.

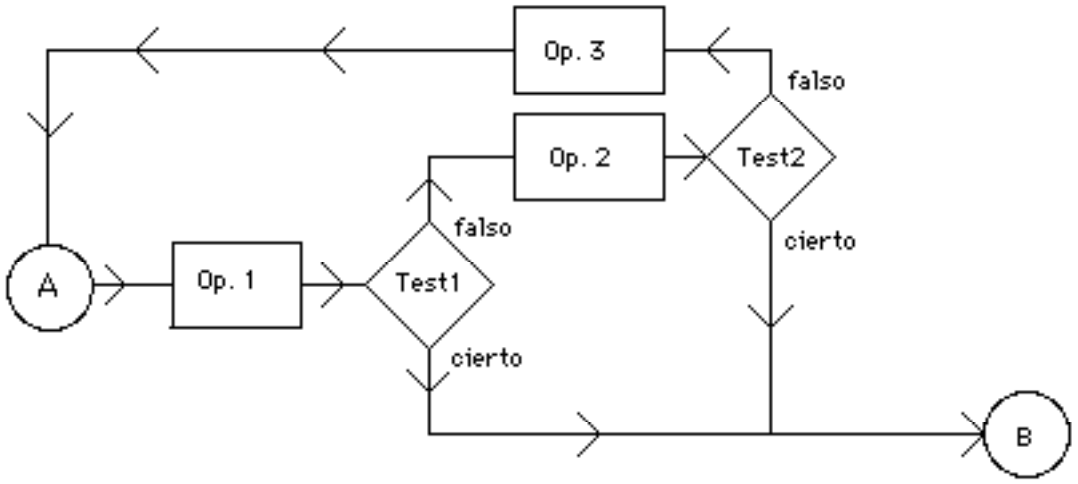


FIGURA XIII.

De hecho, resulta imposible reducir dicho flujograma a un bloque de una sola entrada y una sola salida. Cuando no se cumple la condición Test2, se realiza la operación 3 y luego se regresa al punto A por una instrucción GOTO, lo cuál hace que el programa no puede reducirse a una sola estructura y por lo tanto no es estructurado.

CAPITULO IV

EL LENGUAJE FORTAN 77

CARACTERISTICAS GENERALES

I. INTRODUCCION.

Un lenguaje de computación permite la comunicación entre el usuario y el computador mediante programas e instrucciones fácilmente manejables por el usuario y evitando toda ambigüedad acerca de lo que quiere hacer el usuario y lo que debe hacer la máquina. El nivel de un lenguaje de computación lo determina la cercanía al nivel de interpretación del usuario. Así un lenguaje de alto nivel está mas cerca del usuario que de la máquina. Un lenguaje de alto nivel debe ser independiente del tipo de computador, pero no así el compilador el cuál hace la traducción del lenguaje del usuario al lenguaje de la máquina. Esa traducción es una operación necesariamente compleja y por lo tanto diferente para cada tipo de computador.

Lenguajes de alto nivel lo son el Cobol, el Algol, el Pascal, el Fortran, el PL/1, el Basic, . . . El Cobol es el lenguaje mas utilizado en el área administrativa y el Fortran el mas usado en el campo científico, por lo menos a nivel de sistemas importantes. A nivel de computador personal, el Basic ha tomado ahora mayor importancia.

El Fortran, acrónimo de FORmula TRANslator, ha sido creado por IBM, principalmente, en el año 1957. Desde entonces, ha sufrido cambios y evoluciones pasando del Fortran original al Fortran II, y luego al Fortran IV. Este último ha sido estandarizado en 1966 por el ANSI, American National Standarts Institute, bajo el nombre de Fortran ANSI 66. En 1977, se adoptó una nueva revisión, la cuál agrega algunas características, aclara puntos ambiguos y hace cambios menores. Esa versión se conoce como Fortran 77. En 1990, se lanzó al mercado una versión mejorada del Fortran, adaptada al ambiente Windows y a las capacidades actuales de los microcomputadores conocida como Fortran 90.

En este capítulo y en los siguientes, se describe el uso del Fortran 77.

II. REGLAS DE CODIFICACION.

Cuando se va a escribir un programa en Fortran, la codificación debe hacerse siguiendo algunas reglas. Una línea consta de 80 columnas, de las cuales algunas tienen un empleo específico :

Columna	Empleo
1	**Comentarios y opciones especiales.** Un asterisco * o la letra C en primera columna indica que lo que sigue en la linea no es una instrucción Fortran y no será traducido por el compilador. Sin embargo, se imprimirá en el listado del programa, y por lo tanto se usa para comentarios de explicación del mismo.
1 - 5	**Etiqueta de instrucción.** Se utiliza como identificador para hacer referencia a esta instrucción. Puede escribirse en cualquiera de las 5 columnas, pero generalmente se coloca hacia la derecha, dejando espacios libres a la izquierda.
6	**Continuación de línea.** Cuando una instrucción no cabe en una sola línea, puede continuarse en la siguiente, colocando una carácter diferente de cero en la columna 6 de la línea de continuación.
7 - 72	**Instrucciones FORTRAN.** La instrucción puede empezar en cualquiera de las columnas 7 a 72. Para facilitar la lectura del programa, pueden usarse sangrías y espacios
73 - 80	**Numeración de líneas.** Cuando se usaban tarjetas perforadas, las columnas 73 hasta 80 podían ser utilizadas para numerarlas.

Lo importante de lo anterior es que las instrucciones Fortran *deben codificarse entre las columnas 7 y 72, ambos inclusivos.* Sin embargo, los datos pueden suministrarse usando las 80 columnas de la línea. Cabe también destacar la diferencia existente entre la etiqueta que puede tener una instrucción, la cuál sirve para referenciar dicha instrucción en el programa, y la numeración de las líneas por el editor. Esa numeración de líneas por el editor es propia del mismo y no puede ser utilizada por el programa en Fortran.

Se puede recalcar también la confusión posible entre diferentes caracteres al codificar un programa. Particularmente eso puede ocurrir con la letra o y la cifra 0,

y también la I y el 1. Para evitarlo, se acostumbra usar la letra o tal cuál y usar Ø para el cero.

III. CONSTANTES Y VARIABLES.

1. Constantes.

Una constante es una cantidad que no varía, y por lo tanto puede escribirse usando su mismo valor (por ejemplo : 3.14159 o 9.81). Viene representada por un número, con o sin punto decimal según sea real o entera, y debe seguir las reglas del Fortran :

a. No se usa nunca la coma.
b. El signo + es implícito, pero los números negativos deben usar el signo -.
c. Los ceros a la izquierda se ignoran.

Ejemplos :

Constante aritmética.	Entero	Real
1,0	1	1.0
-6,0	-6	-6.0
1.000.000,00	1000000	1000000.00

También se puede usar la notación científica o exponencial :

Aritmética	Científica	Fortran
2.000.000,00	2×10^6	2. E06
0,00001	10^{-5}	1. E-5

Se puede asignar un nombre a una constante para hacer un programa mas general (por ejemplo : PI o G). En este caso, los nombres se rigen por las siguientes reglas :

a. Puede tener hasta 6 caracteres, los cuales deben ser letras o cifras.
b. El primer carácter debe ser una letra.
c. Si la constante es entera, esa letra debe ser I , J , K , L , M o N.
d. Los espacios en blanco se ignoran.

2. Variables.

Una variable es la representación de una cantidad que se desconoce o que puede variar durante la ejecución de un programa. En álgebra, una variable o incógnita se representa mediante un carácter tal como a, ß o x. En Fortran, se usa un nombre formado por una secuencia de uno hasta seis caracteres, siguiendo las mismas reglas expuestas para las constantes.

Ejemplos :

Algebra	Fortran	Tipo
x	X	real
ß	BETA	real
a	A	real
j	J	entera
i_{min}	IMIN	entera
x_{max}	XMAX	real
_	DELTA	real

Ejemplos incorrectos :

Variable	Comentario
AB.21	Incorrecto : Uso del punto .
2XERT	Incorrecto : Empieza por 2
I+J	Incorrecto : Uso del signo +
CAUDALES	Incorrecto : mas de 6 caracteres
X**2	Incorrecto : Uso del signo *
UN POCO	Correcto : Equivale a UNPOCO

IV. OPERADORES ARITMETICOS.

Son los operadores que permiten realizar operaciones usuales en aritmética o sea : sumar, restar, multiplicar, dividir y elevar a una potencia (+, -, x, : y exp.). En Fortran, esos operadores se expresan mediante el uso de otros símbolos

Aritmética	Fortran
+	+
-	-
x	*
:	/
exp.	**

No se pueden usar dos operadores seguidos, pero se puede encerrar cualquier expresión o parte de ella entre paréntesis. En Fortran, las paréntesis tienen el mismo significado que en álgebra y determinan la secuencia de las operaciones que se van a realizar. Se ejecutan siempre desde adentro hacia afuera y de la izquierda a la derecha.

Ejemplos :

Algebra	Fortran
x y	X*Y
a + b	A + B
a + (b - c)	A + (B-C)
Π r^2	PI * R **2
y^n	Y ** N
$\sqrt{(b2 - 4ac)}$	(B**2 - 4*A*C)**0.5
$W^{1/3}$	W **(1./3.)

V. INSTRUCCIONES ARITMETICAS.

Una instrucción aritmética consiste en una variable separada de una expresión aritmética por el signo " = ".

En Fortran, el signo " = " tiene un significado diferente que en matemática : no significa igualdad de las expresiones sino que el resultado de las operaciones efectuadas del lado derecho, debe ser asignado a la variable ubicada a la izquierda del signo " = ". Por ejemplo :

AREA = 3.14159 *R**2

significa que el resultado de la operación de multiplicar R al cuadrado por PI, debe ser asignado a la variable llamada AREA.

En una operación aritmética, no puede haber constantes ni operadores del lado izquierdo del signo =. Ahí debe estar el nombre de una variable. Del mismo

modo, no puede haber variables desconocidas del lado derecho de la instrucción. Los espacios en blanco se ignoran.

Ejemplos :

X = AB** 2. - C equivale a X=AB**2.-C
X = A + B está permitido si A y B son constantes o variables conocidas.
A + B = X no es correcto.

Si se trata de operaciones mixtas, es decir que se usan variables y/o constantes enteras y reales, entonces, el tipo del resultado lo fija el tipo de la variable ubicada del lado izquierdo. Por ejemplo :

X = I + J	el resultado será real. si I=2 y J= 3, X = 5.
N = X**2.	el resultado será la parte entera. si X = 1.5, entonces N = 2
X = M+5*(X - Y)	el resultado será real. si X=2.4, Y=3.2 y M=6, X =2.0

VI. FUNCIONES INTRINSECAS DEL FORTRAN.

Existe una serie de funciones disponibles directamente en Fortran, las cuales forman parte integral del lenguaje y pueden usarse en cualquier instrucción.

Función	Definición	Tipo	Argumento
ABS(X)	Valor absoluto	Real	Real
IABS(N)	Valor absoluto	Entero	Entero
FLOAT(N)	Convertir en real	Real	Entero
IFIX(X)	Convertir en entero	Entero	Real
EXP(X)	Exponencial de x	Real	Real
SIN(X)	Seno de x	Real	Real
COS(X)	Coseno de x	Real	Real
TAN(X)	Tangente de x	Real	Real
SQRT(X)	Raíz cuadrada de x	Real	Real
ALOG(X)	Logaritmo natural de x	Real	Real
ALOG10(X)	Logaritmo decimal de x	Real	Real

CAPITULO V

EL LENGUAJE FORTRAN 77

INSTRUCCIONES DE ENTRADA Y SALIDA

I. LAS INSTRUCCIONES EN FORTRAN 77.

En Fortran, existen dos clases principales de instrucciones, las instrucciones ejecutables y las no-ejecutables.

Las instrucciones ejecutables especifican la acción que el programa debe realizar mientras que las no-ejecutables contienen información sobre esas mismas instrucciones o sobre las variables del programa : características de los operandos, tipo de datos, especificación de formato, etc ...

Ambos tipos de instrucciones, ejecutables o no-ejecutables, pertenecen a una de las seis categorías siguientes :

1. Instrucciones de definición de programa.
2. Instrucciones de especificación.
3. Instrucciones de asignación de valores.
4. Instrucciones de control.
5. Instrucciones de entrada y salida.
6. Instrucciones de parada y suspensión.

En las tablas siguientes, se presentan las principales instrucciones del Fortran 77 según la categoria a la cuál pertenecen y se especifica si es ejecutable (Ej.) o no-ejecutable (No ej.).

TABLA 1

INSTRUCCIONES DE DEFINICION DE PROGRAMAS	
BLOCK DATA	No ej. Identifica el programa como bloque de datos
END	Ej. Especifica el fin de un programa.
ENTRY	No ej. Provee una entrada alterna a un subprograma
FUNCTION	No ej. Identifica el programa como subprograma FUNCTION
PROGRAM	No ej. Identifica el programa como principal
RETURN	Ej. Transfiere el control de vuelta al programa que hizó la llamada
SUBROUTINE	No ej. Identifica el programa como sub-programa

TABLA 2

INSTRUCCIONES DE ASIGNACION DE VALORES	
ASSIGN	Ej. Asigna un valor a una variable usada en un GOTO o un FORMAT.
DATA	No ej. Asigna valores a variables en la ejecución.
asignacion aritmética	Ej. Representa todas la operaciones aritméticas definidas en el capítulo anterior.

TABLA 3

INSTRUCCIONES DE ESPECIFICACION	
COMMON	No ej. Reserva un espacio commun en memoria.
DIMENSION	No ej. Define los limites de un arreglo .
EMA	No ej. Declara argumentos en EMA.
EQUIVALENCE	No ej. Asocia variables en el mismo sitio en memoria.
IMPLICIT	No ej. Especifica el tipo con el primer cáracter.
PARAMETER	No ej. Define valores de constantes.
tipo	No ej. Asigna un tipo a una variable (INTEGER, REAL, CHARACTER,...)

TABLA 4

INSTRUCCIONES DE CONTROL	
CALL	Ej. Transfiere el control a un subprrograma.
CONTINUE	Ej. Continua la ejecución.
DO	Ej. Hace que las instrucciones se repitan.
DO WHILE	Ej. Idéntica al anterior, mientras una condición sigue verdadera.
GO TO	Ej. Transfiere el control a una instrucción.
IF	Ej. Ejecución condicional de una o varias instrucciones.

TABLA 5

INSTRUCCIONES DE ENTRADA Y SALIDA	
CLOSE	Ej. Cierra el acceso a un archivo.
FORMAT	No ej. Describe el formato de E/S de datos.
OPEN	Ej. Permite el acceso a un archivo.
PRINT	Ej. Transfiere datos hacia afuera.
READ	Ej. Tranfiere datos hacia adentro.
REWIND	Ej. Posiciona un archivo al inicio.
WRITE	Ej. Transfiere datos hacia afuera.

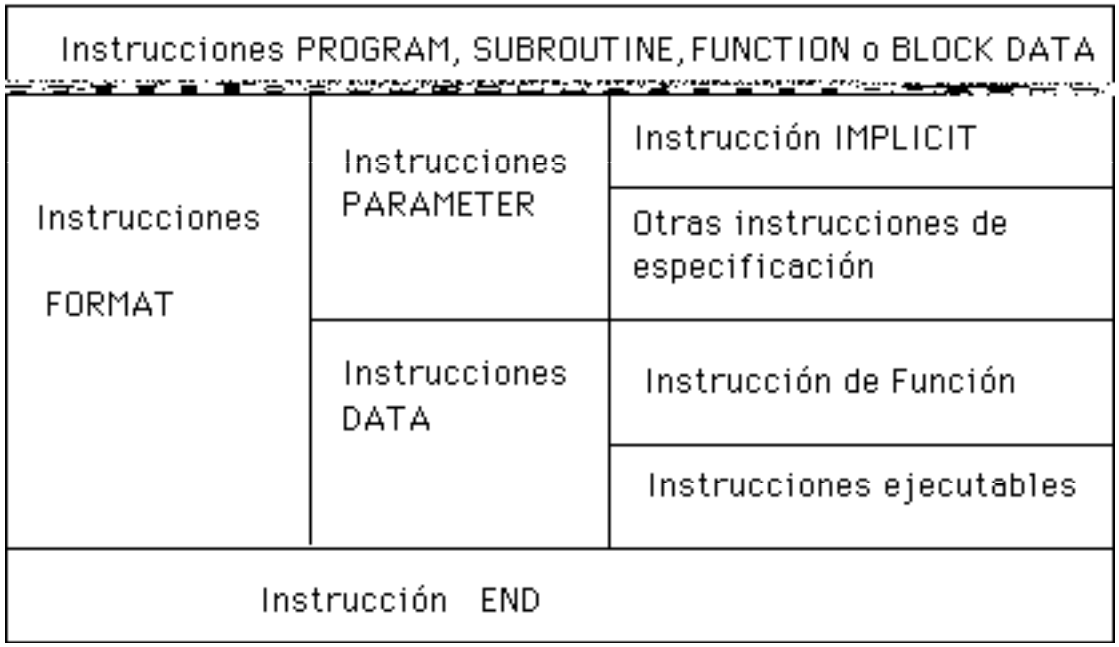
TABLA 6

INSTRUCIONES DE PARADA O SUSPENSION	
STOP	Ej. Termina la ejecución de un programa.
PAUSE	Ej. Provoca la suspensión temporal de un programa.

Cada una de esas instrucciones ejecutables o no-ejecutables, tiene un sitio definido en un programa, es decir que , por ejemplo, una instrucción COMMON no puede estar ubicada en cualquier parte. En la Tabla 7, se presenta el orden que deben seguir las instrucciones dentro del programa. La primera instrucción debe ser PROGRAM o SUBROUTINE o FUNCTION o BLOCK DATA. La última debe ser la instrucción END. Entre esas dos, la instrucción FORMAT puede estar en cualquier

sitio. La instrucción PARAMETER debe estar antes de cualquier instrucción ejecutable, así como las instrucciones de especificación, tales como IMPLICIT, REAL, DIMENSION, INTEGER, etc. ...

TABLA 7



Esas reglas deben respetarse rigurosamente para que el programa pueda ser ejecutado correctamente. En caso de no hacerlo, se producirán errores durante la compilación del programa.

II. INSTRUCCIONES DE ENTRADA/SALIDA SIN FORMATO.

Los datos pueden ser introducidos o sacados del computador de diferentes maneras, según se use o no los formatos de especificación de entrada y salida. En este párrafo, se presentan las instrucciones de E/S sin formato.

1. La instrucción READ.

La forma general de la instrucción READ sin formato o con formato libre es la siguiente :

READ *, V1,V2,V3, ... ,VN

donde V1 nombre de la variable correspondiente al primer dato de entrada.
V2 nombre de la variable correspondiente al segundo dato de entrada.
. . .
. . .
VN nombre de la variable correspondiente al último dato de entrada.

Los diferentes valores deben separarse por comas o espacios en blanco. Los enteros no deben tener punto decimal. Esos datos se leerán por la unidad normal de entrada (terminal).

Ejemplo :

READ *, A ,B, C

Se leerá por pantalla tres valores a los cuales se le asignará los nombres A, B, C. Esos tres valores deberán darse separados por comas o blancos.

2. La instrucción PRINT.

La instrucción PRINT transfiere datos de la memoria a la unidad normal de salida (impresora). En formato libre, se da como :

PRINT *, list

donde list es una constante, una variable, un arreglo y/o una expresión alfanumérica encerrada entre apóstrofos.

Ejemplo :

PRINT *,' El caudal es ',Q

Se imprimirá el valor de la variable Q almacenado en memoria precedido de la expresión entre apóstrofos. Si Q es igual a 25.5, la impresión será :

El caudal es 25.50

III. LAS INSTRUCCIONES DE ENTRADA/SALIDA CON FORMATO.

1. La instrucción READ con formato.

Su forma general simplificada es :

READ (unidad, fmt, END = num1, ERR = num2) list

donde :
unidad es la unidad lógica de entrada. Puede ser un archivo.
fmt es el número de la instrucción FORMAT que especifica el formato de los datos de entrada. Si es un asterisco, el formato es libre.
num1 es el número de una instrucción a la cuál irá el programa en el caso de encontrar un "fin de archivo " (EOF),en los datos.
num2 es el número de una instrucción a la cuál irá el programa en el caso de encontrar un error en la lectura de datos.
list es una lista de los nombres de las variables de entrada.

Ejemplos :

READ(1,*) AA leerá la variable por pantalla en formato libre.

READ(13,110) XA,XB leerá las variables XA y XB por la unidad 13 (archivo) con el formato indicado en la instrucción 110.

READ(23,290,END=90) X1 leerá la variable X1 por la unidad 23 y con el formato 290. En caso de encontrar una marca de fin de archivo, EOF, irá a la instrucción 90.

READ(8,30,ERR=99) AB leerá la variable AB por la unidad 8 con el formato 30. En caso de encontrar un error en los datos de entrada, irá a la instrucción 99.

2. La instrucción WRITE con formato.

Su forma general simplificada es :

WRITE (unidad, fmt) list

donde unidad, fmt y list tienen la misma significación que para la instrucción READ. Adicionalmente, fmt puede contener la propia instrucción FORMAT entre apóstrofos y paréntesis y list puede representar una expresión aritmética.

Ejemplos :

WRITE(1,*) CAUDAL dará por pantalla la variable CAUDAL con formato libre.

WRITE(6,120) A, CC imprimirá las variables A y CC de acuerdo al formato 120.

WRITE(23,'("caudal",F8.2)') Q transferirá al archivo 23 el valor de la variable Q de acuerdo al formato indicado entre apóstrofos.

3. La instrucción FORMAT.

Una instrucción FORMAT es una instrucción no-ejecutable, en forma de una lista de especificaciones de formato y de edición, la cuál se seguirá al leer o al imprimir datos o variables.

3.1 Especificaciones de formato.

Las principales especificaciones de formato son las siguientes :

a. Especificación I.

La especificación I se usa para números enteros; se da como :

Iw donde w es el número de posiciones del campo de lectura o escritura.

b. Especificación F.

Se usa para números reales y se da como :

Fw.d donde w es el tamaño del campo y d el número de posiciones a la derecha del punto decimal.

c. Especificación E.

Se usa para números reales en formato exponencial y se da como :

Ew.d donde w y d son los mismos anteriores.

d. Especificación A.

Se usa para caracteres alfanuméricos y se da como :

Aw donde w es el tamaño del campo.

e. Ejemplos.

Entrada de datos :

Especificación	Dato	Valor leído
I4	1	1

I2	-1	-1
I2	123	12
F6.4	123456	12.3456
F6.4	123.45	123.45
E8.3	7.1 E 5	$7.1x10^{5}$
E9.4	45 E35	$0.0045x10^{35}$
A 4	DATO	DATO
A10	CAUDAL DIARIO	CAUDAL DIA

Comentario :
- los blancos a la derecha se ignoran.
- el punto decimal leído prevalece sobre la especificación w.

Salida de datos :

Especificación	Valor en memoria	Salida
I4	+452.25	452
I2	623	**
I5	-52	-52
F5.2	+10.657	10.65
F3.1	-254.2	***
F4.1	23	23.0
E10.3	-12.3454	-.123E+02
E10.8	0.00456532	.456532E-02
A10	CAUDAL	CAUDAL
A4	CAUDAL	****

3.2 Especificaciones de edición.

Las diferentes especificaciones, que sean de formato o de edición, vienen separadas por coma cuando se trata de un mismo registro, o por barra cuando son registros consecutivos.

1. Edición de alfanuméricos.

Se hace con comillas o apóstrofos, o con la especificación H. En este último caso, se da como nH donde n es el número de caracteres que siguen a H.

Ejemplos :

' CAUDAL' se imprimirá como CAUDAL
"DATO" se imprimirá como DATO

6HLLUVIA se imprimirá como LLUVIA

2. Posicionamiento horizontal.

Se usa la especificación X, la cuál se da como nX donde n es le número de saltos de posiciones hacia la derecha . Por ejemplo :

10X salta 10 posiciones hacia la derecha.

3. Repeticiones.

Cuando se quiere utilizar la misma especificación para diferentes variables, no es necesario repetirla, sino que se coloca el número de veces que se quiere delante de la especificación :

3F5.2 equivale a F5.2,F5.2,F5.2

5I4 equivale a I4,I4,I4,I4,I4

También se puede repetir un grupo de especificaciones, encerrándolo entre paréntesis :

3(5X,F10.3) equivale a 5X,F10.3,5X,F10.3,5X,F10.3

Cuando las especificaciones se usan para mas de un registro, se utiliza la barra, /, para separarlos. Por ejemplo :

FORMAT(5X,F8.3,5X,3I2/5X,5F8.3)

En este caso se leerá el primer registro o línea con el formato 5X,F8.3,5X,3I2 y el segundo con 5X,5F8.3.

Ejemplos :

ABS(-14.) = 14.0
IABS(-12) = 12
FLOAT(15) = 15.0
IFIX(56.78) = 56
EXP(2.5) = $e^{2,5}$
SEN(3.14159) = sen(180°)
COS(3.14159/2.) = cos(90°)
TAN(3.14159/4.) = tan(45°)
SQRT(49.0) = $\sqrt{49}$
ALOG(20.) = Ln (20.)
ALOG10(2,8) = log(2.8)

CAPITULO VI

LAS INSTRUCCIONES DE CONTROL

En este capítulo, se describirán las instrucciones de control y transferencia mas comúnmente utilizadas en Fortran 77. Esas son :

- La instrucción DO, la cuál puede ser :
  - DO etiquetado
  - Bloque DO
  - DO WHILE
- La instrucción IF, la cuál puede ser :
  - IF aritmético
  - IF lógico
  - Bloque IF
- La instrucción GOTO, la cuál puede ser :
  - GOTO incondicional
  - GOTO calculado
  - GOTO asignado
- La instrucción CALL

- La instrucción CONTINUE

I. LA INSTRUCCION DO

I. 1 El DO etiquetado.

Se da como :

DO [etiqueta], [index]=[inicio],[final],[paso]

donde :

etiqueta es el número de una instrucción ejecutable, la cuál será la última ejecutada en el lazo DO.

No puede ser :
- un GOTO, u otro DO
- cualquier instrucción de un IF.
- un RETURN, un STOP o un END

index es una variable que controla el lazo DO. Puede ser entera o real. La coma antes del index es opcional.

inicio es una variable o expresión que representa el valor inicial de la variable index en el lazo DO.

final es una variable o expresión que representa el valor final de la variable index en el lazo DO.

paso es una variable o expresión que representa el incremento de la variable index en el lazo DO. No puede ser igual a cero. Por defecto, es igual a 1.

Las variables index, inicio, final y paso deben ser del mismo tipo. Si no lo son, entonces las tres últimas son convertidas al tipo de index. La última instrucción del DO puede ser la instrucción CONTINUE para evitar terminar con una instrucción no permitida. Las instrucciones comprendidas entre el DO y la última (la que lleva etiqueta) se ejecutan cada vez que se completa un ciclo del lazo.

Ejemplos :

```
      DO 10, I=1,20
      WRITE (1,*) I
10    CONTINUE
```
Se imprimirán los números del 1 al 20

```
      DO 20 X=1.0,2.0,0.1
      . . . . .
      . . . . .
20    CONTINUE
```
Se ejecutarán 11 veces las instrucciones incluidas entre el DO y la nº 20

```
      DO 100 J=10,2,-1
      . . .
100   CONTINUE
```
Se ejecutarán 9 veces lo incluido entre el DO y la instrucción nº 100.

Cuando se ejecuta un lazo DO, la siguiente secuencia de operaciones se realiza :

1. Se calculan los valores de index, inicio, final y paso.
2. El número de veces que se ejecutará el lazo se calcula como :
   Parte entera de ((final -inicio + paso)/paso)
3. Si el número resultante es inferior o igual a cero, no se ejecuta el lazo y se pasa a la instrucción siguiente.
4. Se ejecuta el lazo DO.
5. Si no se ha ejecutado el lazo completo, se incrementa el valor de index y se regresa al punto 4.

Dentro del DO, modificaciones de index, inicio, final o paso no afectan el número de veces que se efectúa el lazo, ya que este valor se calcula al entrar al lazo DO.

I. 2 El Bloque DO.

El Bloque DO es idéntico al DO etiquetado, con la única diferencia de no usar etiqueta al final del lazo, sino un END DO. Es una extensión del Fortran 77 de la ANSI. Por ejemplo :

```
DO J=1,20        El bloque de instrucciones se ejecutará 20 veces
. . . .          y luego saltará a la instrucción que sigue END DO
. . . .
END DO

DO I=0,10,2      El bloque de instrucciones se ejecutará 6 veces
. . . .          y saltará a la instrucción que sigue al END DO.
. . . .
END DO
```

I. 3 El DO WHILE.

La instrucción DO WHILE constituye también una extensión al Fortran 77, pero es un elemento muy importante de la programación estructurada.

Se da como :

```
DO WHILE ( expresión lógica )
. . . .
. . . .
END DO
```

y se ejecuta de la siguiente manera :

La expresión lógica se evalúa al inicio del lazo DO. Si es cierta, se ejecutan las instrucciones comprendidas entre el DO y el END DO, y se vuelve a chequear la expresión lógica. Si no es cierta, el lazo DO se termina y se ejecuta la instrucción que sigue al END DO.

Ejemplos :

```
DO WHILE (I.NE.99)     Se leerán valores de I y se ejecutarán
READ(1,*) I            las instrucciones siguientes hasta que
. . . .                se lea un valor de I igual a 99.
. . . .
END DO

DO WHILE (J . LT . 10)
```

```
. . .
    DO WHILE (I . GE . J)
    . . .
    . . .
    END DO
. . .
END DO
```

En el segundo ejemplo, se tiene dos lazos DO anidados. El primero se ejecuta mientras J sigue menor que 10, y el segundo mientras I es superior o igual a J. Cuando se tienen varios lazos DO anidados, cada uno debe terminar por un END DO diferente.

Nótese la diferencia entre el DO WHILE y el Bloque DO o el DO etiquetado. En el primer caso, la condición se chequea al inicio del lazo, mientras que en los otros, se chequea al final del mismo.

II. LA INSTRUCCION IF.

II. El IF aritmético.

La instrucción IF aritmética transfiere el control a una de tres instrucciones alternativas. Su forma es :

IF (expre) etiq1,etiq2,etiq3

donde :
expre es una expresión aritmética cuyo signo se chequea.
etiq1, etiq2, etiq3 son etiquetas de instrucciones ejecutables a las cuales va el programa según el valor de expre (negativo, nulo o positivo).

Ejemplos :

IF (A+B) 10,20,30

Si A+B es negativo, se va a la instrucción 10
Si A+B es igual a cero, se va a la instrucción 20
Si A+B es positivo, se va a la instrucción 30

IF (X) 10,10,25

Si X es inferior o igual a cero, se va a la instrucción 10
Si X es mayor que cero, se va a la instrucción 25

El uso del IF aritmético no se recomienda en programación estructurada, ya que no respeta siempre la condición de una sola entrada, una sola salida. Se recomienda el uso del IF lógico o del Bloque IF.

II. 2 El IF lógico.

La instrucción IF lógica evalúa una expresión lógica y ejecuta una instrucción si el resultado de la evaluación es cierto.

Se da como :

IF (expre) inst

donde **expre** es la expresión lógica a evaluar
inst es la instrucción a ejecutar si la expresión es cierta.
No puede ser :

- un DO
- un END
- un IF

Ejemplos :

IF (X. GT . Y) X=Y	Si X es mayor que Y, se hace X igual a Y
IF (A . EQ . B)GOTO 99	Si A igual B, se va a la instrucción 99

El IF lógico actúa como una selección de dos vías posibles : si la condición es verdadera, se ejecuta la instrucción contenida en la instrucción IF, sino esa no se ejecuta y se pasa a la instrucción siguiente.

Los operadores aritméticos permitidos dentro de una expresión lógica, son

.EQ.	IGUAL A
.**NE.**	DIFERENTE DE
.**LT.**	INFERIOR A
.**LE.**	INFERIOR O IGUAL A
.**GT.**	SUPERIOR A
.**GE.**	SUPERIOR O IGUAL A

También se pueden combinar varias expresiones lógicas dentro de un mismo IF en base a los operadores lógicos siguientes :

. AND.	Ambas expresiones deben ser ciertas.
. OR .	Una u otra expresión debe ser cierta.
. NOT .	La proposición inversa de la segunda deben ser cierta.

Ejemplos :

IF (exp1.OR.exp2)	cierto si exp1 o exp2 ciertos
IF(exp1.AND. .NOT. exp2)	cierto si exp1 cierto y exp2 falso
IF(exp1. AND. exp2)	cierto si exp1 y exp2 ciertos

II. 3 El Bloque IF.

El bloque IF es una extensión del IF lógico que permite ejecutar un o dos bloques de instrucciones dependiendo del valor cierto o falso de la expresión lógica. También es muy importante en programación estructurada. Se da como :

```
IF( expre1 ) THEN
. . . .
ELSE [ IF ( expre2 ) THEN]
. . . .
[ELSE]
. . . .
ENDIF
```

donde expre1 y expre2 son expresiones lógicas. Un bloque IF puede contener cualquier número de sub-bloques ELSE IF, pero solamente un sub-bloque ELSE. Un bloque IF puede contener cualquier número de bloques IF anidados, pero cada uno debe terminar por un ENDIF.

Ejemplos :

```
IF ( X.EQ.Y ) THEN          Si X es igual a Y , se hace X igual a X+1
X=X+1
ENDIF
```

```
IF( A.LT.B.OR.A.EQ.C) THEN          Si A es menor que B   o si A es igual
```

```
. . . .                                  a C, se ejecuta el bloque siguiente.
ELSE IF(A.EQ.B) THEN                     De lo contrario, si A es igual a B,se
. . . .                                  ejecuta el segundo bloque.
ELSE                                     Finalmente, si A no es igual a B, se
. . . .                                  ejecuta el tercer bloque.
ENDIF
```

```
IF(NI.EQ.O) THEN                         En este ejemplo, se muestra el uso de
NI=NJ                                    varios IF anidados.
J=J+1
        IF(J.LT.K) THEN
        K=K-1
        ELSE IF(J.EQ.K) THEN
        K=K+1
        ENDIF
ELSE
NI=I
K=NI
ENDIF
```

III. LA INSTRUCCION GOTO.

III. 1 El GOTO incondicional.

La instrucción GOTO transfiere el control a una instrucción ejecutable etiquetada, dentro del mismo programa. Se da como :

GOTO etiq o GO TO etiq

donde **etiq** es la etiqueta de la instrucción a la cuál debe ir el programa. Esta instrucción puede estar ubicada antes o después de la instrucción GOTO, pero debe estar en el mismo programa.

Ejemplo :

```
GOTO 20          El control se transfiere a la instrucción que lle-
                 va la etiqueta 20.
```

III.2 El GOTO calculado.

La instrucción GOTO calculado transfiere el control a una de varias instrucciones dependiendo del resultado de la evaluación de una expresión. Se da como :

GOTO (etiq1,etiq2,etiq3,. . . ,etiqn), expre

donde

etiq la etiqueta de una instrucción ejecutable.

expre es una expresión entera o real que se evalúa.

El uso de una expresión real es una extensión del Fortran 77. El resultado de la expresión se convierte a entero, tomando la parte entera del mismo. Este valor indica la posición de la etiqueta en la lista del GOTO.

Ejemplos :

GOTO(20,30,40,50) A Si A=1, va a la instrucción 20
Si A=2, va a la instrucción 30
Si A=3, va a la instrucción 40
Si A=4, va a la instrucción 50.

B=1.5
X=2.
GOTO(10,11,12,13,14,15) B+X

En este caso, la parte entera de B+X es igual a 3 y el control pasa a la instrucción 12.

III.3. El GOTO asignado.

El GOTO asignado transfiere el control a una instrucción cuya etiqueta ha sido asignada a una variable en la instrucción GOTO por una instrucción ASSIGN.

Se da como :

GOTO ivar,(etiq1,etiq2,etiq3,. . .)

donde

ivar es una variable entera.
etiq es la etiqueta de una instrucción ejecutable (opcional)

A la variable ivar, se le debe asignar antes un valor por una instrucción ASSIGN. Cuando se ejecuta el GOTO, el control se transfiere a la instrucción que lleva la etiqueta correspondiente.

Ejemplos :

ASSIGN 10 TO SPOT El control pasa a la instrucción 10.
GOTO SPOT

ASSIGN 100 TO TIME
GOTO TIME(90,100,150) El control pasa a la instrucción 10

IV. LA INSTRUCCION CALL.

La instrucción CALL transfiere el control a un subprograma. Se da como :

CALL nombre (arg1,arg2,arg3,. . . .)

donde

nombre es el nombre de un subprograma.
arg son los argumentos que se pasan al subprograma.

Ejemplos :

CALL LECIM Transfiere el control a la subrutina LECIM.

CALL VALOR(X1,X2,X3) Transfiere el control a la subrutina VALOR así como los valores X1,X2,X3.

CAPITULO VII

LAS VARIABLES CON SUBINDICES
Y
LAS INSTRUCCIONES DE ESPECIFICACION

I. LAS VARIABLES CON SUBINDICES.

Cuando se quiere usar en un mismo programa un conjunto de variables similares, en vez de usar nombres diferentes para cada una, se pueden usar variables con subindices. Por ejemplo, si P es la precipitación de un día, las precipitaciones del mes de Enero pueden llamarse P1,P2,P3. . . . hasta P31 o en forma mas sencilla agruparlas en una sola variable que será P_i, i variando de 1 hasta 31. En este caso, la variable P representa un vector que almacena los valores de precipitación díaria. En Fortran, se puede usar también esas variables con subindices en la forma siguiente :

P(I), I= 1,31

y la precipitación del primer día será P(1), la del segundo P(2), . . .lo cuál facilita la estructuración del programa.

En el ejemplo anterior, P(I) es un vector o un arreglo unidimensional que contiene los valores de precipitación del mes de Enero. Si se quiere usar los valores de precipitación de todo el año, se podría usar un vector de 365 valores, o mejor un arreglo de dos dimensiones en el cuál el primer índice representaría el número del mes y el segundo el día dentro del mes, P(I,J), con I=1,12 y J=1,31.

En este último ejemplo, se tiene que :

P(2,24) es la precipitación del día 24 de febrero.
P(10,5) es la precipitación del día 5 de octubre.

Un arreglo de dos dimensiones como el anterior puede también considerarse como una matriz en la cuál el primer índice es el número de la fila y el segundo el número de la columna. Los arreglos pueden tener mas de dos dimensiones. En Fortran 77, se permiten arreglos de hasta 7 dimensiones. Esos arreglos deben ser declarados como tal en una instrucción de especificación (INTEGER, REAL, DIMENSION, COMMON, ...) en la cuál se especifica el número de dimensiones y los limites de cada dimensión.

Eso viene en la forma siguiente :

nombre(d1,d2,d3, . . .)

donde

nombre es el nombre del arreglo, el cuál debe seguir las mismas reglas de los nombres de las variables.

d_i es la especificación de dimensión del arreglo. Debe haber una especificación para cada dimensión.

Esa especificación se da como :

n o **n:m**

donde :

n es el limite superior
m es el limite inferior

Si se especifica solamente el limite superior, el limite inferior se toma como 1. El valor de ambos limites puede ser positivo, negativo o nulo. Sin embargo, el limite superior debe ser mayor (o igual) que el limite inferior. Los limites pueden ser expresiones aritméticas, en las cuales las constantes o variables deben ser enteras. Esos limites indican el número máximo de elementos en cada dimensión del arreglo. El número de elementos en cada dimensión puede ser calculado como n-m+1.

Ejemplos :

A(0:10) Arreglo unidimensional (vector) de 11 elementos.

TABLA(5,12) Arreglo bidimensional de 5 filas y 12 columnas.
XX(2,2,2,3) Arreglo de cuatro dimensiones y 24 elementos.

CAB(-5:3,2:12,-1,5) Arreglo tridimensional de 693 elementos.

Los subindices designan un elemento específico del arreglo. Para referenciar un elemento de un arreglo, se debe indicar el nombre del mismo, así como los subindices del elemento en cada dimensión, separados por coma. Cada subindice debe entrar dentro de los limites definidos para cada dimensión. El error cometido al usar un subindice fuera de los limites definidos no se detecta, ni a la compilación, ni

durante la ejecución, pero los resultados son imprevisibles. Los subindices pueden ser expresiones aritméticas; después de evaluar la expresión, se usa su parte entera.

Ejemplos :

ARR(I+4,J-1)

TAB((3*X+1)/4) Si X=3.6, la expresión vale 2.95, lo cuál da un subindice igual a 2.

Los arreglos unidimensionales se almacenan como lista de valores. Los arreglos de mayores dimensiones se almacenan haciendo variar primero el primer índice, luego el segundo, y asi sucesivamente hasta el último índice, el cuál varia mas lentamente.

Ejemplos :

A(10) se almacena como :

A(1),A(2),A(3),A(4),. . . ,A(9),A(10)

B(3,5) se almacena como :

B(1,1),B(2,1),B(3,1),B(1,2),B(2,2),B(3,2),B(1,3),B(2,3),. . .
B(1,5),B(2,5),B(3,5)

La lectura y escritura de las variables con subindices se hace en una forma un poco diferente. Se puede usar un DO implícito para entrada y salida de variables contenidas en arreglos. Se da como :

READ(unidad, fmt) (lista, index = inicio, final, paso)

o

WRITE(unidad, fmt) (lista, index = inicio, final, paso)

donde unidad, fmt, index, inicio, final, paso tienen la misma significación definida anteriormente. Por ejemplo :

READ(1,100) (A(I),I=1,20)

En este caso, las variables se leerán en el orden A(1),A(2),. . ., A(19),A(20)

WRITE(6,120) ((B(I,J),I=1,2),J=1,2)

El orden de impresión será :

B(1,1),B(2,1),B(1,2),B(2,2)

Es decir que el lazo interno se ejecutará cada vez que se incrementa el lazo externo. Esos DO implícitos se permiten para arreglos de cualquier número de dimensiones.

II. LAS INSTRUCCIONES DE ESPECIFICACION.

Las instrucciones de especificaciones mas usadas son las siguientes :

INTEGER, REAL, CHARACTER
DIMENSION, IMPLICIT, COMMON
PARAMETER
EQUIVALENCE

II. 1. Las especificaciones de tipo.

Las instrucciones de especificación de tipo definen el tipo de las variables. Las especificaciones **INTEGER** y **REAL** se usan cuando el nombre de la variable va en contra de la declaración implícita por el primer carácter del nombre.

Ejemplos :

INTEGER RUN, TIME

REAL IMAX, LARGO, N

En esos dos casos, los nombres de las variables contradicen el tipo de las mismas, por lo que se hace necesario esa especificación de tipo. Normalmente, las variables se declaran sin especificar el número de palabras que usan en memoria. Sin embargo, en algunos casos, se puede cambiar ese tamaño. Por eso, se usa :

INTEGER*2 o INTEGER *4

REAL*4 o REAL*8

Respectivamente, se usarán enteros de dos o cuatro palabras, y reales de cuatro u ocho palabras.

La especificación **CHARACTER** se usa para declarar que una variable es alfanumérica.

En el caso de las variables alfanuméricas, es necesario indicar la longitud de la variable. La instrucción se da como :

CHARACTER*long

donde **long** especifica la longitud de la variable. Por defecto, se toma igual a 1.

Ejemplo :

NOMBRE*20 La variable NOMBRE tiene 20 caracteres.

DIREC*40 La variable DIREC tiene 40 caracteres.

II. 2. La instrucción DIMENSION.

La instrucción DIMENSION se usa para especificar las dimensiones de las variables con subindices. Se da como :

DIMENSION nom1(limit1), nom2(limit2), . . .

donde

nom son los nombres de los arreglos.

limit son las declaraciones de dimensionamiento de esos arreglos, tales como se definieron anteriormente.

Ejemplo :

DIMENSION A(10,20),TAB(2,5,10)

Las declaraciones de dimensionamiento de los arreglos deben darse una sola vez, sea en la instrucción DIMENSION o en una declaración de tipo, pero nunca en ambas. Por ejemplo :

INTEGER A(100) declara el arreglo A como entero y teniendo 100 elementos.

y es equivalente a

INTEGER A
DIMENSION A(100)

INTEGER A(100)
DIMENSION A(100) está prohibido ya que se dimensiona dos veces el arreglo A.

II.3 La instrucción COMMON.

La instrucción COMMON especifica un espacio común de almacenamiento en memoria, el cuál puede ser usado por diferentes programas y subprogramas. Se da como :

COMMON /nom1/ list1,/nom2/ list2, . . .

donde :

nom es el nombre de un COMMON etiquetado. Ese nombre es opcional

list es la lista de las variables.

Ejemplos :

COMMON A,B,C
Las variables A,B,C se almacenan en un COMMON sin etiqueta.
COMMON /TSF/ X,X1,X2
Las variables X,X1,X2 se almacenan en el COMMON etiquetado TSF.
COMMON /A/A1,A2,/X/ X1,X2
Las variables A1 y A2 se almacenan en el COMMON A y , X1 y X2 en el COMMON X.

En una instrucción COMMON, no deben aparecer los nombres de variables que están en la lista de argumentos de una subrutina, los nombres de subrutinas o de funciones, ni variables que están en una instrucción DATA. Puede haber varias instrucciones COMMON en un solo programa. Lo importante en una instrucción COMMON no es el nombre de la variable, sino su ubicación dentro del COMMON.

Por ejemplo, si en un programa aparece lo siguiente :

COMMON I(4),J(6)

y en un subprograma :

COMMON J(4),I(6)

los valores del arreglo I del programa principal serán los valores del arreglo J del subprograma. Igualmente, si en el programa principal aparece :

COMMON I(4),J(6), SAM

Y en la subrutina :

COMMON GEO,M(10)

la equivalencia será :

Programa principal	Subprograma
I(1)	GEO
I(2)	M(1)
I(3)	M(2)
...	...
J(6)	M(9)
SAM	M(10)

II. 4. La instrucción IMPLICIT.

La instrucción IMPLICIT especifica el tipo de las constantes o variables, asociado con el primer carácter de su nombre. Se da como :

IMPLICIT tipo,(rango1,rango2, . .),tipo(rango1, . . .)

donde :

tipo es una especificación de tipo (INTEGER,REAL,...)

rango es una sola letra o un rango de letras asociadas con el tipo especificado.

Ejemplo :

IMPLICIT REAL(I,J,K),INTEGER(A-C)

Esa instrucción significa que las variables cuyo nombre empieza por I, J o K serán reales, y que las variables cuyo nombre empieza por una letra comprendida entre A y C serán enteras.

II. 5. La instrucción PARAMETER.

La instrucción PARAMETER se usa para definir constantes con nombres. Después de la instrucción PARAMETER, el uso del nombre es equivalente al uso de la constante. Se da como :

PARAMETER (nom1=exp1,nom2=exp2,. . .)

donde :

nom es el nombre de la constante.

exp es una expresión cuyo valor se atribuye a la constante.

Ejemplos :

```
PARAMETER (MIN=10,MAX=50)
DIMENSION A(MIN,MAX)

CHARACTER*6 ARCHI
PARAMETER (ARCHI=?RIO)
OPEN(10,FILE=ARCHI,STATUS='OLD')
PARAMETER (N=4)
CHARACTER*N TOTO
PARAMETER (INF=0,ISUP=20)
DIMENSION XER(INF:ISUP)
DO I=INF,ISUP
. . .
ENDDO
```

```
PARAMETER (PI=3.14159)
. . .
SURF=PI*R**2
```

El uso de la instrucción PARAMETER define la constante únicamente en el programa en el cuál se usa. Una constante puede ser definida una sola vez en una instrucción PARAMETER. Si su tipo no es implícito, debe ser declarado previamente en una instrucción de especificación.

II. 6. La instrucción EQUIVALENCE.

La instrucción EQUIVALENCE asocia variables para que comparten el mismo espacio en memoria. Se da como :

EQUIVALENCE (list1),(list2), . . .

donde :

list	es una lista de dos o mas variables o arreglos que se asocian en el EQUIVALENCE.

Ejemplos :

EQUIVALENCE (A,B),(C(2),DE,E)

Las variables A y B comparten el mismo espacio en memoria, y las variables C(2), D y E comparten otro espacio común.

Cuando se usan arreglos la equivalencia se hace elemento por elemento. Por ejemplo :

```
DIMENSION A(3),B(5)
EQUIVALENCE (A(2),B(4))
```

indica que los elementos A(2) y B(4) comparten el mismo espacio en memoria. Eso implica también que los otros elementos tienen equivalencia uno a uno :

```
                A(1)  A(2)  A(3)
B(1)  B(2)      B(3)  B(4)  B(5)
```

Por la tanto, la equivalencia de arreglos se hace en base a la forma en la cuál se almacenan los elementos del arreglo.

Los tipos de los elementos en la instrucción EQUIVALENCE pueden ser diferentes. Esa instrucción no implica conversión de tipo de variables. Cada variable conserva su tipo.

III. LA INSTRUCCION CONTINUE.

La instrucción CONTINUE crea un punto de referencia dentro de un programa. Se da como :

CONTINUE

Ejemplo :

```
      DO 20 I=1,10
10    X=X+I
      IF(X.LT.25) GOTO 20
      GOTO 10
      CONTINUE
```

En este caso, la última instrucción del DO es un GOTO, lo cuál no está permitido. Entonces, se usa la instrucción CONTINUE.

La instrucción CONTINUE debe tener siempre una etiqueta para servir de punto de referencia en un programa. Particularmente, se usaba en los lazos DO como última instrucción. Ahora, con el uso de bloque DO y del END DO, su uso ya no se justifica tanto.