FORTRAN

FERNANDO ALONSO AMO
CARLOS GARCIA CODINA
J. J. SANCHIS LLORCA

MINISTERIO DE EDUCACION Y CIENCIA

FORTRAN

FERNANDO ALONSO AMO
CARLOS GARCIA CODINA
J. J. SANCHIS LLORCA

MINISTERIO DE EDUCACION Y CIENCIA

Edita: Servicio de Publicaciones del Ministerio de Educación y Ciencia
Imprime: R. García Blanco. Avda. Pedro Díez, 3. Madrid-19
ISBN: 84-369-0844-9
Depósito Legal: M. 39.338-1981
Impreso en España

PROLOGO

La presente publicación surge como texto complementario de las clases que, sobre el lenguaje FORTRAN, se vienen impartiendo en el 2º curso de la Facultad de Informática de Madrid.

Se ha intentado, en todo momento, ser preciso y riguroso tanto en la definición como en la descripción de los formatos de cada una de las sentencias que componen el lenguaje. Entendemos que no es un texto idóneo, como introducción al FORTRAN, para una persona ajena al tema informático, aunque esperamos sea válido y útil para el alumnado de informática que se inicia en este lenguaje. Este es el fin que tuvieron "in mente" sus autores.

Es muy importante que un texto, sobre un lenguaje de programación, esté basado en un ordenador en concreto, por ello se ha tomado como referencia el FORTRAN IV adaptado a las peculiaridades que presenta el ordenador Univac 9400. El haber elegido este equipo es circunstancial, se debe únicamente a ser este ordenador el del Centro de Cálculo de la Facultad sobre el que, básicamente, van dirigidas las prácticas de clase.

Agradecemos la valiosa colaboración del Subdirector General de Organización y Automación, Pedro Maestre Yenes, que ha impulsado esta publicación, y la de Juan Pazos Sierra que ha ayudado en la corrección del texto.

No podemos cerrar este Prólogo sin mencionar la labor de Fernando Bernal en la delineación de los diagramas, y, muy especialmente, la de nuestra secretaria Asunción Guerra por el cariño y las horas de dedicación entregadas a este texto.

Madrid, (Febrero, 1981)

Los autores

INDICE

CAPITULO 1

INTRODUCCION

El fin de este texto es el de proporcionar al estudiante un conocimiento suficiente del lenguaje FORTRAN de modo que le permita utilizar el ordenador como una herramienta de cálculo científico.

La estructura del FORTRAN (FORmula TRANslator) fue diseñada en 1954 por John Backus, y las especificaciones completas del lenguaje aparecen en julio 1955. El nombre inicial que recibe es el de FORTRAN FORMAT. En 1958 al incluirle diferentes implementaciones se transforma en FORTRAN II y luego en 1962 en FORTRAN IV. La definición actual de esta versión se presenta en el documento X3.9-1966 del American National Standards Institute. Posteriormente se habla del FORTRAN V, y diferentes casas constructoras lo incluyen con este nombre, pero la realidad es que nunca se llegaron a formalizar las especificaciones de esta versión.

Aunque el FORTRAN es un lenguaje " primitivo", y no se acopla a la metodología actual de la programación estructurada, es un hecho que es el lenguaje más compatible y más ampliamente utilizado.

En este texto se explican las sentencias básicas del FORTRAN IV que acepta el compilador de la serie UNIVAC 9400/9480 funcionando bajo el sistema operativo OS/4. La referencia utilizada es la del manual UP-7693.

1.1. CONFIGURACION MINIMA PARA SOPORTAR EL FORTRAN UNIVAC 9400 EN DISCO (DFOR)

Todo programa informático exige unas necesidades de configuración para poder funcionar, las correspondientes al com-

pilador FORTRAN son:

. Memoria principal.- 131 Kbytes, siendo ocupadas por el compilador en su ejecución -- 32 Kbytes.

. Lector de tarjeta perforada de 80 columnas

. Impresora

. Memoria masiva.- Dos unidades de discos magnéticos, con espacio suficiente para mantener el compilador FORTRAN.

Una configuración usual (ej. la que dispone la Facultad de - Informática de Madrid) es la siguiente:

. Memoria principal.- 131 Kbytes

. Lector de tarjetas.- Univac 1004 de 400 tarjetas/minuto.

. Impresora.- Univac 1004 de 600 líneas/minuto.

. Memoria masiva.- Dos unidades (8414) con una capacidad de 29.176 Megabytes.

1.2. ORDEN REQUERIDO DE LAS SENTENCIAS FUENTE EN UN PROGRAMA FORTRAN

Todo lenguaje está formado por un conjunto de sentencias, bien sean declarativas de objetos, bien sean ejecutivas de las acciones que se deben realizar con esos objetos, y que son las que - conforman la estructura del lenguaje y definen su potencia. Estas sentencias fuente, normalmente, precisan un orden de prelación en su tratamiento. En la Tabla 1.1. se indica el orden - que exige el compilador FORTRAN. En la columna "Orden" figuran agrupadas las sentencias en 5 apartados. Para diferentes valores de "Orden" se deben colocar antes las sentencias de "valor" inferior. Las correspondientes a un mismo apartado se - pueden escribir en cualquier secuencia.

En la Tabla 1.1. no figuran las sentencias FORMAT, NAMELIST, DATA y DEFINE FILE por no estar encuadradas en ningún apartado, ya que pueden aparecer en cualquier parte del módulo unidad, excepto en el caso de que el módulo unidad sea una función ó subrutina que deberán aparecer detrás de las sentencias FUNCTION ó SUBROUTINE respectivamente.

Se define un "programa FORTRAN" como el formado por un conjunto de sentencias fuente y comentarios FORTRAN agrupados en módulos unidad. Un módulo unidad puede ser de 4 tipos diferentes:

a) Módulo principal o programa principal.- No precisa ninguna sentencia diferencial para su definición.
b) Subprograma FUNCTION.- Su primera sentencia debe ser - FUNCTION.
c) Subprograma SUBROUTINE.- Su primera sentencia debe ser SUBROUTINE.
d) Subprograma BLOCK DATA.- Su primera sentencia debe -- ser BLOCK DATA.

Todo módulo debe finalizar con la sentencia END.

En un programa FORTRAN ejecutable, es obligatoria la presencia del módulo a), siendo opcional la de los otros tres módulos.

TABLA 1.1

INSTRUCCION FUENTE	ORDEN
FUNCTION, SUBROUTINE, BLOCK DATA	1º
COMMON COMPLEX DIMENSION DOUBLE PRECISION EQUIVALENCE EXTERNAL IMPLICIT INTEGER LOGICAL REAL	2º
FUNCION "FORMULA"	3º
Asignación aritmética IF aritmético ASSIGN GO TO asignado BACKSPACE CALL GO TO calculado Asignación lógica IF lógico CONTINUE DO ENTRY	4º

TABLA 1.1 (Continuación)

INSTRUCCION FUENTE	ORDEN
END FILE PAUSE PRINT READ PUNCH RETURN REWIND STOP GO TO incondicional WRITE FIND	4º
END	5º

1.3. CONJUNTO DE CARACTERES DE UN PROGRAMA FORTRAN

Cada compilador FORTRAN presenta, como alfabeto del lenguaje, un conjunto de caracteres que no siempre son válidos para otros compiladores, especialmente cuando los compiladores corresponden a casas constructoras diferentes.

El Univac 9400 dispone de un conjunto de 64 caracteres, que - los vamos a dividir entre los que son utilizados por el compilador FORTRAN y el resto.

a) Conjunto de caracteres FORTRAN

Totalizan 48, desglosados en:

- Letras: A, B, C, D, . . . , Z, $ (Son 27)
- Digitos: 0, 1, 2, 3, . . . , 9 (Son 10)
- Símbolos FORTRAN especiales =, ( ) + - * /. & (Son 10)
- Blanco: b (a nivel de impresión o grabación es un espacio en blanco).

Al grupo formado por las letras y los dígitos se les denomina usualmente como "conjunto de caracteres alfanuméricos".

b) Conjunto de caracteres especiales 9400

Lo componen 16 símbolos y son:

¢ > < | % ! ; : @

? _ (Subrayado) ' (apóstrofo) "(comillas)

\ ¬

- El código de representación de cada carácter es el - EBCDIC (Extended Binary Coded Decimal Interchange Code) y tiene una ocupación de 8 bits por caracter

Ejemplos.-

Caracter	Código EBCDIC (en hexadecimal)	Representación binaria
A	C1	11000001
B	C2	11000010
b	40	01000000

1.4. FORMATO DE LAS SENTENCIAS FUENTE

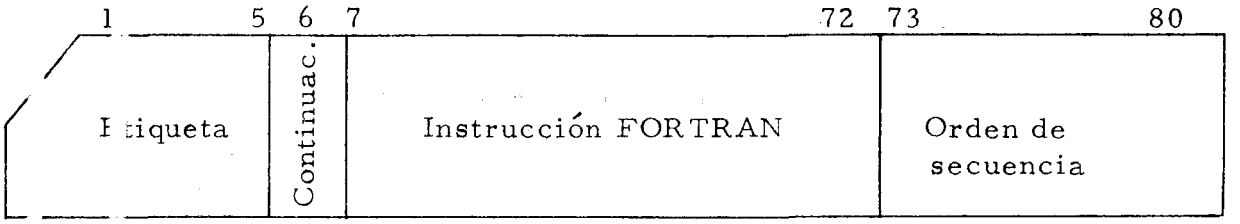
Al programa escrito en el lenguaje FORTRAN se le denomina "programa fuente" (o programa simbólico). Una vez tratado por el ordenador se desglosa en un conjunto de instrucciones máquina, que ya pueden ser ejecutadas. A este programa "máquina" se le llama "programa objeto", y a la operación de transformar la primera forma en la segunda se la denomina COMPILACION-ENSAMBLAJE. La COMPILACION es la traducción de las sentencias fuente a instrucciones máquina, formando un programa, todavía no ejecutable, denominado "programa reubicable". Esta traducción la realiza el compilador FORTRAN.

Con objeto de poder efectuar una traducción correcta, es preciso utilizar ciertas reglas en la escritura del "programa -- fuente". Estas reglas convencionales son de dos tipos: Las que tratan la sintaxis del lenguaje, y las que regulan la ortografía.

Por todo ello las sentencias fuente se deben ajustar a un formato dado, que permita, al compilador, conocer en cada instrucción la acción que se pretende realizar.

Tomando como base una tarjeta perforada de 80 columnas, a continuación se describen los campos que componen una sentencia fuente:

FORMATO:



Las características de cada campo son las siguientes:

ETIQUETA (Columnas 1 a 5).- Sirve para referenciar una instrucción FORTRAN y con ello permitir la bifurcación al número señalado en este campo de etiqueta.

Consta de 1 a 5 dígitos, por lo que puede admitir un número comprendido entre 1 y 99999.

Ceros de cabecera, blancos en medio de la etiqueta o al final, son ignorados por el compilador.

E.- 00125 01b25 125bb = Todas equivalen a la etiqueta bb125

Cada etiqueta debe ser única en el programa.

CONTINUACION (Columna 6).- Sirve para indicar que la instrucción de la tarjeta perforada (ó línea, a nivel de hoja de codificación) precedente continúa sobre esta tarjeta.

Cualquier caracter diferente al blanco o al cero en la columna 6 indica que esta tarjeta es continuación de la anterior.

Pueden existir hasta 19 líneas de continuación.

Una línea de continuación puede seguir a su línea inicial, a otra de continuación, o a una línea de comentarios.

INSTRUCCION FORTRAN (Columnas 7 a 72).- En este campo se escriben las instrucciones simbólicas del programa, que se detallan en capítulos posteriores.

La instrucción puede empezar en la columna 7 o en cualquier otra columna posterior de este campo.

. Cuando no quepa una instrucción entre las columnas 7 y 72 se rellenará la siguiente línea a partir de la columna 7 consignando para ello el campo de continuación como se ha especificado anteriormente.

ORDEN DE SECUENCIA (Columnas 73 a 80).- Permite al usuario colocar pequeños comentarios, enumerar las tarjetas del programa, etc. Aunque este campo es ignorado por el compilador se imprime en el listado del programa fuente.

NORMAS GENERALES

. Una línea sólo puede contener como máximo una sentencia fuente.
. Todos los caracteres de la línea están restringidos al "Conjunto de caracteres FORTRAN", excepto en el caso de comentarios y literales.
. Pueden existir líneas en blanco. El compilador las ignora, pero aparecen en el listado del programa fuente.
. A fin de eliminar las dudas que se pueden presentar para -- distinguir el cero de la letra o, al codificar un programa se - suelen barrar los ceros (ϕ).

Ejemplos

1 5	6	7 72	73 80
12		A = A+SIN(X)	1ϕϕ
		B = B+A	11ϕ
		IF(B.GT.C)WRITE (6,5) A,B	12ϕ
	1	,C	13ϕ
5		FORMAT (1X, 3(I3,2X))	14ϕ

1.5. COMENTARIOS

Su fin es el de facilitar la lectura del programa, permitiendo también con ello que otra persona distinta del propio programador pueda seguir el mecanismo de su funcionamiento.

El FORTRAN del 9400 admite dos tipos de comentarios:

a) Línea de comentarios

Se indica colocando la letra C en la columna 1. Admite cualquier conjunto de caracteres: tanto los del FORTRAN, como los especiales del 9400. Se pueden utilizar las 80 columnas de la línea de codificación para el comentario. Las líneas de comentario son ignoradas por el compilador, aunque se imprimen en el listado del programa fuente.

b) Comentario compacto

Si aparece un punto y coma (;) en el campo "Instrucción FORTRAN" (columnas 7 a 72) significa para el compilador que la información que sigue en esa línea se debe tratar como un comentario, excepto en el caso en que el punto y coma (;) figure como carácter de un literal.
El comentario compacto no forma parte de los estándares del FORTRAN IV, es una ampliación que presenta el FORTRAN 9400.

Ejemplos

```
1        5  6  7
C     PRO G  RAMA PARA EL CALCULO
C            DEL FACTORIAL ---      } → línea comentario
             A = A+B ; SUMO VARIABLES → Comentario compacto
          10 FORMAT(1X,'CARACTER;SERA SEPARADOR') → Literal
```

1.6. IDENTIFICADORES

Son nombres simbólicos que sirven para referenciar a las variables, funciones y subrutinas de un programa FORTRAN

Las características que definen un identificador son:

. Pueden tener una longitud máxima de 6 caracteres alfanuméricos.

. El primer carácter debe ser una letra.

. Se aconseja no utilizar el $ como el tercer carácter del nombre de una subrutina o función, pues lo utiliza el sistema -- como formato estandar de sus propias rutinas.

Ejemplos

Identificadores válidos	no válidos
ANCHO	ANCHURA
A1Ø	1A1Ø
$ABC	AbBC

. SUBROUTINE CA$MPO (A, B, C).- No es aconsejable una definición de este tipo, pues podría existir un identificador idéntico definido por el sistema.

El lenguaje FORTRAN no tiene identificadores reservados ("palabras reservadas"), ello quiere decir que el - usuario puede utilizar como identificadores para sus variables, funciones, etc. los nombres simbólicos que definen las sentencias FORTRAN.

Ejemplo

```
IF = 432
RETURN = RETURN + 1
```

CAPITULO 2

TIPO DE DATOS

2.1. CONSTANTES

Una constante es un valor o dato que está disponible durante la ejecución de un programa pero no puede ser redefinido ó modificado.

Las constantes pueden ser de siete tipos diferentes

- enteras
- reales
- de doble precisión
- hexadecimales
- complejas
- lógicas
- literales

2.1.1. CONSTANTE ENTERA

Denominada también de "punto fijo". Son números enteros positivos, negativos o nulos cuyo rango de variación depende del tipo de ordenador.

En el Univac 9400 presentan la siguiente estructura:

Formato: SXXXXX...

Siendo.-

S: Es el signo y puede ser
- + ó b: constante positiva
- \- : constante negativa

XXXX: Conjunto de dígitos (Ø al 9), con una longitud de 1 9 dígitos.

Una constante entera se almacena en el ordenador, en decimal empaquetado, en 5 bytes con el siguiente formato:

n n	n n	n n	n n	n S

Siendo.-

n: Dígito del Ø al 9

S : El signo, y toma el valor:

- C(16 : si la constante es positiva
- D(16: si la constante es negativa

El máximo valor absoluto de una constante entera es: 999999999

Ejemplos

12435 ⟶	0 0	0 0	1 2	4 3	5 C
- 3568 ⟶	0 0	0 0	0 3	5 6	8 D
+ 456795789 ⟶	4 5	6 7	9 5	7 8	9 C
- 3456795789 ⟶	Constante fuera de rango				

2.1.2. CONSTANTE REAL

Denominada de "punto flotante", se compone de una mantisa (que siempre existe) y, eventualmente, de un exponente (potencia de 10).

Las constantes reales pueden ser de tres tipos:

a) Constante real básica

No presenta en su escritura el campo de exponente. La estructura en el Univac 9400 es:

Formato: SXX. XXX...

Siendo.-

S: Es el signo y puede ser
- + ó b : constante positiva
- - : constante negativa

XXX: Conjunto de dígitos (Ø al 9), admitiendo un máximo de siete dígitos.

punto (.): Dentro del conjunto de dígitos XXX... se incluye como un elemento más el punto (equivalente a nuestra coma decimal), y sirve para separar la parte entera de la parte decimal.

El punto puede estar situado:

- Precediendo a la constante (ej.- .4325)
- En medio de la constante (ej.- 4.325)
- Al final de la constante (ej.- 4325.)

b) Constante real básica seguida por un exponente decimal

Es el tipo de constante descrita anteriormente que incluye la parte de exponente.

Formato: SXX.XXX... ESNNN

Siendo.-

S: Es el signo de la constante y del exponente, y puede ser: + ó b: constante o exponente positivo; - : constante o exponente negativo

XXX y punto (.): Lo señalado para el apartado a)

E: Letra obligatoria que precede al exponente

NNN: Es una constante entera, con un máximo de tres dígitos, que indica el exponente. Es decir, señala la potencia de 10 por la que se debe multiplicar la constante real básica para obtener su valor real.

Ejemplo.- 3.14E-10 equivale a $3,14 \times 10^{-10}$

c) Constante entera seguida por un exponente

Se compone de una mantisa seguida de la parte de exponente

Formato: SXXXX...ESNNN

Donde.-

S, E y NNN: Tienen el mismo significado que en b)

XXXX: Es una constante entera.

Ejemplo.- 314E+10 equivale a 3140000000000.

Representación de una constante real en el Univac 9400

Una constante real, de cualquiera de los tres tipos definidos, se almacena en el ordenador en 5 bytes, en el formato denominado de punto flotante: " En el primer byte va el exponente o característica (con un "offset" de $128_{(10}$ ó $200_{(8}$)lo que permite representar los números con exponente negativo (del $0_{(8}$ al $200_{(8}$) y los de exponente positivo (del $200_{(8}$ al $377_{(8}$). En los 4 bytes siguientes va la fracción o mantisa en formato empaquetado, con espacio para 7 dígitos decimales y el último medio byte para el signo de la constante.

Es decir:

0	1	2	3	4	← bytes
exp	n1 | n2	n3 | n4	n5 | n6	n7 | S	

Siendo.- exp: La característica de la constante

ni: Dígitos del 0 al 9

S: El signo de la fracción { $C_{(16}$: +
{ $D_{(16}$: -

Ejemplos

314159.E-5 →	8 | 1	3 | 1	4 | 1	5 | 9	0 | C
-.15E-06 →	7 | A	1 | 5	0 | 0	0 | 0	0 | D
1.E+04 →	8 | 5	1 | 0	0 | 0	0 | 0	0 | C

De acuerdo con lo señalado anteriormente, el rango de validez de los números reales, con este tipo de representación es de: $10^{-128} \leq |R| \leq 10^{+127}$ y pueden tomar el valor cero.

2.1.3. CONSTANTE EN DOBLE PRECISION

Responde a las mismas características generales que la constante real, con la ventaja de almacenar un mayor número de cifras significativas. En el caso 9400 el número de dígitos significativos llega hasta 17.

Una constante en doble precisión admite los tres formatos señalados para las reales, con la salvedad de cambiar la letra E - que indica el exponente por la D, es decir:

a) SXX.XXX...

Donde.- XXX: Debe tener de 8 a 17 dígitos

b) SXX.XXX...DSNNN

c) SXXXX... DSNNN

Ejemplos

a)+3.14159265 →Tiene 9 dígitos

-89.654132556→Tiene 11 dígitos

b) 31.4159 D-1

-8.9654132 D+1

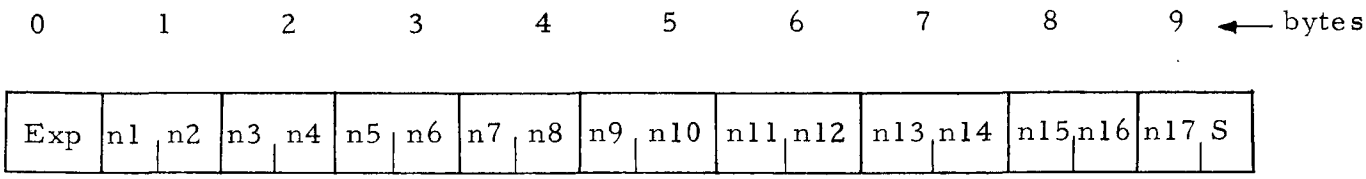
c) 314159 D-5

-896541 D-4

Representación de una constante en doble precisión en el UNIVAC 9400

Una constante en doble precisión se representa con 10 bytes, en el formato de punto flotante: "En el primer byte va el exponente o característica (con un "offset" de $128_{(10}$ ó $200_{(8}$) y en los 9 bytes siguientes va la fracción o mantisa en formato empaquetado, con espacio para 17 dígitos y el último medio byte para el signo de la constante.

Es decir:



Siendo.- exp: La característica de la constante

ni: Dígitos del 0 al 9

S: El signo de la fracción $\begin{cases} C_{(16} : + \\ D_{(16} : - \end{cases}$

El rango de validez de los números en doble precisión, con esta representación, es de: $10^{-128} \leq |R| \leq 10^{+127}$ y pueden tomar el valor cero.

2.1.4. CONSTANTE HEXADECIMAL

La constante hexadecimal, como su nombre indica, sirve para representar números en base 16 por lo que además de los dígitos 0 a 9 se utilizan en su representación las letras A a F con los valores respectivos de 10 a 15.

La estructura de una constante hexadecimal es:

Formato : ZXXXX...

Siendo.-

Z: Letra obligatoria que precede a la constante y señala que es una constante hexadecimal.

XXXX: Dígitos numéricos del 0 al 9, y las letras comprendidas entre la A y la F ambas inclusive.

El formato binario de los dígitos hexadecimales es el siguiente:

0-0000	4-0100	8-1000	C-1100
1-0001	5-0101	9-1001	D-1101
2-0010	6-0110	A-1010	E-1110
3-0011	7-0111	B-1011	F-1111

El número máximo de dígitos permitidos en una constante hexadecimal viene determinado por el tipo de variable asociada a la constante. Si el número de dígitos excede el máximo permitido, la constante queda truncada por la izquierda hasta ese máximo permitido. Si el número de dígitos de la constante es menor que el número máximo permitido, se "padea" o rellena la constante por la izquierda con ceros. Las constantes hexadecimales sólo se pueden usar con la sentencia DATA.

Ejemplo :

a) DATA I/Z4325AB/

se almacena :

0	1	2	3	4 ← bytes
00 00	43	25	AB	

b) DATA I/Z 123456789ABC/

se almacena:

0	1	2	3	4 ← bytes
34	56	78	9A	BC

2.1.5. CONSTANTE COMPLEJA

Una constante compleja consta de un par de constantes reales, que pueden llevar o no signo, y que van separadas por una coma y entre paréntesis. La primera constante del par es la -- parte real, y la segunda constante la parte imaginaria.

La estructura de una constante compleja es:

Formato : (R_1, R_2)

Siendo.-

R_1 y R_2: Dos constantes reales con el formato descrito en (2.1.2.) R_1 y R_2 ocupan 5 bytes cada una, y se situan en la memoria en posiciones adyacentes. R_1 en la primera parte y R_2 a continuación

Ejemplos

(2.0, -56.78E4) $\begin{cases} R_1 = 2.\emptyset \\ R_2 = -567800 \end{cases}$

(0., 0.) parte real e imaginaria nulas

(3.14, 0.0) $\begin{cases} R_1 = 3.14 \\ R_2 = \text{nula} \end{cases}$

2.1.6. CONSTANTE LOGICA

Una constante lógica o de Boole es la que puede tomar uno de estos dos valores:

. TRUE. $\left.\begin{matrix} \\ \end{matrix}\right\}$ Los puntos forman parte de la constante

. FALSE. $\left.\begin{matrix} \\ \end{matrix}\right\}$

Normalmente ocupa un byte, excepto cuando figura en una sentencia COMMON que ocupa 5 bytes. El valor ". TRUE." se representa internamente en memoria con $255_{(10}$ y el - ". FALSE." con 0.

Es decir:

$255_{(10} = 11111111_{(2} = .\text{TRUE.}$

$0_{(10} = 00000000_{(2} = .\text{FALSE.}$

Ejemplos:

```
A  = .TRUE.
SW = .FALSE.
DATA IF/.TRUE./
```

2.1.7. CONSTANTE LITERAL (O CONSTANTE HOLLERITH)

Una constante literal, denominada también constante Hollerith ó constante alfanumérica, consiste en una tira de uno o más caracteres del conjunto de caracteres admisibles del - 9400 que se vió en la Sección 1.3

Admite las siguientes representaciones:

Formatos:	a) wH	$e_1 e_2 \ldots e_n$
	b)	$'e_1 e_2 \ldots e'_n$

Siendo.-⟨e_i⟩: un caracter Univac 9400
⟨w⟩ la longitud de la tira ($\emptyset < w \leq 255$).

Nota.- En el formato b, un apóstrofo (') --- se representa por dos apóstrofos seguidos.

Ejemplo.- Sea el literal.-Cambio apóstrofo (') por asterisco (*)

a) 39 H CAMBIO APOSTROFO (') POR ASTERISCO (*).

b) ' CAMBIO APOSTROFO ('') POR ASTERISCO (*)'.

2.2. VARIABLES

Una variable es la representación simbólica de una determinada dirección de memoria cuyo contenido es susceptible de irse modificando en el transcurso de la ejecución del programa.

Las variables se referencian, como ya se comentó en la Sección 1.6., por un nombre simbólico (de 1 a 6 caracteres alfanuméricos, el primero de los cuales debe ser una letra).

Cada variable lleva asociada un tipo, que debe ser el mismo que el de los datos que representa, y de acuerdo con ese tipo operará el ordenador con el contenido de la dirección de memoria que define esa variable.

El FORTRAN efectúa una dicotomía y, si no se declara otra cosa, todas las variables del programa serán enteras o reales de acuerdo con los siguientes criterios:

. Si la primera letra del nombre de la variable es una de las comprendidas entre la I y la N (ambas inclusive), entonces la variable se considera entera.
. El resto de las variables se consideran reales.

Tipos de variables y ocupación en memoria

Existen cinco tipos de variables cuyas características de: ocupación de memoria y formato de representación, son idénticas que las de las constantes asociadas a ese tipo.

Estos tipos son:
- Variables enteras
- Variables reales
- Variables de doble precisión
- Variables complejas
- Variables lógicas

2.2.1. VARIABLES ENTERAS

No necesitan declararse mediante ninguna sentencia especial pues, como se ha dicho anteriormente, el compilador considera como tales, si no se le especifica lo contrario, a aquellas variables cuya primera letra está comprendida entre la I y la N ambas inclusive.

También se pueden declarar variables enteras utilizando la sentencia de tipo:

INTEGER

En el Univac 9400 una variable entera ocupa 5 bytes y su formato de representación es el de la constante entera.

Ejemplos

INI MONTE	JOTA NUME	INTEGER CAMPO, ZETA

2.2.2. VARIABLES REALES

Tampoco precisan declararse con ninguna sentencia especial, basta que la primera letra del nombre de la variable sea distinta de las comprendidas entre la I y la N.

También se pueden declarar variables reales utilizando la sentencia de tipo : REAL,

En el Univac 9400 una variable real ocupa 5 bytes y su formato es el de las constantes reales.

Ejemplos

ALTO BASE CONO	REAL IZERO, LA, M12

2.2.3. VARIABLES EN DOBLE PRECISION

Necesariamente deben ser declaradas con la sentencia de tipo: DOUBLE PRECISION. En el 9400 ocupan 10 bytes y su formato de representación es el de las constantes en doble precisión.

Ejemplo

DOUBLE PRECISION ALFA, BETA, LANDA, I1

2.2.4. VARIABLES COMPLEJAS

Estas variables deben ser declaradas con la sentencia tipo: COMPLEX. En el 9400 ocupan 10 bytes y su formato es el de las constantes complejas.

Ejemplo

```
COMPLEX BETA, ZO, ZR, I
```

2.2.5. VARIABLES LOGICAS

Estas variables también deben ser declaradas. Se utiliza para ello la sentencia de tipo: LOGICAL.

En el UNIVAC 9400, una variable lógica ocupa 1 byte, -- excepto si está dentro de un COMMON ó EQUIVALENCE que ocupa 5 bytes. En este caso el compilador sólo trata el byte menos significativo (más a la derecha), los otros 4 bytes los ignora.

Su formato de representación es el de las variables lógicas.

Ejemplo

```
LOGICAL A1, MAN, TRU, FALSO
```

2.3. ARRAYS O TABLAS

Un Array o Tabla es un conjunto de variables, agrupadas bajo un único nombre genérico (el nombre del Array), y siendo todas las variables del mismo tipo. Se referencia,o llama a una de ellas,con el nombre de la tabla y entre paréntesis tantos sub

índices, separados por comas, como dimensiones tenga la tabla. Los subíndices son números o expresiones aritméticas que individualizan un elemento dentro del conjunto tabla.

Una tabla nos permite definir estructuras lógicas que no tienen - porque ser las típicas estructuras matriciales del álgebra.

Un array FORTRAN 9400 tiene las siguientes características:

. Puede ser declarado con un máximo de 7 dimensiones, y se deben declarar mediante la sentencia DIMENSION ó COMMON o de tipificación explícita como se verá en el Capítulo 5.

Ejemplo

DIMENSION GENERA (5, 6, 4, 20, 5, 2, 3)

. En la declaración cada subíndice debe ser una constante entera sin signo.

. En el caso de subprogramas: funciones o subrutinas, se pueden utilizar como subíndices, en la declaración de las tablas, variables enteras. En este caso las tablas deberán estar también declaradas en el módulo principal y se deberá transmitir su nombre al subprograma bien como argumento, bien en la zona común. El valor de la variable entera, correspondiente a cada subíndice, no deberá exceder al de la respectiva constante entera asignada al subíndice de la tabla, en el módulo principal. Esta forma de dimensionamiento se denomina "dimensionamiento ajustable". Las variables enteras que determinan las dimensiones se pueden transmitir mediante un COMMON ó como argumentos.

Ejemplo

Módulo principal	Subprograma
DIMENSION TAB1 (7, 8)	SUBROUTINE SUMA (TAB1, RESU)
COMMON M, N	COMMON M, N
M = 7	DIMENSION TAB1 (M, N)
N = 7	--
CALL SUMA (TAB1, RESU)	--
--	--
--	--
	--
	--

Ejemplo de tabla

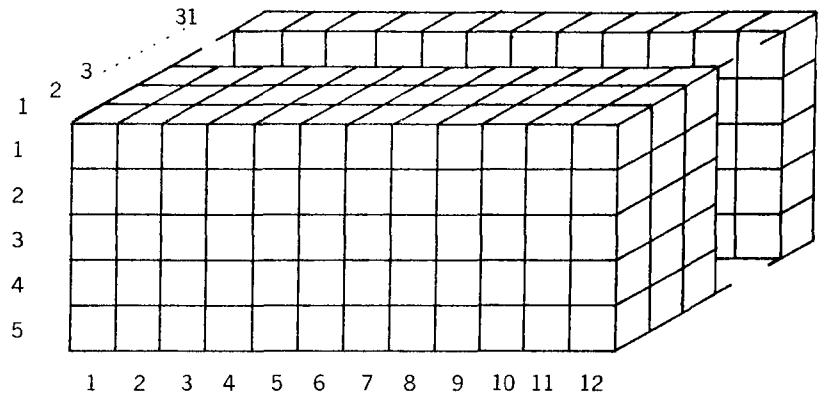
Queremos representar las horas de sol, de cada día, en un período de 5 años, de forma que las de un día en concreto sean accesibles dando el año, mes y día.

Una tabla sencilla que defina todos los días, en ese período, tendrá tres dimensiones y será de la forma:

DIMENSION HORAS (5, 12, 31)

- La primera dimensión define los años y podrá tomar valores de 1 a 5.
- La segunda dimensión define los meses y podrá tomar valores de 1 a 12.
- La tercera dimensión define los días de un mes y podrá tomar valores de 1 a 31.

Es decir, se puede representar la tabla como el ortoedro de la figura:



cuyo número de elementos o datos es de 5 x 12 x 31 = 1860

2.3.1. REFERENCIA A UN ELEMENTO DE UN ARRAY

A un elemento de una tabla se accede de la siguiente forma:

Formato.- nombre $(s_1, s_2, \ldots, s_n)$

Siendo.-

<nombre> : Nombre del Array

$<s_i>$: Expresión entera o real . Se evalúa en tiempo de ejecución como un entero cuyo valor debe ser mayor que cero.
(El concepto de expresión entera y real se detalla en el Capítulo 3).

<n> : Debe ser igual al número de subíndices o dimensiones de la declaración del array.

En la sentencia EQUIVALENCE se admite que el número de subíndices sea 1, pues en este caso se considera la posición relativa dentro del array.

Ejemplo anterior (continuación)

Poner EQUIVALENCE (HORAS(1, 2, 1), A) es idéntico que poner EQUIVALENCE (HORAS(6), A) porque HORAS (1, 2, 1) es el elemento 6º de HORAS. Ello es debido a que, los elementos de una tabla se almacenan variando más rápidamente los subíndices de la izquierda. En nuestro ejemplo quedaría almacenada en memoria, en posiciones consecutivas, en el mismo orden que se indica a continuación:

- El primer día de cada mes en los 5 años

	1º	2º	3º	4º	5º
Primer elemento en memoria →	(1, 1, 1)	(2, 1, 1)	(3, 1, 1)	(4, 1, 1)	(5, 1, 1)
	(1, 2, 1)	(2, 2, 1)	(3, 2, 1)	(4, 2, 1)	(5, 2, 1)

	(1, 12, 1)	(2, 12, 1)	(3, 12, 1)	(4, 12, 1)	(5, 12, 1)

- El último día de cada mes en los 5 años

(1, 1, 31)	(2, 1, 31)	(3, 1, 31)	(4, 1, 31)	(5, 1, 31)

(1, 12, 31)	(2, 12, 31)	(3, 12, 31)	(4, 12, 31)	(5, 12, 31)

2.3.2. SITUACION DE UN ELEMENTO EN UNA TABLA FORTRAN

Como se ha indicado anteriormente y hemos visto en el ejemplo anterior, los elementos de una tabla FORTRAN se almacenan en memoria de una forma lineal y con un orden tal que su primer subíndice es el que varía más rápido.

En la Tabla 2.1 se da una fórmula de direccionamiento para cada una de las dimensiones que admite el FORTRAN. En la columna declaración, las letras A, B... G son los tamaños que se han dado a cada dimensión en la sentencia DIMENSION que define la tabla. La columna subíndices especifica, con a, b, ..., g, el valor de los subíndices al referenciar un elemento. La columna posición relativa da el orden de cada elemento, dentro del orden de almacenamiento descrito anteriormente, teniendo en cuenta que al primer elemento de la tabla se le asigna el cero.

m indica la ocupación de un elemento. Vale 5 para los casos de elementos enteros y reales, 10 para doble precisión y complejos, y 165 (según el apartado 2.2.5) para los lógicos.

Tabla 2.1

Nº de dimensiones	Declaración	Subíndices	Posición relativa
1	(A)	(a)	(a-1)m
2	(A, B)	(a, b)	((a-1)+A*(b-1))m
3	(A, B, C)	(a, b, c)	((a-1)+A*(b-1)+A*B(c-1))m
--	-------	---------	------------------------
7	(A, B, C, D, E, F, G)	(a, b, c, d, e, f, g)	((a-1)+A*(b-1)+A*B(c-1) .. +A*B*C*D*E*F(g-1))m

CAPITULO 3

SENTENCIAS DE ASIGNACION

3.1. EXPRESIONES

Se entiende por expresión FORTRAN a un conjunto de constantes, variables, operadores y funciones, separados por paréntesis y/o, símbolos de operación, de forma que estructuren una expresión - con significado matemático o lógico, según el caso.

En FORTRAN se pueden evaluar dos tipos de expresiones:

3.1.1. EXPRESION ARITMETICA

Conforma fórmulas matemáticas. Una expresión aritmética siempre proporciona un valor numérico.

Su estructura es:

Formato. - <operando$_1$> <operador> <operando$_2$>

Siendo. -

<operando$_1$> y <operando$_2$> : Una constante entera, real, de doble precisión o compleja.

- Una variable (o elemento de un array) entera, real, de - doble precisión o compleja.
- Una referencia a una función
- Otra expresión aritmética.

<operador>: Uno de los siguientes

**	Indica la exponenciación
*	Indica la multiplicación

/ Indica la división

+ Indica la suma

- Indica la resta

Nota.- Cuando <operando $_1$> es un valor complejo y <operador> es **, <operando $_2$> sólo puede ser un valor entero. Un valor complejo no se puede utilizar como exponente.

El tipo de formato en que se efectúa el cálculo de la expresión depende de los operandos que lo forman, y se realiza de acuerdo con el siguiente cuadro:

Operando$_1$ \ Operando$_2$	Entero	Real	Doble precisión (D.P.)	Complejo
Entero	Entero	Real	D.P.	Complejo
Real	Real	Real	D.P.	Complejo
D.P.	D.P.	D.P.	D.P.	Complejo
Complejo	Complejo	Complejo	Complejo	Complejo

Quiere decir que si, por ejemplo, el primer operando es entero y el segundo es real (o viceversa) el operando entero se pasa automáticamente a formato real y la exprexión se calcula en formato real.

Nota.- En el caso de enteros, reales y doble precisión con complejos, al operando no complejo lo toma el compilador como la parte real de un operando complejo.

Ejemplos de expresiones aritméticas

2.+C/P-Z*COS(PI)

X+Y

3*X+4

4/3*2.142835*B**4

3.1.2. EXPRESION LOGICA

Una expresión lógica es aquella que al evaluarla proporciona como resultado los valores lógicos .TRUE. ó ---
.FALSE.

Se puede formar de dos modos:

a) Utilizando operadores lógicos

Formato.- <operando$_1$> <operador lógico> <operando$_2$>

Siendo.-

<operando$_1$> y <operando$_2$>: . Una constante lógica
- Una variable lógica
- Un elemento de un array lógico
- Una función lógica
- Otra expresión lógica

<operador lógico>: Puede ser.-

.NOT. Indica negación

.AND. Indica conjunción

.OR. Indica disyunción

Seguidamente se representa la " tabla de verdad" :

Sea <operando$_1$> → a y <operando$_2$> → b

a	b	a.OR.b	a.AND.b	.NOT.a
.TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.

Interesa señalar que el operador .NOT. es unario (sólo precisa un operando) y los operadores .OR. y .AND. - son binarios.

b) Utilizando operadores de relación o de comparación

Formato .- <operando$_1$> <operador de relación> <operando$_2$>

Siendo.-

<operando$_1$> y <operando$_2$>: . Una expresión aritmética (tal como se definía en el apartado 3.1.1.)

. No se admiten operandos que sean valores complejos o constantes literales.

<operador de relación>: Puede ser uno de los siguientes:

.GT. Indica mayor que (>)

.GE. Indica mayor o igual que (≥)

.LT. Indica menor que (<)

.LE. Indica menor o igual que (≤)

.EQ. Indica igual que (=)

.NE. Indica no igual o desi gual (≠)

Ejemplos de expresiones lógicas

```
LOGICAL L1, L2, L3, L4
L2.AND.L3.OR.L4
.NOT.L4
L2.AND.(A.GT.B)
L1.OR.B.LT.C.OR.C.EQ.2.35* A
```

3.1.3. ORDEN DE EVALUACION EN UNA EXPRESION

La evaluación de las expresiones se hace siempre por parejas, y de acuerdo con el orden jerárquico de los operadores que se indica a continuación:

. Orden jerárquico de los operadores

El número de orden más pequeño señala mayor prioridad

Operadores	Orden
. Aritméticos:	
**	1º
*, /	2º
+, -	3º
. De relación:	
.GT., .GE.,.LT., .LE.,.EQ.,.NE.	4º
. Lógicos:	
.NOT.	5º
.AND.	6º
.OR.	7º

En la expresión lo primero que se evalúan son las llamadas a funciones.

Si en la expresión existen paréntesis, los valores encerrados entre paréntesis son los primeros que se calculan, comenzando por el paréntesis más interno. Dentro de un mismo nivel de paréntesis (o de toda la expresión si no existen paréntesis) el orden de cálculo es de acuerdo con el orden jerárquico de los operadores. Cuando se encuentren operadores del mismo orden,el cálculo se realiza de izquierda a derecha.

La única excepción en la evaluación de izquierda a derecha para un mismo orden jerárquico, es el caso de dos o más exponenciaciones consecutivas (Ej.- A**B**C se evalúa A**(B**C)).

3.1.4. REGLAS PARA LA FORMACION DE EXPRESIONES

A la hora de codificar una expresión, el programador se habrá ajustado a todo lo señalado en los apartados anteriores, y además deberá tener presente las siguientes reglas de formación:

- Todas las constantes, variables, elementos de un array y llamadas a funciones deben estar separadas por un operador.
- No pueden ir juntos dos operadores, sólo está permitido cuando el segundo operador sea un + ó un -, en este caso deben ir separados por paréntesis (Ej.- A*-B debe ir A*(-B)).

- En la expresión deben existir el mismo número de paréntesis de apertura que de cierre.
- Aunque es aconsejable, los datos que conforman una expresión aritmética no precisan ser del mismo tipo. Se admite la mezcla de tipos señalada en el apartado 3.1.1.

Ejemplos

a) Expresión matemática	Expresión escrita en FORTRAN
$R+\frac{A-B}{C-X}$	R+(A-B)/(C-X)
$1/3B.H+4/3\pi R^3$	B*H/3+4/3*3.1415*R**3

b) Sean

IA Variable entera, con contenido 128

IB variable entera, con contenido 328

AA Variable real, con contenido 12,4=12.4

AB Variable real, con contenido 0,03 = 0.03

LA Variable lógica, con contenido .TRUE.

LB Variable lógica, con contenido .FALSE.

DA Variable en doble precisión, con contenido $12 \times 10^2 =$ = 12D+2

DB Variable en doble precisión, con contenido 0,002 = = 0.2D-2

Y dada la expresión:

<expresión>:.NOT.IA+AA+AB-DB.GT.140.OR.LB.AND. DA*IB.LT.1.1E+5

Su evaluación es como sigue y en el siguiente orden:

1º) DA**IB = 393600(Se calcula en doble precisión)
2º) IA+AA = 140,4 (Se calcula en real)
3º) 2º+AB = 140,43 (Se calcula en real)
4º) 3º - DB = 140,428 (Se calcula en doble precisión)
5º) 4º.GT.140 = .TRUE.
6º) 1º.LT.110000 =.FALSE.
7º) .NOT.5º = .FALSE.
8º) LB.AND.6º =.FALSE.
9º) 7º.OR.8º = .FALSE.

El resultado de la expresión es .FALSE., es decir,es una expresión lógica.
Utilizando paréntesis, la expresión anterior sería equivalente a:

(.NOT.((IA+AA+AB-DB).GT.140)).OR.(LB.AND.(((DA**IB)).LT.1.1E+5))

3.2. SENTENCIA DE ASIGNACION ARITMETICA

Una sentencia de asignación aritmética está constituída por un nombre de variable (o elemento de un array) seguido del signo igual (=) y de una expresión aritmética.

Es decir:

Formato.- v = e

Siendo.- <v>:Una variable entera, real, de doble precisión o compleja.

. Un elemento de un array entero, real, de doble precisión o complejo.

<e> Una expresión aritmética, como se ha descrito en el apartado 3.1.1.

Descripción.- La sentencia de asignación aritmética efectúa la siguiente operación: "Sustituye el valor que tenía la variable <v> ,situada a la izquierda del signo igual, por el resultado de la expresión <e> situada a la derecha de dicho signo. Como es evidente, el contenido de la variable <v> se modifica (su nuevo valor puede ser el mismo que tenía anteriormente) con la ejecución de esta sentencia.

Conviene remarcar que la expresión < e > se calcula de acuerdo con la mezcla de tipos indicados en el apartado 3.1.1., pero este valor se transfiere a < v > con el mismo tipo -- que tenga esta variable.
Cuando < v > sea una variable compleja y < e > sea una expresión de tipo entero, real o de doble precisión, el valor de < e > se transfiere a la parte real de < v >.

Ejemplo: Sean A variable real , con contenido 4.0
I variable entera, con contenido 3
K variable entera, con contenido 12

La ejecución de la sentencia:

K = A/I

Se realiza en los siguientes términos:

1º) Expresión (A/I): 4.0/3 = 1.3333... (se calcula en real)

2º) Como la variable K es de tipo entero, el resultado de la expresión (A/I) se pasa a formato entero: 1.3333 → 1.

3º) Se transfiere este valor a la variable K, cuyo contenido pasa de ser 12 a ser 1.

"Habitualmente se dice, aunque no sea muy apropiado - que la variable K pasa de valor 12 a valor 1". Como es evi dente no nos deberíamos referir a la variable K sino al con tenido de la variable K, ya que como se vió en la sección 2.2. una variable representa una dirección de memoria.

3.3. SENTENCIA DE ASIGNACION LOGICA

Una sentencia de asignación lógica está formada por un nombre de variable (o elemento de un array) lógica seguido del signo - igual (=) y de una expresión lógica.

Es decir:

Formato. - v = e

Siendo. -

⟨v⟩ : Una variable lógica
Un elemento de un array lógico

⟨e⟩ : Una expresión lógica, como se ha descrito en el apar tado 3.1.2.

Descripción.- La sentencia de asignación lógica sustituye el valor que tenía la variable lógica <v>, situada a la izquierda del signo igual, por el resultado de la expresión lógica <e>, que deberá ser .TRUE. ó .FALSE., situada a la derecha de dicho signo.

Ejemplo

Sean A variable lógica, con contenido .TRUE.
B variable entera, con contenido 2
C variable entera, con contenido 3

La sentencia de asignación lógica:

A = B.GT.C

dará como resultado .FALSE., y este valor será asignado a la variable A.

3.4. SENTENCIA <ASSIGN>

Esta sentencia no definida en el FORTRAN IV estandar, presenta la siguiente estructura:

Formato.- ASSIGN<etiqueta> TO<variable entera>

Siendo.-

<etiqueta>: Constante entera sin signo, declarada en el programa fuente en el campo ETIQUETA (Columnas 1 a 5)

<variable entera>: Definida en el apartado 2.2.1.

Descripción.- Asigna una etiqueta a una variable entera para poder utilizarla posteriormente en un GO TO

asignado o como referencia de una sentencia FORMAT (ambas sentencias se describen en capítulos posteriores).

Nota.- Las variables así asignadas no se pueden utilizar para otra acción diferente (ej.- dentro de una expresión aritmética) hasta que no vuelvan a ser redefinidas mediante una sentencia de asignación aritmética o una sentencia READ.

Ejemplos.-

```
ASSIGN   8 TO J
      :
GO TO J, (8, 34, 17)
      :
ASSIGN 34 TO J
      :
ASSING 17 TO J
      :
```

CAPITULO 4

SENTENCIAS DE CONTROL

Un programa FORTRAN lo constituyen un conjunto de instrucciones que se van ejecutando en secuencia, es decir,una vez ejecutada una de ellas se pasa el control a la que le sigue inmediatamente. Muchas veces interesa alterar la secuencia normal y transferir el control a otro punto del programa que no sea la instrucción inmediatamente siguiente. Las sentencias que permiten efectuar esta "ruptura de secuencia" se denominan sentencias de CONTROL.

Las sentencias de control son de dos tipos:

a) Las que exigen una acción obligada sin depender de ninguna condición previa (Ej.- STOP, GO TO incondicional).

b) Las que son dependientes de que se verifique una determinada condición (Ej.- IF aritmético ó lógico).

Son sentencias de Control del FORTRAN 9400:

- IF aritmético, IF lógico
- GO TO incondicional, calculado y asignado
- DO
- CONTINUE
- PAUSE, STOP
- END

Nota.- El DO y CONTINUE aunque no están dentro del grupo de "sentencias de ruptura de secuencia", también se las incluye, -- normalmente,dentro del grupo de sentencias de control.

4.1. SENTENCIA ⟨IF⟩ aritmético

Tiene la siguiente representación:

Formato.- IF (e) et_1, et_2, et_3

Siendo. - ⟨e⟩ :Una expresión aritmética. No está permitida la compleja.

⟨et_1, et_2, et_3⟩:Son tres etiquetas correspondientes a sentencias ejecutables del programa,separadas entre sí por comas.

Descripción. - La sentencia'IF aritmético"permite la transferencia del control del programa en tres direcciones, a las instrucciones etiquetadas con et_1, et_2 ó et_3según sea el valor de la expresión ⟨e⟩:

Si ⟨e⟩ es menor que cero, el control se pasa a et_1.

Si ⟨e⟩ es igual a cero, el control se pasa a et_2.

Si ⟨e⟩ es mayor que cero, el control se pasa a et_3.

Se permite no especificar alguna de las tres etiquetas pero en este caso es preciso consignar -- las comas de cabecera o intermedias para señalar ausencia de etiqueta, no siendo necesario poner las comas de cola.

Según la etiqueta que esté ausente, el formato del IF será:

Etiquetas ausentes	Formato del IF
et_1	IF(e), et_2, et_3
et_2	IF(e) et_1,,et_3
et_3	IF(e) et_1, et_2
et_1 y et_2	IF(e) ,, et_3
et_1 y et_3	IF(e), et_2
et_2 y et_3	IF(e) et_1

Cuando no se especifica etiqueta y la condición del IF se verifica para esa salida sin etiqueta, el control se transfiere a la -- siguiente instrucción en secuencia que sea ejecutable.

Ejemplos

```
a)      IF(I-4) 10,20,30
                =
                =
                =
   10     A=A+SIN(x)
                =
                =
                =
   20     I=I+1
                =
                =
                =
                =
   30     WRITE(6,5)A1
```

Si I-4 < 0 va a la sentencia etiquetada con 10
Si I-4 = 0 va a la sentencia etiquetada con 20
Si I-4 > 0 va a la sentencia etiquetada con 30

```
b)      IF (IA-IB),,3
        IA = IA+1
                =
                =
                =
   3    IB = IA+COS(x)
```

Si IA≤IB, el control se transfiere a la sentencia IA = IA+1
Si IA>IB, el control se transfiere a la sentencia etiquetada con 3 -- (IB = IA+COS(x))

4.2. SENTENCIA⟨IF⟩ Lógico

Tiene la siguiente representación:

Formato.- IF (e) sentencia

Siendo.- ⟨e⟩ : Una expresión lógica.
⟨sentencia⟩: Cualquier instrucción ejecutable, excepto un DO u otro IF lógico.

Descripción.- La sentencia "IF lógico" permite comprobar una condición lógica definida por ⟨e⟩. Si la expresión lógica⟨e⟩es cierta (.TRUE.) se eje

cuta la ⟨ sentencia ⟩ escrita al lado y luego el programa continua en secuencia. Si la expresión ⟨ e ⟩ es falsa - (.FALSE.) el control pasa a la siguiente instrucción sin ejecutarse la ⟨ sentencia ⟩ situada en la misma línea que el IF lógico.

Ejemplos.-

a) IF (A.GT.B) SUM = SUM+ 1

- Si A⟩B.- El contenido de la variable SUM se incrementa en 1
- Si A⟨B.- No se ejecuta la instrucción SUM = SUM+1

b) No estaría permitida la siguiente instrucción, por ser ⟨sentencia⟩ otro IF lógico:

IF(A.LT.B) $\underbrace{\text{IF(C.GT.D) X = SIN(Y)}}_{\text{Sentencia}}$

4.3. SENTENCIA ⟨ GO TO⟩ incondicional

Tiene la siguiente estructura:

Formato.- GO TO et

Siendo.-⟨et⟩ : La etiqueta de una instrucción ejecutable.

Descripción.- Permite la transferencia incondicional del control a la instrucción que lleva esa etiqueta. Es decir, el programa continua su ejecución en la instrucción señalada por esa etiqueta. Cualquier sentencia ejecutable que siga inmediatamente a una sentencia GO TO incondicional deberá llevar etiqueta, en caso contrario nunca podrá ejecutarse.

Ejemplo

GO TO 10.- Bifurca a la instrucción FORTRAN etiquetada con 10.

```
      GO TO 10
      ...
10    A = SIN (X)
```

4.4. SENTENCIA <GO TO> Calculado

Se ajusta al siguiente formato:

Formato.- GO TO (et_1, et_2, . . . , et_n), variable entera

Siendo.- $\langle et_i \rangle$: La etiqueta de una instrucción ejecutable.

<variable entera>: Como se definió en el apartado 2.2.1. Debe ser mayor que cero y menor o igual a n.

El valor máximo de n para el FORTRAN 9400 es de 99.

Descripción.- Transfiere el control a et_k, donde 'k' es el valor actual de la variable entera. Es decir, si variable entera = 1 el control se transfiere a la instrucción etiquetada con et_1, si variable entera = 2 se transfiere a la etiquetada con et_2, y así sucesivamente.

Si $k \leq 0$ ó $k > n$, se produce un diagnóstico de error durante la ejecución y el programa termina.

Una misma etiqueta puede repetirse tantas veces como sea necesario (ej: GO TO (5, 5, 6, 5), I.).

Ejemplo

```
I = 2
GO TO (10, 20, 21, 11), I→Bifurca a la instrucción FORTRAN
                          etiquetada con 20.
```

4.5. SENTENCIA <GO TO> Asignado

Tiene la siguiente estructura:

Formato.- GO TO variable entera, $(et_1, et_2, \ldots . et_n)$

Siendo.- <et_1>: La etiqueta de una instrucción ejecutable

<variable entera> Su valor debe ser idéntico a una de las etiquetas de la lista (ej. igual a et_1 ó a $et_2, \ldots .$). Este valor sólo se puede asignar, a la <variable entera>, con la sentencia ASSIGN descrita en la Sección 3.4.

Si el contenido de la <variable entera> correspondiera a una etiqueta del programa, y esta etiqueta no está representada por ningún <et> de la sentencia GO TO, se produce, en tiempo de compilación, un diagnóstico de error.

Descripción.- La sentencia "GO TO asignado" transfiere el control del programa a la instrucción etiquetada con el valor actual que tiene la <variable entera>.

Ejemplo

```
ASSIGN 10 TO I
       ═══
       ═══
GO TO I, (11, 10, 12)  ──►Bifurca a la instrucción cuya etiqueta es 10
```

4.6. SENTENCIA <DO>

Esta sentencia es una de las más importantes del FORTRAN. Presenta los siguientes formatos:

Formato.-

a) DO etiqueta i= par_1, par_2, par_3

b) DO etiqueta i = par_1, par_2

Siendo.- <etiqueta> :Es la etiqueta de la instrucción que finaliza el DO.

<i>: Es el nombre de una variable entera. Se le denomina variable de control del DO. Esta variable sólo puede tomar valores válidos si se verifica que $(m_2 - m_1) * m_3 > 0$.

<par_1>: Valor inicial de la variable de control <i>. Debe ser una constante o variable entera.

<par_2>: Valor final de <i>. Debe ser también una constante o variable entera.

<par_3>: Incremento de la variable de control <i>. Debe ser una constante o variable entera distinta de cero. En el caso del formato b) en que par_3 no se especifica, se asume $par_3 = +1$.

Descripción.- La sentencia DO, denominada de "bucle", permite ejecutar repetidamente un conjunto de instrucciones que son las comprendidas entre el DO y la instrucción etiquetada con el valor <etiqueta>, descrito anteriormente. A estas sentencias se las denomina rango del DO.

El número de veces que se ejecutan estas sentencias viene gobernado por la progresión de la variable de control <i> , que inicia con el valor <par 1> , y toma los sucesivos valores de <par 1> + <par 3> ,<par 1> + 2x<par 3> , ... y así sucesivamente hasta que <i> alcance un valor superior a <par 2> . En este momento, continúa la ejecución del programa en secuencia, es decir, fuera del rango del DO.

Ejemplo

```
DO 5 J = 10, 20, 3
       =
       =
5  CONTINUE
----------
```

. El DO se ejecuta para los siguientes valores de J: 10, 13, 16, 19

4.6.1. CONSIDERACIONES SOBRE EL RANGO DEL DO

- La última sentencia del rango de un DO (sentencia etiquetada con <etiqueta>) no puede ser ni una instrucción GO TO, ni un IF, ni otro DO.
- El incremento de la variable de control <i> y su posterior comparación con <par 2> se efectúa al final del DO. Ello quiere decir que aunque los parámetros del DO sean incorrectos, el rango se ejecuta por lo menos la primera vez.
- En el rango de un DO no se permiten sentencias que asignen un valor a la variable de control <i> , ni que alteren <par 1> , <par 2> ó <par 3>

Ejemplo

```
   DO 2 I = 1, 5
          =
   I = I + 1  no está permitido
          =
2     CONTINUE
```

- Si un bucle DO contiene a otro (DO, s anidados), el rango del segundo debe estar totalmente contenido dentro del rango del primero.

. El rango de un DO interno puede contener la última sentencia del rango DO externo inmediato. En este caso, una transferencia de control a la última sentencia de los DO, s finaliza la ejecución del DO más interno.

Ejemplo

```
      DO 1 I = 1,4
      DO 1 J = 1,5
      ------------
      ------------
      ------------
1     CONTINUE
```

. Se dice que un conjunto de DO, s están completamente anidados si ningún rango de DO termina antes de que empiece otro rango de DO.

Ejemplo de DO, s completamente anidados

```
      DO 10 I = 1,5
      DO 11 J = 1,6
      DO 12 K = 1,4
12    CONTINUE
11    CONTINUE
10    CONTINUE
```

4.6.2. TRANSFERENCIAS DE CONTROL

. En cualquier momento, el control del programa puede ser transferido fuera del rango de cualquiera de los bucles DO's de que conste. Cuando se produce este hecho el control también puede ser transferido dentro del rango de un DO, pero únicamente bajo las siguientes condiciones:

1) Si el rango del DO no tiene DO's anidados, se puede introducir en él mediante:

   . Un GO TO.

   . Un IF lógico, conteniendo un GO TO.

   . Un IF aritmético puede transferir el control a otra sección del programa que contenga una de estas -- sentencias señaladas anteriormente y que transfieren el control al mismo bucle del DO.

2) Si hay DO's anidados el control se puede transferir fuera del DO, y entonces retornar al mismo rango - del DO bajo las condiciones que se han descrito en - el apartado anterior.

   En otras palabras, no está permitido transferir el control del programa dentro del rango de un DO si no es por su cabeza (sentencia DO) o bajo las condiciones señaladas anteriormente.

   . Cuando el control del programa se transfiere fuera del rango de un DO, sus parámetros permanecen - definidos. Si la sección del programa, que recibe el control, retorna a ese DO, no se deben redefinir los parámetros del DO.

   Ejemplo (no permitido)

```
      CALL SUB(&2)
      DO1 I = 1,5
2     A = A+SIN(X)
1     CONTINUE
```

4.7. SENTENCIA <CONTINUE>

Es una sentencia ejecutable FORTRAN, pero ficticia en el sentido de que no produce ninguna instrucción real en el programa objeto.

Formato.- CONTINUE

Descripción.- Permite definir una etiqueta a la cual poder transferir el control del programa.

La sentencia CONTINUE se emplea principalmente como instrucción terminal del rango de un DO, --- cuando la última sentencia del bucle es una de las de control no permitidas.

Al no generar código puede situarse en cualquier - parte del programa sin afectar a la secuencia de - ejecución.

Ejemplo

```
      DO  100  I = 1,20, +2
      ------------------
100   CONTINUE
```

4.8. SENTENCIA <STOP>

Presenta los siguientes formatos:

Formato.-
a) STOP
b) STOP n

Siendo.- <n>:Constante entera sin signo no superior a 5 dígitos.

Descripción.- La sentencia STOP finaliza la ejecución del programa y retorna el control al Sistema Operativo. Cuando se ejecuta STOP ó STOP n se visualiza en la consola del operador el mensaje que se indica a continuación. El ordenador no espera ninguna respuesta del operador a dicho mensaje.
El mensaje tiene la siguiente forma, dependiendo del formato de la sentencia STOP:

- Para el caso a)

←——————→FTØ1bSTOP

(11 blancos)

- Para el caso b)

←——————→FTØ1b STOP: bn

(11 blancos)

Ejemplos.-

STOP 123

STOP

4.9. SENTENCIA <PAUSE>

Admite los siguientes formatos:

Formato.-

a) PAUSE

b) PAUSE n

c) PAUSE 'mensaje'

Siendo.- <n>:Constante entera sin signo no superior a 5 dígitos
<mensaje> :Literal no superior a 255 caracteres.

Se permite todo el juego de caracteres UNIVAC 9400.

Descripción.- Esta sentencia provoca, en el momento de la ejecución, una parada momentánea del programa, y produce la visualización de un mensaje en la consola del operador. El programa reanuda o no su ejecución con la sentencia inmediatamente siguiente a la PAUSE, dependiendo de la respuesta que se de en la consola de operación.
El mensaje que se visualiza en la consola depende del formato empleado:

- Para el caso a)

←——————→FTØØbPAUSE:
(11 blancos)

- Para el caso b)

←——————→FTØØbPAUSE: b n
(11 blancos)

- Para el caso c)

←——————→FTØØbPAUSE: b mensaje
(11 blancos)

Nota.- Si el mensaje excede de 56 caracteres, cada conjunto de 56 caracteres se visualiza en línea siguiente.

. Contestación del Operador

El operador desde la consola puede contestar:

GO.- El programa ejecuta la siguiente instrucción en secuencia

EOJ.- El Ordenador transfiere el contrd a una macrointrucción EOJ (end of job) del supervisor y finaliza le ejecución del programa.

Ejemplos.-

```
PAUSE

PAUSE 12

PAUSE 'fin de iteración'
```

Nota.- Esta sentencia es un vestigio de las primeras versiones FORTRAN y se mantiene a efectos de compatibilidad. Debe ser evitada en lo posible porque su utilización origina molestias al operador.

4.10. SENTENCIA <END>

Se escribe de la siguiente forma:

Formato.- END

Descripción.- Es la última sentencia de cada módulo unidad y sirve para señalar al compilador FORTRAN que ha finalizado el programa fuente de ese módulo.
Es una sentencia ejecutable y admite etiqueta.
Cuando se ejecuta, visualiza un mensaje en la consola de operación y seguidamente pasa el control del programa al Sistema Operativo. A los efectos de ejecución se comporta como la sentencia STOP.
El mensaje que visualiza en la consola es:

(11 blancos) ⟵⟶ FTØ2bSTOP: bEND

Nota.- En la mayor parte de los compiladores la sentencia END no es una instrucción ejecutable y no admite etiqueta.

Ejemplo.-

```
      SUBROUTINE SUMA (A, B, C)
      ------
      ------
      ------
      END
```

CAPITULO 5

SENTENCIAS DE ESPECIFICACION

5.1. GENERAL

En este capítulo se describe cómo se declara una tabla o array en un programa FORTRAN, y las sentencias:

- DIMENSION
- INTEGER
- REAL
- DOUBLE PRECISION
- COMPLEX
- LOGICAL
- IMPLICIT
- EQUIVALENCE
- COMMON
- DATA
- EXTERNAL

5.2. DECLARADOR DE UN ARRAY

En la sección 2.3. se detallaba el concepto de array, y se le presentaba como un conjunto ordenado de elementos identificados todos ellos por un nombre simbólico, que es el nombre del array.

El formato de un "declarador de array" es:

Formato. - Nombre del array ($dim_1 \ldots, dim_n$)

Siendo. -

⟨Nombre del array⟩ : Nombre simbólico que identifica al array.

⟨dim_i⟩ : Constante entera sin signo (y/0 variable entera en el caso de funciones y subrutinas). Ver sección 2.3.

<n>:Número de dimensiones del array. En el 9400 debe ser igual o menor que 7.

Un array puede ser declarado en una sentencia DIMENSION, COMMON ó de declaración de tipo.

Ejemplo

DIMENSION SUM(5, 4, 8)

Declara un array con nombre SUM, y que tiene tres dimensiones: 1ª Varía de 1 a 5
2ª Varía de 1 a 4
3ª Varía de 1 a 8

Con un total de 160 elementos.

5.3. SENTENCIA <DIMENSION>

Presenta la siguiente forma:

Formato.-
DIMENSION declarador 1, . . . , declarador k

Siendo.- <declarador i>: Descriptor de una tabla o array, como se acaba de especificar en 5.2.

Descripción.- Sirve para declarar tablas, utilizando para ello un declarador específico para cada tabla.

Ejemplo

Si se desea declarar un array que representa un calendario de un año, que diferencia el mes y la semana a que pertenece un determinado día, la sentencia DIMENSION apropiada es:

DIMENSION CALEN (7, 5, 12)

- 12 → meses del año
- 5 → semanas posibles en un mes
- 7 → días de la semana

5.4. SENTENCIAS PARA DECLARACION DE TIPOS

En la sección 2.2., cuando se hablaba de tipos de variables se especificaba el tipo automático que asigna el FORTRAN, creando una dicotomía entre variables enteras y reales. Así mismo se señalaba que las variables de doble precisión, complejas y lógicas exigían una declaración explícita de tipo para poder ser definidas.

Existe la posibilidad de cambiar, para las variables que lo precisen, esta asignación automática de tipo que efectúa el FORTRAN, y de poder declarar otro tipo de variables, además de las enteras y reales, utilizando para ello las sentencias de declaración de tipo que se describen a continuación.

5.4.1. SENTENCIAS DE TIPIFICACION EXPLICITA

Se ajustan a la siguiente estructura general:

Formato.-

$\langle tipo \rangle \quad a_1/e_1/, a_2/e_2/, \ldots, a_n/e_n/$

Siendo.-

$\langle tipo \rangle$: Uno de los siguientes identificadores:

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

$\langle a_i \rangle$: Es el nombre de una variable, de un array, de un declarador de array o de una función.

$\langle e_i \rangle$: Es opcional. Sirve para dar un valor inicial al elemento $\langle a_i \rangle$ que le precede.
Si el elemento $\langle a_i \rangle$ que le precede es:

- Una variable, entonces $\langle e_i \rangle$ debe ser una constante del mismo tipo que la variable.

(Ej.- INTEGER CONO/2/, ALTO/5/)

- Un array, $\langle e_i \rangle$ debe ser una lista del tipo:

$$/c_1, c_2, \ldots, c_n /$$

Siendo.- $\langle c_i \rangle$:Una constante del mismo tipo que el array

(Ej.- REAL NORMA(3) /3.2, 4.3, 1.5/)

También se admite que los elementos $\langle c_i \rangle$ de la lista sean de la forma:

$$j * k$$

Siendo.- $\langle j \rangle$:una constante entera, que señala el número de elementos del array que van a tener el valor $\langle k \rangle$

$\langle k \rangle$:Una constante del tipo del array

(Ej.- DOUBLE PRECISION A(7)/6 * 2.13 D-5, 10.54 D+4/)

Descripción.- Estas sentencias permiten declarar como enteras, reales, de doble precisión. complejas y lógicas a las variables , arrays y funciones que lo precisen, en la codificación del programa FORTRAN.

Ejemplo.-

REAL I, MA/5.2/, IC(3,4,5)/2..3..58*1.2/

Esta sentencia define a:

I.- Variable real

MA.- Variable real, a la que se asigna el valor inicial 5.2

IC(3,4,5).- Declarador de array, formado por 60 elementos a los que se les asigna inicialmente los valores:

IC(1,1,1).→2.

IC(2,1,1).→3

IC(3,1,1), IC(1,2,1).... IC(3,4,5).→1.2

Notar la diferencia que existe entre nombre de un array, declarador de un array y elemento de un array. En el ejemplo anterior IC es el nombre del array y. por ejemplo, IC(2,1.1) cuyo contenido es 3 es un elemento del array.

Debido a la forma de declarar un array existirá un elemento - IC(3,4,5), con contenido 1.2,que no tiene que ver nada con el declarador IC (3,4,5).

5.4.2. SENTENCIA⟨IMPLICIT⟩

IMPLICIT $\langle tipo_1 \rangle (a_1, a_2, \ldots, a_n)$, $\langle tipo_2 \rangle (b_1, b_2 \ldots b_n)$, $\ldots \langle tipo_n \rangle (z_1, z_2, \ldots, z_n)$

Siendo.-

$\langle tipo_i \rangle$: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL

$\left\{ \begin{matrix} a_i \\ b_i \\ \vdots \\ z_i \end{matrix} \right\}$: Es una letra, separada de otra letra por una coma (,) o un (-), indicando en este último caso un rango de letras.

Descripción.- Esta sentencia permite al usuario especificar el tipo que desea para cada grupo de variables del programa que empiezan por una determinada letra.

Se diferencia de las sentencias de declaración de tipo explícito en que, el explìcito sólo sirve para asignar tipos a nombres concretos, mientras que con la IMPLICIT se asignan tipos a bloques de variables que empiezan por una determinada letra que se especifica en la propia sentencia. Mientras no se indique lo contrario mediante una sentencia de tipificación explícita, todas las variables que empiezan con la misma letra tendrán el mismo tipo, que será el automático ó el declarado en la sentencia IMPLICIT.

Ejemplos.-

a) IMPLICIT REAL (I, K-N)

Todas las variables que empiecen por I, K, L, M ó N serán declaradas como reales.

b) IMPLICIT LOGICAL (N), INTEGER ($, A, X, Z)

Todas las variables que empiecen por N serán consideradas como variables lógicas; y todas las que empiecen por $, A, X ó Z como enteras.

Observaciones.-

. Los nombres simbólicos declarados en el programa antes de la sentencia IMPLICIT son del tipo que les corresponda por la declaración automática de tipo.

. Solo puede existir una sentencia IMPLICIT en cada módulo unidad (es decir una en el programa principal, y una en cada una de las subrutinas, funciones ó BLOCK DATA que forman el programa completo).

. Con independencia del lugar en que estén ubicadas las sentencias de declaración de tipo en los módulos unidad, la tipificación explícita manda sobre la IMPLICIT.

Ejemplo.-

```
IMPLICIT INTEGER (A-$)
REAL MAT,
LOGICAL S1
```

Todas las variables serán de tipo entero, con excepción de las variables MAT (real) y S1 (lógica).

. Se precisa tener mucho cuidado al utilizar la sentencia IMPLICIT, pues como asigna un tipo a todos los nombres simbólicos que empiezan por una determinada letra, puede ocurrir que, sin querer, cambiemos de tipo a funciones estandar incorporadas en las librerías del Sistema (Ej. SIN, COS...), y cuya ejecución con el nuevo tipo nos devolverá un resultado erróneo ó finalizará por error.

Por ello si se cambia el tipo de una de estas funciones es preciso desafectarlo utilizando la declaración de tipo explícito.

Ejemplo.-

```
IMPLICIT INTEGER (R-Z).
REAL SIN (en caso de que en ese módulo se utilizara la --
          función seno).
A = SIN(X)
```

5.4.3. COMPATIBILIDAD CON OTROS COMPILADORES FORTRAN

Para permitir la compatibilidad con otros compiladores -- FORTRAN, el del 9400 acepta las siguientes declaraciones de tipo, ajustándose a la siguiente ocupación de memoria:

Declaración	Tipo	Ocupación en bytes
INTEGER*2	entero	5
INTEGER*4	entero	5
REAL*4	real	5
REAL*8	real	5
COMPLEX*8	complejo	10
COMPLEX*16	complejo	10
LOGICAL*1	lógico	1(5 si aparece en un COM-

Declaración	Tipo	Ocupación en bytes
		MON o en un EQUIVALENCE)
LOGICAL*4	lógico	1(5 si aparece en un COMMON o en un EQUIVALENCE).

5.5. SENTENCIA <EQUIVALENCE>

Presenta la siguiente forma general:

Formato.-

EQUIVALENCE (nom_1, nom_2, . . . , nom_n), (nom'_1, nom'_2, . . , nom'_k), ...

Siendo.- $\langle nom_i \rangle$ y $\langle nom'_i \rangle$: El nombre de una variable
. El nombre de un elemento de un array
. El nombre de un array

Descripción.- Es una sentencia no ejecutable, que permite que - compartan memoria, dentro del mismo módulo unidad, los nombres simbólicos $\langle nom_i \rangle$ ó $\langle nom'_i \rangle$ - encerrados en un mismo par de paréntesis.
La sentencia EQUIVALENCE asigna la misma dirección de memoria a cada uno de los nombres simbólicos situados en la misma lista encerrada entre paréntesis. La equivalencia se produce en relación con el byte más significativo de la entidad especificada.
La sentencia EQUIVALENCE se utiliza a fines de optimización de memoria y también, en muchos casos, para reducir el número de sentencias de asignación; pues permite y facilita la reutilización de - una misma área de memoria con diferente estructura y tipo.

El FORTRAN 9400 precisa que si todos los nombres simbólicos que intervienen en la sentencia EQUIVALENCE fueran nombres de variables el número máximo permitido sería de 38.
El número total de nombres simbólicos que se pueden declarar en un EQUIVALENCE viene delimitado por la siguiente expresión:

$$\left[(4xNG) + (8xNV) + \sum_{I=1}^{7} ((4 + 4xI) \times NAI)\right] < 400$$

Siendo.- NG: Número de grupos, entre paréntesis, que aparecen en la sentencia EQUIVALENCE
NV: Número de nombres de variables que hay en total
NAI: Número de nombres de arrays de dimensión I.

Ejemplo.-

```
DIMENSION I1(8), I2(2, 3)
EQUIVALENCE (I1(3), I2(1, 1))
```

I1→	1	2	3	4	5	6	7	8
I2→			1, 1	2, 1	1, 2	2, 2	1, 3	2, 3

Es decir, el elemento I1(3) comparte su zona de memoria con el I2(1, 1); el I1(4) con el I2(2, 1), y así sucesivamente.

Si en el programa existiera la sentencia I1(6) = 20, ello nos permitiría manejar ese valor 20 tanto desde el array I1 (con el elemento I1(6)), como desde el array I2 (con el elemento I2(2, 2)).

5.6. SENTENCIA <COMMON>

Presenta la siguiente forma general:

Formato.-

COMMON/bloque nom_1/n_1, n_2, . . . , n_n/bloque nom_2/
n'_1, n'_2, . . . , n'_k/. . .

Siendo.- <bloque nom_i> : Es opcional. Es el nombre simbólico que define una zona común. Si no existe se asigna como zona común el : BLANK COMMON

<n_i> y <n'_i> : Es el nombre de una variable
Es el nombre de un array
Es el nombre de un declarador de array

Descripción.- Es una sentencia no ejecutable, que permite que compartan memoria nombres simbólicos definidos en diferentes módulos unidad (ej. entre el módulo principal y una subrutina).

Dentro de un mismo módulo unidad, la sentencia COMMON puede aparecer varias veces. En el caso de que esto ocurra y en dichas sentencias el <bloque nom_i> sea el mismo, los nombres simbólicos definidos con posterioridad irán ocupando zonas consecutivas de esa zona común <bloque nom_i>

Así :
```
COMMON  X, Y, Z/ET1/W
COMMON /ET1/A, B, C//Q, R, S
```

equivale a: COMMON /ET1/W, A, B, C//X, Y, Z,
Q, R, S

ó

```
COMON /ET1/W, A, B, C,
COMMON X, Y, Z, Q, R, S   ←BLANK COMMON
```

No es obligatorio que los nombres simbólicos de los diferentes COMMON, correspondientes a los distintos módulos unidad sean los mismos, basta con que las variables (ó elementos de array) situadas en posiciones homólogas ocupen en memoria zonas de la misma longitud (por ejemplo 5 bytes). La correspondencia se realiza por la posición relativa que ocupan las variables (ó elementos de array) dentro de la zona común.

Así.- Módulo principal: COMMON A(6)
Subprograma nº 1 : COMMON X, Y, Z
Subprograma nº 2: COMMON X(2, 3),

Corresponde al:

Módulo principal	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
Subprograma nº 1	X	Y	Z			
Subprograma nº 2	X(1, 1)	X(2, 1)	X(1, 2)	X(2, 2)	X(1, 3)	X(2, 3)
BLANK COMMON						

En el FORTRAN 9400 la dimensión máxima de un bloque COMMON es de 65, 536 bytes, y se permite un total de 950 nombres simbólicos.

Nota.- No se pueden colocar en una sentencia EQUIVALENCE dos nombres simbólicos que se encuentren en COMMON. Si uno de los nombres simbólicos que aparecen en una sentencia EQUIVALENCE está en COMMON, todos los asociados a él estarán en ese COMMON.

5.7. SENTENCIA <DATA>

La forma general de esta sentencia es:

Formato.-
DATA nom1, nom2, ... , nomn/r * const1, ... , r* constn/,
nom'1, nom'2, ... , nom' k/ r' * const'1, ..., r' * const'k/,

Siendo.- <nomi> y <nom'i> : . El nombre de una variable
. El nombre del elemento de un array.
. Un DO implicito (se describe en la sección 6.2).

<consti> ó<const'i>: Es una constante entera, real, de doble precisión, compleja, hexadecimal, lógica o una -- constante literal.

<r*> ó <r'*> : Es un contador de repetición y su uso es opcional. Precisa el el número de veces que se ha de repetir una constante <consti> ó <const'i> . Debe ser una constante entera sin signo.

(Ej: DATA A, B, C.D / 4* 5.2/ equivale a:
DATA A, B, C, D/5.2, 5.2, 5.2, 5.2/

Descripción.- La sentencia Data es una instrucción no ejecutable y tiene como función inicializar, en tiempo de compilación, las zonas de memoria especificadas en las listas de variables con los valores indicados en las listas de constantes. Las constantes se deben co-- rresponder en número, orden y tipo con las varia-- bles de la lista. Cuando en la lista de variables --

aparezca un nombre de array, habrá que indicar - tantas constantes como elementos compongan el - array, estas constantes se deberán corresponder con los elementos del array que resultan de hacer - variar más rápidamente los primeros subíndices - (orden natural). El tipo de constante debe ser -- - acorde con el de la variable, pues esta sentencia no efectúa ningún tipo de conversión. Así por ejemplo:

DATA A/3/

inicializará la variable A con el valor 3 como constante entera, sin efectuar ninguna conversión a real. Cuando posteriormente se referencie la variable - real A su valor no será 3, si no el valor real a que corresponda la representación del número 3 en formato entero, pues se considera que tiene estructura real.

Ninguno de los items de la lista de variables puede estar en un BLANK COMMON. Cuando la sentencia DATA aparece en un subprograma "Block Data" - puede inicializar valores de zonas de memoria situadas en un bloque común con etiqueta.

La sentencia DATA no puede asignar valores a argumentos ficticios (ver Capítulo 7) de un subprograma, ya que en tiempo de compilación no se conocen sus direcciones pues éstas pertenecen a la zona de datos del programa referenciante. La utilización de la sentencia DATA es aconsejable en dos casos concretos:

a) Cuando en un programa se desee inicializar -- unas determinadas zonas de memoria siempre con los mismos valores.

b) Cuando se utilicen tablas con valores fijos a lo largo del programa.

En estos dos casos, las ventajas de su utilización, con respecto a las instrucciones de asignación o lectura, son claras pues, al no generar código ejecutable, el número de instrucciones es menor y por consiguiente también es menor el tiempo de ejecución del programa.

Ejemplos.-

a)
```
DOUBLE PRECISION DP
DIMENSION I(5), A(3,2)
DATA I, A(2,1)/1,2,3,4,5,7.4/, DP/ 7.43 D* 2/
```

b)
```
DIMENSION I1 (4)
DATA I1, A, B/3*2, 5, 4.3, 2.1/
```

Resultando las variables inicializadas así:

I1(1)	I1(2)	I1(3)	I1(4)	A	B
2	2	2	5	4.3	2.1

Nota.- I1, A y B no tienen porqué estar en posiciones contiguas.

5.8. SENTENCIA <EXTERNAL>

Tiene la siguiente forma:

Formato.-

EXTERNAL $a_1, a_2, \ldots, a_n$

Siendo.- $\langle a_i \rangle$.- Nombre simbólico de una función o subrutina.

Descripción.- Cuando en la llamada a una función o subrutina figura como argumento:

- El nombre de una función estandar (ej: SIN, .)
- El nombre de una función de usuario FORTRAN
- El nombre de una subrutina FORTRAN.

es necesario que estos nombres de funciones y subrutinas se declaren previamente. Para ello se emplea la sentencia EXTERNAL. En otras palabras, en la sentencia External deben figurar los nombres de las funciones o subrutinas usadas como argumentos actuales de un procedimiento externo.

Ejemplo.-

Un programa FORTRAN está formado por 3 módulos unidad (es decir que se compilan independientemente): MODULO PRINCIPAL, SUBRUTINA A y FUNCION B.

Si se diseña una estructura de programa de la siguiente forma:

Módulo Principal	SUBRUTINA A	FUNCION B
	SUBROUTINE SUM(RR)	FUNCTION RESUL (A)
CALL SUM (RESUL (Y))	X = RESUL (Y)	
	RETURN	RETURN
	END	END

Se observa que, al ser compilaciones independientes, el compilador FORTRAN detecta:

- Que RESUL es una función cuando compila la SUBROUTINA A, pues existe en este módulo la sentencia X = RESUL (Y).
- Que SUM es una subrutina, cuando compila el MODULO Principal, por existir en su módulo una CALL SUM. Pero no sabe el tipo de RESUL (Y), pues no existe ninguna acción que se lo indique.

Ello se obvia colocando, en el módulo principal, la sentencia:

EXTERNAL RESUL

CAPITULO 6
ENTRADAS/SALIDAS

6.0. INTRODUCCION

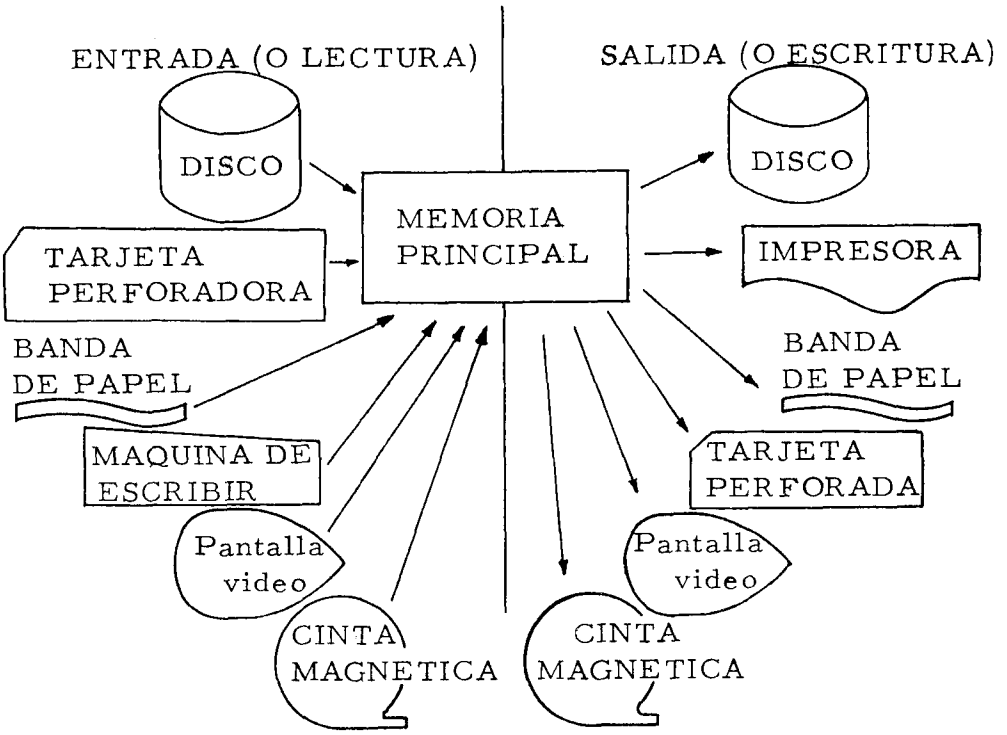
La flexibilidad de los ordenadores se basa en el hecho de que una vez confeccionado un programa, éste se puede utilizar tantas veces como sea preciso para resolver el mismo problema con datos diferentes. Esto se consigue combinando el concepto de variable con las instrucciones de entrada/salida.

Como se ha definido en el Capítulo1, una variable es la representación simbólica de una dirección de memoria con cuyo contenido se efectúan las operaciones aritmético-lógicas. En un momento determinado, el contenido de una dirección es función del resultado de una sentencia de asignación, ó del valor almacenado mediante una sentencia DATA o similar, ó del almacenado por una sentencia de entrada (denominada también de lectura).

En la ejecución de un programa, una vez obtenidos los datos necesarios para el tratamiento y efectuado éste, se necesita conocer de alguna forma el resultado de dicho tratamiento, para ello se utilizan las sentencias de salida (también conocidas como de escritura).

Las sentencias de entrada/salida (E/S) sirven, pues, para establecer la comunicación entre el exterior y la memoria principal del ordenador. La función de estas sentencias, con respecto al ordenador, es semejante a la desarrollada por las personas al leer o escribir, así cuando una persona lee, almacena en su memoria una información, y cuando escribe plasma sobre un soporte información que estaba en su mente.

De la misma forma que las personas pueden captar y emitir información utilizando diferentes dispositivos físicos, así los ordenadores pueden tratar a través de sus unidades de E/S, la informaciòn contenida en distintos soportes, lo que gráficamente se puede representar:



La función de las sentencias de E/S no consiste unicamente en transferir datos entre el exterior y la memoria central, sino que, debido a las diferentes formas de representación de la información (interna y externa) deben también efectuar la conversión apropiada, para lo cual utilizan las especificaciones de formato. Estas especificaciones pueden omitirse bien porque se - tomen las estandar o por no ser necesarias debido a que la forma de representación externa sea la misma que la interna (es el caso de la representación binaria).

De lo hasta ahora dicho, se desprende que en una operación de E/S se debe suministrar la siguiente información:

1) Sentido de la operación (entrada ó salida)

2) El tipo de soporte en que se ubica la información, precisándose para ello la unidad lógica asignada al dispositivo físico por el que se va a efectuar la transmisión.

3) Zonas de memoria implicadas en la transmisión y orden en que ésta debe efectuarse. Normalmente serán nombres de variables y entonces se les denomina lista de E/S.

4) Especificaciones de formato, en las que se indican la colocación física de los datos sobre el dispositivo externo y el tipo de conversión que se desea.

No siempre es necesario indicar de forma explícita la información correspondiente a los puntos 3 y 4; como veremos posteriormente existen instrucciones de E/S sin lista o con lista vacía y sin especificaciones de formato.

La información relativa a los tres primeros puntos se indica en la propia instrucción de E/S y la correspondiente al cuarto punto la proporciona la sentencia FORMAT. Así por ejemplo, para almacenar dos datos, que esten grabados en tarjeta perforada, en las posiciones de memoria que tengan asignadas los nombres de variables IA e IB, se podríancodificar las siguientes sentencias:

```
      READ (5, 1) IA, IB
1     FORMAT (I2, I3)
```

Tarjeta perforada

En donde se especifica que:

a) Es una operación de entrada (READ)

b) La operación se efectuará por la unidad lógica 5. En la mayoría de las instalaciones está asociada a la lectora de tarjetas.

c) El contenido de la tarjeta se almacenará en las variables IA e IB y en este orden.

d) El 1 que aparece en la sentencia READ debe ser, como de hecho lo es, la etiqueta de una sentencia FORMAT que contenga las especificaciones de formato a tener en cuenta en esta transferencia. Estas especificaciones son I2, I3, e indican que para ambos datos se desea una conversión entera (código I), que el primer dato lo forman las dos primeras posiciones y el segundo las tres siguientes.

Dependiendo del tipo de acceso que se haga en el soporte exterior las operaciones de E/S se clasifican en dos grandes grupos: de -- acceso secuencial, para acceder a una información determinada hay que, de alguna forma, tratar toda la que se encuentra físicamente delante de ella, y de acceso directo, a una determinada información se puede acceder directamente sin tener en cuenta la que le precede, este tipo de acceso sólo se podrá efectuar en aquellos dispositivos que permitan direccionamientos directos, por ejemplo tambores ó discos magnéticos.

6.1. INSTRUCCIONES DE ENTRADA/SALIDA CON FORMATO

Se pueden utilizar las siguientes formas:

```
READ (n, l1) lista
READ (n, l1, EOF = l2) lista
READ (n, l1, END = l2) lista
READ (n, l1, END = l2, ERR=l3) lista
WRITE(n, l1) lista

READ l1,lista
PUNCH l1,lista      Se mantienen por compatibilidad con
PRINT l1,lista      el FORTRAN -II
```

Donde:

- READ, WRITE, PUNCH y PRINT: Son los códigos de instrucción, e indican el sentido de la operación de entrada ó salida. En las tres últimas sentencias los códigos READ, PUNCH y PRINT, también indican respectivamente, que la operación se debe efectuar por el dispositivo estandar de lectura, perforación o impresión.

- <n>: Constante o variable entera no subindicada que indica el dispositivo por el que se efectuará la operación de E/S, su valor debe estar comprendido entre 0 y 41. Normalmente en todas las instalaciones se asigna un 5 para la lectora de tarjetas y un 6 para la impresora, dejando los otros números para asignar otras unidades lógicas o dispositivos físicos.

- $\langle l_1 \rangle$ puede ser:
  . Una etiqueta de una sentencia FORMAT: Constante entera sin signo o variable entera a la que se le ha asociado una etiqueta mediante la sentencia ASSIGN.

. Nombre de un "namelist". Se describe en la Sección 6.6.

. Nombre de un "array" que deberá contener las especificaciones del formato. Se describe en la Sección 6.5. con el título de "formato variable".

- $\langle l_2 \rangle$ y $\langle l_3 \rangle$: Etiquetas de instrucciones ejecutables.

- EOF ó END = l_2 : Indica que si durante la ejecución de una sentencia de lectura, se encuentra una marca de fin fichero, el control se transferirá a la instrucción con la etiqueta $\langle l_2 \rangle$

- ERR = l_3: Si en la ejecución de una sentencia de lectura se produce un error, el control se transferirá a la instrucción con la etiqueta $\langle l_3 \rangle$.

Las cláusulas END y ERR son opcionales y entre ellas pueden aparecer en cualquier orden. Si al ejecutar una orden de lectura, en la que no se ha especificado la cláusula END, aparece una marca fin de fichero, el programa termina por error, lo mismo ocurre si no existe la cláusula ERR y aparece un error en los datos. Si se pone la cláusula ERR el control se transfiere a la sentencia que lleva la etiqueta $\langle l_3 \rangle$ y no se almacena ningún dato en las variables de la lista asociada a esa sentencia de lectura.

Nota.- La dimensión máxima de los registros con formato en el FORTRAN 9400 es de 255 caracteres, aunque el soporte físico puede establecer otras restricciones como, por ejemplo, que el número máximo de caracteres de una tarjeta no podrá ser superior a 80 y el de una línea de impresión normalmente esta limitado a 132 .

- Lista: Conjunto ordenado de items, separados por comas, que especifican las zonas de memoria involucradas en la operación de entrada o salida.

6.2. FORMAS QUE PUEDE TENER UNA LISTA DE E/S

Una lista de E/S puede estar formada:

- Por uno o más items separados por comas, estos items pueden ser: variables, elementos de arrays o nombres de arrays.

- Por una lista entre paréntesis.

- Por una lista entre paréntesis, incluyendo un DO implícito.

- La lista puede estar vacía, en cuyo caso la información a transmitir se obtendrá de (WRITE) ó almacenará en (READ) la zona de memoria ocupada por las especificaciones de formato. Normalmente sólo se utiliza en operaciones de escritura para imprimir cabeceras. Como caso especial se puede considerar el NAMELIST.

  Los valores transmitidos se corresponden a los items de la lista y precisamente en el mismo orden de aparición, es decir, el primer valor transmitido será el asociado al primer item de la lista y, si este es una variable simple, el segundo valor será el asociado al segundo item de la lista y así sucesivamente hasta el final. Si un item de la lista es una variable o un elemento de un array se transmitirá una cantidad unitaria, por el contrario cuando un elemento de una lista es el nombre de un array se transmitirán tantas cantidades como elementos tenga el array completo, correspondiéndose con los elementos del array que se obtendrían al

hacer variar más rápidamente los subíndices de la izquierda (orden secuencial de los elementos de los arrays).

Con independencia de las especificaciones de formato, una instrucción de E/S o termina por error o no termina hasta que se hayan transmitido todos los valores correspondientes a los items de la lista, en este sentido se puede decir que la lista manda sobre el formato y siempre se transmiten tantos items de información como en ella se indiquen.

Un "DO implícito" tiene la siguiente forma:

(lista, d)

donde ⟨d⟩ es una especificación de la forma:

i = par1, par2, par3

ó

i = par 1, par 2

siendo ⟨par 1⟩, ⟨par 2⟩ y ⟨par 3⟩ los parámetros del DO e ⟨i⟩ su variable de control.

La función del DO implícito es similar a la de un DO normal ó DO explícito (ver Sección 4.6), es decir, repite la lista comprendida en su rango tantas veces como indican los parámetros. El rango del DO implícito se delimita con paréntesis, desapareciendo por tanto la etiqueta del DO explícito.

En un DO implícito lo que se repite es la lista dentro de la misma instrucción de E/S, a diferencia de cuando la instrucción está contenida en el rango de un DO explìcito en que se repite la instrucción completa. En lo referente al contenido y orden de las variables a transmitir no hay diferencia entre utilizar una forma u otra, la diferencia estriba en que cada vez que se inicia la ejecución de una operación de E/S

la unidad correspondiente se posiciona al principio del registro, hecho que obligatoriamente se produce cuando, para transmitir una serie de variables, se repite la instrucción completa en lugar de repetir unicamente la lista, o parte de ella, mediante un DO implícito.

Los DO's implícitos pueden estar anidados hasta un máximo de 7 niveles.

Para optimizar el tiempo de ejecución de un programa se aconseja efectuar el menor número posible de operaciones de E/S por seroperaciones lentas. Conviene, por tanto, -- que las listas de E/S sean lo más completas posibles, para lo cual es de gran ayuda la utilización de los DO's implícitos.

Ejemplos de listas de E/S.-

Notas: Se supone que las sentencias que aparecen en los - siguientes ejemplos pertenecen a programas independientes y que los arrays están correctamente declarados.

a) READ (5, 1) J

b) READ (5, 1) B1, B2, C(3, 2), D(I, J)

Las variables I, J deberán tener asignado un valor apropiado en el momento de la ejecución de esta sentencia.

c) READ(5, 1) I, J, A(I, J)

Esta sentencia leerá desde tarjeta o imagen de - tarjeta (unidad 5) tres valores. Los dos primeros son los subíndices que indican el elemento del -- array A en el que se almacenará el tercer valor.

d) READ (5, 1) (I, J, (A(I, J)))

Se permite la redundancia de paréntesis, funcionalmente esta sentencia es idéntica a la del ejemplo anterior.

e) WRITE (6 , 1) B,(D, A(3, 4), I = 1, 2 Ø)

Imprimirá (unidad 6) el contenido de la variable B y 20 veces el correspondiente al par D y A(3, 4), en el que D es una variable y A(3, 4) un elemento de un array.

f) READ (5, 1) A

Suponemos que A está declarado como un array de (15, 15) (se puede considerar que es una matriz de 15 filas y 15 columnas).

Esta sentencia leería la matriz completa por columnas, o sea un total de 225 valores, el primero se - almacenaría en la posición A(1, 1), el segundo en la posición A(2, 1), el tercero en la posición A(3, 1)..., el décimoquinto en la posición A(15, 1), el decimosexto en la posición A(1, 2) y así sucesivamente hasta que se transmitieran todos.

g) READ (5, 1) ((A(I, J), I = 1, 15), J= 1, 15)

Equivalente al ejemplo anterior, pero utilizando DO implícito.

h) READ (5, 1) ((A(I, J), J = 1, 15), I = 1, 15)

Leería la matriz completa por filas, o sea los 15 primeros valores se almacenarían en la primera fila, los 15 siguientes en la segunda fila, ... , etc.

i) READ (5, 1) (A(I, I), I = 1, 15)

Leería 15 datos y los almacenaría en la diagonal principal de la matriz A.

j) READ (5, 1) ((A(I, J), I = 1, 15, 2) , J = 1, 15)

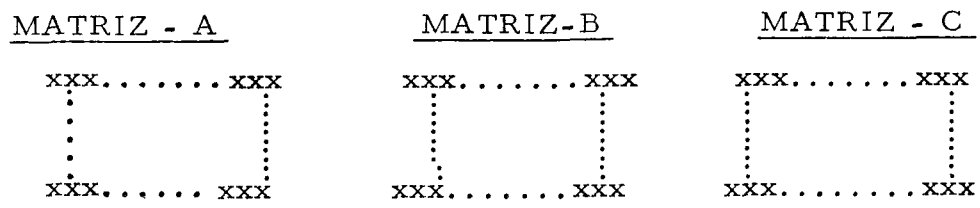
Almecenaría los valores leídos en las filas impares variando por columnas. O sea el primer valor lo almacenaría en A(1, 1),el segundo en A(3, 1), el tercero en A(5, 1), . . . etc.

k) READ (5, 1) L, K, ((A(I, J), J=1, L), I= 1, K)

Leería por filas una submatriz de A, cuyos límites se dan en los dos primeros datos de la tarjeta.

l) Este ejemplo, destaca la ventaja de utilizar el DO - implícito dentro de las sentencias de E/S.

"En un programa que calcula la suma de las matrices A y B y deja el resultado en la matriz C, todas ellas de 15 por 15 se desea listar las matrices A, B y C para comprobar los resultados. Se pide que las matrices se listen por filas, incluyendo en cada línea de impresión la fila correspondiente a la matriz A y a continuación las mismas filas de las matrices B y C; o sea un listado de la forma:



Utilizando el DO implícito la impresión se podría reducir a la sentencia:

```
      WRITE (6, 1) ((A(I, J), J = 1, 15), (B(I, K), K = 1, 15),
     *(C(I, L), L = 1, 15), I=1, 15)
```

Por el contrario, para poder obtener el mismo resultado, sin utilizar el DO - implicito, se tendrían que al menos escribir las siguientes sentencias:

```
      DO 2 I = 1, 15
      WRITE (6, 1) A(I, 1), A(I, 2), A(I, 3)....., A(I, 15), B(I, 1)...,
     *B (I, 15), C(I, 1), C(I, 2), ....., C(I, 15)
2     CONTINUE
```

6.3. OPERACION DE RELECTURA

Formato.- READ (i, etiqueta) lista

Siendo.-<i>:Una constante o variable entera de valor 29

<etiqueta> La etiqueta de un formato.

Descripción.- Permite volver a leer, con el mismo o diferente FORMAT, el último registro leído. Se utiliza para que el programa pueda determinar la clase de información contenida en el registro.

Esta orden ni selecciona, ni inicia acción alguna sobre un dispositivo periférico. En ella no se permite la cláusula ERR.

La relectura no puede seguir, lógicamente, a una sentencia WRITE, BACKSPACE, REWIND ó END FILE del mismo dispositivo.

6.4. SENTENCIA <FORMAT>

Proporciona información de conversión y/o edición entre la representación interna y los items correspondientes a una sentencia de E/S.

La sentencia FORMAT debe tener una etiqueta. Esta etiqueta puede ser referenciada desde más de una sentencia de E/S con formato.

Formato.- FORMAT $(g_1 t_1 z_1 t_2 z_2 \ldots\ldots t_n z_n g_2)$

Siendo cada.-

$\langle g_i \rangle$: Un grupo de uno o más slashs /. Es opcional

$\langle t_i \rangle$: Un descriptor o grupo de descriptores de campos, como se detallan a continuación.

$\langle z_i \rangle$: Un separador de campos, consiste en uno o varios slashs o una coma.

6.4.1. DESCRIPTORES DE CAMPOS (t)

Entre los descriptores de campos cabe distinguir dos tipos, los de conversión y los de edición. Los primeros deben ir asociados a un item de la lista de E/S, y deben ser acordes con el tipo de item. Los segundos no llevan asociado ningún item de la lista y normalmente se utilizan

para: saltar campos o registros, escribir literales, posicionarse en un punto determinado del registro exterior, etc.

Los descriptores de campos son:

rIw
xPrEw.d
xPrFw.d
xPrDw.d
xPrGw.d
rLw
wH h_1 h_2.....h_n
'h_1 h_2.....h_n
rAw
wX
Tq
rZw

Donde:

- Las letras I, E, F, D, G, L, H, A, X, T, Z indican el tipo de conversión o la función de edición a ejecutar en la operación de E/S.
- ⟨w⟩ : Es un entero sin signo mayor que cero, y menor que 256, que indica el número de posiciones caracter del campo en el soporte externo.
- ⟨d⟩ : Es una constante entera sin signo que indica el el número de dígitos de la parte fraccionaria, en el soporte exterior, cuando se tratan valores reales o en doble precisión.
- ⟨r⟩: Es un entero sin signo distinto de cero y menor que 256 que indica el número de veces que se repite el descriptor básico que le sigue a continuación. Si se omite se toma 1.

- <x P>:Es opcional, siendo x un entero sin signo que representa el factor de escala.
- $<h_i>$: Representa un caracter del juego de caracteres.
- <q> : Constante entera sin signo comprendida entre 1 y 255 que indica una posición en un registro exterior (posicionamiento del tabulador).

Con los descriptores que efectúan conversiones numéricas (I, E, F, D y G) se cumplen las siguientes normas generales:

- En entrada, los blancos se toman como ceros. el signo + es opcional y el - obligatorio.
- En salida los valores se ajustan a la derecha del campo rellenando las posiciones sobrantes de la izquierda, si las hay, con blancos, (esta regla se cumple para todos los códigos de conversión). Cuando el valor del item de la lista no puede expresarse en las <w>posiciones del campo, éste se rellena a asteríscos. Cuando un valor es positivo el signo + no se imprime, cuando es negativo el signo - apareçe delante de la primera cifra significativa.

Se pueden utilizar paréntesis para agrupar descriptores de campos y, estos paréntesis, pueden ir precedidos por una constante ≤ 255 que indica el número de veces que se repiten esos descriptores.

A continuación se describen cada uno de los descriptores:

a) Descriptor entero (rIw).-

El item asociado de la lista de E/S, debe ser entero.

En entrada (READ).-

El campo debe ser una constante entera. Si el valor excede del rango de magnitud de las constantes enteras, sólo se almacenan los 9 dígitos menos significativos con el signo que tuviera.
Se imprime un diagnóstico en ejecuciones continuadas.

Ejemplo.-

	READ(5, 1)IA	Dato	Contenido de IA
1	FORMAT (I12)	123456789099	IA=456798099

En salida (WRITE).-

El valor del item aparecerá como una constante entera ajustada a la derecha del campo, precedido por el signo - si es negativo y por un blanco, si la longitud del campo lo permite, si es positiva.

Ejemplo.-

	WRITE(6, 1)IA	Resultado
1	FORMAT (1X, I12)	bbb456789099

b) Descriptor real - Conversión E (xPr Ew.d)

El item asociado de la lista de E/S, debe ser real en simple precisión.

En entrada

El campo externo consiste en una cadena de dígitos precedidos por blancos o ceros que - opcionalmente puede contener un punto en - cuyo caso anula a la especificación⟨d⟩
La cadena de dígitos puede ir seguida de una notación exponencial, una E ó D seguida por una constante entera opcionalmente con signo

Ejemplo. -

```
      READ (5, 1) (A(I), I = 1, 3)
1     FORMAT (2 E7.2, E8.1)
```

Datos

```
3241.28 b324129 43.45E+2
```

Contenido de A

A(1) = 3241.28; A(2) = 3241.29
A(3) = 4345.00

En salida. -

El campo externo tiene el siguiente formato (ajustado a la derecha):

```
b 0. n1 n2 ..... nd ± YYY
```

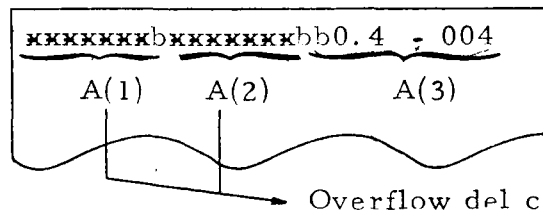
Donde: $(n_1 \ldots\ldots n_d)$: Son las ⟨d⟩ cifras significativas, la última de las cuales está redondeada
⟨YYY⟩: Tres digitos con el valor del exponente
Restricción. - $(w - d) \geq 7$ ya que además de los dígitos ⟨ni⟩ , existen 7 caracteres fijos, que son b0. - YYY

Ejemplo. -

Imprimir el contenido de A del ejemplo de la entrada:

```
      WRITE (6, 1) (A(I), I = 1, 3)
1     FORMAT (1X, E7.2, 1X, E7.2, 1X, E8.1)
```

Resultado



c) Descriptor real - Conversión F (xPrFwd)

Se utiliza para transmitir items reales sin formato exponencial.

En entrada .- Igual que para el descriptor real - Conversión E.

En salida .- El campo externo tiene el siguiente formato (ajustado a la derecha):

b n_1 $n_2 \ldots n_a$. $m_1 m_2 \ldots m_d$

Siendo.- $< n_i >$: Dígitos de la parte entera

$< m_i >$: Dígitos de la parte decimal

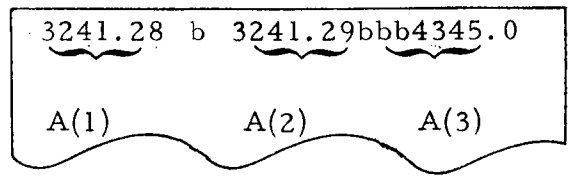
El blanco de cabecera (b) está previsto para un signo menos, no obstante si se sabe que todos los valores son positivos, no es necesario este blanco.

Ejemplo.-

Si imprimimos el contenido de A del ejemplo del apartado b):

```
      WRITE (6, 1) (A(I), I = 1, 3)
1     FORMAT (1X, F7.2, 1X, F7.2, 1X, F8.1)
```

Resultado

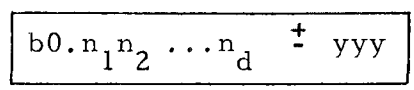


d) Descriptor de doble precisión-Conversión D(xPrDw.d)

Se utiliza para items de la lista de E/S definidos en doble precisión.

En entrada.- Igual que para el descriptor real-Conversión E

En salida.- El campo externo se ajusta al siguiente formato:



con el mismo significado y con las mismas restricciones que para el descriptor real-Conversión E.

Ejemplo de entrada.-

```
      READ (5, 1)  (A(I), I = 1, 2)
1     FORMAT (2  D7.2)
```

Dato

```
1234567123.4D1
```

Contenido de A

A(1) = 12345.67

A(2) = 1234.Ø

e) Descriptor G (xPrGw.d)

Es un formato generalizado que se puede utilizar para datos reales, en doble precisión, enteros y lógicos.

a) Para datos reales, en entrada este descriptor - acepta cualquier dato con o sin exponente y en salida utiliza el formato F si el ancho del campo lo permite; sino utiliza el E.

b) Para doble precisión.- Se convierte en el descriptor D.

c) Para enteros y lógicos.- Se interpreta como I ó L respectivamente.

Los campos‹d› y ‹xP› se ignoran en el caso c), y por lo tanto pueden omitirse.

f) Descriptor lógico (rLw)

Permite la entrada/salida de valores lógicos.

En entrada.- El primer caracter no blanco debe ser una T ó F, el resto de los caracteres del campo se ignoran.

En salida.- Coloca una T o una F, ajustada a la derecha con blancos a la izquierda si la longitud del campo > 1.

Ejemplo.-

Sea BOOL una variable lógica con valor .TRUE.

```
      WRITE (6, 1)  BOOL
1     FORMAT (1X, L3)
```

Resultado

```
 bbT
```

g) Descriptor Hollerith (wH)

No se asocia ningún item de la lista con este tipo de descriptor.

En entrada.- Los w caracteres transferidos desde el dispositivo externo reemplazan a los datos actuales Hollerith especificados en la instrucción FORMAT.

Ejemplo .-

```
      READ (5, 1)
1     FORMAT (13 HbbMATEMA-
                TICAS)
```

Datos

```
 bbFISICA
```

. Contenido del FORMAT después de la lectura:

(13 HbbFISICAbbbbb)

En salida.- Los actuales datos Hollerith contenidos en la instrucción FORMAT son transferidos al dispositivo externo.

Ejemplo

```
      WRITE (6, 1)
1     FORMAT (1X, 5HTOTAL)
```

Resultado

```
 TOTAL
```

h) Descriptor literal ('h1 h2 ..... hn')

Su función es la misma que la del descriptor Hollerith, y por ello no se debe asociar ningún item de la lista de E/S con este descriptor. La tira de caracteres que va entre apóstrofos debe ser $\leq$ 255 caracteres.

En entrada y salida.- Para indicar el caracter apóstrofo (') dentro del literal hay que colocar dos apóstrofos consecutivos ('). Se debe tener en cuenta que todos los apóstrofos internos causan lectura de caracter. Cada apóstrofo en el campo de entrada es tratado como un caracter. Todos los caracteres (incluído blancos) entre los apóstrofos de apertura y cierre se imprimen como datos de salida.

Ejemplo :

```
      READ(5, 1)                 |  Datos
1     FORMAT ('DON''T')          |  ABCDEFG
```

Contenido del FORMATO después de la lectura:

('ABCDEF')

```
      WRITE (6, 2)               |  Resultado
2     FORMAT (1X, 'DON''T')      |  DON'T
```

i) Descriptor Alfanumérico (Aw).-

El descriptor de campo Aw lee o escribe w caracteres Hollerith en o desde un elemento de la lista de E/S. Supongamos que m es el número máximo de caracteres -- que pueden almacenarse en la variable asociada de la lista (normalmente m valdrá 1, 5 ó 10).

En entrada.- Si el descriptor w es menor o igual que m, el dato se almacena en memoria ajustado a la izquierda con blancos a la derecha.

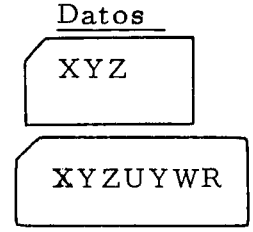
Si el descriptor w es mayor que m, se transfieren a memoria los m caracteres más a la derecha, ignorándose el resto, es decir, los más significativos.

Ejemplos.-

```
a)        READ (5, 1) A
      1   FORMAT (A3)

b)        READ (5, 1) A
      1   FORMAT (A7)
```

Datos



Contenido de A en Memoria

a)	X	Y	Z	b	b
b)	Z	U	V	W	R

En salida.- Si w es menor o igual que m, se transfieren desde memoria los w caracteres más a la izquierda.

Si w > m se transfieren los m caracteres ajustándolos a la derecha del campo y rellenando con blancos a la izquierda.

Ejemplo.-

Supongamos que A contiene: ZUVWR

```
      WRITE (6, 1) A
1     FORMAT (1X, A3)
```

Resultado

```
ZUV
```

```
      WRITE (6, 2) A
2     FORMAT (1X, A7)
```

```
bbZUVWR
```

j) Descriptor de blancos (wX)

Al igual que los descritores Hollerith y literal, este descriptor tampoco lleva asociado un item de la lista de E/S.

En Entrada.- Ignora los w caracteres consecutivos del campo de entrada.

Ejemplo

```
READ (5, 1)    IA
1 FORMAT (3X, I3)
```

Dato

```
1 2 3 4 5 6
```

Contenido de IA: 456

En salida.- Inserta w espacios en el registro de salida.

Ejemplo.- Sea la variable IA con contenido 456

```
      WRITE (6, 1) IA
1     FORMAT (3X, I3)
```

Resultado: bb456

(ver "control de carro". Sección 6.4.5)

k) Descriptor de posición del registro (Tq)

Especifica la posición. en el registro FORTRAN. donde debe empezar la transferencia de datos.

La entrada y salida se puede comenzar en cualquier caracter del registro (ó línea) con el empleo del descriptor Tq. El valor q representa la posición de comienzo ($q \leq 255$).

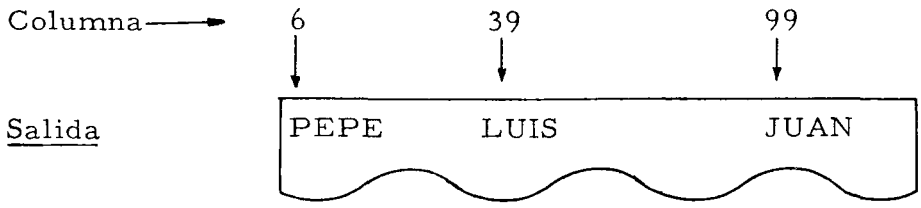
Este descriptor permite romper el orden de transferencia establecido en la lista de E/S.

Ejemplo .-

a)
```
      READ (5, 2) Y, B
2     FORMAT (T20, F10.3, T1, F5.1)
```

Entrada	B	Y
	XXX.X	XXXXXX.XXX
Columna →	1 5	20

b)
```
      WRITE (6, 10)
10 FORMAT (T7, 4HPEPE, T100, 4HJUAN, T40, 4HLUIS)
```



Nota.- Para registros de impresión, la posición especificada por T se imprime en la q-1 porque el primer caracter de un registro de impresión se interpreta como el caracter de control, que no se imprime.

1) Descriptor hexadecimal (rZw)

Un byte contiene dos dígitos hexadecimales. El descriptor hexadecimal (lo mismo que el alfanumérico Aw) se puede asociar con cualquier tipo de variable.

De acuerdo con lo especificado en el Capítulo 1, las variables pueden tener el siguiente número de dígitos hexadecimales:

Enteras: 10 dígitos hexadecimales.
Reales: 10 dígitos hexadecimales.
D.P.: 20 dígitos hexadecimales.
Complejas: 20 dígitos hexadecimales
Lógicas: 2 ó 10 (esto último si la variable está declarada en un COMMON ó EQUIVALENCE).

En entrada.- Los caracteres deben ser dígitos hexadecimales y se almacenan 2 por byte, ajustados a la derecha con Ø a la izquierda.

En salida.- El valor hexadecimal se imprime precedido de blancos.

Ejemplo.-

a)
```
      READ (5, 1) (A (I), I = 1, 2)
    1 FORMAT (2Z5)
```

Datos

```
01ABCDEF99
```

Contenido de A

0 0	0 0	0 0	1 A	B C	(A(1))
0 0	0 0	0 D	E F	9 9	(A(2))

---

b)
```
      WRITE (6, 1)  (A(I)), I = 1, 2)
    1 FORMAT (1X, 2Z10)
```

Resultado

```
bbbbbb1ABCbbbbbDEF99
```

A(1) A(2)

6.4.2. AGRUPAMIENTOS DE DESCRIPTORES DE CAMPOS

Para agrupar los descriptores de campos se utilizan -- paréntesis.

El paréntesis de la izquierda puede ir precedido por una constante entera sin signo (máximo 255) que señala el número de veces que los descriptores, incluídos entre paréntesis, se han de repetir. Cuando esta constante no se indica, su valor implícito es 1.

Ejemplo

```
      READ (5, 1) (A(I), I = 1, 5)
1     FORMAT (F7.3, 2 (F7.1, F7.Ø))
```

Contenido de A
A(1) = 1234.567
A(2) = 12345.6
A(3) = 1234567.
A(4) = 222.567
A(5) = 33333.7

Datos

```
123456712345.61234567222.56733333.7
```

Nota.- El FORMAT del ejemplo anterior equivale a un FORMAT (F7.3, F7.1, F7.0, F7.1, F7.0)

. Interesa señalar que el punto (.) contenido en el campo de los datos, manda sobre el descriptor del campo (véase para A(4) y A(5)).

. Si no existe punto (.) en el campo de los datos, la parte decimal se toma de acuerdo con el descriptor del campo (véase para A(1)).

. El nivel máximo de paréntesis que se acepta es el de segundo orden, considerando a los paréntesis externos de orden cero.

Ejemplo

a) Válido

```
FORMAT (F7.3, 2(F7.1, 3(F7.4, I5)))
        ↑      ↑      ↑
        0      1      2
```

b) No válido

FORMAT (F7.3, 2(F7.1, 3(F7.4, 5(I5))))

6.4.3.- REEXPLORACION DEL FORMATO

Como la lista siempre se satisface, cuando en una operación de E/S el número de items de la lista es superior al número de descriptores en las especificaciones de formato (al menos debe existir un código de conversión), los códigos de conversión a utilizar son los que se encuentren en el primer grupo completo (descriptores entre paréntesis) que se encuentre efectuando una reexploración del formato de derecha a izquierda. Siempre que se produce una reexploración el dispositivo exterior se posiciona al principio del registro siguiente.

Ejemplo.-

```
READ (5, 1) (A(I), I = 1, 10)
1 FORMAT (F7.3, 2(F7.1, F7.Ø))
```

Asigna los descriptores:

F7.3 → A(1)

F7.1 → A(2), A(4), A(6), A(8), A(10)

F7.0 → A(3), A(5), A(7), A(9)

6.4.4. ESPECIFICACION <FORMAT> PARA REGISTROS MULPLES

La barra oblicua o slash es un delimitador de registro y un separador de campo. Si en una especificación de formato aparece un / el resto del registro que se está transmitiendo se ignora en la entrada o se rellena con espacios en la salida. Los códigos de edición que siguen a un / se utilizan para editar el próximo registro. El último paréntesis ')' del FORMAT también es un delimitador si quedan elementos de la lista E/S por registrar. Si hay n slashs, al comienzo o al final de las especificaciones de formato, se saltarán n registros; en cualquier otra parte se termina el proceso del registro actual y se saltan n-1 registros.

Ejemplos.-

a)

```
      READ (5, 1) (L(I), I = 1,2)
    1 FORMAT (I3/I4)
```

Datos

```
123456789
897654321
```

Contenido de L

L(1) = 123

L(2) = 8976

```
      WRITE (6,2) (L(I), I = 1, 2)
    2 FORMAT (1X, I3/1X, I4)
```

Resultado

```
123
8976
```

b)

```
      READ (5, 1) (L(I), I = 1,2)
    1 FORMAT (///  I3, I4)
```

Datos

3 tarjetas

```
123456789
```

Contenido de L

L(1) = 123

L(2) = 4567

Nota.- Salta tres tarjetas (debido a los 3 slash) y toma para L los datos de la cuarta tarjeta.

```
      WRITE (6, 2) (L(I), I = 1, 2)
   2  FORMAT (/// 1X, I3, I4)
```

Resultado

→ } 3 líneas en blanco

1234567

c)
```
      READ (5, 1) (L(I), I = 1, 2)
   1  FORMAT (I3)
```

Datos	Contenido de L
123456789	L(1) = 123
897654321	L(2) = 897

Nota.- El efecto es semejante a un FORMAT (I3/I3)

```
      WRITE (6, 2) (L(I), I = 1, 2)
   2  FORMAT (1X, I4)
```

Resultado

b123

b897

Nota.- El efecto es semejante a un FORMAT (1X, I4/1X, I4).

6.4.5. CONTROL DE CARRO

La primera posición de un registro de salida, para impresora, no se imprime sino que determina el control del carro.

b → avanza 1 línea

0 → avanza 2 líneas

\+ → no avanza

1 → salto a nueva página (cabeza)

\- → avanza 3 líneas

4 al D → Salta al correspondiente canal de acuerdo con el lazo de la banda de papel que controla la impresora.

Si la impresora no lleva control por banda de papel, su efecto es semejante al blanco (avanza una línea).

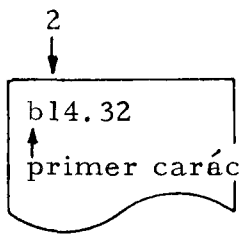
Nota.- Para algunos compiladores FORTRAN (ej: FORV-UNIVAC) ésta posición de control del carro, aunque no se imprime, ocupa en salida una posición de la impresora. Es decir, si la línea de impresión es de 132 caracteres, sólo se ocupan los 131 restantes, quedando el primero a blanco.

Ejemplos.- (En FORV - UNIVAC)

a) . Sea el contenido de A = 14.32

```
      WRITE (6, 2) A
2     FORMAT (1X, F5.2)
```

Resultado

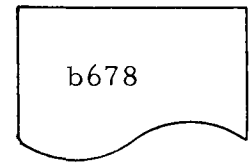


↑ primer carácter del registro que se utiliza para control del carro.

b) . Sea el contenido de L(1) = 1234 y L(2) = 5678

```
      WRITE (6, 2) L(2)
2     FORMAT (I4)
```

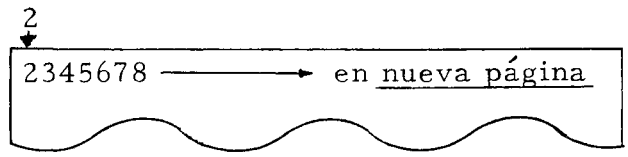
Resultado



(Nota.- Si la impresora no lleva control de banda de papel el 5 se interpreta como b).

```
      WRITE (6, 2) L
2     FORMAT (2I4)
```

Resultado



(Nota.- El resultado es debido a que el registro de salida deja la siguiente información:

1	2	3	4	5	6	7	8	b	b

Toma este carácter como control del carro; es decir, salta a nueva página.

6.4.6. FACTOR DE ESCALA

Utilizando los códigos de conversión F, E, G y D, puede efectuarse la E/S con factores de escala positivos o negativos mediante una potencia de 1Ø. El código de conversión tiene el formato xP precediendo inmediatamente a una especificación de repetición (si existe alguna); donde x es una constante entera que puede ir precedida de signo El efecto de x es multiplicar o dividir el correspondiente item de la lista por una potencia de 1Ø.

- Cuando se inicia el control de formato se establece un factorde escala de cero. Una vez establecido el factor de escala, se aplica a todos los descriptores de campo F, E, G y D subsiguientemente interpretados hasta que se encuentra y se establece otro factor de escala. Para restablecer el factor de escala inicial o sea el factor Ø se puede poner ØP.

- El factor de escala no se aplica para entradas que contengan exponente.

- En las entradas con conversiones E, F, G y D y sin exponente el campo externo, el valor representado internamente es el valor externo dividido por 10^x.

- Para salidas con conversión F, el valor externo es el interno multiplicado por 10^x.

- Para salidas con conversión E ó D, la parte significativa de la constante se multiplica por 10^x y el exponente se reduce en x. O sea, el valor no cambia, solo cambia la posición del punto decimal.

- Con el código de conversión G el factor de escala se rige por las mismas normas que con el código E cuando efectivamente se utilice este código; el factor de escala se ignorará cuando el código efectivo a utilizar sea el F.

Ejemplo.- (En FORV - UNIVAC)

a) Entrada

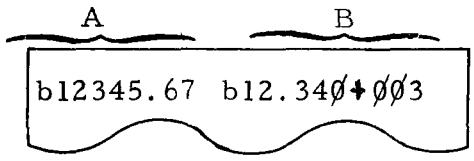
```
READ (5, 1) A, B
1 FORMAT (2PF9.2, E11.4)
```

b12345.67ØØ12.34E+Ø3 → al leer la tarjeta anterior se almacenará

A ← 123.4567
B ← 1234Ø.

b) Salida

```
      WRITE (6,2) A, B
    2 FORMAT (1X 2PF9.2, E11.4)
```



6.5. FORMATO VARIABLE

En cualquiera de las sentencias de E/S la referencia a la sentencia FORMAT se puede sustituir por el nombre de un 'array', que debe contener las especificaciones de formato, como caracteres a utilizar en un determinado momento. Los valores de este "array" pueden modificarse como los de cualquier variable, cambiando con ello el tipo de formato; de ahí la denominación de formato variable. La forma más normal de almacenar las especificaciones de formato es mediante sentencias de lectura.

Ejemplo.-

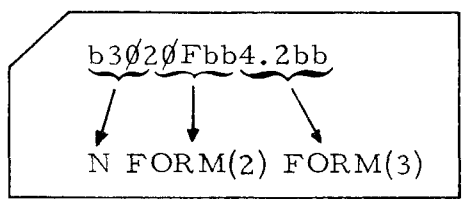
Se dispone de un programa que efectúa unos cálculos estadísticos, con los resultados de una serie de tests, sobre una muestra de tamaño considerable. Se parte de que el número máximo de preguntas de cada test es inferior a mil.

Para no desaprovechar el soporte exterior y poder ajustar las especificaciones del formato a la precisión de cada experimento concreto, sin tener que modificar la sentencia FORMAT, lo que llevaría consigo la recompilación del programa, se ha decidido utilizar un vector para contener las especificaciones de formato. Se supone que sólo se desea calcular la puntuación media de cada pregunta de un determinado test. Suponiendo que cada test consta de 30 preguntas y que cada pregunta se ha cuantificado de la forma X.XX (un entero y dos decimales) y que se ha perforado 20 preguntas por tarjeta (o sea cada test ocupará dos tarje-

tas, la primera con 20 datos y la segunda con 10). Las sentencias para efectuar la lectura correcta de estos datos podrían ser:

```
      DIMENSION A(1ØØØ), FORM(4), B(1ØØØ)
      DATA FORM (1)/'('/, FORM(4)/')'/
      READ (5, 1ØØ)N, FORM(2), FORM(3)
1ØØ   FORMAT (I3, 2A5)
1     READ (5, FORM, END = 55) (A(I), I = 1, N)
      DO 2 I=1,N
2     B(I) = B(I) + A(I)
      NTEST =NTEST + 1
      GO TO 1
55    DO 3 I = 1,N
3     B(I) = B(I)/NTEST
      WRITE(6, 1Ø1) (B(I), I = 1,N)
1Ø1   FORMAT (1X, F7.2)
      STOP
      END
```

La primera tarjeta a leer deberá ser



y a continuación todas las de datos propiamente dichos, que se leerían con el formato deseado o sea el: (2ØF4.2)

6.6. SENTENCIA <NAMELIST>

El NAMELIST, combinado con las sentencias de E/S, permite - transferir, desde un dispositivo exterior a memoria y viceversa, datos con formato sin necesidad de utilizar la sentencia FORMAT.

a) Su formato general es:

NAMELIST / nombre 1/a_1, a_2, ... a_n/ nombre 2/a_1, a_2, ..., a_n

Siendo.-

⟨nombre$_i$⟩ .- Es el nombre de un 'namelist'. - Un 'namelist' consta de 1 a 6 caracteres alfanuméricos, el primero de los cuales debe ser una letra. Una-vez que este nombre es definido, por aparecer en la sentencia NAMELIST, no puede ser redefinido en ninguna - otra instrucción y únicamente puede utilizarse en una sentencia de entrada/salida.

⟨a$_i$⟩ .- Nombre de una variable o de un arrray de cualquier tipo. La lista de variables o arrays pertenecientes al nombre de un 'namelist', finaliza - con un nuevo nombre de 'namelist'- incluido entre slash (/) ó con el fin de la sentencia NAMELIST.

Una variable o un array pueden pertenecer a más de una lista de nombres de 'namelist'.

b) El formato de E/S para un NAMELIST es:

```
READ (unidad, nombre del 'namelist' END = l1,
      ERR = l2)

WRITE (unidad, nombre del 'namelist')
```

Nota.- El END y el ERR son opcionales.

c) El formato general de los datos de entrada, que utilizan la sentencia NAMELIST es:

```
b& nombre del 'namelist' b nom1 = dato, nom2 =
   = dato, ... , nom_n = dato,& END
```

donde:

∠nombre del 'namelist'∠ .- Debe ser idéntico a un∠nombre∠ de la sentencia NAMELIST.

∠nomi∠ .- Debe ser una variable, elemento de un array ó nombre de un -- array, definido en la sentencia NAMELIST. Es decir un $\angle a_i \angle$.

∠dato∠ .- Constante del mismo tipo que - el ∠nomi∠ asociado. Si ∠nomi∠ es un array, ∠dato∠ es una lista de una o más constantes, separadas por comas (,).

Se admite, en el caso de arrays, asignar varios elementos utilizando la notación: j x c, siendo c una constante del tipo del array asociado y j una constante, entera sin signo, multiplicadora. No es necesario que un array sea completado por su lista de ∠dato∠.

Si las variables son lógicas, ∠dato∠

puede ser de la forma T ó -
..TRUE., F ó .FALSE.

Observaciones

. El blanco (b) y el primer ampersand (&) son obligatorios y señalan que viene el nombre de un 'namelist'. El blanco que precede a ⟨nom1⟩ señala el fin del nombre del 'namelist'.

  El segundo ampersand (&) seguido por END indica el fin del registro.

. Constantes literales pueden ser transferidas usando los descriptores (wH) ó (' ').

Ejemplo , -

MAT = 'LATIN' ó MAT = 5HLATIN

. La sentencia NAMELIST es una instrucción no ejecutable que especifica uno o varios nombres simbólicos (nombres del 'namelist') involucrados en la transferencia de datos.

. La sentencia READ referida al nombre de un 'namelist' ejecuta las siguientes acciones:

  1- Busca en los datos ese nombre de 'namelist' que figura en la sentencia READ.

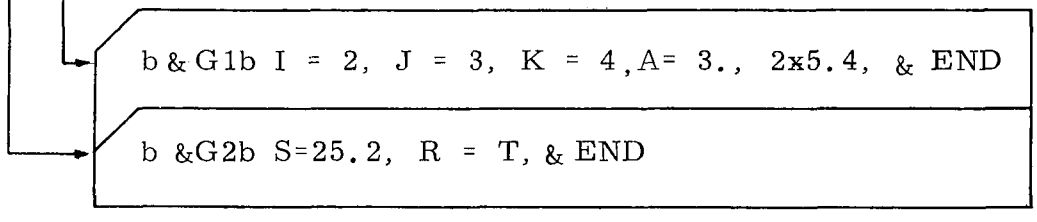
  2- Si lo encuentra, lee la primera variable

o array y la compara con la lista de nombres (a_i) definidas en la sentencia NAMELIST. Si encuentra esa variable o array en la lista, asigna los datos- (dato) y accede al siguiente nombre de variable o array. Si esa variable o a- rray no aparece en la lista (a_1, a_2.. a_n) de ese nombre de 'namelist', aparece un mensaje de error y finaliza el programa.

3- Si ese registro no contiene ese nombre de 'namelist', lee sucesivos registros- hasta que lo encuentr a ó finaliza por - error si no existe.

Ejemplo

```
      NAMELIST /G1/ I, J, K, A / G2/R, S
      DIMENSION A(3)
      LOGICAL R
      ———
      ———
      ———
      READ (5, G1)
      READ (5, G2)
```



6.7. ENTRADAS/SALIDAS SIN FORMATO

La ejecución de una sentencia de Entrada o Salida con formato es un proceso más complejo de lo que parece a simple vista, - ya que supone una conversión de tipo, rellenos, etc. Además como hay casos en que el propio formato se puede leer como - dato, es frecuente que la rutina de edición (p. ej.) sea un peque ño intérprete del formato de salida. Todo esto hace que si la - entrada/salida es intermedia (esto es, son datos creados por otro programa o con destino a otro programa) sea lo más práctico y eficiente el usar sentencias de entrada y salida sin formato, que es como si le dijéramos al compilador:

"escribe estas variables de forma tal que luego seas capaz de leerlas"

Lo normal es que el compilador se limite a hacer un volcado de las variables a escribir con lo que el proceso es muy rápido.

Formatos.-

```
READ (n, EOF = etiqueta 1, ERR = etiquet 2) lista

READ (n, END = etiqueta 1, ERR = etiqueta 2) lista

WRITE (n) lista
```

Donde todos los campos tienen el mismo sentido que en las sentencias de E/S con formato. (Ver Sección 6.1.)

Descripción.-

Las instrucciones de Entrada/Salida sin formato inician y controlan la transferencia de datos sin formato entre un dispositivo periférico determinado y memoria.

6.8. INSTRUCCIONES DE ENTRADA/SALIDA AUXILIARES

Controlan el movimiento de los periféricos de E/S y su posicionamiento a unos determinados puntos de referencia. Veamos las sentencias más importantes:

a) REWIND n

Siendo.- <n> : Una constante o variable entera que designa un fichero.

Descripción.-

Esta sentencia activa la unidad n y posiciona la cabeza lectora/grabadora al principio del fichero justamente delante del primer registro.

b) BACKSPACE n

Siendo.- <n> : Una constante o variable entera, que designa un fichero.

Descripción.-

"Produce el retroceso de un registro del fichero n".

. En FORTRAN se crea un registro de la siguiente forma:

1) Un registro con formato se define por las siguientes acciones.
   - La terminación de una instrucción WRITE.
   - La aparición de un slash (/) en un FORMAT.
   - El último paréntesis de cierre ')' encontrado en el FORMAT, cuando se produce reexploración de formato.

2) Un registro sin formato se define simplemente por la instrucción WRITE.

Nota.- La sentencia BACKSPACE no produce efecto si el fichero asociado con esa unidad está posicionado inmediatamente anterior al primer registro.

c) ENDFILE n

Siendo.- <n> : Una constante o variable entera designando un dispositivo de salida. La unidad debe especificar un dispositivo de salida, en cinta o en ficha.

Descripción.-

Cierra el fichero asociado con la unidad especificada (en el caso de la cinta magnética escribe una marca especial de fin de fichero).

Ejemplo de E/S sin formato

Escribir un programa que lea, desde tarjeta, una serie de números enteros y los imprima ordenados en valor ascendente. Cada número irá en una tarjeta, con formato I6. Se supone que los números están perforados correctamente.

Solución.- Leeremos las tarjetas y grabaremos los números en un fichero en cinta magnética, con número de unidad lógica 7. Posteriormente ordenaremos estos datos, con la ayuda del fichero 8, ascendentemente, y los imprimiremos:

```
      LOGICAL SW
      IB = 7
      IA = 8
      REWIND IB
    1 READ (5, 100, END = 2)N
  100 FORMAT (I6)
      WRITE (IB)N
      GO TO 1
    2 ENDFILE IB
    8 SW = .FALSE.
      REWIND IB
      REWIND IA
      READ (IB, END = 6)N1
    5 READ (IB, END = 7)N2
      IF(N1-N2) 3, 3, 4
    3 WRITE (IA)N1
      N1 = N2
      GO TO 5
    4 WRITE (IA)N2
      SW = .TRUE.
      GO TO 5
    6 WRITE (6, 101)
  101 FORMAT (1H1, 'No hay datos')
      STOP
    7 WRITE (IA)N1
      ENDFILE IA
      IF(SW) GO TO 11
      REWIND IA
   10 READ (IA, END = 9)N
      WRITE (6, 102)N
  102 FORMAT (1X, I6)
      GO TO 10
   11 IBB = IB
      IB = IA
      IA = IBB
      GO TO 8
    9 END
```

6.9. REQUERIMIENTOS DE E/S EN EL FORTRAN 9.400

a) Entrada de tarjetas

Los datos deben ir precedidos por una tarjeta cuya $1^{a.}$ y $2^{a.}$ columnas lleven los símbolos / $ y finalizar con otra que contenga en $1^{a.}$ y $2^{a.}$ columna un /*.

Además, las fichas de datos se deben colocar en el flujo de control inmediatamente detrás de la sentencia de control de trabajos // EXEC.

b) Salida de tarjetas

Una instrucción ENDFILE produce una tarjeta /*. No produce la tarjeta / $ que es por cuenta del usuario, si luego quiere procesar este fichero.

c) Cintas y discos

El FORTRAN 9400 sólo acepta carretes con fichero y etiquetas estandar. Es decir, estos ficheros han debido ser creados mediante un programa FORTRAN 9.400.

En el caso de cintas magnéticas, solo permite la generación de ficheros de un solo volumen. El modo de E/S de una cinta viene determinado por su primera referencia. Si la primera referencia es una READ, el modo es de entrada. Si la primera referencia es una WRITE o ENDFILE, el modo es de salida. Las sentencias BACKSPACE y REWIND no ponen modo de E/S al

fichero. Si se produce un error de paridad en una sentencia READ, en la cual se especifica la cláusula ERR, la salida se produce a la sentencia que señale esta cláusula, en caso contrario finaliza el trabajo. La salida por EOF se produce despues de encontrar la siguiente secuencia: marca de cinta, EOF marca de cinta, (esta secuencia la escribe la sentencia ENDFILE).

El FORTRAN usa el método de acceso directo, con direccionamiento relativo, para leer y escribir en disco. Antes de empezar a escribir un fichero en disco debe ser preformateado, ell quiere decir que hay que inicializar las pistas del fichero con registros de datos de la longitud apropiada. Los ficheros secuenciales, sobre disco, deben ser preformateados con una longitud de registro de 258 bytes. En los ficheros de acceso directo su longitud debe estar de acuerdo a la definida en la sentencia DEFINE FILE. (Ver Apartado 6.10.1)

6.10. OPERACIONES DE ACCESO DIRECTO

El término "acceso directo" define la posibilidad de acceder a un registro específico de un fichero, sin tener que tratar todos los registros que le preceden como ocurre con un fichero soportado por un dispositivo secuencial, tal como la lectora de fichas o una unidad de cinta magnética.

Para que se pueda realizar una operación de "acceso directo" es preciso que el dispositivo lo permita, como es el caso del tambor magnético, disco, etc. En estos dispositivos, cuando no se utilizan las posibilidades del acceso directo, también se pueden mantener ficheros secuenciales.

El FORTRAN 9400 sólo admite el disco magnético, como dispositivo de acceso directo.

Existen cuatro sentencias para efectuar este acceso: READ, WRITE, DEFINE FILE Y FIND. Las sentencias READ y WRITE efectúan la transferencia de datos entre el fichero y la memoria principal. La sentencia DEFINE FILE identifica y describe las características de los ficheros a utilizar durante las operaciones de acceso directo. La sentencia FIND solapa la localización del registro, desde un dispositivo de acceso directo, con los cálculos del programa; su misión es posicionar las cabezas de acceso en la pista del disco en que se encuentra un determinado registro. Las sentencias READ ó WRITE pueden o no hacer referencia a una sentencia FORMAT.

En un fichero de acceso directo cada registro tiene un único número de registro asociado con él. En las sentencias READ, WRITE y FIND el programador no solo tiene que especificar el número de referencia al fichero, como en las operaciones de E/S secuenciales, sino también el número del registro a leer, escribir o buscar.

6.10.1. SENTENCIA <DEFINE FILE>

Para realizar el acceso directo a un fichero, es necesario definir, previamente a su utilización, las características del fichero en cuestión. Las características de uno o más ficheros se declaran mediante la sentencia no ejecutable DEFINE FILE. Para cada fichero se debe especificar:

- Identificación del fichero.
- Número de registros del fichero.

- Tamaño máximo de cada registro
- Identificación del formateado deseado para los datos.

Formato.-

DEFINE FILE $f_1(r_1, m_1, x_1, v_1), f_2(r_2, m_2, x_2, v_2), \ldots\ldots$
$f_n(r_n, m_n, x_n, v_n)$

Siendo.- $\langle f_i \rangle$: Una constante entera que especifica el número de referencia de un fichero.

$\langle r_i \rangle$: Una constante entera que especifica el número de registros en el fichero.

$\langle m_i \rangle$: Un entero que indica el tamaño máximo del registro en el fichero, en términos de caracteres (bytes), posiciones de memoria (bytes), o unidades de almacenamiento; dependiendo de la designación elegida para x;x representa una de tres posibles letras que indican una opción del control del formato.

$\langle v_i \rangle$: Una variable entera no subíndicada, llamada variable asociada. Después de la ejecución de una READ ó WRITE, esta variable contendrá la posición que ocupa el siguiente registro en el fichero, con lo que se garantiza un tratamiento secuencial automático, con solo poner la variable asociada en el campo de número de registro en una sentencia READ ó WRIT Después de la ejecución de una FIND la variable asociada contendrá el valor del registro indicado.

$\langle x_i \rangle$: Una de las tres letras siguientes:

L-La transferencia de los datos se puede efectuar con o sin formato; el tamaño (m) del registro se mide en bytes

E-La transferencia de los datos se efectúa con formato, el tamaño (m) del registro se mide en caracteres (bytes).

U- La transferencia de los datos se efectúa sin formato; el tamaño (m) del registro se mide en unidades de almacenamiento. Para determinar el tamaño del registro en unidades de almacenamiento: se divide por cinco el número total de bytes necesarios para almacenar todos los items de la lista de E/S, y cuando el resto sea distinto de cero se redondea al entero - inmediato superior.

Ejemplo. -

DEFINE FILE 3(100, 12Ø, L, FILE3)

El fichero 3 se compone de 100 registros y el tamaño - máximo del registro es 120 bytes. L indica que el tamaño del registro se especifica en bytes. Si la sentencia de E/S contiene una referencia a un formato, se transfieren- con formato 120 bytes de datos, si nó los datos se transfieren sin formato.

6.10.2. SENTENCIA <READ EN ACCESO DIRECTO>

Formato. -

READ (fich'pos-reg, formato, ERR = 1_1) lista

Siendo.- <fich> : Una constante o variable entera que, seguida de un apóstrofo, identifica el fichero.

<pos-reg> :Una expresión entera que designa la posición del registro en el fichero -- (El primer registro tiene como valor pos-reg = 1)

<formato> :(Opcional) Una etiqueta de una sentencia FORMAT, nombre de array, o variable entera (lo mismo que en 6.1).

$<l_1>$: (Opcional) Una etiqueta de la sentencia a la que se cederá el control cuando se produzca una condición de error durante la transferencia de datos desde disco a memoria.

Ejemplo.-

Para leer datos, con formato, desde disco:

```
INTEGER FILE3
DEFINE FILE 3(100, 512, L, FILE3)
    ===
    ===
FILE3 = 0
    ===
    ===
READ (3'FILE3+ 1, 87, ERR = 11Ø)A, B, (C(I), I = 1, 3Ø)
87 FORMAT (32F16.4)
    ===
    ===
```

Cuando la sentencia READ se ejecuta por primera vez, se transfiere el primer registro del fichero 3 a memoria. En posteriores ejecuciones,la orden READ transfiere el registro siguiente del fichero a memoria. El descriptor ---

32 F16.4 indica que cada unidad de datos consta de 16 bytes y que se transfieren 32 items, de tales unidades. O sea se transfieren a memoria los 512 bytes del registro (32*16).

El slash, en una especificación de formato, puede controlar el punto inicial de transferencia de los datos en el fichero. Si en el ejemplo anterior la sentencia FORMAT fuera FORMAT (//32F16.4), la primera ejecución de la sentencia READ transferiría el tercer registro del fichero, la segunda ejecución transferiría el sexto registro, ... etc.

6.10.3. SENTENCIA <WRITE EN ACCESO DIRECTO>

Se ajusta a la siguiente descripción

Formato.-

WRITE (fich'pos-reg, formato) lista

Siendo.- ⟨fich⟩: Una constante o variable entera que, seguida de un apóstrofo, identifica al fichero.

⟨pos-reg⟩: Una expresión entera que designa la posición del registro en el fichero.

⟨formato⟩: (opcional) Una etiqueta de una sentencia FORMAT, nombre de matriz o variable entera.

Ejemplo.-

De una operación WRITE:

```
      INTEGER FILE4
      DEFINE FILE  4(150, 36, L, FILE4)
      LOGICAL L
      DOUBLE PRECISION D
      FILE4 = 2
      WRITE (4'FILE4+1,2) I,R,D,L
    2 FORMAT (I8, F12.2,D15.5,L1)
```

Se transfieren treinta y seis bytes (8+12+15+1) desde memoria al tercer registro del fichero 4. La especificación del formato indica el número de bytes para los valores, entero, real, doble precisión y lógico transferidos. Si la sentencia WRITE no especifica una referencia a formato, se ejecutará una escritura sin formato. En este caso se transferirán 21 bytes o sea:

NOMBRE VARIABLE	TIPO	BYTES QUE OCUPAN
I	Entera	5
R	Real	5
D	Doble Precisión	10
L	Lógica	1
		21

6.10.4. SENTENCIA <FIND>

Formato.-
FIND (fich' pos-reg)

Siendo.- <fich> : Una constante o variable entera que, seguida de un apótrofo, identifica el fichero.

<pos-reg>: Una expresión entera que designa la posición de un registro en el fichero.

Descripción.- La sentencia FIND permite desminuir el tiempo necesario para ejecutar un programa objeto que solicita registros de disco.

Esta sentencia posiciona los brazos de acceso a la pista especificada por el campo - ⟨pos-reg⟩ . Al mismo tiempo que los brazos buscan la posición solicitada, se posibilita que el programa continúe su ejecución. Si se ejecuta una sentencia READ, para acceder a un registro, en el que previamente se ha hecho una FIND, se ganará tiempo total de programa, pues al procesarse una -- sentencia READ el programa se detiene y no vuelve a tomar control hasta que la operación ha concluído (posicionamiento y transferencia).

Ejemplo. -

Este ejemplo muestra la relación entre las sentencias -- READ y FIND

FIND (4'2Ø)

READ (4'2Ø)

Mientras que se posicionan los brazos de acceso. se ejecutan las sentencias comprendidas en la FIND y la READ.

CAPITULO 7
FUNCIONES Y SUBRUTINAS

7.1. GENERAL

En el capítulo 1 definíamos un "programa FORTRAN" como un conjunto de módulos unidad que se compilaban independientemente, siendo necesaria la presencia del denominado "modulo principal ó programa principal" y siendo optativa la existencia de los demás módulos. El resto de los módulos unidad reciben el nombre de subprogramas ó procedimientos. En otras palabras, se denomina subprograma a un conjunto de sentencias que ayudan a resolver una parte del programa general y que se encuentran fuera del "módulo principal", es decir, que se compilan independientemente. Aún cuando lo más común es que un subprograma dependa directamente del "programa principal" (sea llamado por este), también puede estar conectado con otro subprograma.

La utilización de los subprogramas nos permite:

- Que los procesos más frecuentes ya estén programados, como ocurre con muchas funciones matemáticas incorporadas al propio compilador.
- Dividir los programas en unidades más pequeñas con lo que un programa complejo puede abordarse por partes y por distintos programadores.
- La segmentación de los programas, o sea mantener en memoria únicamente la parte del programa que en ese momento se vaya a ejecutar (ver sección 7.10, apartado j).

En este capítulo estudiaremos tres tipos de procedimientos:

- Funciones
- Subrutinas
- Función "fórmula"

7.1.1. PARAMETROS DE UN PROCEDIMIENTO

Los procedimientos utilizan normalmente un conjunto de parámetros ó argumentos cuya función es permitir que el subprograma se ejecute con valores diferentes.

Los parámetros declarados en la definición del procedimiento se denominan argumentos ficticios (ej. una declaración podría ser SUBROUTINE SUM (A, B), donde A y B son argumentos ficticios).

A los parámetros transferidos desde la instrucción de llamada se les denomina argumentos actuales. (Una llamada al procedimiento anterior podría ser, por ejemplo, CALL SUM (X, Y). - X e Y son argumentos actuales). Los agumentos actuales se deben corresponder en número, tipo y orden con los argumentos ficticios del procedimiento (Ej. CAL SUM (X, Y)→SUBROUTINE SUM (A, B)
X debe ser del mismo tipo que A e Y debe serlo con respecto a B).

7.1.2. ARGUMENTOS DE UN PROCEDIMIENTO

Seguidamente se detallan las formas permitidas, según el procedimiento, en el FORTRAN 9400.

¿Están permitidos como argumentos actuales?	Función (2)	Subrutina (2)	Función "Fórmula"
Nombres de variables	SI	SI	SI
Expresiones	SI	SI	SI
Referencias a funciones (4)	SI	SI	SI
Elementos de un array	SI	SI	SI
Nombres de array	SI	SI	NO
Constantes literales	SI (3)	SI (3)	NO
Etiquetas (una etiqueta precedida de un &)	NO	SI	NO
Nombre de un procedimiento externo (1)	SI	SI	NO

Notas.-

(1) Los nombres de los procedimientos externos, que aparezcan como argumentos actuales, deberán ser declarados en una Sentencia EXTERNAL.

(2) No se admiten más de 255 argumentos.

(3) Como máximo se transmiten los 10 caracteres más a la izquierda del literal aunque el número de caracteres aceptados por la función o la subrutina dependen del tipo del argumento ficticio.

(4) Si la función referenciada es parte de una expresión suscrita en una lista de E/S, la constante no debe contener un paréntesis abierto, cerrado o el caracter igual.

7.1.3. LISTA DE ARGUMENTOS DE UNA SUBRUTINA CON ETIQUETAS DE INSTRUCCIONES.

Como se ha visto en la tabla 7.1.2, sólo está permitido para los procedimientos tipo SUBRUTINA el que un argumento sea la etiqueta de una instrucción ejecutable.

Sea una definición de subrutina del tipo:

SUBROUTINE SUMA (X, Y, *, *)

Los asteríscos (*) representan salidas a instrucciones etiquetadas del módulo de llamada.

La salida de la subrutina, a una de estas etiquetas, se efectúa con la sentencia RETURN K, siendo < K > una constante o variable entera que señala la posición ordinal de la etiqueta dentro de la lista de argumentos. Los argumentos normales no se contabilizan.

Ejemplo.- Si la llamada a esa subrutina fuera de la forma:

CALL SUMA (A, B, & 10, & 20)

Un RETURN 2, retorna a la instrucción del módulo de llamada cuya etiqueta es 20.

En el FORTRAN 9400 se pueden incluir un máximo de 15 etiquetas en las listas de argumentos, y los argumentos actuales de retorno a instrucciones etiquetadas deben ir precedidos por un ampersand (&) mientras que los argumentos ficticios, en la definición de la SUBROUTINE, correspondientes a retornos de etiquetas, deben ser asteríscos (*).

Ejemplo

Módulo principal

```
      DIMENSION A(20),  B(20),  C(20)
      N = 20
      CALL SUM (A, B, C, &, 5, N)
    5 WRITE (6, 6)
    6 FORMAT (1X,'LA SUMA DE LOS DOS VECTORES ES Ø')
      END
```

```
      SUBROUTINE SUM (A, B, C, *, M)
      DIMENSION A(M),  B(M),  C(M)
      DO  1 I = 1,  M
      C(I) = A(I) + B(I)
    1 CONTINUE
      DO 2 I = 1, M
      IF(C(I).NE.Ø) GO TO 3
    2 CONTINUE
      RETURN 1
    3 RETURN
      END
```

. En este ejemplo la subrutina SUM, nos suma dos vectores: A y B, dejando el resultado en el vector C.

Si todos los elementos de C son cero, la rutina retorna por (*) a la instrucción etiquetada con 5 del módulo principal, y nos imprime un mensaje que -

nos indica este hecho. En los demás casos retorna, al módulo principal, a la instrucción inmediatamente siguiente a la CALL SUM.

7.2. SENTENCIA <FUNCTION>

Una función es un subprograma formado por un conjunto de instrucciones destinadas a efectuar ciertos cálculos, utilizando para ello variables y/o constantes transmitidas por los módulos que la llamen. Su primera sentencia debe ser:

Formato.-
<tipo> FUNCTION <nombre * S> (a1, a2, ..., an)

Donde.- <tipo> : Indica el tipo de la función y es opcional. Si se especifica deberá ser: Integer, Real, Double Precisión, Complex ó Logical.
Si se omite, el tipo de la función viene determinado por el tipo automático de su nombre (ver 2.2) o por el tipo que se asigne a <nombre> en el cuerpo de la función mediante la sentencia IMPLICIT o la de tipificación explícita.

<nombre> : Nombre simbólico que identifica a la función cuando es referenciada.

<* S> : Sirve para especificar la longitud de acuerdo con lo señalado en el apartado 5.4.3. (Su utilización es opcional).

$<a_i>$: Argumentos ficticios. Se pueden considerar como pseudovariables y deben ser nombres simbólicos.
En el FORTRAN 9400 el número de argumentos permitidos es de 1 a 255.

Descripción.- Define un subprograma función que se caracteriza por ser un programa incompleto por si mismo, y que al menos debe contener una sentencia FUNCTION, una RETURN y una única sentencia END.

Al llamar desde otro módulo unidad a un subprograma función, éste devuelve al módulo de llamada un valor en el lugar en que aparece el nombre de la función con los argumentos actuales.

Por ello, alguna de las instrucciones del subprograma FUNCTION debe asignar un valor al nombre de la función, bien sea mediante una sentencia READ, una CALL o una aritmética (debiendo aparecer el nombre de la función a la izquierda del signo igual).

Una función puede llamar a otros subprogramas, pero no puede llamarse a sí misma, ello es debido a que el FORTRAN no es un lenguaje recursivo.

Cuando se llama a la función, los argumentos ficticios se sustituyen por los valores de los argumentos de la sentencia de llamada (argumentos actuales) precisando, por ello, que ambos sean del mismo tipo, estén en el mismo orden y haya igual número de ellos. Cada argumento ficticio, que sea una variable, debe aparecer al menos una vez en una instrucción ejecutable de la función. Las variables que -

intervienen en el subprograma son independientes de las de otro módulo. por tanto, pueden llevar el mismo nombre simbólico.

Ejemplos.-

a) INTEGER FUNCTION ELIPSE (F1 ,F2)
define una función entera de nombre ELIPSE y con dos argumentos: F1 y F2

b) FUNCTION ELIPSE (F1, F2)
es similar a:
REAL FUNCTION ELIPSE (F1, F2)

Nota.- Como se ha señalado en la ección 1.6. interesa en el FORTRAN9400 , evitar usar el $ como el tercer caracter del nombre de una función, ya que éste es el formato usado para las rutinas del sistema.

7.3. REFERENCIA A UNA <FUNCTION>

La llamada a un subprograma función, desde un módulo unidad, se ajusta a la siguiente codificación:

Formato.-
nombre función $(a_1, a_2, \ldots, a_n)$

Siendo.- <nombre función> : Nombre simbólico que identifica a la función. Debe ser el especificado en la sentencia FUNCTION

$<a_i>$: Argumentos actuales. Pueden ser:
- Nombres de variables

- Expresiones
- Referencias a otras funciones
- Elementos de un array
- Nombres de arrays
- Constantes literales
- Procedimientos externos

Descripción.- La llamada a una función se utiliza para de
volver un valor, en el nombre de la función
a la expresión que contiene la llamada, basa
do en los argumentos de la lista.

Ejemplo.-

Cálculo de la distancia de un punto (X1, Y1) a una recta --
AX+BY+C = 0

READ (5, 1Ø) X1, Y1, A, B, C	FUNCTION DISTAN (X, Y, A1, A2, A3)
D = DISTAN (X1, Y1, A, B, C) (referencia a una FUNCTION)	A = A1*X + A2*Y + A3
	DISTAN = A/SQRT(A1**2 + A2**2)
	RETURN
	END

Nota.- SQRT es una función estandar del Fortran (ver tabla II)

7.4. FUNCIONES ESTANDAR DEL FORTRAN

EL FORTRAN UNIVAC 9400 proporciona dos tipos de funciones:

a) Las intrínsecas.- Son funciones que se incorporan en la propi
compilación del programa fuente; es decir
no precisan de ensamblaje posterior (Se d
criben en la Tabla I).

b) Las externas.- Residen, en reubicable, en las librerías d
sistema y precisan de un posterior ensam
blaje (Se describen en la Tabla II).

Nota.- Si el tipo de una función intrínseca o externa se cambia con una IMPLICIT, la IMPLICIT se debe anular con una declaración de tipo explícito para realizar una correcta referencia a la función.

TABLA I

FUNCIONES INTRINSECAS

NOMBRE DE LA FUNCION	Nº DE ARGUMENTOS	FUNCION	ARGUMENTO	VALOR DE LA FUNCION
ABS IABS	1	Determinar el valor absoluto del argumento	real entero	
AINT INT	1	Truncado. Elimina parte decimal	real real	real entero
AMOD MOD	2	Resto. Definido como $a_1-INT(a_1/a_2)*a_2$	real entero	
AMAXØ AMAX1 MAXØ MAX1	≥ 2	Selecciona el valor mayor	entero real real	real real entero
AMINØ AMIN1 MINØ MIN1	≥ 2	Selecciona el menor valor	entero real real	real real entero
FLOAT	1	Convierte el argumento de entero a real	entero	real
IFIX HFIX	1	Convierte el argumento de real a entero	real	entero
SIGN ISIGN	2	Sustituye el signo del primer argumento por el signo del segundo	real entero	
DIM IDIM	2	Diferencia positiva Resta el menor de los dos argumentos del primero	real entero	

TABLA I (continuación)

NOMBRE DE LA FUNCION	Nº DE ARGUMENTOS	FUNCION	ARGUMENTO	VALOR DE LA FUNCION
DMOD	2	(Ver MOD, AMOD)	doble precisión	doble precisión
DABS	1	(Ver ABS, IABS)	doble precisión	doble precisión
CABS CDABS	1	(Ver ABS, IABS)	complejo	real
IDINT	1	(Ver INT)	doble precisión	entero
DMAX1	≥2	(Ver MAX1)	doble precisión	doble precisión
DMIN1	≥2	(Ver MIN1)	doble precisión	doble precisión
DFLOAT	1	(Ver FLOAT)	entero	doble precisión
DSIGN	2	(Ver SIGN)	doble precisión	doble precisión
SNGL	1	Convertir doble precisión a real	doble precisión	real
REAL	1	Obtener la parte real de un número complejo	complejo	real
AIMAG	1	Obtener la parte imaginaria de un número complejo	complejo	real

TABLA I (continuación)

NOMBRE DE LA FUNCION	Nº DE - ARGU MENTOS	FUNCION	ARGU- MENTO	VALOR DE FUNCION
DBLE	1	Convertir de real a doble precisión	real	doble precisión
CMPLX	2	Convertir dos argumentos reales a un número complejo	real	complejo
CONJG DCONJG	1	Obtener el conjugado de un número complejo	complejo	complejo

TABLA II

FUNCIONES EXTERNAS

NOMBRE DE LA FUNCION	N°DE ARGUMENTOS	FUNCION	TIPO DE ARGUMENTO	TIPO DE RESULTADO
EXP	1	Exponencial (e^x)	real	real
EXP10	1	Exponencial (10)	real	real
ALOG	1	Logaritmo natural	real	real
ALOG10	1	Logaritmo decimal	real	real
SIN	1	Seno	real	real
COS	1	Coseno	real	real
TAN	1	Tangente	real	real
ASIN ARSIN	1	Arco seno	real	real
ACOS ARCOS	1	Arco coseno	real	real
ATAN	1	Arco tangente	real	real
ATAN2	2	Arco tangente del cociente (a_1/a_2)	real	real
SINH	1	Seno hiperbólico	real	real
COSH	1	Coseno hiperbólico	real	real
TANH	1	Tangente hiperbólica	real	real
SQRT	1	Raíz cuadrada	real	real
CBRT	1	Raíz cúbica	real	real

TABLA II (Continuación)

NOMBRE DE LA FUNCION	Nº DE ARGUMENTOS	FUNCION	TIPO DE ARGUMENTO	TIPO DE RESULTADO
DEXP	1	e^x	doble precisión	doble precisión
CEXP CDEXP	1	e^{+x}	complejo	complejo
DLOG	1	Logaritmo natural	doble precisión	doble precisión
CLOG CDLOG	1	Logaritmo natural	complejo	complejo
DLOG10	1	Logaritmo decimal	doble precisión	doble precisión
DASIN DARSIN	1	Arco seno	doble precisión	doble precisión
DACOS DARCOS	1	Arco coseno	doble precisión	doble precisión
DATAN	1	Arco tangente	doble precisión	doble precisión
DATAN2	2	Arco tangente de la razón (a_1/a_2)	doble precisión	doble precisión
DSIN	1	Seno	doble precisión	doble precisión
CSIN CDSIN	1	Seno	complejo	complejo
DCOS	1	Coseno	doble precisión	doble precisión

TABLA II (Continuación)

NOMBRE DE LA FUNCION	Nº DE ARGUMENTOS	FUNCION	TIPO DE ARGUMENTO	TIPO DE RESULTADO
CCOS CDCOS	1	Coseno	complejo	complejo
DTAN	1	Tangente	doble precisión	doble precisión
COTAN	1	Cotangente	real	real
DCOTAN	1	Cotangente	doble precisión	doble precisión
DSQRT	1	Raíz cuadrada	doble precisión	doble precisión
CSQRT CDSQRT	1	Raíz cuadrada	complejo	complejo
DTANH	1	Tangente hiperbólica	doble precisión	doble precisión
DSINH	1	Seno hiperbólico	doble precisión	doble precisión
DCOSH	1	Coseno hiperbólico	doble precisión	doble precisión

7.5. SENTENCIA <SUBROUTINE>

Define un subprograma tipo subrutina, y debe ser la primera instrucción del módulo.

Formato.-

a) SUBROUTINE nombre $(a_1, a_2, \ldots, a_n)$

b) SUBROUTINE nombre

Siendo.- <nombre> : Nombre simbólico que identifica a la subrutina cuando es referenciada.

$< a_i >$: Argumentos ficticios. Se ajustarán en todo a los argumentos actuales de la sentencia de llamada. El Fortran 9400 admite un máximo de 255 argumentos.

Descripción.- Inicialmente los lenguajes FORTRAN sólo admitían que se devolviera un valor de retorno, en el caso de las funciones, con el nombre de la función, estando prohibido que los argumentos pudieran retornar ningún valor. Es decir, los argumentos de la sentencia FUNCTION sólo eran parámetros de entrada. Por ello siempre que, al utilizar un subprograma, era necesaria la existencia de más de un valor de retorno se hacía obligatorio el uso del subprograma subrutina.

La subrutina devuelve valores por medio de sus argumentos $< a_i >$, definidos en la sentencia SUBROUTINE y en ningún caso por medio del <nombre> de la subrutina. Este, únicamente, sirve para referenciarla.

El módulo subrutina, al igual que el de la función, exige una sentencia RETURN, y una última sentencia END.

En lo que respecta a la recursividad, y a las características que deben reunir los argumentos ficticios con respecto a las actuales, es válido todo lo expuesto en la " Sentencia FUNCTION".

Otra característica que diferencia a la subrutina de la función, es el hecho de que la subrutina puede no llevar argumentos, mientras que la función exige - por lo menos uno.

Ejemplo.-
Obtener el área de un rectángulo y su perímetro.

Módulo principal

```
      READ (5, 1) A, B
    1 FORMAT (2F7.3)
      CALL RECT  (A, B, AREA, PERIM)
      ≡≡≡
      END
```

Módulo subrutina

```
      SUBROUTINE  RECT(BASE, ALTURA ,  A, B)
      A = BASE * ALTURA
      B = 2.* (BASE +ALTURA)
      RETURN
      END
```

7.6. SENTENCIA <CALL>

La llamada a un subprograma subrutina, desde un módulo unidad, se efectúa mediante la sentencia CALL, que se ajusta a la siguiente codificación:

Formato.-

a) CALL nombre $(a_1, a_2, \ldots, a_n)$

ó

b) CALL nombre

Siendo.- <nombre>: Nombre simbólico que identifica a la subrutina. Debe ser el especificado en la sentencia SUBROUTINE.

$<a_i>$: Argumentos actuales. Pueden ser:

- Nombre de variables
- Expresiones
- Referencias a otras funciones
- Elementos de un array
- Nombres de arrays
- Constantes literales
- Etiquetas de instrucciones ejecutables
- Procedimientos externos.

Descripción.- La sentencia CALL permite referenciar a una SUBROUTINE. No precisa, como en el caso de la FUNCTION, que se tenga que retornar un valor en la ejecución de la llamada. Una vez ejecutada la subrutina el control del programa vuelve a la sentencia siguiente a la de llamada (sentencia CALL), a no ser que exista algún argumento formal que sea una etiqueta (ver 7.1.3.) en cuyo caso puede volver a la sentencia, del programa referenciante, con esa etiqueta.

Ejemplo.-

Utilizar una subrutina para saltar una página de impresora e imprimir FIN DE EJECUCION

Módulo unidad	Módulo subrutina
	SUBROUTINE PAGE
	WRITE (6, 1)
	1 FORMAT (1H1)
CALL PAGE	WRITE (6, 2)
	2 FORMAT (////'FIN DE EJECUCION'////)
	WRITE (6, 3)
	3 FORMAT (1H1)
	RETURN
	END

7.7. FUNCION "FORMULA"

Las funciones "fórmula", denominadas igualmente "funciones aritméticas definidas", consisten en expresiones que se ajustan a las reglas enunciadas para la escritura de fórmulas aritméticas, y que figuran en el propio módulo unidad.

Formato.-

nombre de la función $(a_1, a_2, \ldots, a_n)$ = expresión

Siendo.- ⟨nombre de la función⟩ : Un nombre simbólico que identifica el procedimiento.

⟨a_i⟩ : Argumentos ficticios, que son nombres de variables.

⟨expresión⟩ : Es una expresión aritmética o lógica, que no puede contener elementos de un array, o una referencia a otra función "fórmula".

Descripción.- La definición de la función debe aparecer en el programa antes que cualquier instrucción ejecutable.

La función "fórmula" es un procedimiento que puede ser referenciado en una expresión.

Admite la asignación de un tipo, utilizando para ello una sentencia de tipo explícito o una IMPLICIT.

Se pueden especificar un máximo de 50 argumentos ficticios, en una sentencia de este tipo.

Todos los argumentos ficticios $\langle a_i \rangle$ deben ser referenciados en la ⟨expresión⟩ . Son reemplazados, durante la ejecución, por las verdaderas variables (argumentos actuales) que figuren en la lista de llamada.

Los argumentos ficticios pueden aparecer (sus nombres) como nombres de variables en el mismo programa; no existiendo ninguna relación entre estos nombres de variables y los nombres de los argumentos ficticios de la función "fórmula".

Si la función "fórmula" aparece en un procedimiento FUNCION ó SUBROUTINE la ⟨expresión⟩no puede contener un nombre simbólico idéntico al nombre del procedimiento o un nombre de entrada (entry point) del mismo subprograma.

Ejemplo.-

```
SUMA (I1, I2) = 2.34 * I1 + I1*I2
    ————
    ————
A = SIN (X) + SUMA (A1, A2)
```

. La función "fórmula" SUMA, tiene como argumentos ficticios: I1 e I2.

. La ⟨expresión⟩ es: 2.34*I1 + I1*I2, y no puede contener elementos de un array, ni referencias a otra función "fórmula".

Ejemplo (No válido)

```
MUL (I, J) = ABS(I) + I * MOD (I, J)
    ————
    ————
SUMA1 (I1, I2) = (I1 + I2) * MUL (I1, I2)
```

7.8. SENTENCIA ⟨RETURN⟩

La última instrucción ejecutada en el subprograma función o subrutina debe ser la sentencia RETURN.

Su formato es:

Formato.-

a) RETURN

ó

b) RETURN i

Siendo.- <i> : Una constante entera sin signo o variable entera que apunta a la i-ésima etiqueta de la lista actual de argumentos (los argumentos normales no cuentan).

Descripción.- Tiene como misión devolver el control al módulo - unidad que ha realizado la llamada a ese subprograma. Si no se desea el retorno al módulo de llamada, en lugar de la sentencia RETURN se puede utilizar - la sentencia STOP, que hará detener la ejecución del programa en ese punto. En un subprograma función sólo se admite el formato a). El formato b) se puede emplear, en las subrutinas, para efectuar una salida a una instrucción ejecutable del módulo de llamada distinta de la que sigue en secuencia a la sentencia CALL.

Ejemplo.-

a) Módulo unidad (de llamada)

```
      CALL SUB (X, & 100, Y, &200)
```

b) Subprograma Subrutina

```
      SUBROUTINE SUB (A, *, B, *)

      RETURN 2  (retorna el control a la instrucción con etiqueta
                 200)

      RETURN 1 (retorna el control a la instrucción con etiqueta
                100).
```

7.9. SENTENCIA <ENTRY>

Permite tener puntos de entrada adicionales en un procedimiento función o subrutina.

Formato.-
ENTRY <nombre> (a_1, a_2, ..., a_n)

Siendo.- <nombre> : Es un nombre simbólico que indica un punto de entrada al procedimiento.

$<a_i>$: Argumentos ficticios, se deben corresponder con los argumentos actuales en orden, número y tipo. El número máximo de argumentos es 255.

Los argumentos son opcionales el "entry" se encuentra en un subprograma subrutina. En el caso de subprogramas función, el "entry" debe llevar por lo menos un argumento.

Todo lo dicho para la sentencia FUNCTION o para la sentencia SUBROUTINE es respectivamente válido para - la sentencia ENTRY, dependiendo de que sea un punto de entrada a un procedimiento función o a un procedimiento subrutina.

Ejemplo.-

Módulo subrutina

```
SUBROUTINE SUMA (X, Y)
      ======
      ======
      ======

ENTRY SUM (X)
      ======
      ======
      ======
      ======
      ======

RETURN

END
```

Módulo principal

```
      ======
      ======

CALL SUMA (A, B)
      ======
      ======
      ======

CALL SUM (C)
      ======
      ======
```

7.10. SUBRUTINAS BASICAS

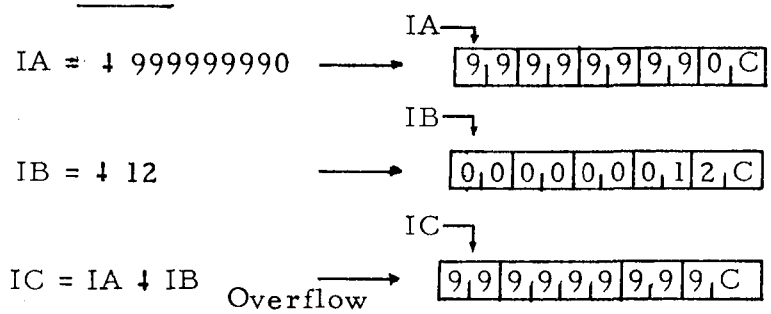
El FORTRAN UNIVAC 9.400 dispone de un amplio conjunto de subrutinas básicas. Pasemos a describir algunas de - ellas:

a) SUBRUTINA <OVERFL>

La subrutina OVERFL detecta si, en la ejecución de una operación, se ha producido o no desbordamiento en la capacidad de un registro, es decir, informa al programador si el resultado de una operación no está dentro del rango de magnitud permitido, según se trate de un valor entero, real o de doble precisión. Para lo cual - consulta un indicador, llamado de "overflow", que se activa cuando en una operación se produce "overflow", "underflow", o ambos.

Overflow.- Se produce este hecho si el resultado de una operación excede la magnitud máxima permitida para ese tipo de dato. El indicador de "overflow se activa señalando esta circunstancia, y el resultado que se almacena es el correspondiente al valor máximo permitido para ese tipo de dato.

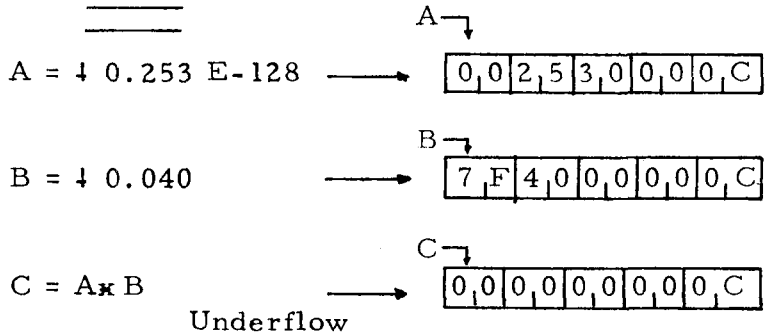
Ejemplo.-



Underlow.- Se produce si, el resultado de una operación, siendo diferente de cero, es menor que la mínima magnitud permitida para ese tipo de dato. El indicador de " overflow" se activa señalando este hecho, y el resultado que se almacena es cero.

Interesa señalar que sólo se puede producir "underflow" en operaciones con datos reales o de doble precisión.

Ejemplo.-



El formato de llamada es:

CALL OVERFL (nombre de variable entera)

Donde.-

⟨variable entera⟩ : Toma el valor 1, si se ha producido una condición de "overflow" ó ambas condiciones de "overflow" y "underflow".

. Toma el valor 2, si no se ha producido - ni "overflow",ni "underflow".

. Toma el valor 3, si sólo se ha producido "underflow".

Nota.-
El indicador de "overflow" se desactiva automáticamente después de cada llamada a la subrutina OVERFL.

Ejemplo.-

```
IA =↓ 999999990
IB =↓ 12
IC = IA ↓ IB
CALL OVERFL (ISO)
```

En este ejemplo la variable ISO toma el valor 1, por haberse producido previamente una condición de "overflow"

b) SUBRUTINA⟨DVCHK⟩

Cuando se realiza una división por cero se activa un "indicador de división", y el resultado de la operación es cero.

La subrutina DVCHK testea este "indicador de división" e informa al programador si se ha producido o no una división por cero.

El formato de llamada es:

CALL DVCHK (nombre de variable entera)

Donde.-

⟨variable entera⟩ : Toma el valor 1, si se ha intentado una división por cero. En los demás casos toma el valor 2.

Nota.-
El "indicador de división" se desactiva después de cada llamada a la subrutina DVCHK.

Ejemplo

```
A = Ø
B = 14.2
C = B/A
CALL DVCHK (ISD)
D = B/2
CALL DVCHK (ISD)
```

En este ejemplo la variable ISD toma el valor 1 en la primera CALL DVCHK; y el valor 2 en la segunda CALL.

c) SUBRUTINA ⟨ERROR⟩

Esta subrutina permite detectar si se produce un error en una función. La subrutina testea el estado de un "indicador de error de función" y seguidamente lo desactiva, caso de estar activado.

El formato de llamada es:

```
CALL ERROR (nombre de variable entera)
```

Donde.-

<variable entera>: Toma el valor 1, si se produce un error, es decir, el "indicador de error de función" está activado. En los demás casos toma el valor 2.

Nota.-
Si a una función externa estandar o intrínseca se la referencia con un argumento que no está dentro del rango -- permitido, se produce un mensaje de error y finaliza la ejecución del programa.

d) SUBRUTINA <ERROR 1>

Se usa en conjunción con la subrutina ERROR. Activa el "indicador de error de función", que luego se puede testear con la subrutina ERROR.

El formato de llamada es:

```
CALL  ERROR 1
```

Ejemplo

```
      Z = XRAY (Q)
      CALL ERROR (I)
      CO TO (30, 40), I
40    CONTINUE
      ═══
30       rutina por condición de error
      ------------------
      FUNCTION XRAY (B)
      IF (B) 10, 20, 10
20    CALL ERROR 1
      RETURN
10    CONTINUE
      ------------------
```

e) SUBRUTINA ⟨SLITE⟩

Activa uno de los cuatro indicadores especificados por el valor del argumento ⟨I⟩ . El testeo de estos indicadores lo realiza la subrutina SLITET, que se describe posteriormente.

El formato de llamada es:

```
CALL SLITE (I)
```

Donde.-

⟨I⟩ : Es una expresión entera.

Si I = 0, se desactivan los cuatro indicadores

Si I = 1, 2, 3, ó 4; se activa el indicador correspondiente.

Si I = -1, -2, -3, ó -4; se desactiva el indicador correspondiente.

f) SUBRUTINA <SLITET>

Se utiliza en conjunción con la subrutina SLITE, y permite al programa determinar el estado de un indicador específico.

El formato de referencia es: .

```
CALL SLITET ( I, J)
```

Siendo.-

<I> : Expresión entera. Puede tomar los valores 1 a 4

<J> : Nombre de una variable entera

Descripción.- Esta subrutina, testea el indicador correspondiente al valor de la expresión <I> . Si el indicador está activado la variable <J> toma el valor 1, en caso contrario toma el valor 2.

El estado de los indicadores no se altera por la ejecución de esta subrutina.

Ejemplo.-

```
CALL SLITE (3)
I = 2
I1 = 5
CALL SLITET (I1-I, J1)
CALL SLITE (Ø)
```

En este ejemplo, 1º se activa el indicador 3, luego se pregunta por él con la CALL SLITET recibiendo J1 el valor 1; y posteriormente se desactivan los 4 indicadores.

g) SUBROUTINE <SSWTCH>

En el ordenador UNIVAC 9400 se pueden simular por software 8 llaves (estas llaves existían físicamente en la consola de operación, en los ordenadores de diseño más antiguo: Ej. IBM 1620) que se activan con la sentencia de control // SET UPSI; y se pregunta por su estado con esta subrutina.

La forma de llamada es:

```
CALL SSWTCH (I,J)
```

Siendo.-

<I> : Una expresión entera cuyo valor está comprendido entre 0 y 7.

<J> : El nombre de una variable entera.

Descripción.-

Cada valor de <I> representa una "llave" de contenido binario, es decir, 'ON' ⟶ activada y 'OFF' ⟶ desactivada. Si la llave que indica <I> está activada, la variable <J> toma el valor 1. en caso contrario toma el valor 2.
El estado de las llaves no se altera por la ejecución de esta subrutina.

h) SUBRUTINA <DUMP>

Permite vaciar, en un momento dado, la memoria principal asignada al programa. La ejecución del programa continua después del vaciado.

La forma de referenciarla es:

```
CALL DUMP
```

i) SUBRUTINA <EXIT>

Esta subrutina termina el programa. Es equivalente a la sentencia STOP del FORTRAN.

La forma de referenciarla es:

```
CALL EXIT
```

j) SUBRUTINA <FETCH>

Tanto la subrutina FETCH, como la LOAD se utilizan cuando se trata de programas "segmentados", y se van introduciendo los segmentos en memoria según precisa el programador. Esta subrutina carga un programa principal en memoria, desde disco y transfiere el control a ese programa.

El formato de llamada es:

```
CALL FETCH (nombre-segmento)
```

Donde.-

<nombre-segmento> :Es un nombre simbólico, encerrado entre apóstrofos, con un máximo de 8 caracteres.

Ejemplo

```
DOUBLE PRECISION NOM/'PERIODO'/
CALL FETCH (NOM)
        ó
CALL FETCH ('PERIODO')
```

El control se transfiere al programa-segmento PERIODO, después de cargarlo en memoria.

k) SUBRUTINA <LOAD>

Carga un subprograma en memoria, cuando lo necesita el programa unidad.

A diferencia de FETCH, el proceso continúa en la instrucción siguiente a la posición de llamada del programa unidad que lo carga y no bifurca al subprograma cargado.

La forma de referenciarla es:

CALL LOAD (nombre-segmento)

Donde.-

<nombre-segmento> :Tiene las mismas características que en CALL FETCH.

CAPITULO 8

BLOCK DATA

8.1. SUBPROGRAMA BLOCK DATA

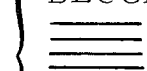
Un subprograma "block data" es una unidad de programa, que se utiliza para inicializar valores a zonas de memoria situadas en bloques comunes etiquetados.

La primera sentencia de un subprograma "block data" tiene que ser la sentencia BLOCK DATA y la última la sentencia END, y solamente puede contener las sentencias no ejecutables:

- DATA
- EQUIVALENCE
- DIMENSION
- COMMON
- Instrucciones de tipificación explícita: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL.
- IMPLICIT (una como máximo)

El orden de estas instrucciones viene especificado en la tabla 1.1.

Su estructura es:

BLOCK DATA

END

8.2. SENTENCIA BLOCK DATA

Formato.-

BLOCK DATA

Descripción.- Sirve para indicar el principio de un subprograma "block data", por lo que debe ser la primera sentencia de este tipo de subprogramas, y solamente puede aparecer allí.

Como esta sentencia no lleva argumentos ni nombre de subprograma, el subprograma "block data" no puede referenciarse desde otra unidad de programa: lo que debe hacerse es, una vez compilado, introducirlo en la colección en el lugar que se crea conveniente. Cuando un programa esta segmentado, y una fase contiene un subprograma "block data" los bloques comunes etiquetados se reinicializan cada vez que se cargue esta fase.

Ejemplo

```
BLOCK DATA
INTEGER B2, BR4
DOUBLE PRECISION E
LOGICAL LR
COMMON/BLK1/LR(2)/BLK2/B2, BR4(2), E
DATA LR/2*.TRUE./, B2/-1/, E/2.7182D01/
END
```

8.3. EJEMPLO DE UTILIZACION DEL BLOCK DATA

Una empresa financiera que trabaja al 16 % de interés y con unos periodos de financiación de 1 a 60 mensualidades, desea un programa que, partiendo del capital financiado y del plazo de amortización, calcule las primas de amortización.

La fórmula a aplicar es:

$$A = C * \frac{(1 + r)^n * r}{(1 + r)^n - 1}$$

Donde:

A	Prima de amortización
C	Capital financiado
r	Tanto por uno mensual, en este caso $\frac{0.16}{12}$
n	Número de periodos de amortización

Se ha decidido que, en lugar de calcular la fórmula para cada caso, se calculen a parte los 60 coeficientes que son los que se incorporan al BLOCK DATA.

$\frac{(1 + r)^n * r}{(1 + r)^n - 1}$ correspondientes a los 60 periodos de amortización y se almacenen en una tabla, reduciéndose así el cálculo a una simple multiplicación.

El subprograma block data podría ser:

```
      BLOCK DATA
      COMMON/TABLA/COEFIC (60)
      DATA COEFIC/1.013333, .051002 , ..........
     *....., .02431/
      END
```

y el programa Fortran:

```
      COMMON/TABLA/COEF (6Ø)
      READ (5, 1ØØ) NPER, CAPI
      IF (NPER.LE.Ø.OR.NPER.GT.6Ø) GO TO 1
1ØØ   FORMAT (I2, F1Ø.Ø)
      AMORT = CAPI * COEF(NPER)
      WRITE (6, 1Ø1) AMORT
1Ø1   FORMAT (3X, F1Ø.2)
      STOP
1     WRITE (6;1Ø2)
1Ø2   FORMAT (3X' PERIODOS DE AMORTIZACION FUERA DE
     *RANGO')
      STOP
      END
```

CAPITULO 9

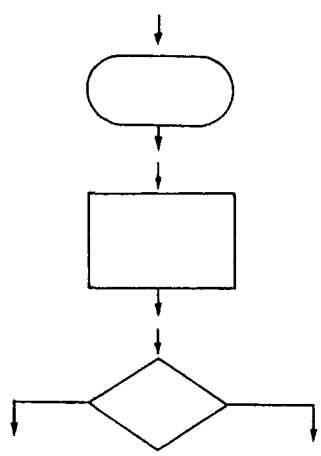
CONSTRUCCION DE PROGRAMAS (*)

9.1. INTRODUCCION

Dada la gran importancia que, en los Departamentos de Programación de los Centros de Cálculo, van teniendo las actuales técnicas de estructuración de programas, se ha considerado importante dedicar a las mismas éste Capítulo. En la Sección 9.2. se presentan los diagramas básicos utilizados en la programación estructurada y se desarrolla la "técnica N-S" u "Organigrama de Chapin". En la Sección 9.3. y bajo el título de "Fortran estructurado" se intenta aplicar estas técnicas al Fortran.

9.2. ORGANIGRAMAS ESTRUCTURADOS

El organigrama es la representación gráfica de un algoritmo. La programación no estructurada se basa en la técnica del "IF-GO TO", en ella se utiliza, lo que ha venido a llamarse, organigrama clásico. Estos organigramas constan de los siguientes tipos de figuras o bloques básicos:



Bloque de comienzo o final

Bloque de proceso o acción

Bloque de decisión

(*) Programación con el lenguaje PASCAL, F.J. SANCHIS LLORCA y A. MORALES LOZANO (Cap. 10). Editorial Paraninfo.

Los bloques de "comienzo o final" del programa o subprograma tienen la misma figura y llevan dentro la palabra COMIENzo o FIN respectivamente.

El bloque de "proceso o acción" es un rectángulo que dentro - lleva una descripción de la acción que realiza.

El bloque de "decisión" es un rombo. Es el único bloque que tiene dos salidas, escogiendo una u otra según que sea verdadera o falsa la condición contenida en su interior.

Estos tres tipos de bloques son los esenciales para representar cualquier programa. Con todo mencionamos algunos bloques adicionales que suelen usarse para una mayor documentación del programa:

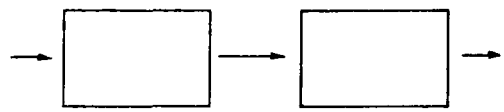
- Entrada o salida por fichas.
- Salida por impresora.
- Entrada o salida en disco magnético.
- Entrada o salida en cinta.
- Entrada o salida por teclado.
- Llamada a otro módulo: subprograma.

de los que no damos las figuras correspondientes.

Estos organigramas clásicos están estandarizados y existen unas plantillas que facilitan su uso.

Böhm y Jacopini demostraron que cualquier programa se - puede diseñar empleando sólo tres tipos de bloques (denominados diagramas privilegiados):

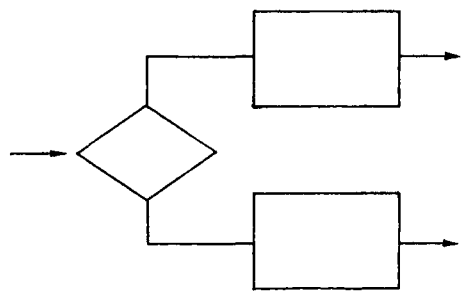
El de acción secuencial "begin - end", Figura 1, que no modifica el flujo de ejecución del programa (sentencia de asignación, de entrada o salida, etc.).



Secuencial

Figura 1

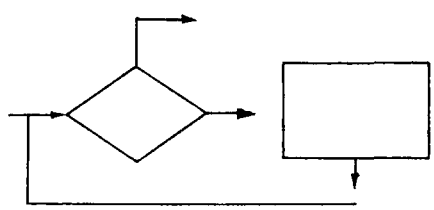
El condicional "if - then- else", Figura 2, que según sea el valor de la comparación o relación ejecutará una u otra de las alternativas.



if - then - else

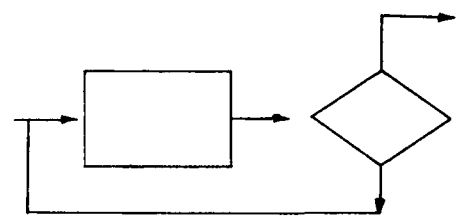
Figura 2

El de repetición o "bucle" "do - while" ó "do-repeat", Figuras 3 y 4, que permite realizar una iteración, cero o más veces.



do - while

Figura 3



do - repeat

Figura 4

Se denomina Programación estructurada a un conjunto de técnicas para desarrollar programas que sean:

- fáciles de depurar y de poner a punto
- legibles por una persona
- modificables o mantenibles, etc.

si bien el concepto de programación estructurada se ha ido haciendo cada vez más impreciso, a nosotros nos interesa destacar el aspecto mencionado, de Böhm y Jacopini, de los tres bloques básicos de un programa: el secuencial, el condicional y el de repetición; esta noción se la llama teorema fundamental de la programación estructurada y es la que vamos a usar en lo sucesivo.

9.2.1. DESCRIPCION DE LOS BLOQUES DEL ORGANIGRAMA DE CHAPIN.

A continuación tratamos una nueva forma de hacer organigramas (llamados "organigramas de Chapin") en programación estructurada. Este método no facilita, de entrada, la tarea del programador; de hecho le pide una mayor comprensión. Pero, una vez pasada esta barrera, los beneficios que se obtienen en la depuración, documentación y mantenimiento de los programas compensan con creces este esfuerzo inicial.

Creemos que esta forma de hacer organigramas es una cosa muy trivial, pero es muy concreta, y, como se irá viendo, favorece (e incluso obliga) la modularidad de los programas, su tamaño reducido, su programación estructurada; en fín, una serie de objeti-

vos dispersos que se consiguen también actualmente de otras formas, pero con un esfuerzo mayor.

Veamos ahora la descripción de los bloques estructurados según la modalidad de Chapin.

El bloque secuencial es un rectángulo, Figura 5, que se puede emplear para representar una sentencia de asignación o de entrada - salida, así como la llamada a un subprograma. Se pueden añadir características para - distinguir entre sí las tres sentencias mencionadas. La más importante es quizás la llamada de un procedimiento para lo que se coloca una elipse interior al -- rectángulo con el nombre del procedimiento llamado.

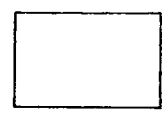


Figura 5

La secuencia de ejecución va siempre de arriba hacia abajo; no hacen falta flechas descendentes; por ejemplo, en la Figura 6 se tienen tres procesos a ejecutar en el orden indicado (en los organigramas clásicos estarían unidos por flechas verticales hacia abajo).

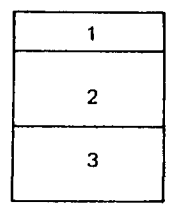


Figura 6

El bloque condicional "if - then - else" se representa como se indica en la Figura 7 y se le llama también - bloque o símbolo de decisión, en el que el triángulo central contiene una expresión lógica y los triángulos izquierdo y derecho contienen una "T" y una "F" respectivamente, para representar los dos bloques de - proceso alterantivos (parte "then" y parte "else").

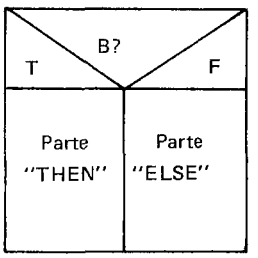


Figura 7

La estructura formada por bloques "if - then - else" resalta muy claramente las diferentes posibilidades lógicas existentes. Por ejemplo, en la Figura 8 el proceso escrito en el lugar de cada uno de los dígitos 1 a 7 respondería a las expresiones lógicas siguientes:

1: A y B
2: A y no B y C
3: A y no B y no C
4: No A (y luego seguirá explorando D y, en su caso, E).
5: No A y D y C
6: No A y D y no E
7: No A y no D

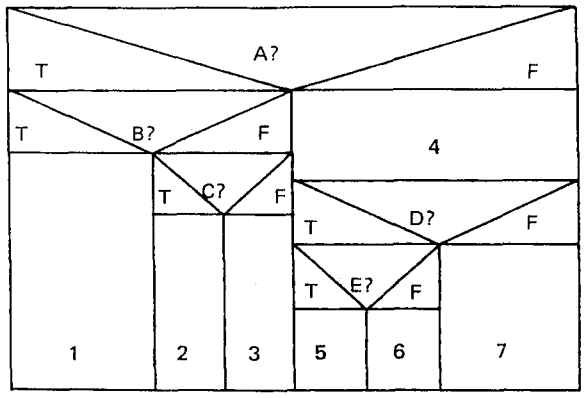


Figura 8

El bloque "do - while" se representa gráficamente como indica la Figura 9. El cuerpo abarcado por la sentencia - se aprecia perfectamente y puede contener a su vez varias sentencias, así como otros bloques "do - while".

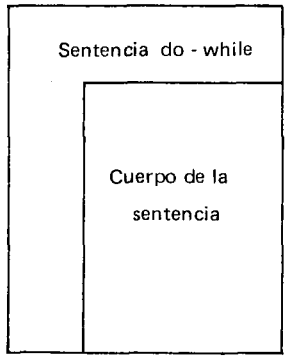


Figura 9

El bloque "do -repeat" cuya estructura lógica viene dada en la Figura 4 tiene en el "diagrama N - S" un grafismo similar al del bloque "do - while". La representación es entonces la de la Figura 10. (comparese con la Figura 9).

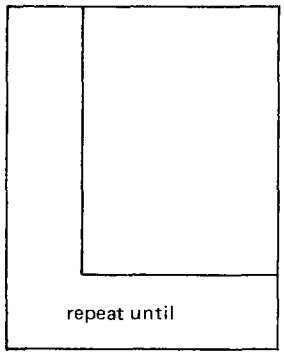


Figura 10.

Debido a su amplia utilización vamos a introducir dos bloques más. El bloque "do - for" que, en programación estructurada es la combinación de los diagramas "secuencial" y "do - while", y se representa en diagrama estructurado como se indica en la Figura 11.

Vc - Variable control
Vi - Valor inicial
Vf - Valor final
K - Incremento variable de control
f - Tratamiento

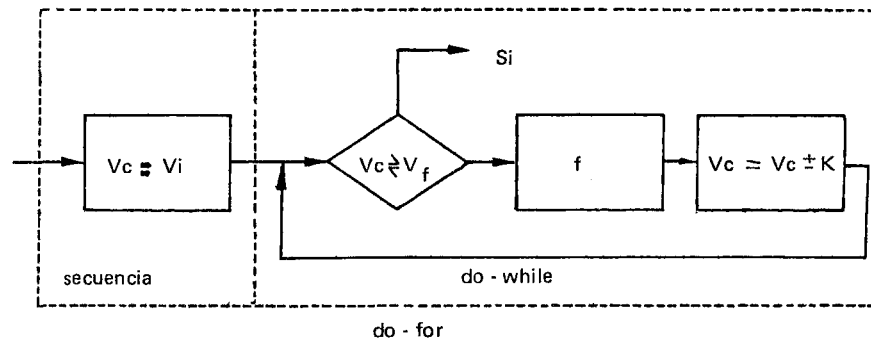


Figura 11

En el organigrama de Chapin se utiliza para el "do-for" el mismo grafismo del bloque "do - while": escribiremos "for" (y su ley o progresión aritmética), en vez de "while".

El bloque "case" representa, en programación estructurada, un "if - then - else" encadenado y se representa como se indica en la Figura 12.

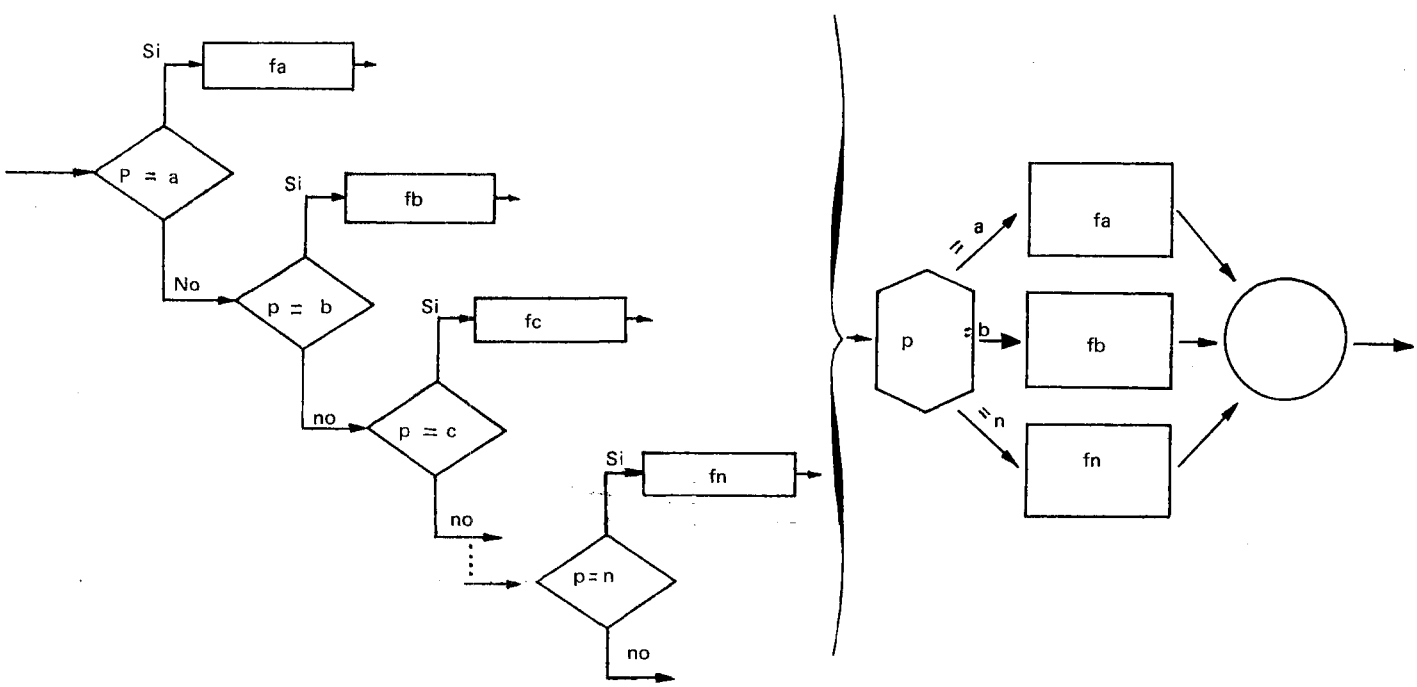


Figura 12

En el organigrama de Chapin el "case" se puede representar con el bloque de la Figura 13 en la que los procesos a la izquierda del vértice del triángulo son las posibles salidas según el valor de la variable de control y el bloque de la derecha es la salida por defecto.

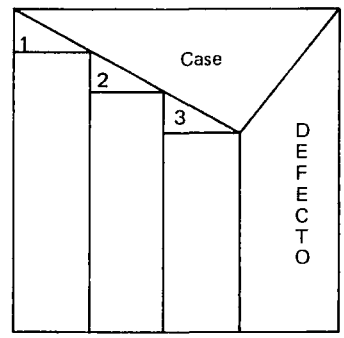


Figura 13

Según el lenguaje de programación empleado, se puede introducir un símbolo (Figura 14) para representar un bloque BEGIN-END - (tal como aparece en los lenguajes PASCAL, PL/1, ALGOL ó SIMULA).

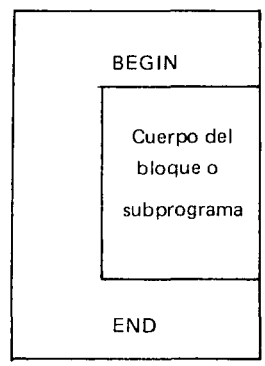


Figura 14

La ausencia, en lo visto hasta ahora, de la sentencia GOTO hace que los programadores se acostumbren a prescindir de ella. D.E. Knuth afirma que a veces el suprimir una sentencia GOTO obliga a forzar la programación en exceso; entonces se puede mantener el GOTO llegando a un compromiso. La sentencia GOTO en estos organigramas se puede simbolizar con una flecha horizontal hacia la derecha y la etiqueta correspondiente.

9.2.2. VENTAJAS DE LOS ORGANIGRAMAS ESTRUCTURADOS

La construcción de un organigrama de este tipo nos ayuda a ver (mejor que con los organigramas clásicos) la estructura y enlace entre los módulos. Además, como no es facil introducir más de 20 símbolos en una página (y no hay conectores de página), se consiguen programas o procedimientos cortos (que es uno de los objetivos de la programación estructurada); pero si el programa o módulo resultara extenso, es más fácil, a la vista del organigrama, el aislar un trozo bastante grande para tomarlo como un subprograma independiente, e incluso se facilita el conseguir que la parte a separar responda a un aspecto funcional claro y aislado.

Chapin hace notar que sus organigramas son más compactos que los clásicos (en los clásicos una gran parte del espacio lo consumen las flechas y líneas de conexión) ocupan aproximadamente la mitad, debido a que no necesitan dejar tantos espacios para su claridad y facilidad de lectura. Además, su trazado no tiene que guardar proporciones entre sus componentes.

Se puede usar una convención para destacar la mayor o menor importancia de una función o procedimiento: las funciones importantes se colocan más elevadas y hacia la izquierda; las menos importantes (por ejemplo, un tratamiento de errores) se puden poner hacia la derecha y abajo. También se pueden aplicar las elipses con el mismo fin.

Este método favorece el diseño y desarrollo "top down" (o descendente): un programa empieza como un rectán-

gulo vacío que se va llenando desde arriba, y que, como se ha indicado, no puede ser muy grande. Se diferencian perfectamente los tres bloques básicos de programación estructurada (Chapin incluye unas barras oblicuas en la parte vertical de una sentencia de bucle, para destacar mejor su presencia y amplitud).

La recursividad es muy fácil de representar con estos bloques, puesto que basta emplear una elipse con el mismo nombre.

Una cosa que no representan bien estos bloques son los diferentes medios con los que se realizan las entradas/salidas (impreso, ficha perforada, cinta, disco); en esto, los organigramas clásicos son más claros. Con todo, en los sistemas operativos actuales y con la "independencia de dispositivos", cada vez va siendo menos seguro que los registros se lean o escriban en el medio indicado originalmente.

9.3. FORTRAN ESTRUCTURADO

9.3.1. INTRODUCCION

A continuación se menciona una forma muy sencilla de escribir programas estructurados en FORTRAN. Se supone que no se dispone de compilador FORTRAN con comentarios incorporados en la misma línea. Si se dispone (como es el caso del FORTRAN 9400) de dichos comentarios no hace falta preprocesador.

Con fines docentes se estudió la posibilidad de que el FORTRAN fuera estructurado, considerándose como trabajo previo el desarrollar un preprocesador que leyera el programa fuente estructurado FORTRAN y diera como salida un programa fuente FORTRAN normal. Este fuente lo procesaría posteriormente el compilador.

El preprocesador que se comenta aquí es tan sencillo que es más adecuado denominarle programa de utilidad (o a veces "preprocesador", así entre comillas).

9.3.2. FORTRAN ESTRUCTURADO

Existen muchos programas preprocesadores para FORTRAN estructurado, basados todos ellos en una gran modificación del fuente en el que se introducen las sentencias especiales:

* IF - THEN - ELSE
* DO - WHILE
* DO - REPEAT
* CASE

no admisibles en FORTRAN y que el preprocesador deberá convertir. Veamos otra forma de resolver - el problema; para ello tomemos la sentencia

```
I = 150; NUMERO DE CASOS
```

que nos dará error al toparse con ella el compila-- dor FORTRAN. Pero si la convertimos a la

```
I = 150
```

suprimiendo desde el punto y coma inclusive hasta el final de la línea, la sentencia es admisible.

Pues bien, en lo dicho radica la forma para estructurar un programa FORTRAN que se describirá a - continuación.

Para la codificación, tomamos p. ej. las columnas 21 a 25 de las sentencias fuente de entrada y en ellas se pone nuestra información estructurada adicional.

Designamos con B una expresión de Boole y con S -- una o varias expresiones aritméticas:

En la Tabla I se encuentra la codificación de las sentencias de control estructuradas más empleadas. Se puede, como ejercicio, escribir alguna de las siguientes sentencias estructuradas que no figuren en dicha tabla, por ejemplo:

- Sentencias condicionales encadenadas
- Sentencia de bucle y medio

Un comentario no lo acepta el FORTRAN si no lleva una C en la columna 1. Para evitar este hecho se incluye a veces en la Tabla I una sentencia inútil habiéndose escogido la III = 0, (o cualquier otra que no altere el programa, como I = I, etc.).

Columnas del fuente					Observaciones
	6	7	21	25	
			IF		
	.	(.NOT.(		B	1) Sentencia IF
	.	))GOTO 1010;		THEN	sin parte ELSE
				S'	
1010		CONTINUE;	FI		
			IF		
	.	(.NOT.(		B1	2) Sentencia IF
	.	))GOTO 2010;		THEN	completa
				S1	
		GOTO 2020;		ELSE	
2010				S2	
2020		CONTINUE;	FI		
		III = 0;	WHILE		
3010		IF (.NOT.(		B2	3) Bucle
	.	))GOTO 3020;	DO		WHILE
				S3	
		GO TO 3010;	ENDWL		
3020		CONTINUE			
		III = 0;	REPEAT		
4010				S'4	4) Bucle
		III = 0;		UNTIL	REPEAT
		IF (.NOT.(		B3	
	.	))GO TO 4010;	ENDRPT		
			DO 5010 I=1,N		
				S5	5) Bucle
5010		CONTINUE;	END I		DO-COUNT

Tabla I

9.3.3. CONCLUSION

Creemos que este programa de utilidad (o el escribir directamente las sentencias estructuradas en FORTRAN - 9400) resuelve la programación con FORTRAN estructurado. Dicho fuente queda muy legible (la parte derecha es - la que debemos leer) y quizás poco denso por los cierres de sentencia que ocupan una línea adicional.

Como ventaja vemos que la estructura real de la sentencia está ahí y uno no se puede olvidar de ella, ya que la tiene que escribir. Esto es importante para el aprendizaje de - programación estructurada. Se puede combinar, si se desea, con los organigramas estructurados (*).

Algunos detalles menores de implementación son los siguientes:

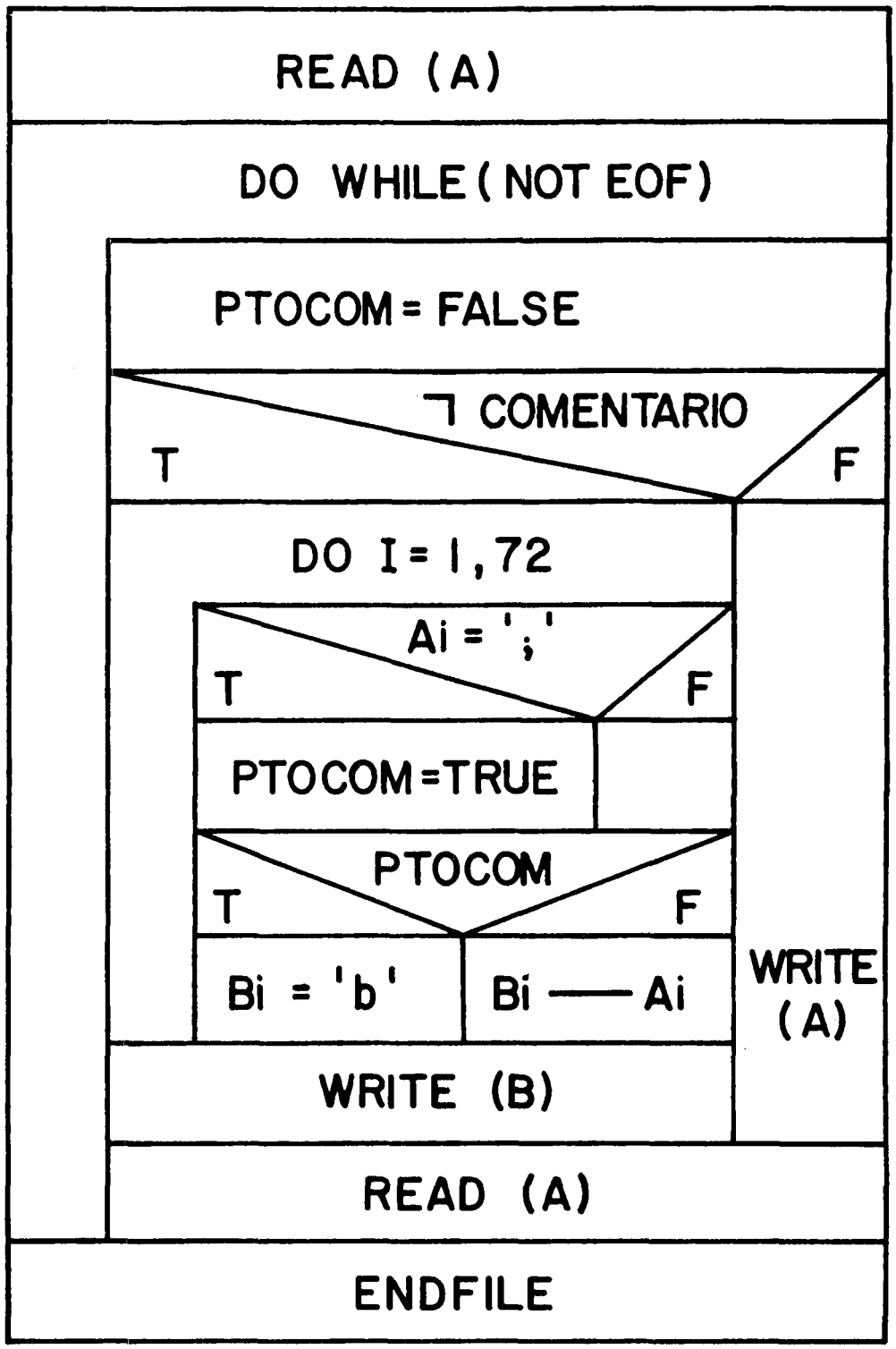
a) Si se carece del carácter punto y coma en perforación o en la impresora empleada, se puede sustituir por -- otro carácter especial que no se use en FORTRAN - (@ de arrobas, caracter sostenido, símbolo de pesetas..)

b) El programa de utilidad que puede construirse pasa a blancos todos los comentarios. Pero si, como es frecuente, el programa fuente leído consta de un principal seguido inmediatamente por sus funciones y subrutinas, entonces el listado resultante del "preprocesador" queda monolítico. Esto se puede obviar introduciendo un separador entre cada END y la FUNCTION ó SUBROUTINE siguiente.

(*) Sanchis Llorca, F, J.: "Organigramas para la programación estructurada". Rev. Proceso de Datos, Nº 73, pags. 17 - 22 (Nov 77).

Cuando el "preprocesador" encuentra este separador (por ejemplo el caracter 1 en la primera columna) salta de página, dando así un listado separado para cada módulo FORTRAN.

Se incluye a continuación utilizando la "técnica N-S" un organigrama estructurado de un programa de utilidad (preprocesador) que suprimiera los comentarios incorporados, si fuera necesario, como se comentó anteriormente.



Organigrama estructurado de un programa para suprimir comentarios.

BIBLIOGRAFIA

- Tharin, M.: "Exercices commentés de programmation en langage FORTRAN" 1972, Masson, París.
- García Merayo, F.: "El lenguaje FORTRAN" 1976, Paraninfo, Madrid.
- Dorn: "Matemáticas y computación con programación FORTRAN".
- Forsythe: "Programación FORTRAN"
- James; Smith; Wolfords: "Métodos numéricos aplicados a la computación digital con FORTRAN".
- Luthe: Lenguaje FORTRAN IV.
- Mc Cracken: "Programacion FORTRAN"
- Mc Cracken: "Programación FORTRAN IV"
- Mc Cracken; Dorn: Métodos numéricos y programación FORTRAN".
- Organic: FORTRAN IV
- Salazar Romero: "FORTRAN, teoría y práctica".
- Smith y Hohnson: "FORTRAN, texto programado".
- Vickers: "FORTRAN IV: un enfoque moderno"
- Sherman, P.: "Programación con FORTRAN". Prentice Hall International.
- Cress: "Programación con WATFOR/WATFIV" Prentice Hall International.
- Parker, J.L., Bohl, M.: "Fortran Programming and WATFIV" SRA, 1973.

- Manning, W.A.; Garnero, R.S.: "A. FORTRAN IV problem Solver", Mc Graw Hill 70.
- Lamoitier, J.P.: "Le Langage FORTRAN", Dunod 71
- Groboillot, J.L.; et al: "Initiation an langage FORTRAN" Dunod 68.
- Mann, R.A.: "An IBM 1130 FORTRAN primer", International texbook Co, 69.
- Plumb: "FORTRAN for an IBM s/360", SRA 68.
- Prager, W: "Introduction to Basic FORTRAN programming and numerical methods", Baisdell, 65.
- UNIVAC: FORTRAN IV, Operating System OS/4 UNIVAC 9400/9480 (UP-76 93)
- IBM: FORTRAN IV Language, Sistem 360/370 GC28-6518
- Dreyfus, M.: FORTRAN IV, Dunod.

Servicio de Publicaciones
del Ministerio de Educación y Ciencia