lenguaje de programación

FORTRAN

2 MAY. 1975

H/ 5062

FORTRAN

lenguaje de programación

Redactado por el
Instituto de Informática

EDITA: Servicio de Publicaciones del Ministerio de Educación y Ciencia
Imprime: R. García Blanco - Avda. Pedro Díez, 3 - Madrid
Depósito Legal M. 25.594 - 1971 - 2.ª edición

Estas lecciones de Fortran fueron dadas por los profesores de Lenguajes del Instituto de Informática durante el curso lectivo 1969-1970, primer curso académico celebrado para Programadores de Aplicaciones.

El nivel del Lenguaje explicado es, en líneas generales, FORTRAN IV Básico adaptable para cualquier tipo de ordenador. Se incluyen, sin embargo, algunas sentencias que no son comunes a todos los ordenadores, por ser imprescindible su conocimiento ya que su uso está muy generalizado.

La teoría aquí expuesta se complementó con las explicaciones dadas en clase por los Profesores: Srta. Acedo, Sr. Becerril, Sr. G. Marcos, Sr. G. Merayo, Srta. G. Mendoza y Sr. Ramiro, y con los ejercicios, problemas y prácticas que los alumnos realizaron a lo largo del curso, utilizando, para llevar a cabo estas últimas, diferentes ordenadores.

TEMA 1

DEFINICION DE UN PROGRAMA FORTRAN.- Un programa FORTRAN es un conjunto de instrucciones escritas con un orden lógico que se emplean para resolver un problema.

Este programa escrito en FORTRAN se llama programa fuente. Existe otro programa llamado COMPILADOR FORTRAN encargado de traducir o compilar cada una de las instrucciones a un lenguaje inteligible para la máquina. Dicho programa traducido se denomina programa objeto.

Para que el compilador "pueda" traducir las sentencias FORTRAN necesita que éstas vengan escritas según unas ciertas reglas cuyo conjunto forma el lenguaje FORTRAN.

Para la escritura de un programa FORTRAN existen unas hojas especiales de codificación en las que han de tenerse en cuenta las siguientes reglas.

De la columna 1 a 5 se escribe un cierto número que identificará a la sentencia que venga a continuación. No importa el orden ni la secuencia y sólo es necesario escribirlo cuando en algún punto del programa tengamos que individualizar o referenciar dicha sentencia.

De la columna 7 a 72 y empezando siempre en la columna 7 se escribe la sentencia que puede llegar hasta la 72 como máximo. Si la sentencia fuera de mayor longitud se puede continuar en la columna 7 de la siguiente línea escribiendo en la columna 6, llamada de continuación, un signo distinto de blanco o de cero.

De la columna 73 a 80 se puede escribir alguna identificación del programa, por ejemplo, secuencia de las fichas, etc. Estas columnas no influyen para nada en la compilación.

Con objeto de documentar el programa se puede escribir una C en la columna 1; de este modo todo lo escrito en las columnas 2 a 80 de la misma línea no influye tampoco en la compilación saliendo simplemente en el listado del programa como comentario o referencia.

ELEMENTOS QUE LO INTEGRAN. - El lenguaje FORTRAN está compuesto por caracteres alfabéticos, numéricos y especiales. Estos se unen para formar palabras y una o más palabras escritas en un cierto orden forman las sentencias o instrucciones.

CONCEPTO DE INSTRUCCION. - Las instrucciones son un conjunto de palabras, signos, etc., que tienen un sentido específico para el compilador si están escritas cumpliendo las reglas que dicho compilador exige.

Las instrucciones se dividen según la función que realizan, en ejecutivas y no ejecutivas.

INSTRUCCIONES EJECUTIVAS. - Son aquellas que obligan al ordenador a realizar alguna operación. Pueden ser:

ARITMETICAS: en ellas el resultado de realizar los cálculos indicados reemplaza al valor actual de la variable escrita a la izquierda del signo igual, por ejemplo, A = B + C.

DE CONTROL: controlan el orden de ejecución de las diferentes instrucciones del programa proporcionando bifurcaciones, etc.

DE ENTRADA Y SALIDA: sirven para llevar a cabo la transferencia de datos desde un dispositivo externo a la memoria principal y viceversa.

INSTRUCCIONES NO EJECUTIVAS. - Se emplean junto con otras sentencias y colaboran en su ejecución, especificando o definiendo variables, funciones, subprogramas.

Se subdividen en:

DE ESPECIFICACION: describen, por ejemplo, cómo es el formato de los datos en una operación de entrada/salida, definen propiedades de las variables, etc.

DE FUNCIONES Y SUB PROGRAMAS: sirven para nombrar o definir funciones y subprogramas.

CONCEPTO DE DATOS. - Son aquellos valores alfabéticos, numéricos o caracteres especiales con los cuales operan las instrucciones del programa. Normalmente no forman parte del programa sino que son leídos cuando se necesitan mediante las correspondientes instrucciones de entrada. Los resultados de las operaciones realizadas con las sentencias son extraídos de la memoria principal al dispositivo deseado mediante las instrucciones de salida.

TIPOS DE DATOS. - Existen en FORTRAN varios tipos de datos:

Enteros
Reales
Doble precisión
Lógicos
Complejos
Literales o alfabéticos.

Algunos compiladores FORTRAN no admiten todos estos tipos de datos como sucede con los complejos y los lógicos.

Los tipos de datos más usuales son los tres primeros, es decir, enteros, reales y de doble precisión, los cuales estudiaremos más profundamente en el tema correspondiente a Constantes y Variables.

TEMA 2

CONSTANTES. - Las constantes son valores que permanecen fijos a lo largo de toda la ejecución del programa. Existen seis tipos.

ENTERAS. - Llamadas también de coma fija. Son números enteros positivos, negativos o nulos que la máquina representa en binario. Ocupan normalmente una palabra de memoria y por lo tanto su valor máximo y mínimo depende de su longitud.

Si dicha palabra es de 32 bits, el valor máximo sería $2^{31} - 1$. El menor será -2^{31}.

En un ordenador cuya palabra es de 36 bits, dichos límites serían $2^{35} - 1$ y -2^{35}, respectivamente.

Nótese que el signo ocupa un bit y que el cero es considerado como positivo.

En FORTRAN las constantes enteras están formadas por una serie de dígitos decimales del 0 al 9 precedidos o no de signo y escritos sin punto decimal.

Ejemplos:

Constantes válidas:

+68
-32
+0
-0

+7284632
784

Constantes no válidas:

35.64	contiene punto decimal
7, 8	la coma no es válida
3E4	la E no es válida
9 348765434563148	excede la capacidad permitida
-34+48, 36	contiene dos signos y coma decimal

REALES. - Llamadas también de coma flotante por el modo de representarse en la máquina. Normalmente ocupan una palabra y por tanto esa longitud condiciona al número. Suelen estar formadas por 7 ó 9 cifras decimales con punto decimal, con o sin signo, que pueden ir seguidas, opcionalmente, de la letra E con una o dos cifras decimales con o sin signo, constituyendo estas últimas el exponente de la potencia de diez a que está elevado el número precedente que recibe el nombre de parte fraccionaria o mantisa. Dicho exponente también se denomina característica.

Son constantes reales válidas:

+93.	
93.0	
93.00	
+0.93E02	($+0,93 \times 10^2$)
0.00093E+5	($0,00093 \times 10^5$)
9300000.0E-05	($9300000,0 \times 10^{-5}$)

Nótese que todas representan el mismo valor. Por su formato la que ocupa el cuarto lugar y debido a la posición del punto se dice que está normalizada, entendiendo por tal, toda constante real acompañada de exponente, con punto decimal al principio y de tal forma que la primera cifra que le sigue es significativa.

Son constantes reales no válidas:

0 -3	no tienen punto decimal
6.34E+256	tiene un exponente de tres cifras.

-64392.694385	más cifras de las permitidas
63.E	le falta exponente.

DE DOBLE PRECISION. - Son análogas a las reales anteriormente estudiadas pero pueden escribirse con más cifras con lo que se obtendrá una mayor precisión. Además el exponente va precedido por una D. El número de cifras puede ser de 16 ó 17 y depende del tipo de ordenador.

Ejemplos:

Válidas:

0.3476943856D+37
0.0D0
641.D-8
0.641D+01
4681012.D-6

Inválidas:

764D3	no tienen punto decimal.
6.54E7	sería válida como real de simple precisión.
0.38.46D-03	contiene dos puntos.
7,56D+4	contiene coma decinal.
6.3D+3542	el exponente tiene más de dos cifras.

LOGICAS. - Son constantes que representan una situación verdadera o falsa y su forma es:

.TRUE.	Verdadero.
.FALSE.	No verdadero o falso.

ALFANUMERICAS. - Pueden representar datos que no sean numéricos. Su longitud depende de la máquina habiendo ordenadores que admiten 256 caracteres y otros sólo 6. También se escriben de diferente manera, bien poniendo los caracteres entre comillas 'PAMPLØNA','CØNSTANTE ALFANUMERICA' o bien anteponiéndoles la letra H (Hollerith) y un número que indica la longitud.

Ejemplos: VIAJE
MALØ
BIEN

COMPLEJAS. - Son números complejos que constan de parte real e imaginaria.

Van escritos entre paréntesis y tanto la parte real como la imaginaria son constantes reales. Ambas partes se separan por una coma.

Ejemplos: (+6.3, 3.E+3)
(-25., +3.4000E-03)
(64.3569, 32.00)
(0., 0.)

El primero de ellos, por ejemplo, significa (6, 3 + 3000i).

TEMA 3

VARIABLES. - Una variable es la representación simbólica de un cierto valor o valores. Realmente representan la dirección de una cierta posición de memoria.

Han de cumplir las siguientes reglas: (1) Constar de 1 a 6 caracteres alfanuméricos, el primero de los cuales tiene que ser forzosamente alfabético y (2) el tipo de una variable debe ser el mismo que los datos que representa. Existen cinco categorías distintas.

ENTERAS. - No necesitan declararse mediante ninguna sentencia especial sino que simplemente se toman como tales aquellas variables cuya primera letra está comprendida entre la I y la N, ambas inclusive.

Contienen un valor numérico de coma fija y ocupan normalmente una palabra.

Son variables enteras las siguientes:

J1234
IRADIØ
MØNTE
KARGA
K12345
I
J
K

Existe una instrucción que permite hacer que un nombre, cualquiera que sea su primera letra, sea tomado como variable entera. La instrucción es INTEGER.

Así las variables ABDC
RADIØ
PMM
XM001

no serían enteras a menos que se escribiese la instrucción:

INTEGER, ABCD, RADIO, PMM, XM001

REALES. - Tampoco necesitan declararse con ninguna instrucción especial. Basta con que la primera letra sea distinta a las comprendidas entre la I y la N. Contienen constantes de coma flotante y ocupan una palabra.

Son variables reales: ABCD
RADIØ
PP
P1
QRSTUV
Q1
Q2

Existe también una instrucción de especificación que permite hacer que un nombre, cualquiera que sea su primera letra, sea tomado como variable REAL. La instrucción es REAL.

Ejemplo: REAL I, J, H, I1, K1, hará que dichos nombres representen variables reales.

DE DOBLE PRECISION. - Contienen constantes de doble precisión. Ocupan una doble palabra y necesitan la instrucción DØUBLE PRECISIØN para ser declaradas explícitamente.

Ejemplo: DØUBLE PRECISIØN A, B1, O2, I37, DATØ, Z240

LOGICAS. - Contienen una constante lógica y ocupan una palabra. Necesitan la instrucción de especificación LØGICAL.

Ejemplo: LØGICAL, ARBØL, MESA, VERDAD, FALSØ

COMPLEJAS. - Contienen constantes complejas y ocupan una doble palabra. Necesitan la instrucción de especificación CØMPLEX.

Ejemplo: CØMPLEX RESIS, ZO, ZR, I

TEMA 4

CONJUNTOS DE VARIABLES. - En los capítulos anteriores hemos estudiado las variables que se suelen denominar simples o elementales; sin embargo, el FORTRAN, como otros muchos lenguajes, permite declarar explícitamente grupos de variables identificadas bajo un mismo nombre, pudiéndose referir a cada una de las variables que componen el grupo o conjunto mediante la utilización de subíndices.

Supongamos un conjunto denominado IRADIØ que consta de 5 variables cuyos valores son, respectivamente, 200, 300, 400, 500, 600.

IRADIØ (1) representa la primera variable del conjunto cuyo valor es 200.
IRADIØ (2) la segunda cuyo valor es 300.
IRADIØ (3) la tercera cuyo valor es 400.
IRADIØ (4) la cuarta cuyo valor es 500.
IRADIØ (5) la quinta cuyo valor es 600.

Así pues, IRADIØ representa todo el conjunto, mientras que IRADIØ (I) es la variable I-ésima del conjunto. El conjunto anterior se dice que es de una sola dimensión y consta de cinco elementos.

Del mismo modo existen conjuntos de dos, tres, etc., dimensiones, hasta un máximo de siete en los compiladores FORTRAN más potentes.

Supongamos un conjunto de dos dimensiones denominado TABLA. La primera dimensión, denominada normalmente filas tendrá cuatro elementos y la segunda, que se denomina columnas, tendrá dos. El total de elementos será 4 x 2 = 8. Dichos elementos son:

	Col. 1	Col. 2
Fila 1	TABLA (1, 1)	TABLA (1, 2)
Fila 2	TABLA (2, 1)	TABLA (2, 2)
Fila 3	TABLA (3, 1)	TABLA (3, 2)
Fila 4	TABLA (4, 1)	TABLA (4, 2)

Así el elemento numérico TABLA (I, J) será el elemento del conjunto TABLA cuya fila es I y cuya columna es J.

Análogamente el elemento COSTE (3, 2, 8, 5,) sería la variable elemental que ocupa el tercer elemento de la primera dimensión, el segundo de la segunda, el octavo de la tercera y el quinto de la cuarta.

Cada uno de los elementos de un conjunto representados de esta forma se dice que es una variable suscrita.

DEFINICION DE SUBINDICE. - Un subíndice es pues un cierto número que individualiza un elemento dentro de un conjunto. Este número puede adoptar diversas formas dependiendo de los tipos de máquinas que se emplean siendo los más conocidos los que a continuación se exponen.

REGLAS DE FORMACION DE SUBINDICES. - Un subíndice puede representarse en una de las formas siguientes, donde V representa una variable entera sin signo y C y C' constantes enteras sin signo:

V
C
V+C
V-C
C*V
C*V+C'
C*V-C'

Nótese que no son permitidos subíndices de la forma C+V, C-V, V*C, etc. Han de tenerse en cuenta, además, las tres normas siguientes: (1) todo subíndice debe estar expresado en modo de coma fija; (2) cuando existan varios subíndices, éstos deben ir separados por comas y (3) un subíndice no puede contener, a su vez, otro subíndice.

Ejemplos:

Variables suscritas válidas

MATRIZ (I, J)
TABLA (19)
M (K-5)
A (5*I)
OMEGA (4*J-7)

Variables suscritas inválidas

ARRAY (-1)	no está permitido signo
COSTE (A +2)	A es una constante real
MATRIZ (-7*J)	signo no permitido
MATRIZ (J*7)	estaría permitido escribir MATRIZ (7*J)
TOTAL (0)	no se permite el valor cero.

DECLARACION DEL TAMAÑO DE UN CONJUNTO. - Para declarar explícitamente un conjunto debemos dar su nombre, indicar cuantas dimensiones tiene y expresar el número máximo de elementos dentro de cada dimensión. Todo esto se logra empleando la sentencia DIMENSIØN, seguida del nombre simbólico del conjunto y escribiendo entre paréntesis su tamaño.

Así la sentencia,

DIMENSIØN A (8)

reservaría en memoria un conjunto que se denomina A, de una sola dimensión y con 8 elementos.

De igual manera,

DIMENSIØN TABLA (4, 4, 2)

reservaría un conjunto de nombre TABLA, con tres dimensiones, cada una de las cuales con 4, 4 y 2 elementos, respectivamente.

Por último,

DIMENSIØN COSTE (5, 3)

haría la reserva para un conjunto llamado CØSTE que tiene 2 dimensiones: la primera dimensión con 5 elementos y la segunda con 3.

La sentencia DIMENSIØN debe preceder, en el programa fuente, a la primera aparicion de la variable a la cual dimensiona.

DISPOSICION DE LOS CONJUNTOS EN MEMORIA. -Los conjuntos se almacenan normalmente en posiciones de memoria ascendente, y de forma tal que el valor del primer subíndice se incrementa con mayor rapidez que el valor del segundo o restantes.

Ejemplos:

El conjunto anterior denominado A, de una dimensión y 8 elementos, aparecería en memoria de la siguiente manera:

A (1) , A (2), A (3), ..., A (7), A (8)

Análogamente el conjunto declarado con DIMENSIØN CØSTE (5, 3) se almacenaría del siguiente modo:

CØSTE (1. 1)	CØSTE (2, 1)	CØSTE (3,1)	CØSTE (4, 1)
CØSTE (5,1)	CØSTE (1, 2)	CØSTE (2, 2)	CØSTE (3, 2)
CØSTE (4, 2)	CØSTE (5, 2)	CØSTE (1, 3)	CØSTE (2, 3)
CØSTE (3, 3)	CØSTE (4, 3)	CØSTE (5, 3)	

Es decir, que varía más rápidamente el primer subíndice hasta su valor máximo; luego se incrementa el segundo en una unidadvolviendo a incrementarse el primero hasta su valor máximo y así sucesivamente hasta que ambos subíndices lleguen a alcanzar los valores más altos.

NOTA. - El tamaño de un conjunto (número entre paréntesis) debe indicarse mediante una constante entera sin signo.

TEMA 5

CONCEPTO DE EXPRESION. - Expresión FORTRAN es un conjunto de constantes, variables, operadores y funciones, separados por paréntesis y/o símbolos de operación, escritos en un orden aceptado por el compilador y de modo que formen una expresión con significado matemático.

Los símbolos de operación u operadores aritméticos son los siguientes:

+	para indicar	la	SUMA
−	"	"	RESTA
*	"	"	MULTIPLICACION
* *	"	"	POTENCIACION
/	"	"	DIVISION

REGLAS DE FORMACION DE LAS EXPRESIONES. -

a. Todas las constantes o variables deben estar separadas por un operador.

b. Nunca pueden ir juntos dos operadores; éstos deben separarse por paréntesis.

c. Debe existir el mismo número de paréntesis de apertura que de cierre.

d. Las cantidades que formen la expresión no necesitan ser del mismo tipo, es decir, pueden estar integradas por cantidades reales y enteras.

e. Los exponentes pueden ser, en general, números enteros o reales.

Ejemplos: En FORTRAN las expresiones que siguen se escribirán como se indica:

$F = m.a$	F = M* A
$Y = \frac{A}{B} + \frac{B}{C}$	Y = A/B + B/C
$Y = \frac{A + B}{C + D}$	Y = (A+B)/(C+D)
$V = \frac{4}{3} \pi R^3$	Y = 4/3*3.1415*R **3
$Z = \frac{V}{I} 10^{(3+H)}$	Z = V/I*10** (3+H)

Son expresiones FORTRAN válidas:

VØLTS / AMPERS	(expresión real)
FUERZA**2 + 3*MASA	(expresión mixta)
A+B-C* D** E	(expresión real)
A + (B-C) * D* (3** 48 +C)	(expresión mixta)

Son expresiones no válidas:

A +0. 32E+27+ -34	dos operadores juntos
P-Q, I37	coma ilegal
6-3+(43*-2)	dos operadores juntos
A+(B-C)) -37	falta o sobra un paréntesis.

ORDEN DE CALCULO. PRIORIDAD DE LOS OPERADORES. - Cuando en una expresión no se emplean paréntesis la prioridad con que se realizan las operaciones en ella contenidas es la siguiente:

Prioridad o nivel máximo	** Potenciación.
2ª Prioridad	* y / Multiplicación y división.
3ª Prioridad	+ y - Suma y resta.

Cuando en la misma expresión se encuentren operadores de la misma prioridad, el orden de cálculo es de izquierda a derecha, realizándose en primer lugar la operación que aparezca primero.

Ejemplos: En la expresión X+Y-Z**3*4/6-12+1 el orden de operaciones es la siguiente:

1º Cálculo de Z**3, cuyo resultado llamamos A.
2º [illegible] 4, cuyo resultado llamamos B
3º B/6, cuyo resultado llamamos C
4º X+Y, cuyo resultado llamamos D
5º D-C, cuyo resultado llamamos E
6º E-12, cuyo resultado llamamos F
7º F+1, que es la evaluación final de la expresión.

UTILIZACION DE PARENTESIS. - La utilización de paréntesis dentro de las expresiones aritméticas lleva consigo la alteración del orden del cálculo en la expresión. Así, cuando existan paréntesis, las operaciones se ejecutarán de tal manera que se realizan en primer lugar las que estén [illegible]rados en los paréntesis más internos siguiendo luego hacia el exterior y teniendo siempre en cuenta las prioridades estudiadas anteriormente.

Ejemplo: Sea la expresión,

A+B**(3-C)+(4/(3**8+D))

La primera operación realizada sería el cálculo del paréntesis más interno, esto es, 3**8+D cuyo resultado llamaremos Z. A continuación se calcularía 3-C cuyo resultado llamaremos Y. Después se calcularía 4/Z cuyo resultado llamaremos X quedando la expresión.

A+B**Y+Z

Finalmente se calcularía B**Y y por último las sumas que quedan, empezando por la izquierda.

SENTENCIAS ARITMETICAS. - La forma general de una sentencia aritmética es

a = b

siendo a una variable real o entera, con o sin subíndice y b una expresión FORTRAN. El resultado de tal expresión sustituye al valor que tenga la variable situada a la izquierda del signo igual.

Ejemplo:

A = A+B

En esta sentencia aritmética el contenido de la variable A se suma al contenido de la variable B y el resultado sustituye al valor anterior de A.

Nótese que las sentencias aritméticas no tienen el carácter de ecuación matemática sino de sustitución.

Son también sentencias aritméticas las siguientes:

ANS (I) = A (J) + B (K)

I = I + 1

A = I	Convierte un número entero en real
A = B	Almacena B en A
I = B	Convierte un número real en entero

REGLAS DE CONVERSION DE LOS TIPOS DE VARIABLES DE LAS SENTENCIAS ARITMETICAS. - Como ya hemos dicho, en una expresión no tienen porque ser todas las variables del mismo tipo. Así, pueden venir mezcladas variables reales, constantes enteras, variables de doble precisión, etc. Entonces el resultado será del tipo de la que tenga un rango o categoría más elevado. Los rangos o categorías son de mayor a menor, los siguientes:

DOBLE PRECISION
REAL
ENTERA

La evaluación de las expresiones se hace siempre por parejas de operandos, pudiendo construirse el cuadro siguiente:

Operando 1

Operando 2	E	R	D
E	E	R	D
R	R	R	D
D	D	D	D

E : Entera
R : Real
D : Doble precisión

La conversión se lleva a cabo automáticamente y el resultado es del tipo de la de mayor rango.

Ejemplos:

I = J*K No hay conversión pues todas las variables son enteras. Se dice que la expresión es de modo entero.

I = A+J J se convierte en real se suma con A y el resultado se asigna a I previa conversión a entero, puesto que la variable I es de ese tipo.

J = A+B A se suma con B, la suma se convierte en entero almacenándose en J.

A = I + A I se convierte en Real, se suma con A y el resultado se almacena en A sin conversión, pues también es real.

TEMA 6

SENTENCIAS DE CONTROL. - Normalmente, las sentencias de un programa FORTRAN se ejecutan en secuencia. Es decir una vez efectuada una de ellas se realiza la que sigue inmediatamente. Vamos a estudiar ahora, una serie de sentencias llamadas de CONTROL que se utilizan para alterar la secuencia normal de la ejecución de las sentencias de un programa FORTRAN.

Estas sentencias son:

GØ TØ	incondicional
GØ TØ	condicionado
IF	aritmético

SENTENCIA GØ TØ INCONDICIONAL. - La forma general de esta sentencia es:

```
GØ TØ  n
```

donde n es una constante entera positiva que indica un número de sentencia. Esta sentencia hace que se transfiera el control a la especificada por el número n. Cualquier sentencia ejecutable que siga inmediatamente a una sentencia GØ TØ debe llevar un número de sentencia; en caso contrario nunca podrá ejecutarse.

SENTENCIA GØ TØ CONDICIONADA. - La forma general de esta sentencia es:

GØ TØ (n_1, n_2 ....., n_n), I

donde:

n_1, n_2, ..., n_n, son constantes enteras positivas que indican números de sentencia. Van separados entre sí por comas y encerrados todos ellos entre paréntesis.

I es una variable en coma fija no suscrita, separada del paréntesis derecho por una coma.

Esta sentencia hace que se transfiera el control a la sentencia numerada n_1, n_2, n_3, ... n_n, según que el valor de I sea 1, 2, ... N, respectivamente. La lista de números de sentencia entre paréntesis puede ser tan larga como se desee y un mismo número puede repetirse tantas veces como sea necesario. Sin embargo, la variable I ha de tener un valor no superior al número de sentencias de que conste la lista. En caso contrario, es decir, si el valor de I está fuera del rango, se ejecuta la sentencia siguiente.

Ejemplo:

GØ TØ (3, 10, 7, 3, 101), I

En este caso para I = 1 e I = 4 el control se transfiere a la sentencia 3; para I = 2 a la sentencia 10; para I = 3 a la sentencia 7 y para I = 5 a la sentencia 101.

Para I = 5 el programa sigue en secuencia es decir se efectuaría la sentencia que sigue inmediatamente a la GØ TØ.

SENTENCIA IF ARITMETICO. - Su forma general es:

IF (a) n_1, n_2, n_3

donde:

a es una expresión aritmética y n_1, n_2, n_3 son números de sentencias ejecutables separadas entre sí por comas.

Esta sentencia hace que se transfiera el control a la sentencia numerada n1, n2 ó n3 según sea el valor de la expresión (a), inferior, igual o mayor que cero. Dos números de sentencia que se indican como opción pueden ser iguales. Por ejemplo con IF (A) 7, 7, 8 se indica al ordenador que ejecute la sentencia 8 solamente si A es estrictamente positivo, es decir, no negativo y no nulo. En caso contrario se ejecutará la sentencia 7.

La expresión aritmética a está sujeta a todas las reglas de jerarquía y modo que son aplicables a las expresiones descritas en las sentencias aritméticas.

TEMA 7

SENTENCIA DO. - Una de las técnicas más importantes de programación es el "bucle" o iteración, es decir, la posibilidad de efectuar varias veces una serie de sentencias. Para ejecutar dichas iteraciones se emplea en FORTRAN la sentencia DØ. Su forma general es:

```
DØ n  i = m1, m2, m3
```

en donde:

- n es el número de una sentencia ejecutable que aparece despues del DØ.
- i es una variable entera no suscrita.
- m1, m2, m3 son o bien constantes enteras sin signo, superiores a cero, o bien variables enteras no subindicadas, sin signo, cuyo valor es superior a cero.

La sentencia DØ hace que se ejecuten varias veces todas las sentencias que le siguen incluida la sentencia cuyo número es n.

Estas sentencias forman el rango del DØ. i es el índice del DØ, es decir, la variable que sirve para contar el número de iteraciones. La primera vez el índice i tomará el valor de m1, la segunda vez el valor de i será m1 + m3 y así sucesivamente hasta que i alcance un valor superior a m2.

Las variables o constantes m1, m2, m3 se denominan valor inicial, límite superior e incremento, respectivamente. El índice m3, es

opcional; si se omite, se supone que su valor es 1. En este caso la coma que precede a m3 debe también omitirse. El número de veces que se ejecutan las sentencias del rango del DØ viene dado por la expresión:

$$\frac{m_2 - m_1}{m_3} + 1$$

Ejemplo:

```
      DØ 20  K = 3, 7, 2
      A ( K ) = B ( K + 1 ) + 3.
   20 C ( K ) = A ( K ) * 2.
```

En este caso y para el primer valor de K, K = 3, se ejecutarán las sentencias A (3) = B (4) + 3. y C (3) = A (3) * 2. . La segunda vez será K = 3+2 = 5, A (5) = B (6) + 3. y C (5) = A (5) * 2.

La tercera y última vez K toma el valor K = 5+ 2 = 7 y se ejecutan las sentencias A (7) = B (8)+3. y C (7) = A (7) * 2. El siguiente valor de K sería K = 7+2 = 9 y por ser superior a m2 = 7 no se volverá a efectuar el bucle, ejecutándose la sentencia siguiente al rango del DØ.

CONSIDERACIONES SOBRE EL EMPLEO DEL DØ. -

1) Los parámetros i, m1, m2, m3 no pueden ser alterados en una sentencia dentro del DØ.

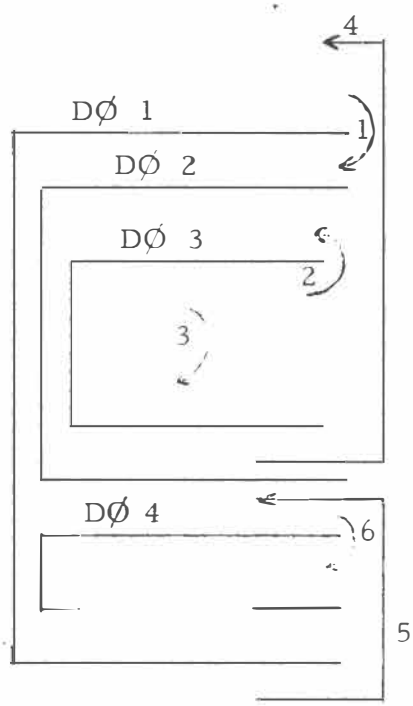
2) La última sentencia del rango de un DØ (sentencia cuyo número es n) no puede ser ni una sentencia GØ TØ, ni una sentencia IF, ni otra sentencia DØ.

3) Puede haber otras sentencias DØ dentro del rango de una sentencia DØ.

4) En cualquier momento puede existir una transferencia fuera del rango de cualquier bucle DØ.

5) Los parámetros i, m1, m2, m3, pueden cambiarse mediante sentencias aritméticas fuera del rango del DØ únicamente si no se vuelve a transferir al rango de la sentencia DØ que utiliza estos parámetros.

En la figura se muestran algunos casos de transferencias válidas e inválidas.



Las transferencias 1, 5 y 6 son incorrectas y las 2, 3 y 4 son válidas.

Las transferencias 1 y 5 se hacen desde fuera del rango de la sentencia DØ 1 y por lo tanto son inválidas. La transferencia 6 es también inválida pues indica una bifurcación desde el DØ 1 al DØ 4.

TEMA 8

SENTENCIA CØNTINUE. - Su forma general es:

```
CØNTINUE
```

La sentencia CØNTINUE es una sentencia ficticia que puede situarse en cualquier parte del programa sin afectar para nada a la secuencia de ejecución.

En el caso de que un DØ deba terminar con una sentencia de bifurcación (GØ TØ, IF, etc.) se utiliza como última sentencia de dicho DØ la sentencia CØNTINUE.

SETENCIA PAUSE. - Su forma general es:

```
PAUSE n
```

en donde n es una constante entera positiva.

Dicha sentencia hace que se efectúe una parada momentánea del programa el cual reanudará su ejecución con una intervención manual del operador. Después de dicha intervención el programa prosigue con la sentencia que sigue inmediatamente a la sentencia PAUSE.

SENTENCIA STØP. - Su forma general es:

```
STØP  n
```

en donde n es una constante entera positiva.

Esta sentencia produce una parada en la ejecución del programa. La diferencia con la sentencia PAUSE estriba en que aunque exista una intervención manual del operador el programa no reanudará su ejecución de nuevo.

SENTENCIA END. - Su forma general es:

```
END
```

La función de la sentencia END consiste en poder identificar la última sentencia física del programa. Esta sentencia no origina ninguna instrucción para el ordenador y únicamente le indica, en tiempo de compilación que el programa ha terminado.

TEMA 9

SENTENCIAS DE ENTRADA Y SALIDA. - Las sentencias de entrada/salida permiten al programador transferir datos entre la memoria central y un fichero cualquiera que se encuentra situado en un dispositivo de entrada/salida.

CONCEPTO DE DISPOSITIVO DE ENTRADA Y SALIDA. -TIPOS DE DISPOSITIVOS. - Se denomina dispositivo a la unidad física donde se encuentra el fichero de datos. El dispositivo se direcciona por un número entero sin signo que está asignado por el compilador, en forma de constante o bien de variable definida con anterioridad a dicha sentencia.

Cuando el programador escribe una sentencia de entrada/salida, con la que introducirá los datos procedentes de un cierto fichero, incluirá en la misma, el número del dispositivo que lo soporta. Así la máquina sabrá dirigirse al fichero indicado.

Existen varios tipos de dispositivos, que corresponden, generalmente, a la organización de los ficheros que soportan. Estos responden a dos categorías:

a) Organización de tipo secuencial, en la que los datos entran o salen de una manera continua, es decir, unos detrás de otros, de manera que si quiere accederse a unos datos que se encuentran en la mitad del fichero, deberá haberse leído toda la información precedente.

Los datos, dentro de un fichero de este tipo, se agrupan en lotes que se llaman bloques o registros físicos.

Entonces un bloque será la unidad de información más pequeña que se transfiere a la memoria central.

Dentro del bloque físico, el programador puede agrupar los datos en unidades más pequeñas llamadas registros lógicos.

El manejo de los bloques corre a cuenta de la unidad física, pero el manejo de los registros lógicos lo lleva a cabo el programador.

Según esto, los dispositivos de acceso secuencial son:

- la lectora de fichas
- la perforadora de fichas
- la impresora
- la cinta magnética.

b) Organización de acceso directo. En este tipo de organización, la información almacenada es direccionable y por lo tanto, se puede acceder a una parte de ella (registro lógico) sin necesidad de leer la información precedente, pues se conoce su dirección dentro del fichero.

Esta organización sólo la soportan las unidades de almacenamiento en disco, tambores magnéticos y tarjetas magnéticas de película fina.

Ejemplo:

Cuando el programador escribe una sentencia para leer unas fichas que tienen perforados los datos del problema,

- Las fichas constituirán un fichero de entrada.
- La lectora es el dispositivo de entrada.
- La organización del fichero será secuencial.
- El bloque o registro físico será la propia ficha.
- El número asociado con el dispositivo podría ser el 1.

La sentencia sería pues:

READ (1, 100) A, B, C, D

siendo A, B, C y D los datos o registros lógicos perforados en la ficha.

La palabra READ da orden de leer al dispositivo (lectora) número 1 y el 100 indica el número de una sentencia de formato que se explicará más adelante.

CONCEPTO DE CONJUNTO DE DATOS. - Se llama conjunto de datos al fichero que contiene los datos que van a tratarse, es decir, una colección organizada de información homogénea.

Esta información estará organizada en bloques para que pueda ser leída o escrita. Estos bloques físicos de información pueden dividirse por el programador en registros lógicos que a su vez pueden ser subdivididos en unidades más pequeñas, variables, conjuntos, etc.

LISTA DE DATOS. - La transferencia de la información contenida en un registro hacia la memoria central, se realiza a través de la lista de datos. En una operación de entrada, los datos pasan del registro externo a los lugares de la memoria especificados por los nombres de variables de la lista.

En una operación de salida, el contenido de las variables que especifica la lista pasará a formar un registro de salida apto para ser escrito.

En el ejemplo anterior

READ (1, 100) A, B, C, D

la lista de datos la constituyen los nombres de variables en coma flotante A, B, C, D y por lo tanto, se infiere que el programa ha leído un registro con cuatro números en coma flotante pertenecientes al conjunto de datos del dispositivo 1 (lectora) y los ha almacenado en cuatro palabras llamadas A, B, C, D.

Si en este registro hubiera, por ejemplo, seis números en coma flotante los otros dos restantes se perderían, puesto que el programador sólo ha especificado cuatro en la orden de lectura.

La lista puede contener nombres de variables o nombres de conjuntos ordenados; pero no permite nombres de funciones y expresiones aritméticas.

Cuando un conjunto ordenado de números aparece en una lista de variables de entrada, la tabla entera se almacenará en memoria, siguiendo el orden de almacenamiento de los conjuntos anteriormente explicado en el tema 4.

CONCEPTO DE FORMATO. - Se denomina formato interno de máquina al modo en que se encuentran almacenados los datos en la memoria principal, es decir, en modo binario. La transmisión de la información desde la memoria a un dispositivo periférico, puede realizarse conservando el formato que acaba de describirse o alterándolo. En este último caso es necesario emplear para ello una sentencia o especificación de formato.

El uso de la sentencia de formato presupone una conversión de los datos expresados en un determinado sistema de numeración (decimal o hexadecimal) a forma binaria o viceversa.

Si se opera sin especificación de formato, la información se transfiere de la memoria principal a los dispositivos de entrada/salida o viceversa en forma binaria, en cuyo caso la longitud de los registros viene determinada por la propia lista de variables a transferir.

TEMA 10

SENTENCIAS DE ENTRADA Y SALIDA EN MODALIDAD SECUENCIAL.- Existen cinco sentencias de este tipo: READ, WRITE, END FILE, REWIND, y BACKSPACE.

Las sentencias READ y WRITE causan la transferencia de los registros de un fichero secuencial; la sentencia END FILE define el final del fichero; las sentencias REWIND y BACKSPACE controlan la posición de lectura del fichero.

SENTENCIA READ. - La forma más general de esta sentencia es:

```
READ ( a, b, END = c, ERR = d ) LISTA
```

donde:

a es una constante o variable entera, sin signo, que representa al dispositivo físico o unidad que se emplea.

b es una constante entera que representa el número de una sentencia FORMAT. El uso de b es opcional.

END = c, siendo c una constante entera correspondiente al número de sentencia a que bifurca el programa cuando llegue al final de los datos. Es opcional.

ERR = d, siendo d el número de sentencia a la que se bifurca si se encuentra algún error en la lectura de los datos. Su empleo es opcional.

LISTA representa la relación de variables que van a leerse.

Combinando estas posibilidades se obtienen las sentencias básicas READ siguientes:

a.

```
READ ( a,b ) LISTA
```

sentencia que lee los datos especificados en LISTA, por la unidad a y con formato b.

La lista de variables puede ser:

a_1) Variables no subindicadas.

Ejemplos:

```
READ ( 1, 100 ) A, B, C
READ ( 2, 3 ) ALFA, BETA, GAMMA
```

a_2) Variables simplemente subindicadas.

Ejemplo:

```
READ ( 1, 100 ) A, B (4), C (7)
```

que origina la lectura de la variable A, la variable B (4) y la variable C (7).

a_3) Variables con más de un subíndice.

Ejemplo:

```
READ ( 1,200 ) (( A ( I, J ), J = 1, 10), I = 1, 20 )
```

que equivale a los siguientes DØ explícitos:

```
    DØ 10 I = 1, 20
    DØ 10 J = 1, 10
10 READ ( 1, 200) A ( I, J )
```

es decir, la sentencia originará la lectura de todos los elementos de la matriz A (20, 10) que se almacenará en memoria según las reglas que siguen las variables suscritas.

a_4) Los apartados a_1, a_2 y a_3 pueden mezclarse entre sí.

Ejemplo:

READ (5, 20) A, B, (C (I), I = 1, 10), D (4)

que originará la lectura de las variables A y B, la lectura de 10 variables C (1), C (2) ..... C (10) y la lectura de la variable D (4), del dispositivo de entrada número 5.

b.

READ (a) LISTA

sentencia con la que se transfieren los datos de LISTA desde el dispositivo a a la memoria, en formato binario.

Ejemplo:

READ (9) A, B, C, D

transferirá los datos del dispositivo número 9, codificados en binario, directamente a memoria.

SENTENCIA WRITE. - Su forma general es:

WRITE (a, b) LISTA

donde:

a es un número entero asignado al dispositivo que se emplea.

b es una constante entera que designa el número de una sentencia FORMAT. Es opcional.

LISTA es la lista de variables como en el caso de la sentencia READ.

Ejemplos:

WRITE (3, 4) A, B, C

causa la escritura en la unidad física de salida 3 de las variables A, B y C con el formato indicado en la sentencia número 4.

```
WRITE ( 3,700 ) ( A ( I ), I = 1,100 )
```

que escribe la ordenación unidimensional A (100) por la impresora (dispositivo número 3) según el formato especificado en la sentencia de formato 700.

Esta sentencia equivaldría al DØ explícito.

```
DØ 10 I = 1,100
10 WRITE ( 3,700) A ( I )
```

De la misma manera podría escribirse la ordenación bidimensional B (200, 10) con la sentencia

```
WRITE ( 3,700 ) (( B ( I, J ), J = 1,10), I = 1,200 )
```

También puede combinarse la lista de variables como en la sentencia READ.

Ejemplo:

```
WRITE ( 3, 57 ) A, ( B ( I, 3 ) I = 1,10,2 ), C, D
```

origina la escritura por la impresora, dispositivo número 3, de las variables que figuran en la lista y que son,

```
A, B ( 1, 3 ), B ( 3, 3 ), B ( 5, 3 ), B ( 7, 3 ), B ( 9, 3 ), C, D
```

con el formato especificado por la sentencia número 57.

Como en el caso de la sentencia READ existe la forma

```
WRITE ( a ) LISTA
```

Ejemplo:

```
WRITE ( 9 ) A, C, D ( 4 ), NOMBRE
```

que transfiere los datos de la memoria central al fichero secuencial que se encuentra en el dispositivo número 9, en forma binaria.

SENTENCIA END FILE. - Su forma general es:

```
END FILE a
```

donde:

a es una constante entera que representa el dispositivo o unidad física del fichero.

Esta sentencia pone una señal de final de fichero en el fichero asociado al dispositivo a.

SENTENCIA REWIND. - Su forma general es:

```
REWIND a
```

donde:

a es una constante entera que representa el número del dispositivo o unidad física del fichero.

Esta sentencia coloca el fichero de datos en disposición de que pueda ser leído su primer registro.

SENTENCIA BACKSPACE. - Su forma general es:

```
BACKSPACE a
```

siendo a una constante o variable entera que representa el número de dispositivo o unidad física del fichero.

Esta sentencia hace retroceder un registro lógico al fichero de datos.

En general, cuando se manejan ficheros en organización secuencial en cinta magnética, después de finalizar las operaciones de lectura o escritura deben escribirse las sentencias,

END FILE a

REWIND a

que dejan el fichero preparado y a punto para una nueva operación de escritura o lectura.

TEMA 11

SENTENCIA DE FORMATO. CODIGOS DE FORMATO. - Para lograr que la información puede transmitirse al ordenador a partir de un medio o soporte de almacenamiento externo, tal como la ficha perforada, la banda de papel o la cinta magnética o al contrario, llegar a estos medios externos desde el ordenador, es necesario que en todo momento este último tenga la suficiente información relativa a la clase y cantidad de datos que se van a manejar. Para este fin, se emplean sentencias convenientes de formato, las cuales acompañan a cada una de las sentencias de entrada/salida que existan en el programa.

Toda sentencia de formato tiene dos finalidades: (1) describir la clase de información que se está utilizando y (2) especificar el tipo de conversión que ha de llevarse a cabo entre la representación interna de la información y la notación externa.

Las sentencias de formato no son ejecutables sino únicamente informativas. Por ello pueden colocarse en cualquier lugar del programa fuente, aunque es aconsejable que todas ellas estén reunidas bien al principio o al final del programa.

Los códigos que pueden contener las sentencias de formato para describir y especificar el tipo de conversión son de dos clases:

Códigos para la conversión de datos numéricos y códigos para información alfanumérica. Los primeros son I, F, E y los segundos A, H, X los cuales se estudiarán con todo detalle en los dos próximos temas.

USOS DIVERSOS DE LA SENTENCIA FORMAT. - La forma general de una sentencia de formato, denominada sentencia FØRMAT, es la siguiente:

$$n\ \text{FØRMAT}\ (c_1, c_2, \ldots, c_n / c'_1, c'_2, \ldots, c'_n / \ldots)$$

siendo n una constante entera que representa el número de la sentencia; las c_i y c'_i códigos de formato cuyo significado se describirá en los próximos temas y los / los indicativos del comienzo de un nuevo registro unitario.

Toda sentencia FØRMAT debe tener un número de sentencia al que ha de referirse otra de entrada/salida en algún otro lugar del programa fuente.

Las barras (/) o slash se utilizan para delimitar los registros unitarios, entendiendo por tales los siguientes conceptos:

a. Una línea de impresora con un determinado número de posiciones de impresión (120 ó 132).

b. Una ficha perforada con un máximo de 80 columnas.

c. Un registro de cinta magnética.

Ejemplos:

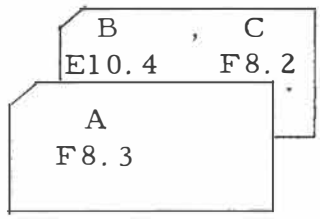
1. En 30 FØRMAT (I5, F8.4), el número de sentencia es 30 y sirve para un solo registro compuesto de dos datos, puesto que se dan dos códigos (I, F) separados por comas.

2. La sentencia 80 FØRMAT (F8.2/ E10.4), tiene 80 como número de sentencia y es capaz de manejar dos registros cada uno de ellos compuesto por un solo dato, siendo los códigos respectivos F y E.

3. Una sentencia de lectura de fichas perforadas con los datos A, B y C, que utilizara el formato,

   n FØRMAT (F8.3/E10.4, F8.2)

   leería correctamente los datos dispuestos en fichas distintas del modo siguiente:



Es posible que en muchas ocasiones deban emplearse códigos de formato repetidos porque exista un cierto número de variables consecutivas que pertenezcan al mismo tipo. En tal caso puede escribirse delante del código correspondiente, I, F o E, un coeficiente o constante numérica igual al número de variables que existan.

Ejemplos:

1. El empleo del código 3I4 con una sentencia de entrada, significa que van a leerse tres datos enteros.

2. La sentencia

```
READ ( 5, 100 ) I, J, A, B, C, K, L
```

podría acompañarse de

```
100 FØRMAT ( I4, I4, F8.2, F8.2, F5.3, I2, I2)
```

o bien de

```
100 FØRMAT ( 2I4, 2F8.2, F5.3, 2I2 )
```

Puede también suceder que un grupo de códigos diferentes se repitan un cierto número de veces. En este caso basta encerrar entre paréntesis tal grupo, haciéndole preceder de un coeficiente o constante numérica igual al número de veces que va a repetirse.

Ejemplo:

El grupo de especificaciones I2, E10.4, F8.2 se repite cuatro veces. Entonces bastaría con escribir,

```
4 ( I2, E10.4, F8.2 )
```

Cada sentencia FØRMAT debe contener tantos códigos de formato como datos existan en la correspondiente lista de la sentencia de entrada/salida.

De no coincidir ambos números pueden darse dos casos:

a) El número de datos de la lista de entrada/salida es menor que el de códigos de formato, en cuyo caso los códigos sobrantes se ignoran.

Ejemplo:

Si los datos A y B se transmiten mediante la sentencia de formato.

100 FØRMAT (F8.2, E10.4, F8.3, I4)

no se tendrían en cuenta los dos últimos códigos F8.3 e I4.

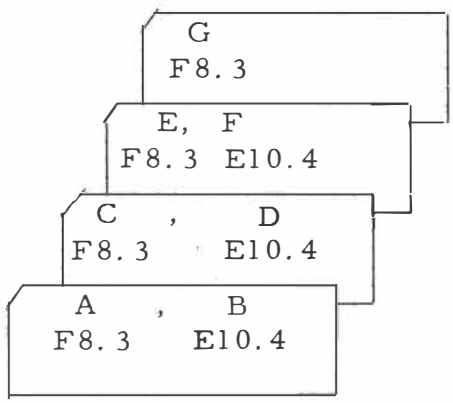
b) El número de datos de la lista de entrada/salida es mayor que el de códigos de formato, en cuyo caso el control de la sentencia FØRMAT se transfiere al paréntesis de apertura precedente, empleándose de nuevo los mismos códigos de formato para los siguientes datos de la lista, pero estos leídos o escritos en registros físicos o unitarios diferentes. Lo dicho se refiere al caso de que en el FØRMAT exista un único nivel de paréntesis.

Ejemplos:

1. Para leer los datos A, B, C, D, E, F, G perforados en fichas y según la sentencia

100 FØRMAT (F8.3, E10.4)

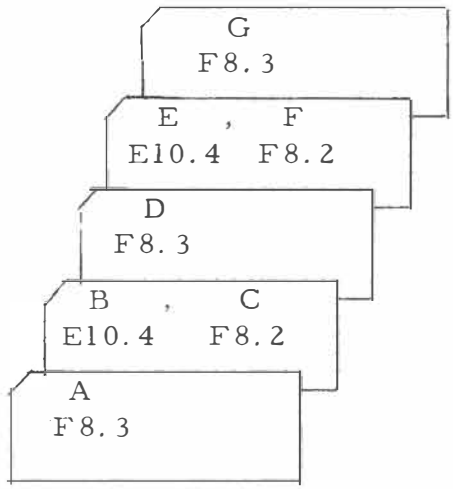
deberían perforarse como se indica:



2. Utilizando para esos mismos datos la sentencia de formato,

100 FØRMAT (F8.3 / E10.4, F8.2)

los datos deberían disponerse así:



Si en el FØRMAT existe más de un nivel de paréntesis, el primer registro corresponde a todos los códigos existentes dentro del nivel más externo y los siguientes se toman desde el nivel más cercano al paréntesis de cierre del nivel primero, hasta el final.

Ejemplos:

1. Se trata de escribir una serie de datos empleando para ello la sentencia de FØRMATO con dos niveles de paréntesis

100 FØRMAT (F8.3, (E10.2, F6.3))

Las líneas impresas tendrían el siguiente formato:

F8.3, E10.2, F6.3
E10.2 F6.3
E10.2, F6.3

2. Si la sentencia empleada fuera

100 FØRMAT ((F8.3, E10.2) , F6.3)

las líneas se imprimirían así:

F8.3, E10.2, F6.3
F8.3, E10.2, F6.3

3. Al emplear

```
100 FØRMAT ( I2 / ( I3, F6.1 ), F9.7 )
```

se producirían las siguientes líneas:

```
I 2
I 3, F6.1, F9.7
I 3, F6.1,F9.7
```

4. Empleando

```
100 FØRMAT (( I2, I3 ), ( I5, F8.2 ), F9.3 )
```

las líneas impresas serían:

```
I2, I3, I5, F8.2, F9.3
I5, F8.2, F9.3
I5, F8.2, F9.3
```

Finalmente, los códigos contenidos en una sentencia de formato deben corresponderse con la clase de los datos a transmitir, es decir, cuando estos sean enteros deben facilitarse en el FØRMAT códigos de conversión entera, etc., todo lo cual se verá con detalle en el tema siguiente.

TEMA 12

CODIGOS DE FORMATO NUMERICO. - El uso de la sentencia FØRMAT está íntimamente ligado como ya se ha dicho con la clase de campos o información que se maneje, según que éstos sean numéricos o alfanuméricos. Se estudiará en esta ocasión todo lo relacionado con los datos numéricos, mientras que en el próximo tema se revisarán los alfanuméricos.

Cuando se usan campos numéricos, el FORTRAN dispone de tres formas de conversión o códigos utilizables tanto en la entrada de datos como en la salida de resultados.

a. Por medio del código E puede convertirse en forma externa de coma flotante con exponente un dato real que estuviera contenido en memoria o bien puede ingresarse en el ordenador un dato externo real que viniera acompañado de exponente.

b. Mediante el código F un dato interno real puede convertirse en información externa también real, pero sin exponente o bien pueden introducirse en el ordenador las cantidades reales que no posean exponente.

c. El código I se utiliza cuando deba ingresarse en el ordenador y en forma entera, un dato externo también entero o en coma fija.

   Lo dicho sirve también para el caso inverso, es decir, este código se emplea para la obtención de resultados enteros contenidos en memoria en forma entera.

Téngase presente que los datos contenidos en memoria y que han sido generados por el lenguaje FORTRAN, únicamente pueden adoptar estas dos modalidades: en forma entera o en forma real con exponente y normalizada.

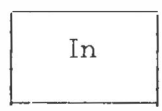
Ejemplos:

1. Si el dato que desea ingresarse en el ordenador, contenido en un soporte externo cualquiera, es el número decimal en coma flotante con exponente 2243.2E+4, deberá escogerse el código E. Lo mismo ocurriría para sacar el número -0.8324E+03 contenido en memoria.

2. Si se trata de almacenar en memoria el número 0.325 se utilizaría el código F. Lo mismo para 42.08 ó 21.3.

3. Si la cantidad a ingresar fuera el dato en coma fija 2343, se emplearía el código I.

FORMATO I. La forma general de este código es:



siendo I el carácter de control o código propiamente dicho y n una constante entera sin signo que indica la longitud del campo, es decir, el número máximo de dígitos que posee el número, más una posición para el signo algebraico cuando sea necesario.

Este código puede emplearse en la entrada o salida de cantidades enteras de la forma siguiente:

a) Entrada: En este caso se transfieren desde el soporte externo a la memoria, n caracteres numéricos, teniendo en cuenta que los blancos, cualquiera que sea el lugar que ocupen, se consideran como ceros.

Ejemplos:

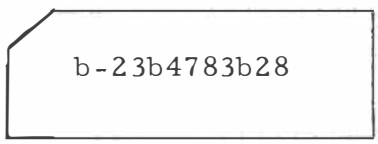
1. Existe un conjunto de fichas cuya información se perfora a partir de la columna 1 y todas ellas se leen con el formato I4. Entonces sería:

	Valor externo	Valor en memoria
Ficha 1	8235	8235
Ficha 2	4b53	4053
Ficha 3	bb28	0028
Ficha 4	132b	1320
Ficha 5	475384	4753
Ficha 6	b-47	0-47

2. Las dos sentencias siguientes

```
      READ ( 2, 100 ) KIL 1, KIL 2, KIL 3
100   FØRMAT ( I3, 2I2 )
```

leerían la ficha perforada según se indica



del modo siguiente:

```
KIL1 = 0-2
KIL2 = 30
KIL3 = 47
```

b) Salida: Se transfieren de memoria al soporte externo tantos caracteres como indica n, teniendo en cuenta que si el número de dígitos significativos que forman la cantidad junto con el signo ocupan en memoria una longitud menor que n, las posiciones izquierda de salida se rellenan con blancos. Si por el contrario, esas posiciones totalizan un número mayor que n, la salida está formada por tantos **asteriscos** (u otros caracteres de control) como indique n.

Ejemplos:

1. Los valores contenidos en la memoria que se indica a continuación se imprimirían con formato I3 según se relaciona:

Valor en memoria	Valor externo
824	824
0	bb0
-5	b-5
1723	***

2. Supongamos las sentencias,

```
      WRITE (3, 100) I, J, K
100   FØRMAT (I4, I3, I5)
```

Si los valores de las respectivas variables I, J, K dentro de la memoria fueran I = -315, J = 23 y K = -217, la línea impresa como resultado sería

-315b23b-217

FORMATO F. - La forma general de este código es:

Fn.d

Siendo F el carácter de control, n una constante entera sin signo que indica la longitud total del campo real y d otra constante del mismo tipo que la anterior que indica el número total de dígitos que siguen al punto decimal, es decir, que componen la parte fraccionaria del número.

Este código puede emplearse para la entrada o salida de la información del modo siguiente:

a) Entrada: En este caso se transfieren desde el soporte externo a la memoria n caracteres de los cuales los d últimos de la derecha serán los decimales. No es necesario perforar el punto decimal, pero si existe ha de tenerse en cuenta en la longitud total n y en ese caso ya no considerará la especificación d. Si se perfora el signo (que debe preceder al dato) también su posición habrá que tenerla en cuenta en n.

Ejemplos:

1. En el conjunto de fichas que sigue la información se perfora a partir de la columna 1 y todas ellas se leen con formato F5.2. Entonces sería:

	Valor externo	Valor en memoria
Ficha 1	b837b	83.70
Ficha 2	32b832	320.83
Ficha 3	472.73	472.7
Ficha 4	3.2531	8.253

2. Supóngase que la información contenida en las veinte primeras columnas de una ficha es b13834 – 61532bb837419 y ésta se lee mediante las sentencias

```
      READ (5, 100) A, B, C
100   FØRMAT (F6.2, F6.1, F8.4)
```

Entonces los valores que tomarán las variables en memoria serían:

A = 138.34
B = -6153.2
C = 83.7419

b) Salida: Se transfieren de memoria al soporte externo tantos caracteres como indica n y únicamente tantos decimales como indica d. En n ha de incluirse una posición por el punto decimal y otra para el signo algebraico, si existe. Si n es menor que las necesidades prescritas se imprimen asteriscos u otro carácter de control; si es mayor el número va precedido por blancos.

Ejemplo:

1. Los valores en memoria se imprimirían como se indica empleando el formato F5.2:

Valor en memoria	Valor externo
83.25	83.25
4.3237	b4.32
12.8	12.80
121.831	*****
324.00	*****

FORMATO E. - La forma general de este código es:

En.d

siendo E el carácter de control, n una constante entera sin signo que indica la longitud total del campo real con exponente y d otra constante del mismo tipo que la anterior que indica el número total de dígitos que siguen al punto decimal.

El empleo de este código es el siguiente:

a) Entrada: Se transfieren a memoria n caracteres. En este número han de incluirse además de las posiciones ocupadas por la parte fraccionaria (d) cuatro posiciones para la parte exponencial, otra para el signo y otra para el punto cuando existan, por lo que en todo caso deberá ser $n \geq d+4$.

Los espacios en blanco que existan se tomarán como ceros cualquiera que sea su posición en la cantidad. Si el punto va indicado y su posición no coincide con la expresada por d, aquél tiene prioridad sobre ésta. Para el exponente son equivalentes las formas siguientes: E+02, +02, +2, E+2, E2.

Ejemplos:

1. Las fichas que siguen se perforan a partir de la primera columna y se leen con formato E10.3. Entonces sería:

	Valor externo	Valor en memoria
Ficha 1	bb3.235+03	3235.0
Ficha 2	bbb4321+02	432.1
Ficha 3	3.8324E+01	38.324
Ficha 4	bb-374.1E+3	Especificación insuficiente
Ficha 5	321876E+02	32187.6

2. Para introducir la cantidad 3.3473E+02 debería emplearsela especificación E10.4. Si el punto decimal no estuviera explícitamente indicado bastaría con E9.4

b) Salida: En la salida todas las cantidades reales con exponente salen en forma normalizada, es decir, con el primer dígito significativo siguiendo al punto decimal. La cantidad ocupará en el soporte externo tantas posiciones como indique n por lo que en esta constante habrá de tenerse en cuenta: las posiciones ocupadas por la mantisa y que están controladas por d; el lugar para el punto decimal; el lugar para el signo cuando exista y cuatro posiciones para el exponente. Si el número de dígitos que forman en memoria la mantisa es mayor que el indicado en d, la parte fraccionaria se trunca por la derecha. Cuando la cantidad n es menor que el número de posiciones requeridas se imprimen asteriscos u otros caracteres de control. Si n es mayor se dejan blancos a la izquierda. Téngase en cuenta que al menos debe ser $n \geq d+6$. Con algunos ordenadores en n ha de incluirse también un lugar para un cero que se sitúa delante del punto decimal.

Ejemplos·

1. Los valores contenidos en memoria, con la especificación E12.3 se imprimirían como se indica:

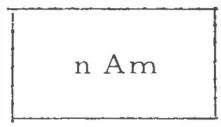
Valor en memoria	Valor externo
.35836788E+03	bbb0.358E+03
.37000000E+05	bbb0.370E+05
-.82470000E+00	bb-0.824E+00

2. Si el número interno -.83745078E+01 se quisiera perforar con E5.2, saldrían asteriscos por dar una especificación insuficiente (la n).

TEMA 13

CODIGOS ALFANUMERICOS. - Con el fin de poder manejar datos alfanuméricos como, por ejemplo, etiquetas o referencias de identificación o los cabeceros de página, el lenguaje FORTRAN dispone de códigos adecuados. Estos son los códigos A y H.

CODIGO DE FORMATO A. - Este código se emplea para poder dar nombres de variables a los datos formados por una combinación cualquiera de números, letras o caracteres especiales, combinaciones que se denominan campos alfanuméricos. La forma general de este código es:



siendo n el número de veces que se repite el formato; A es el código de especificación y m el número de caracteres alfanuméricos que puede albergar una palabra de ordenador. Este último número dependerá del ordenador que se emplee.

Esta especificación con la que pueden leerse o imprimirse datos alfanuméricos, hace que los nxm primeros caracteres contenidos en un soporte determinado sean introducidos en memoria o sean impresos o perforados desde el almacenamiento.

Ejemplo: las sentencias

```
      READ (3, 100) MENSA
  100 FØRMAT (A15)
```

darían lugar a que los quince primeros caracteres contenidos en un soporte determinado (unidad 3) se leyeran en memoria ocupando un campo cuyo nombre es MENSA. Se supone que el ordenador hipotético con el que se trabaja sería capaz de contener en cada palabra quince caracteres. Si no pudiera albergar nada más que uno se emplearía el FØRMAT siguiente:

```
100 FØRMAT (15A1)
```

Puede suceder que una palabra del ordenador no se llene completamente; en este caso y como los caracteres van disponiéndose de izquierda a derecha, la parte vacía se rellena con blancos. Lo mismo puede decirse para la salida de resultados.

Ejemplo: supóngase que hemos leído treinta columnas de una ficha en treinta palabras distintas de memoria. Entonces habríamos empleado la sentencia de formato

```
FØRMAT (30A1)
```

Si quisiéramos ahora imprimir estos campos y para ello empleásemos el formato 30A8, suponiendo que el ordenador admite hasta 8 caracteres por palabra, se escribirían los 30 caracteres que ya habíamos leído pero cada uno de ellos estaría separado del siguiente por 7 blancos.

CODIGO DE FORMATO H. - Este código se utiliza para la lectura o impresión de mensajes constantes, también denominados datos literales. Su forma general es:

nH

siendo n el número de caracteres del mensaje, contando entre ellos los blancos que existan, y H el código de la especificación.

Ejemplo: si quisiéramos imprimir el mensaje LEER NUEVOS DATOS podríamos emplear las dos sentencias

```
WRITE (5, 100)
100 FØRMAT (17HLEER NUEVØS DATØS)
```

Esta especificación puede también emplearse para leer un cierto título e imprimirlo más tarde. Para ello es necesario utilizar con las dos sentencias de entrada/salida la misma sentencia de formato.

Ejemplo: deseamos leer las diez primeras columnas de una ficha que contiene un título que más tarde y en otro lugar del programa vamos a imprimir. Las sentencias empleadas podría ser (b indica blanco):

```
      READ    ( 3, 100 )
      .
      .
      .
      WRITE   ( 5, 100 )
      .
      .
      .
      .
100   FØRMAT  ( 20Hbbbbb...(20)....b )
```

En lugar de emplear el código H, en algunos ordenadores basta con escribir entre comillas el mensaje que quiera utilizarse. Así, el primer ejemplo equivaldría a emplear estas sentencias:

```
      WRITE ( 5, 100 )
100   FØRMAT ('LEER NUEVØS DATØS')
```

Las comillas pueden emplearse tanto para la entrada de caracteres como para la salida.

CONTROL DE CARRO. - Utilizando ciertos códigos y bajo control del propio programa puede hacerse que la impresora conectada a la unidad central pueda ejecutar ciertos movimientos especiales como espaciado entre renglones de escritura o cambio de página para comenzar a escribir en la siguiente. Para ello se emplean normalmente los caracteres b (blanco), 0 y 1, acompañados del código H que acaba de estudiarse, dentro de la sentencia FØRMAT correspondiente. El significado de estos caracteres es el siguiente:

b : escritura a espacio simple entre renglones
0 : doble espaciado
1 : comenzar a escribir en la página siguiente.

Ejemplos:

a) Para escribir el mensaje LEER NUEVOS DATOS en la hoja siguiente emplearíamos el formato

```
100   FØRMAT  (1H1, 'LEER NUEVØS DATØS')
```

b) Si hubiésemos utilizado la sentencia

```
100   FØRMAT (1H0, 'LEER NUEVØS DATØS
```

el mensaje encerrado entre comillas se nos hubiera impreso dejando dos líneas en blanco entre su línea y la anterior.

Este carácter de control debe ser lo primero que contenga la sentencia FØRMAT y puede también ir encerrado entre comillas, si se desea y el ordenador lo permite.

El control del carro en cuanto a la forma de imprimir los renglones puede también ejercerse mediante el empleo en la sentencia FØRMAT de signos /, barras o slash, siendo función del número de estos caracteres y de su situación en la sentencia, la cantidad de registros ignorados en la entrada de datos o del número de líneas en blanco en la salida de resultados. El siguiente cuadro es un resumen de lo expuesto:

Nº de barras	Registros ignorados o líneas en blanco en la impresión
Al principio del FØRMAT existen n	n
En el centro del FØRMAT existen n	n - 1
Al final del FØRMAT existen n	n

Ejemplos:

a) Si se emplea la sentencia FØRMAT (I5/////F8.3), entre los datos indicados por I5 y F8.3 escritos en líneas diferentes quedarán 4 líneas en blanco.

b) Si se leyeran los datos enteros perforados en una misma ficha y para ello se empleara la sentencia de formato

FØRMAT (////I5, I4)

se leerían esos datos de la quinta ficha.

CODIGO DE FORMATO X. - Su forma general es:

nX

siendo n el número de caracteres que se van a ignorar de un determinado soporte cuando empleamos el código para la entrada de datos, o el número de blancos que se van a producir cuando se emplea en la salida de resultados.

Ejemplos:

a) Se tiene una ficha perforada con seis campos de datos enteros, cada uno de ellos ocupando 4 columnas. Si se desea ignorar el segundo de ellos podría emplearse el formato siguiente:

FØRMAT (I4, 4X, 4I4)

b) Si en una línea impresa, entre dos resultados reales se desea dejar 5 blancos debería emplearse la sentencia de formato siguiente:

FØRMAT (F8. 3, 5X, F8. 4)

con la que se produciría por ejemplo la línea impresa.

-532. 472bbbbb-37. 4108

TEMA 14

SENTENCIAS DE ENTRADA/SALIDA EN ACCESO DIRECTO. - Para leer o escribir en un dispositivo de acceso directo se precisan unas sentencias que permitan acceder directamente a un registro sin ningún otro paso intermedio.

Cuando se leía o escribía secuencialmente, únicamente se necesitaba para ello indicar el fichero de que se trataba. Ahora además se necesitará indicar qué registro interesa dentro del fichero. El modo más cómodo de hacer esto, será asociando el registro con un número que será precisamente el número del registro dentro del fichero.

SENTENCIA DEFINE FILE. - Es aquella que define las características de un fichero al que vamos a acceder mediante sentencias de leer o escribir en acceso directo.

Puede colocarse en cualquier lugar del programa con la única limitación de que deberá preceder a cualquiera de las sentencias que se refieran al fichero que nos define. Su forma general es:

DEFINE FILE a_1 (n_1, t_1, f_1, v_1), a_2 (n_2, t_2, f_2, v_2), ...

siendo:

- a_i: Constante entera sin signo, que indica el fichero de que se trata.
- n_i: Constante entera sin signo, que indica el número de registros que tiene el fichero asociado con a.

t_i: Constante entera sin signo, que indica el tamaño de dichos registros expresado bien en caracteres bien en palabras.

f_i: Es una de las siguientes letras y sirve para indicar si la transmisión va a ser con o sin control de formato:

L: Si va a ser con o sin control de formato y entonces el tamaño del registro debe ir expresado en número de caracteres.

E: Con control de formato y tamaño expresado en caracteres.

U: Sin control de formato y tamaño expresado en palabras de memoria.

v_i: Es una variable entera no suscrita. Terminada una operación de lectura o escritura (READ o WRITE), indica el número del registro siguiente al transmitido. Después de una operación de búsqueda (FIND), indica el número del registro encontrado.

Ejemplo:

DEFINE FILE 4(50, 100, E, I2), 5(199, 50, L, U3), 10(100, 7, U, ID)

El primero indicará un fichero que se reconoce por el número 4, que tiene 50 registros de 100 caracteres cada uno de longitud y por tener la letra E la transmisión de la información será con control de formato siendo I2 en este caso la variable asociada. El segundo irá con o sin control de formato, y el tercero sin él y de tal manera que cada registro es de 7 palabras.

SENTENCIA READ. - Permite transmitir los datos, desde un dispositivo de acceso directo, a la memoria. Su forma general es:

READ (a'r, n) LISTA

siendo:

a: Constante o variable entera sin signo que indica el fichero asociado. Debe ir seguida de un apóstrofe.

r: Expresión entera que indicará el número del registro dentro del fichero del que va a leerse la información.

n: Será el número de la sentencia FØRMAT en caso de que la transmisión vaya a efectuarse bajo formato. Si se omite se supone que la transmisión va a ser sin control de formato.

LISTA: En caso de existir indicará el nombre de las variables o conjuntos en donde se van a almacenar los datos leídos.

SENTENCIA WRITE. - Se emplea para transmitir la información desde la memoria a un dispositivo de acceso directo. Su forma general es:

```
WRITE (a'r, n) LISTA
```

donde:

a, r y n tienen el mismo significado que en la sentencia READ.

LISTA: En caso de existir indicará el nombre de las variables o conjuntos que se desean grabar en el dispositivo de acceso directo dentro del registro indicado por r del fichero a.

Ejemplo: Sean las sentencias

```
      DEFINE FILE 4( 50, 4, L, I2), 5 ( 10, 12, L, KA )
      INTEGER H ( 8 )
          .
          .
          .
      KA=2
1     FØRMAT (2I2)
2     READ (4'20, 1) (h(I), I=1, 8)
          .
          .
          .
3     WRITE(5'KA+1)X, Y, Z
```

La sentencia READ 2 hace que desde el fichero asociado con el número 4 bajo el control de la sentencia 1 FØRMAT, se lean ocho partidas de datos y dado que cada registro sólo contiene dos partidas, se leerán cuatro registros comenzando a partir del 20 (20, 21, 22 y 23). La variable asociada I2 valdrá 24 al término de la operación.

La sentencia 3 WRITE hace que se transmitan al fichero asociado con el número 5 y sin control de formato tres variables de tipo real: X, Y, Z.

Como cada registro contiene 12 caracteres se escribirá el registro número 3 y al final de la transmisión KA tendrá el valor 4.

SENTENCIA FIND.- Puede, en algunos casos, resultar interesante que mientras se está efectuando un proceso con unos datos se desee preparar ya en alguna parte del programa los datos siguientes y con ello aumentar la velocidad del programa objeto. Esto es precisamente lo que hace esta sentencia. La forma general es:

```
FIND ( a ' r )
```

siendo:

a: Es una constante o variable entera sin signo que indica el número del fichero al que se refiere.

r: Indica el número del registro dentro del fichero a.

Ejemplo:

```
      DEFINE FILE 7(2000, 1, U, JJJ), 8(1920, 6, E, M)
      IØM = 320
      JJJ = 1
100   N14 = 500 + JJJ
      IMAX = N14 + IØM
111   WRITE (8'JJJ, 1) IMAX
  1   FØRMAT (I6)
 77   FIND (7'N14)
        .
        .
        .
 14   READ ( 7'N14)AAA
      IF (IMAX - 1200) 100, 100, 15
 15   CONTINUE
```

La sentencia 111 WRITE hace que se escriba bajo el control de la sentencia 1 FØRMAT el valor de IMAX (en la primera pasada 821) en el registro 1 del fichero 8).

M quedará con el valor 2.

La sentencia FIND preparará el registro 501 del fichero 7.

La sentencia 14 READ, asignará a AAA el registro 501 quedando JJJ con el valor 502.

En la segunda pasada será N14=1002, IMAX=1322.

111 WRITE escribe en el registro 2 del fichero 8, 1322.

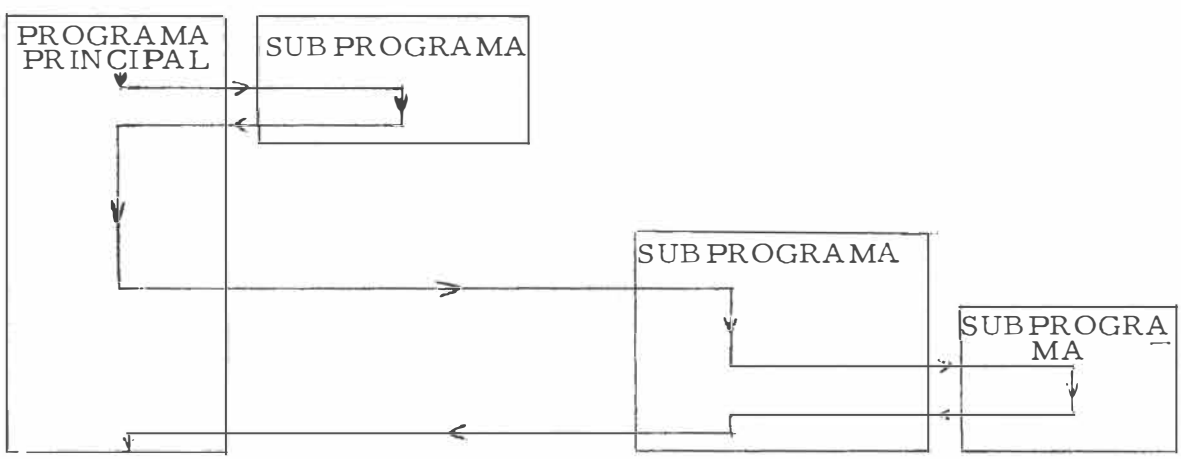
FIND preparará el registro 1002 del fichero 7 continuando la ejecución a partir de la sentencia número 15.

TEMA 15

CONCEPTO DE SUBPROGRAMA. - Se denomina subprograma a un conjunto de sentencias que ayudan a resolver una parte de un problema tratado en un programa general (principal), y que se encuentra fuera de ese programa principal. El subprograma es llamado, en bloque, desde el programa principal.

Hay dos tipos de aplicaciones fundamentales para los subprogramas: una es la necesidad de repetir unas ciertas operaciones fijas, con unos datos variables. La otra es la posible necesidad de fragmentar un programa principal y sus subprogramas forman un bloque de diversas "piezas" conectadas entre sí.

Aun cuando lo más común es que un subprograma esté conectado exclusivamente al programa principal, puede estar también conectado con otros subprogramas.



El uso de subprogramas permite simplificar el programa principal, trabajar simultáneamente distintos programadores en puntos diversos de un programa, efectuar cambios y correcciones sin alterar la estructura fundamental. Facilita también, la localización de errores.

El aumento de tiempo en la ejecución de un programa que tenga subprogramas frente a uno que tenga las funciones que realizan los subprogramas incluidas en el principal, no es apreciable salvo en casos en que los subprogramas se deban ejecutar un gran número de veces, siendo muy cortos ellos mismos. Esto se debe a las operaciones que tiene que realizar la máquina para llevar y devolver un subprograma a memoria.

DENOMINACION DE LOS SUBPROGRAMAS. - Los subprogramas se identifican por caracteres alfanuméricos (en número variable según la máquina empleada).

El primer carácter debe ser siempre alfabético. El único carácter especial que puede utilizarse es el blanco, pero no es válido como separación entre otros caracteres: se le ignora.

El nombre de un subprograma no podrá coincidir nunca con el de una variable.

TIPOS DE SUBPROGRAMAS. - Se puede distinguir cuatro tipos de subprogramas: subrutinas, subprogramas FUNCTIØN, subprogramas FUNCTIØN de FORTRAN, y funciones de sentencia.

Se denominan funciones, en FORTRAN, a la expresión en sentencias FORTRAN de una cierta relación entre constantes y variables. Por tanto, para emplear funciones en FORTRAN será necesario poderlas identificar por su nombre, definir el tipo de operación que van a realizar, y definir las constantes y variables que intervienen en ellas.

FUNCIONES DE SENTENCIA. - Si las relaciones que se encuentran en un programa son lo suficientemente cortas o sencillas como para que puedan incluirse completas en una sola sentencia, pueden expresarse mediante una función de sentencia. La definición de dicha función de sentencia indica las operaciones que se efectúan siempre que aparece el nombre de la función de sentencia incluido en alguna otra sentencia del programa.

Si se tiene por ejemplo la función matemática:

$$Y(t) = 3t^2 - 16,3X + t^3$$

podríamos escribirla en FORTRAN como:

```
Y(T) = 3.0* T**2 - 16.3* X + T**3
```

Si este cálculo tuviéramos que repetirlo para diversos valores de t y X podríamos definir la función de sentencia

```
PAR(T,X) = -16.3*X + 3*T**2 + T**3
```

y entonces cada vez que utilizáramos PAR (T, X) con valores cualquiera de T y X, se realizaría ese cálculo.

Así si tuviéramos que hacer

```
Z = L + Y(t)
```

escribiríamos en FORTRAN

```
Z = L + PAR(T , X)
```

La expresión escrita a la derecha del signo igual, en la definición de función de sentencia no puede tener variables subindicadas ni hacer referencia a la propia función. Los argumentos de la función (en el ejemplo son T y X), no pueden ser subindicados tampoco.

Los argumentos empleados en la definición de función de sentencia son variables ficticias. En el ejemplo hubiéramos podido definir la función de sentencia como:

```
PAR(P, Q) = -16.3*Q + 3*P**2 + P**3
```

y llamarla en la sentencia con T y X

```
Z = L + PAR(T  X)
```

realizándose las operaciones con los valores de T y X.

Estas variables se denominan también pseudovariables, y pueden utilizarse en más de una definición de función de sentencia o bien como variables fuera de dichas definiciones de función de sentencia.

En cambio los argumentos en las sentencias que hacen referencia a dicha función deben ser iguales en tipo, orden y número a los ficticios. Esto es lógico teniendo en cuenta que los valores de los argumentos en sentencias de referencia (sentencia de llamada a función de sen-

tencia) son los valores que adoptan las pseudovariables para efectuar el cálculo de la función.

Supongamos, en el mismo ejemplo que

la variable T = 2.0
la variable X = 5.2

y que se debe calcular el valor de Z en

Z = L + PAR (T, X)

Lo que ocurre es que P toma el valor de T, y Q el de X

P = T = 2.0
Q = X = 5.2

y se calcula la función

PAR (2.0, 5.2)

Los argumentos de la sentencia de referencia podrán ser cualquier tipo de constante o variable subindicada o no subindicada.

En la expresión de una sentencia de definición de función se pueden utilizar variables diferentes a las que forman los argumentos.

Así por ejemplo el llamar a la función ØPER(A, B) definida por una sentencia

ØPER (A, B) = 2.2*A-B/5.0+C

la función se calcula con los valores transmitidos de A y B y con el que tenga C en ese momento.

La definición de función de sentencia debe anteponerse a la primera sentencia ejecutable del programa en que se encuentre dicha función.

SUBPROGRAMAS FUNCTIØN. - Se caracteriza por ser un programa completo por sí mismo, con una sentencia FUNCTIØN, otra RETURN, y otra END, por lo menos.

Desde un programa principal se llama a un subprograma FUNCTIØN con uno o más argumentos que son los valores con los que éste opera, y devuelve al programa principal el valor de la función.

Las variables y números de sentencia del subprograma FUNCTIØN son independientes de los de cualquier otro programa.

No es necesario dar en el programa principal información previa sobre el subprograma FUNCTIØN antes de la sentencia en que se le llama, si éste es el tipo implícito.

La primera sentencia del subprograma es la sentencia FUNCTIØN, que puede tener las formas:

```
          FUNCTION NNNN (a1, a2, a3, ...)
REAL      FUNCTION NNNN (a1, a2, a3, ...)
INTEGER   FUNCTION NNNN (a1, a2, a3, ...)
```

en donde NNNN representa el nombre de la función, y a_j sus argumentos. El nombre de la función no puede ser utilizado como variable.

En caso de que se emplee alguna de las dos últimas formas es necesario indicar en el programa principal respectivamente:

```
REAL NNNN
INTEGER NNNN
```

En un subprograma FUNCTIØN no puede aparecer ni una sentencia SUBROUTINE ni otra sentencia FUNCTION.

Se debe asignar un valor al nombre de la función en algún o algunos puntos del subprograma, ya sea en una sentencia READ, en una CALL, o en una aritmética (a la izquierda del signo igual, como variable).

Los argumentos del subprograma FUNCTIØN se pueden considerar como pseudovariables. Se sustituyen por tanto por los valores de los argumentos de la sentencia de llamada del programa principal. De cualquier forma deben ser del mismo tipo, en el mismo orden, y en igual número que los argumentos de la llamada del programa principal.

Los argumentos de la sentencia de llamada al subprograma FUNCTIØN pueden ser cualquier tipo de constante, cualquier tipo de variable subindicada o no, cualquier expresión aritmética, cualquier nombre de conjunto, o el nombre de un subprograma FUNCTIØN o SUBRØUTINE. Si es el nombre de un conjunto, se deberá especificar en el subprograma FUNCTIØN la dimensión del conjunto con una sentencia DIMENSIØN apropiada.

El subprograma FUNCTIØN debe acabar con dos sentencias:

```
RETURN
END
```

Ejemplo:

Programa principal	Subprogramas	
INTEGER FUNCTIØN BASE		
DIMENSIØN I (45)	FUNCTIØN ACUM (S, V)	FUNTIØN BASE (I)
.	.	DIMENSIØN I (45)
.	.	
.	.	.
.	.	.
A=ACUM (R, T)	RETURN	RETURN
.	END	END
.		
.		
.		
B = Q. -BASE (I(R))		

SUBPROGRAMAS FUNCTIØN DE FORTRAN O FUNCIONES DEL SISTEMA. - Son subprogramas incluidos en el sistema FORTRAN y ya prefijados. Se diferencian de los subprogramas FUNCTIØN normales en que el subprograma ya está hecho, de modo que el programador no necesita prepararlo.

A continuación se indican algunos de estos subprogramas:

Nombre	Valor de retorno	Función realizada
SIN (Arg)	Seno (Arg)	Cálculo del seno (argumento en radianes) Trigonométrico
CØS (Arg)	Coseno (Arg)	Coseno Trigonométrico
EXP (Arg)	e^{Arg}	Exponencial de e
SQRT (Arg)	$(Arg)^{\frac{1}{2}}$	Raíz cuadrada
ALØG (Arg)	$Log_{10}(Arg)$	Logaritmo del argumento
ABS (Arg)	Arg	Valor absoluto

Tanto en las funciones de sentencia, como en los subprogramas FUNCTIØN como en los subprogramas FUNCTIØN de FORTRAN el número de argumentos es 1 o más, y el de valores de retorno es solamente 1, a diferencia de los subprogramas SUBRØUTINE en los que el número de valores de retorno es cero o más.

TEMA 16

SUBPROGRAMAS SUBROUTINE. - Siempre que al utilizar un subprograma sea necesario la existencia de más de un valor de retorno se hace obligatorio el uso del subprograma SUBRØUTINE. En esto reside la diferencia fundamental entre SUBRØUTINE y FUNCTIØN.

Al igual que los subprogramas FUNCTIØN, debe tener al menos una sentencia RETURN, y una última sentencia END.

En los subprogramas SUBRØUTINE son válidas todas las sentencias FORTRAN excepto la FUNCTIØN, las SUBRØUTINE, DEFINE FILE u otra en la que el nombre de la SUBRØUTINE aparezca como variable.

Del mismo modo que en las FUNCTIØN, por ser un programa independiente en los subprogramas SUBRØUTINE se puede utilizar cualquier variable, aunque ya esté empleada en el programa principal, y cualquier número de sentencia.

Los argumentos que aparecen en una sentencia SUBRØUTINE se suelen denominar parámetros. No hay más limitación en cuanto a su número, que la posibilidad de líneas de continuación en la hoja de codificación.

Como para las FUNCTIØN, los parámetros son pseudovariables y deben corresponder en orden, número, tipo y dimensión con los de la llamada del programa principal. No tienen por qué tener el mismo nombre.

Tampoco es obligatorio la existencia de argumentos o parámetros.

Los parámetros que deban transmitir valores al programa principal, no es necesario que todos lo hagan, deben aparecer en el subprograma a la izquierda del signo igual en alguna sentencia aritmética, o en una lista de entrada del subprograma, o como argumento en una CALL o en una llamada de función.

No es necesario dar en el programa principal información previa a la sentencia de llamada a la SUBRØUTINE a menos que el tipo de algún argumento no sea implícito, en cuyo caso en el subprograma se debe definir también el tipo de ese argumento.

Los argumentos pueden ser cualquier tipo de constante, de variable (subindicada o no), nombre de conjunto, expresión aritmética, o nombre de subprograma.

En una sentencia EQUIVALENCE, dentro de un subprograma SUBRØUTINE no puede figurar ninguna de las pseudovariables.

La subrutina queda compilada de modo que no asigna nuevas direcciones de almacenamiento a las variables que aparecen como argumentos, sino que utiliza las mismas direcciones que tienen esos argumentos en el programa principal.

TEMA 17

SENTENCIA CALL.- Esta sentencia se emplea como llamada a un subprograma SUBRØUTINE. Cuando en la ejecución de un programa principal aparece la sentencia CALL el control se transmite a la subrutina llamada. La forma general de esta sentencia es:

```
CALL NNNN (a, b, c, ...)
```

donde NNNN representa el nombre de la subrutina y a, b, c, etc., son los argumentos.

SENTENCIAS RETURN y END.- En el empleo habitual de los subprogramas suele interesar el retorno al programa principal para que éste continúe ejecutándose. Esto se consigue con la sentencia RETURN, que devuelve el control al programa principal, transmitiendo los argumentos con los valores que tengan en ese momento. Si no se desea el retorno al programa principal en lugar de la sentencia RETURN se puede utilizar la sentencia STØP.

En un subprograma puede existir más de una sentencia RETURN.

La sentencia END le señala al compilador el final físico del subprograma. Sólo puede haber una sentencia END.

Seguidamente se exponen diversos casos de utilización de las sentencias y subprogramas anteriores:

Programa principal

```
        .
        .
        .
DIMENSIØN B ( 30 )
        .
        .
CALL RØTU
        .
        .
        .
        .
7 CALL CAMB (A,B,C)
        .
        .
        .
        .
        .
        .
CALL FIN (DI, IJ)

END
        .
        .
```

Subroutines

```
SUBRØUTINE RØTU
WRITE ( 3, 7 )
FØRMAT (10X, 20H PRINCIPIØ CALCULØ
RETURN
END

SUBRØUTINE CAMB ( A, B, C )
DIMENSIØN B ( 30 )
A = B ( 7 ) + 2
IF (A) 5, 7, 5
5 RETURN
7 C = A + 6
RETURN
END

SUBRØUTINE FIN ( PL, K )
        .
        .
STØP
END
```

TEMA 18

SENTENCIA CØMMØN. - La finalidad de la sentencia CØMMØN es reservar una cierta área de almacenamiento en la memoria de la máquina, que pueda ser compartida por varios programas o subrutinas sin que se altere la información existente en ese área de almacenamiento. Esto permite una mayor simplificación de programas, y un ahorro de volumen de memoria ocupada.

La forma de esta sentencia es:

```
CØMMØN a, b, c, ....
```

donde a, b, c, ... son variables o conjuntos dimensionados o no.

Si en un programa aparecen unas variables cualesquiera A, B, C, D y empleamos la sentencia CØMMØN A, B, C, D la máquina reservará una zona de almacenamiento en la memoria para esas variables. Si en un programa o subprograma conectado con el anterior aparece también la sentencia CØMMØN A, B, C, D la zona que tenían asignada esas variables no será alterada, y por tanto los valores que tengan al comenzar el segundo programa o subprograma será el mismo que tenían al acabar el primer programa.

Si en un programa se define

CØMMØN IREV, LDØC, RMN

y en otro programa o subrutina conectado con el anterior se define

CØMMØN LS, J4, T19

y las tres variables de los dos programas se corresponden mutuamente en cuanto a tipo, las del segundo CØMMØN tendrán asignadas las mismas posiciones de memoria que las del primero.

La dirección reservada a las variables que aparecen en el CØMMØN se hace (de modo ascendente o descendente) de acuerdo con el orden con que aparecen en el CØMMØN. Si hay más de una sentencia CØMMØN se asigna según el orden de cada sentencia.

Así, la aparición de:

```
CØMMØN A, B, C, D
CØMMØN E, F
```

en un programa, provocará un almacenamiento en el orden

A, B, C, D, E, F

Las redundancias no están permitidas. Por ejemplo:

CØMMØN IA, L, K, L, B9

Es de gran utilidad este tipo de sentencia para su empleo en las subrutinas. Se puede eliminar con ella los argumentos en la sentencia de llamada y en la subrutina. Así, por ejemplo, si en un programa tenemos:

```
.
.
.
CØMMØN A, B, C
.
.
.
CALL ESTRA
.
.
.
```

y en la subrutina tenemos:

```
SUBRØUTINE ESTRA
CØMMØN A, B, C
```

sustituye a:

```
programa principal:   .
                      .
                      .
                      CALL ESTRA (A, B, C)
                      .
                      .
                      .
subrutina:            SUBRØUTINE ESTRA (A,B,C)
                      .
                      .
                      .
                      .
```

Esto,además de simplificar la programación de las subrutinas (por número de argumentos, similitud de tipos, etc.), hace que el programa se ejecute más deprisa.

Si en un programa principal tenemos tres variables A, B, C y en otro programa relacionado con él solamente utilizamos dos variables DØB, SECT, iguales a A y C respectivamente, podríamos dar en el primer programa una sentencia:

```
CØMMØN A, B, C
```

y en el segundo, otra:

```
CØMMØN DØB, FICT, SECT
```

en donde FICT es una variable ficticia que no se vuelve a mencionar en el programa, pero que permite que las direcciones asignadas a las otras dos parejas de variables sean idénticas.

Cuando se sustituyen los argumentos de una subrutina por variables incluidas en un CØMMØN, éstas están sujetas a las mismas normas que los argumentos.

Se ha indicado anteriormente que las variables del CØMMØN pueden estar dimensionadas. Hay dos formas de indicarlo. Supongamos dos variables A (20) y B (30). Podremos poner:

```
DIMENSIØN A (20), B (30)
CØMMØN A, B
```

o bien:

CØMMØN A (20), B (30) que sustituye a las dos anteriores.

Sería también válida la combinación:

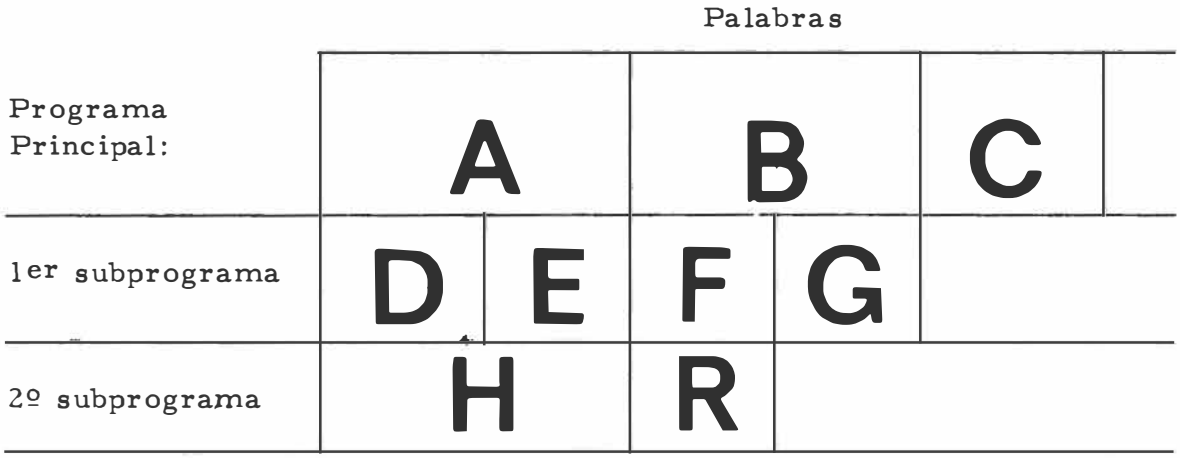
```
DIMENSIØN B ( 30 )
CØMMØN A ( 20 ), B
```

La dimensión de un conjunto debe indicarse en la primera referencia a dicho conjunto (excepto en el caso de argumentos, en que la dimensión se da en la segunda referencia). Así, es válida la forma:

```
REAL IJK ( 100 )
CØMMØN PQ ( 15 ), IJK
```

Supongamos que tenemos un programa principal y dos subprogramas y que en el programa principal se utilizan las variables A, B, C; en el primer subprograma las D, E, F, G; y en el otro programa las H, R.

Supongamos también que las A, B y H son de doble precisión, y el resto de precisión normal. Supongamos también que la doble precisión requiere cuatro palabras por variable, y que la precisión normal requiere sólo dos:



Si en el programa principal tenemos CØMMØN A, B, C y en el primer subprograma tenemos CØMMØN D, E, F, G, ninguno de los valores de A, B, C se podrá transferir correctamente a D, E, F, G a través del CØMMØN. porque ninguna de las variables de cada grupo está almacenada en las mismas direcciones y con igual longitud que las del otro grupo. En cambio, si en el 2º subprograma tenemos CØMMØN H, R, podremos transferir a través del CØMMØN los valores de la variable A a la variable H.

Del mismo modo los valores de la variable F se podrían transmitir a la variable R.

Esto indica que a veces puede ser conveniente la ordenación cuidadosa de las variables en el CØMMØN (por ejemplo; primero la precisión normal, luego las de doble precisión) que permita fácilmente la transferencia de valores entre programas y subprogramas.

SENTENCIA EQUIVALENCE. - Ya se ha indicado anteriormente que la sentencia CØMMØN permite un ahorro de memoria (por ejemplo, en vez de tener que definir dos veces un mismo conjunto, con nombres distintos en un programa y un subprograma, lo incluimos en dos CØMMØN y ocupa así el lugar de un solo conjunto. Es decir si A (I) = B (I), CØMMØN A (d) y CØMMØN B (d) sustituyen a DIMENSIØN A (d), B (d)).

Existe otra sentencia, la EQUIVALENCE, cuya función es exclusivamente la de ahorrar volumen de memoria.

En general las variables y conjuntos de variables se almacenan en distintas posiciones de memoria. En caso de que el programa esté escrito de modo que dos ciertas variables se utilicen consecutivamente, pero no alternadas, nada impide el que los valores que toman la variable utilizada en segundo lugar se almacenen en las posiciones que tomaba la primera variable. Esto, que es también válido para el caso de conjuntos, se consigue con la sentencia EQUIVALENCE.

La forma general de esta sentencia es:

EQUIVALENCE (v_1, v_2, v_3, ...), (x_1, x_2, x_3, ...), ...

donde v_1 y x_i son nombres de variables que pueden estar subindicadas Se pueden presentar tres casos para los subíndices: a) si la variable es una variable simplemente subindicada (un solo subíndice), el subíndice con que aparece en el EQUIVALENCE indica el lugar que ocupa en el conjunto (así una variable V con DIMENSIØN V (30) que apareciera en el EQUIVALENCE como V (17), haría referencia a la variable V que ocupa el lugar 17 en el conjunto). b) Si la variable que aparece en el EQUIVALENCE es una variable perteneciente a un conjunto no simplemente subindicado (es decir, con dos o tres dimensiones), y el subíndice con que aparece en el EQUIVALENCE, es uno solo, éste indica el lugar que ocupa en el conjunto, (si tenemos DIMENSIØN VD (20, 10, 20) y EQUIVALEN-

CE VD (3), el EQUIVALENCE se refiere a la variable VD que ocupa el lugar nº 3 en el conjunto, es decir a la VD (3, 1, 1). c) Si la variable subindicada tiene más de un subíndice puede aparecer en el EQUIVALENCE con tantos subíndices como tenga. Por ejemplo EQUIVALENCE A (12, 9, 14), B (8, 3, 4) significa que están en el EQUIVALENCE, dos variables, no los dos conjuntos.

La sentencia:

EQUIVALENCE (v_1, v_2 ...), (X_1, X_2, ...)

indica que las variables v_1, v_2, v_3, ... tienen asignada la misma, y única, posición de almacenamiento; y lo mismo ocurre con X_1, X_2 ..

En el uso de la sentencia EQUIVALENCE hay que evitar contradicciones.

No puede incluirse en una sentencia EQUIVALENCE dos variables que ya se encuentren en el CØMMØN.

Conviene tener en cuenta que si las variables que se incluyen en el EQUIVALENCE pertenecen a un conjunto, éste queda automáticamente posicionado. Se puede así aumentar el tamaño del CØMMØN, siempre que se haga por el límite superior. Por ejemplo, si tenemos:

```
DIMENSIØN PIL ( 4 )
CØMMØN REA, SPA, DEN
EQUIVALENCE (SPA, PI ( 1 ) )
```

habremos posicionado del siguiente modo:

Por el CØMMØN	Por el EQUIVALENCE
límite inferior REA	
SPA	PIL (1)
límite superior DEN	PIL (2)
	PIL (3)
	PIL (4)

y habremos ampliado el CØMMØN.

En cambio haciéndolo del modo:

```
DIMENSIØN PIL ( 4 )
CØMMØN REA, SPA, DEN
EQUIVALENCE (SPA, PIL ( 3 ) )
```

que posicionaría:

Por el CØMMØN	Por el EQUIVALENCE
REA	PIL (1)
SPA	PIL (2)
DEN	PIL (3)
	PIL (4)

es inválido.

Las sentencias:

```
DIMENSIØN R ( 7 ), S ( 30, 30 ), T ( 10, 8, 8 )
EQUIVALENCE ( R ( 3 ), S ( 5, 2 ), T ( 2, 1, 1 ))
```

que posicionarían en el mismo lugar las variables R (3), S (5, 2) y T (2, 1, 1) implican también (en horizontal direcciones iguales):

```
                    .
                    .
                    .
                    S   ( 30, 1 )
                    S   ( 1, 2 )
                    S   ( 2, 2 )
R ( 1 )             S   ( 3, 2 )
R ( 2 )             S   ( 4, 2 )        T( 1, 1, 1 )
R ( 3 ) ..... S   ( 5, 2 ) ..... T( 2, 1, 1 )
R ( 4 )             S   ( 6, 2 )        T( 3, 1, 1 )
  .                   .                   .
  .                   .                   .
  .                   .                   .
```

Se debe prestar atención a que el alineamiento de fronteras de las variables (ver sentencia CØMMØN) sea el adecuado en el EQUIVALENCE. Si las longitudes de variables no se corresponden, se pueden incluir variables ficticias de longitud conveniente hasta lograr el ajuste.

INDICE

	Página
TEMA 1	5
TEMA 2	9
TEMA 3	13
TEMA 4	17
TEMA 5	21
TEMA 6	27
TEMA 7	31
TEMA 8	35
TEMA 9	37
TEMA 10	41
TEMA 11	47
TEMA 12	53
TEMA 13	61
TEMA 14	67
TEMA 15	73
TEMA 16	81
TEMA 17	83
TEMA 18	85

SERVICIO DE PUBLICACIONES DEL MINISTERIO DE EDUCACION Y CIENCIA