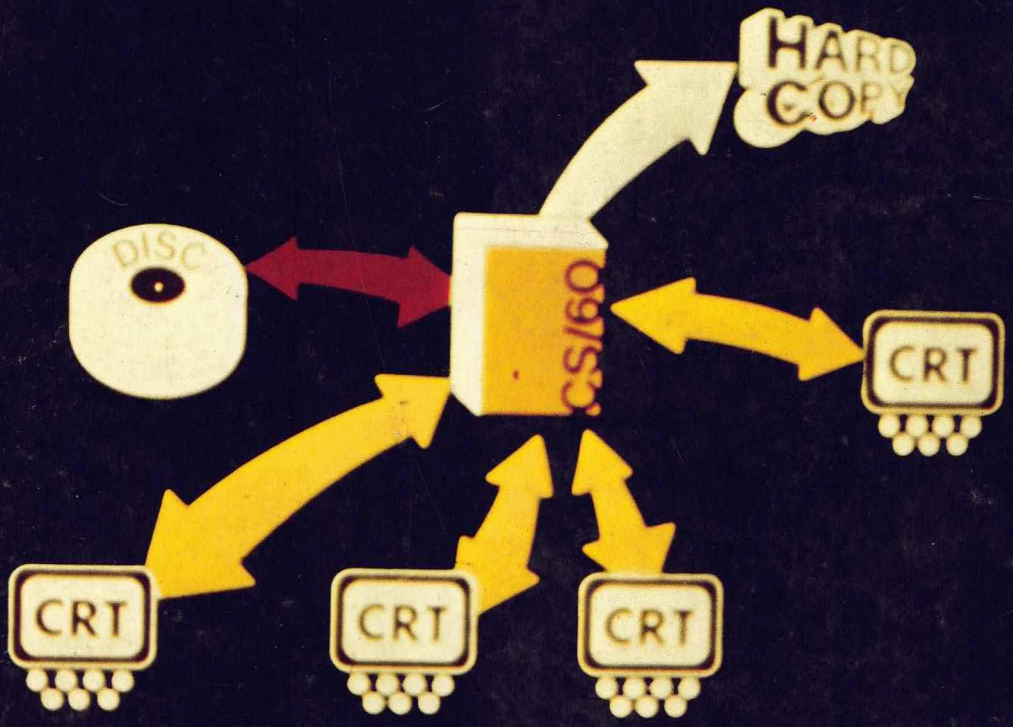
PROGRAMACION DE ORDENADORES

LENGUAJE BASIC



EDITORIAL tesys

TOMO 6

El presente tratado de **Programación de Ordenadores** está compuesto por un conjunto de temas que constituyen la estructura imprescindible para todo aquel que desee un amplio conocimiento del campo de la Programación.

Dichos temas vienen englobados en volúmenes diversos que a continuación se detallan; constituyéndose por separado como unidades monográficas susceptibles de un estudio particular, al margen del conocimiento conjunto y global de todos ellos, como compendio completo para el logro final de programar.

La agrupación por tomos se desarrolla de la siguiente forma:

Tomo 1. **Estructura Básica del Proceso de datos.**

Tomo 2. **Estructura Básica del Proceso de datos.**

Tomo 3. **Estructura Básica del Proceso de datos.**

Tomo 4. **Estructura Básica del Proceso de datos.**

Tomo 5. **Estructura Básica del Proceso de datos.**

Tomo 6. **Lenguaje Basic**

Tomo 7. **Lenguaje Cobol**

Tomo 8. **Lenguaje Cobol**

Tomo 9. **Leyes de Construcción de programas.**

Tomo 10. **Diccionario de Proceso de datos**

El tratamiento de los temas comprende tanto los aspectos teóricos como las aplicaciones al diseño de estructuras básicas de posible uso inmediato, reagrupándose en los siguientes conjuntos específicos que por sí mismos pasan a ser unidades independientes.

Son los siguientes:

Grupo I: Tomos 1, 2, 3, 4 y 5. **(Estructura Básica del Proceso de Datos).**

Grupo II: Tomo 6. **(Lenguaje Basic).**

Grupo III: Tomos 7 y 8. **(Lenguaje Cobol).**

Grupo IV: Tomo 9. **(Leyes de Construcción de Programas).**

PROGRAMACION DE ORDENADORES

LENGUAJE BASIC

PROGRAMACION DE ORDENADORES

LENGUAJE BASIC

AUTOR

JOSE LUIS MORALES PASCUAL
Licenciado en Informática

COLABORADORES

CARLOS ANTONIO MARTINEZ TREVIN
Ingeniero de Telecomunicación

ANDRES J. ERNESTO RUANA GIRONES
Ingeniero Industrial

BARCELONA (ESPAÑA)

© E.C.C. - Barcelona, 1981

Editado por:
EDITORIAL TESYS, Bruch 20, Barcelona-10. (España), 1981

Primera edición: Abril, 1981

Depósito legal: B.: 25.869-1980 (VI)

I.S.B.N. 84-85838-00-9 (Obra completa)
I.S.B.N. 84-85838-06-8 (Tomo 6)

Impreso por:
GRAFOS, Arte sobre papel.
Paseo Carlos I, 157 - Barcelona (13).

Impreso en España - Printed in Spain

CONTENIDO

Introducción 9

1. Elementos del Basic 11

2. Instrucciones del Basic 35

3. Instrucciones de E/S 71

4. Subprogramas y funciones 121

5. Matrices 145

6. Errores de ejecución 169

7. Programas resueltos en Basic 183

Soluciones de los ejercicios 195

Indice general 197

INTRODUCCION

Actualmente existen diversos tipos de lenguajes de programación, cuyas diferencias residen en la orientación del tipo de problemas a resolver, en la facilidad de programación, en la «potencia» del lenguaje y en el tipo de instalaciones en las que se pueden emplear.

Los lenguajes de alto nivel, denominados también lenguajes orientados al problema, tales como FORTRAN, COBOL, BASIC, ALGOL, PL/1, APL, SNOBOL, LISP, etc., poseen la característica fundamental de que, aunque existen diversas variantes de cada uno de ellos, tienen una gran homogeneidad tanto en las instrucciones de lenguaje en sí, como en las *reglas gramaticales* que rigen la escritura de dichas instrucciones. Cualquier programador que aprenda una determinada versión de un lenguaje de alto nivel, podrá seguir programando en otra variante del mismo lenguaje simplemente dando una ojeada al manual que suministra el constructor sobre dicho lenguaje.

En cuanto a los lenguajes de bajo nivel, llamados también lenguajes orientados a la máquina, sólo pueden ser tratados por el ordenador para el que fueron pensados, con lo cual las necesidades del comprador quedan supeditadas a la necesidad de encontrar un programador que conozca dicho lenguaje.

Es evidente la aceptación comercial que poseen los lenguajes de alto nivel, ya que el comprador de un equipo puede centrar el estudio de sus necesidades en la elección de los distintos modelos que le ofrecen los fabricantes, sin la necesidad de tener en cuenta si existirá alguien que conozca el lenguaje del modelo que se ajusta a sus preferencias.

Uno de estos lenguajes de programación de alto nivel es el BASIC, que ha despertado un gran interés desde su creación debido a la sencillez de sus reglas, al rápido acceso que permite al ordenador, a la facilidad de programación de sus instrucciones, y además porque es un lenguaje orientado a trabajar a la vez tanto en *tiempo real* como en *tiempo compartido.*

Por otra parte, su fácil comprensión logra que este lenguaje sea un instrumento útil para entender las bases de la programación, con lo que se facilita mucho la tarea de aprendizaje de otros lenguajes de estructuras más complicadas.

El lenguaje de programación BASIC fue ideado a mediados de la década de los años 60 por John G. Kemeny y Tomas E. Kurtz en el Darmouth College de Hanover (EE.UU.) y por William F. Sharpe en la Universidad de Washington.

La palabra BASIC está formada por las siglas «Beginners All-purpose Simbolic Instructions Code» (Codificación en instrucciones simbólicas de aplicaciones de todo tipo para principiantes).

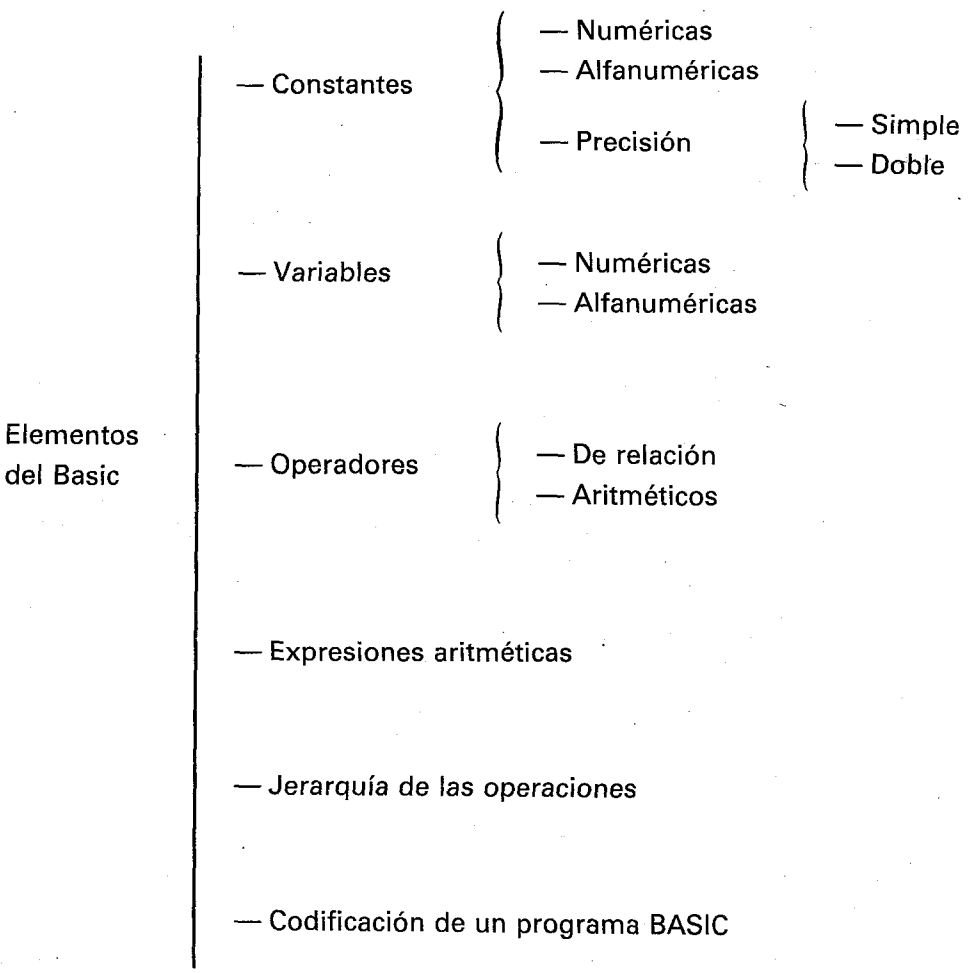
Quizás la mayor ventaja del BASIC ha sido la misma evolución que ha seguido la Informática y el uso cada vez más generalizado de los *mini-computadores,* ya que, debido a la simplicidad de sus reglas gramaticales, es un lenguaje fácilmente adaptable a ordenadores de todo tipo.

El BASIC es un lenguaje artificial de programación cuyas instrucciones son muy similares a la manera en que se escriben las fórmulas matemáticas del álgebra. Gracias a esta similitud es muy sencillo resolver cálculos complicados tanto de ingeniería como de economía, estadística, medicina, contabilidad o de cualquier otra área que tenga la necesidad de un soporte capaz de manejar un gran volumen de datos, de realizar cálculos muy complejos y que, a pesar de las distintas versiones que existan, esté lo suficientemente normalizado como para que cualquier persona que, poseyendo solamente unas ideas generales de Informática y que esté familiarizada con un determinado modelo de ordenador, pueda crear y modificar sus programas en cualquier otro solventando rápidamente los problemas inherentes a la necesidad de adaptación que supone el cambio de ordenador. Finalmente, es un lenguaje muy flexible, lo cual facilita la tarea de introducir modificaciones en un programa.

1.

ELEMENTOS DEL BASIC

CUADRO RESUMEN DE LA LECCION



1.1. CONSTANTES

Una *constante* está formada por una serie de caracteres que permanecen invariables durante la ejecución del programa.

Las constantes aparecen como parte de muchas de las sentencias de un programa BASIC.

Las constantes puede ser de dos tipos:

— Numéricas.

— Alfanuméricas.

1.2.1. Constantes numéricas

En BASIC, se denominan *constantes numéricas* o *números* a las cantidades numéricas.

Los caracteres permitidos en una constante numérica son los siguientes:

— Los diez dígitos decimales (0, 1, ..., 9).

— El punto decimal (.).

— Los signos aritméticos (+ y −).

— El caracter «E».

Las cantidades numéricas pueden expresarse en dos formas: como cantidades enteras y como cantidades decimales.

Ejemplos:

Números enteros	*Números decimales*
38	−38.05
0	0.5
−572	+ 1.974
+ 6	− 0.0062

Las cantidades enteras o números enteros, son las constantes que no poseen el punto decimal. Las cantidades decimales son los números que tienen el punto decimal.

El punto decimal, por tanto, realiza la función de elemento separador entre la parte entera y la parte decimal. Esta función debe ser realizada, en BASIC, por un *punto* (no deben aparecer comas en ninguna parte de la constante numérica).

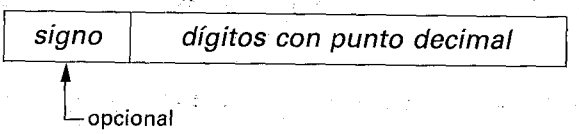
Las constantes numéricas pueden ser: positivas, negativas o cero.

Las constantes numéricas positivas son las que vienen precedidas por el signo +, o bien las que no poseen signo, ya que el signo + es opcional.

Las constantes numéricas negativas son las que vienen precedidas por el signo −, es decir, el primer caracter de la constante es el signo negativo (−).

Una constante numérica puede representarse en cualquiera de las formas siguientes: en *forma no exponencial* o en *forma exponencial.*

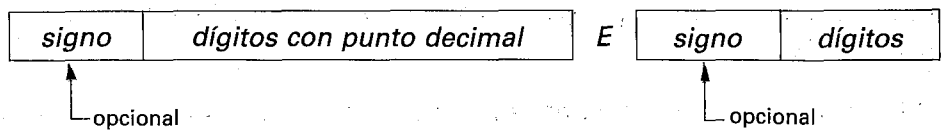
Forma no exponencial:



Ejemplos:

Valor aritmético	*Forma no exponencial*
2529,35	2529.35
−0,05	−0.05
+45678,3001	+45678.3001
123	123

Forma exponencial:



La letra E significa «por 10 a la potencia». Se emplea cuando el valor de la constante es un número real escrito en *forma exponencial* (o *notación científica*).

Los caracteres de la constante numérica que aparecen a la derecha de la letra E constituyen el *exponente* y los caracteres de la constante que se encuentran a la izquierda de la letra E forman la *mantisa.* Por consiguiente, el valor de una constante numérica es igual al valor numérico (entero o decimal) de la mantisa multiplicado por 10 elevado a la potencia que indica el exponente.

El exponente puede ser cualquier número entero que al ser combinado con la mantisa no determine que el valor de la constante numérica sobrepase los valores admisibles por el ordenador.

Dichos valores varían según la marca y modelo de ordenador, por ejemplo, en algunos ordenadores, el valor de la constante debe ser superior a 10^{-50} e inferior a 10^{+50}.

La letra E está siempre precedida de un dígito. Por ejemplo E2 representa una variable mientras que 1E2 representa la constante 100 (es decir, 1×10^2).

Ejemplos:

Valor aritmético	*Forma exponencial*
−12,3	−123 E−1
1,2345	123.45 E−2
1234,5	12.345 E+2
0,1234567	1234567 E−7
12300	+123 E+2
12300	123 E2

Existen varias constantes numéricas cuyo uso es muy frecuente en los procesos de cálculo.

Una de las ventajas que ofrece el BASIC al programador consiste en que dichas constantes numéricas son definidas durante el proceso de traducción realizado por el compilador Basic. Dichas constantes reciben el nombre de *constantes internas.*

Ejemplo de constantes internas son las constantes, π, e, $\sqrt{2}$, ..., cuyos valores respectivos son 3.141592 ..., 2.718281 ..., 1.414213 ..., están definidos en el ordenador con los nombres &PI, &E, &SQR2, ...

1.1.2. Constantes alfanuméricas

Las constantes alfanuméricas están formadas por caracteres alfanuméricos:

— Caracteres numéricos (0, 1, 2, ..., 9).

— Caracteres alfabéticos (A, B, C, ..., Z. Además, en Basic, se consideran caracteres alfabéticos los caracteres: @, $, #.)

— El carácter «blanco» ( ).

— Caracteres especiales (+, −, *, /, %, ?, ...).

Las constantes alfanuméricas se representan entre caracteres limitadores.

El lenguaje BASIC, admite dos tipos de caracteres limitadores distintos, pudiendo ser empleado cualquiera de ellos:

— Las comillas simples (').

— Las comillas dobles (").

La condición que ha de cumplirse con ellos es que tanto el carácter limitador que indica el principio de la constante alfanumérica, como el que indica el final, han de ser el mismo.

Cuando se desea que un carácter limitador aparezca dentro de una constante alfanumérica, debe codificarse por duplicado en la posición que debe ocupar. Con esto se logra que, en la fase de compilación, el compilador entienda que todavía no ha alcanzado el final de la constante y traduzca el carácter limitador (incluido doble dentro de la constante) como un carácter más perteneciente a dicha constante.

Si por el contrario, se utiliza un carácter limitador para definir la constante alfanumérica y se quiere incluir dentro de la definición de dicha constante el otro tipo de carácter limitador, no se debe codificar por duplicado.

Las constantes alfanuméricas se emplean para definir títulos, encabezamientos, etc., en el listado de resultados de un programa, para lograr una mayor legibilidad de los datos obtenidos.

Ejemplos de constantes alfanuméricas son:

Constante	*Significado*
'ABC'	ABC
"ABC"	ABC
'A5'	A5
"X9"	X9
'25 %'	25 %
"VALOR DE 'A' ="	VALOR DE 'A' =
'VALOR DE "A" ='	VALOR DE "A" =
"VALOR DE "" A" " ="	VALOR DE "A" =
"VALOR DE ' 'A' ' ="	VALOR DE "A" =
'VALOR DE " "A" " ='	VALOR DE " "A" " =

1.1.3. Precisión

Para calcular el perímetro de una circunferencia, por ejemplo, se emplea la fórmula:

$$P = 2 * \pi * R$$

siendo:

P = Perímetro de la circunferencia.
R = Radio de la circunferencia.
π = (Pi) constante matemática cuyo valor aproximado es 3,141592653589793...

Si el radio son 3 centímetros y se considera que $\pi = 3{,}14$, el perímetro será:

$$2 * 3{,}14 * 3 = 18{,}84 \text{ centímetros}$$

— Si se considera que $\pi = 3{,}1416$, el perímetro será:

$$2 * 3{,}1416 * 3 = 18{,}8496 \text{ centímetros.}$$

— Si $\pi = 3{,}141593$, entonces el perímetro será:

$$2 * 3{,}141593 * 3 = 18{,}849556 \text{ centímetros.}$$

— Si $\pi = 3{,}141592654$, el perímetro será:

$$2 * 3{,}141592654 * 3 = 18{,}84955592 \text{ centímetros.}$$

Se va viendo que a medida que aumenta la exactitud o precisión del número π, se obtienen más dígitos en el resultado:

18,84 = 4 dígitos significativos.
18,8496 = 6 dígitos significativos.
18,849556 = 8 dígitos significativos.
18,84955592 = 10 dígitos significativos.

En BASIC se puede elegir entre:

— Simple Precisión (8 dígitos significativos).
— Doble Precisión (16 dígitos significativos).

Simple precisión

La instrucción que define la Simple Precisión es:

ENTER BASIC

Cuando se compile un programa BASIC en Simple Precisión, todas las constantes numéricas y todas las variables numéricas, ocuparán campos en memoria con una longitud fija de cuatro octetos, con lo que se logra un máximo de ocho dígitos significativos.

La Simple Precisión se especifica tanto con la instrucción ENTER BASIC, como con su omisión.

Ejemplo de los valores almacenados por un programa compilado en Simple Precisión son:

Constante	*Valor*
1234.5678912	1234,5678
12345678E-1	1234567,8
123456789123E-5	1234567,8
&PI	3,141593
&E	2,718282
&SQR2	1,414214

Doble Precisión

La instrucción que define la Doble Precisión es:

```
ENTER BASICL
```

Cuando se compila un programa BASIC en Doble Precisión, todas las constantes numéricas y todas las variables numéricas del programa, ocuparán campos de memoria con una longitud de ocho octetos, con lo que se logrará un máximo de 16 dígitos significativos. La Doble Precisión es poco empleada ya que normalmente nunca son necesarios tantos dígitos significativos.

Ejemplo de los valores almacenados por un programa compilado en Doble Precisión son:

Constante	*Valor*
12345.67891234567812	12345,67891234567
12345678912345678912E-15	12345,67891234567
12345E12	12345000000000000
&PI	3,141592653589793
&E	2,7182818284859045
&SQR2	1,414213562373095

1.2. VARIABLES

Una variable es la representación simbólica de una serie de caracteres que ocupan un área (o campo) de la memoria. Es decir, una variable es un nombre que representa una constante.

Tanto la asignación de los campos de memoria pertenecientes a un programa, como los nombres simbólicos para referenciarlos, son generados durante la fase de compilación.

Los datos obtenidos dentro de cualquier variable, pueden ser consultados, tratados y modificados durante la ejecución del programa.

Las variables son elementos imprescindibles en cualquier tipo de problemas, ya que permiten almacenar la información necesaria para su resolución.

Para hacer referencia a una variable, bastará con indicar el nombre con el que fue generada, siendo tarea del ordenador el reconocer a qué posición de memoria corresponde dicho nombre.

Al igual que en las constantes, existen dos tipos de variables:

— Variables numéricas.
— Variables alfanuméricas.

1.2.1. Variables numéricas

Una variable numérica hace referencia a un campo de la memoria destinado a almacenar constantes numéricas.

El nombre de una variable numérica se representa por una letra cualquiera del abecedario (o uno de los caracteres @, $, #) o con una letra seguida por uno de los diez dígitos (del 0 al 9). Además, el primer carácter debe ser alfabético.

Algunos ejemplos de nombres de variables numéricas son:

D2	H8	P2	A0
K7	A	Z9	G
@4	X4	X	E2

No sirven como nombres de variables numéricas, por ejemplo, los siguientes:

ABC (porque hay demasiados caracteres alfabéticos).
B72 (demasiados caracteres numéricos).
3K (el primer carácter no es alfabético).

En el instante de empezar a ejecutarse un programa en BASIC, todas las variables numéricas definidas dentro del programa son automáticamente inicializadas con el valor cero.

1.2.2. Variables alfanuméricas

Una variable alfanumérica hace referencia a un campo de la memoria destinado a almacenar constantes alfanuméricas.

Una variable alfanumérica se representa mediante una letra seguida por el signo $.

Por ejemplo, son nombres de variables alfanuméricas los siguientes:

A$ K$ D$ Z$ W$

Las variables alfanuméricas son de longitud fija: siempre tienen 18 caracteres.

En el instante de iniciar la ejecución de un programa en BASIC, todas las variables alfanuméricas definidas dentro de él son automáticamente inicializadas con 18 caracteres blancos.

1.3. OPERADORES

Los operadores son caracteres especiales utilizados para especificar las operaciones que han de efectuarse entre dos constantes, dos variables o una constante y una variable.

Los operadores pueden ser:

— Operadores de relación.
— Operadores aritméticos.

1.3.1. Operadores de relación

Los operadores de relación permiten establecer comparaciones entre dos elementos.

Por medio de los operadores de relación se indican las condiciones de igualdad o desigualdad para que pueda ejecutarse una bifurcación condicional en BASIC.

Los operadores de relación son los siguientes:

Operador	*Significado*
=	igual a
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
<>	no igual a

Los tres últimos operadores también pueden representarse intercambiando los signos:

=<	igual o menor que
=>	igual o mayor que
><	no igual a

Los elementos situados tanto a la izquierda como a la derecha de un operador de relación, han de ser del mismo tipo: numéricos o alfanuméricos, puesto que el ordenador no puede establecer una relación de comparación entre un campo numérico y un campo alfanumérico.

Ejemplos:

```
A$ = B$        W$><7        C2=>9.17
C<>K4          A$ = 'SUMA'  p4=<100
Z1<=Z2         C2>=9        'SUMA' > J$
```

Los siguientes ejemplos son incorrectos:

```
4.95 > B$        Z = C$
```

1.3.2. Operadores aritméticos

Los operadores aritméticos son caracteres especiales que permiten la ejecución de las operaciones aritméticas entre dos elementos de tipo numérico:

Operador	*Significado*
+	Suma
−	Resta
*	Multiplicación
/	División
↑ ó **	Potenciación
−	Negación

Ejemplos:

Operación	*Operador*	*Ejemplo*	*Significado*
Suma	+	A + B	A más B
Resta	−	A − B	A menos B
Multiplicación	*	A * B	A por B
División	/	A / B	A dividido por B
Exponenciación	↑ ó * *	A ↑ B ó A * * B	A elevado a B
Negación	−	− A	menos A

Las cantidades negativas solamente pueden ser elevadas a potencias enteras. Esto se debe a la forma en que el ordenador calcula las potencias:

— Si el exponente es un número entero, la base se multiplica por sí misma tantas veces como indique el exponente.

— Si el exponente es un número decimal, calcula el logaritmo de la base, lo multiplica por el exponente y obtiene el resultado calculando el antilogaritmo.

Un número negativo no puede ser elevado a una potencia decimal (los números negativos no tienen definidos sus antilogaritmos).

Sea el caso, por ejemplo, de H8 ↑ 5.23. Será compilado normalmente, pero si durante la fase de ejecución resulta que el valor de H8 es negativo, se obtendrá un mensaje de ERROR y el ordenador abandonará la ejecución del programa.

1.4. EXPRESIONES ARITMETICAS

Las expresiones aritméticas están formadas por combinaciones de constantes numéricas, variables numéricas y operadores aritméticos. Se utilizan para definir los cálculos del programa.

Ejemplos:

Expresión en Basic	*Significado algebraico*
A + 5.374	a + 5,374
25 /P8	25/p8
B * H/2	b * h/2
− K ↑ 2	$-(K^2)$
K1 − K2 + 84 * 9.2E3/A7	$K1 - K2 + 84 * 9{,}2 * 10^3/a7$

Cuando sea necesario poner dos operadores juntos en una expresión aritmética deben separarse mediante paréntesis, por ejemplo: A5 * (− A4).

1.5. JERARQUIA DE LAS OPERACIONES EN LAS EXPRESIONES ARITMETICAS

Los lenguajes de programación no deben tener ambigüedades. Para que el ordenador sepa exactamente el orden en que deben realizarse las operaciones aritméticas dentro de una expresión, los lenguajes de programación establecen un orden de prioridad (jerarquía) de las operaciones:

Prioridad	*Operación*
1	Exponenciación (↑)
2	Multiplicación (*) y división (/)
3	Suma (+) y resta (−)

Por ejemplo, A * B + C ↑ 2 puede interpretarse como «multiplicar A por B, sumar el resultado a C y elevar a la potencia 2 el resultado de esta suma». Pero según el orden de prioridad de las operaciones lo anterior sería incorrecto. Entonces esa expresión significa: «elevar C a la potencia 2, después multiplicar A por B y por último sumar el resultado de esta multiplicación a C elevado a 2».

En el caso de que tengan que efectuarse varias operaciones del mismo nivel, se realizan de izquierda a derecha. Por ejemplo, A/B/C, primero se divide A por B y el resultado se divide por C.

Las reglas de jerarquía pueden alterarse por el uso de paréntesis. Por ejemplo, en la expresión A + B * C las operaciones no se efectúan en el mismo orden que la expresión (A + B) * C. En este caso se realiza primero la suma (A + B) y después se multiplica el resultado por C.

En todas las expresiones en donde aparezcan paréntesis, las primeras operaciones a efectuar son las que se encuentran dentro de los paréntesis internos, a continuación las de los paréntesis externos y por último las de fuera de paréntesis.

Ejemplo 1:

$$A * (B/(C * D) + E) - F \uparrow 3$$

En primer lugar se efectúa la operación C * D, después la división, a continuación la suma, la exponenciación (F ↑ 3), la multiplicación por A y por último la resta.

Ejemplo 2:

$$[(B + C) - (D/E \uparrow F) \uparrow (G * H) + 2.7] * H/E$$

1.° $X_1 = B + C$
2.° $X_2 = E \uparrow F$
3.° $X^3 = D / X_2$
4.° $X_4 = G * H$
5.° $X_5 = X_3 \uparrow X_4$
6.° $X_6 = X_1 - X_5$
7.° $X_7 = X_6 + 2.7$
8.° $X_8 = X_7 * H$
9.° $X_9 = X_8 / E$
10.° Valor resultado $= X_9$

1.6. CODIFICACION DE UN PROGRAMA BASIC

Las instrucciones de un programa en BASIC están compuestas por constantes, variables y palabras claves.

Una *palabra clave* es una palabra del lenguaje BASIC que indica el ordenador el tipo de tratamiento a realizar.

La codificación de un programa BASIC se rige por las siguientes reglas gramaticales:

— Cada instrucción se escribe en una línea (generalmente, una línea equivale a 72 caracteres).

— Cada instrucción debe ir precedida por un número entero (de 1 a 5 dígitos) llamado *número de línea.*

— No pueden existir, dentro de un mismo programa, dos números de línea iguales.

— Los números de línea indican el orden en que serán ejecutadas las instrucciones.

— Los números de línea están codificados en orden creciente. La primera instrucción a ejecutar será la que tenga el número de línea más pequeño.

— Las instrucciones pueden introducirse en cualquier orden, pero el orden de ejecución será el que indiquen los números de línea (de menor a mayor).

— A continuación del número de línea se codifica una palabra clave que es la que describe el tipo de operación.

— A continuación de la palabra clave hay una expresión que contiene los datos a tratar.

— A continuación, y de manera opcional, puede escribirse un pequeño comentario que haga referencia a la parte del proceso que realiza la instrucción.

El formato general de una instrucción BASIC es el siguiente:

número de línea	*palabra clave*	*datos*	*comentario*

Ejemplo:

290	*PRINT*	*«ESTADISTICA MENSUAL»*	*'CABECERA*
Número de línea	Palabra clave	Datos	Comentario

El BASIC, al ser un lenguaje orientado al programador, permite una cierta libertad a la hora de codificar las instrucciones de un programa para lograr un mayor grado de legibilidad. El lenguaje BASIC, utilizado siguiendo un método racional y lógico, facilita el desarrollo de la lógica del programa aunque dicho programa sea muy complicado.

Una vez escrito un programa en lenguaje BASIC, puede ser ejecutado en cualquier ordenador (siempre que dicho ordenador tenga previsto el lenguaje BASIC como uno de sus lenguajes de trabajo y disponga del compilador Basic).

En la *fase de compilación,* las instrucciones del programa son traducidas (compiladas) al lenguaje máquina por el compilador.

La segunda etapa que sigue un programa BASIC es la *fase de ejecución.* Si el compilador detecta algún error en la codificación de las instrucciones, el programa no pasará a la fase de ejecución y se obtendrá automáticamente un listado en donde el propio compilador indica el tipo de error y en dónde se ha producido, para que el programador realice las oportunas correcciones.

Para lograr una mayor facilidad en la corrección de los errores y en la introducción de modificaciones, es norma general entre los programadores el etiquetar (numerar) las instrucciones de diez en diez en lugar de hacerlo en orden consecutivo. Así, cuando se precisa intercalar nuevas instrucciones, existirán etiquetas intermedias suficientes.

Ejemplo:

Intercalar dos nuevas instrucciones entre las instrucciones 740 y 750.

```
    720 .....
    730 .....
    740 .....
→   750 .....
    760 .....
```

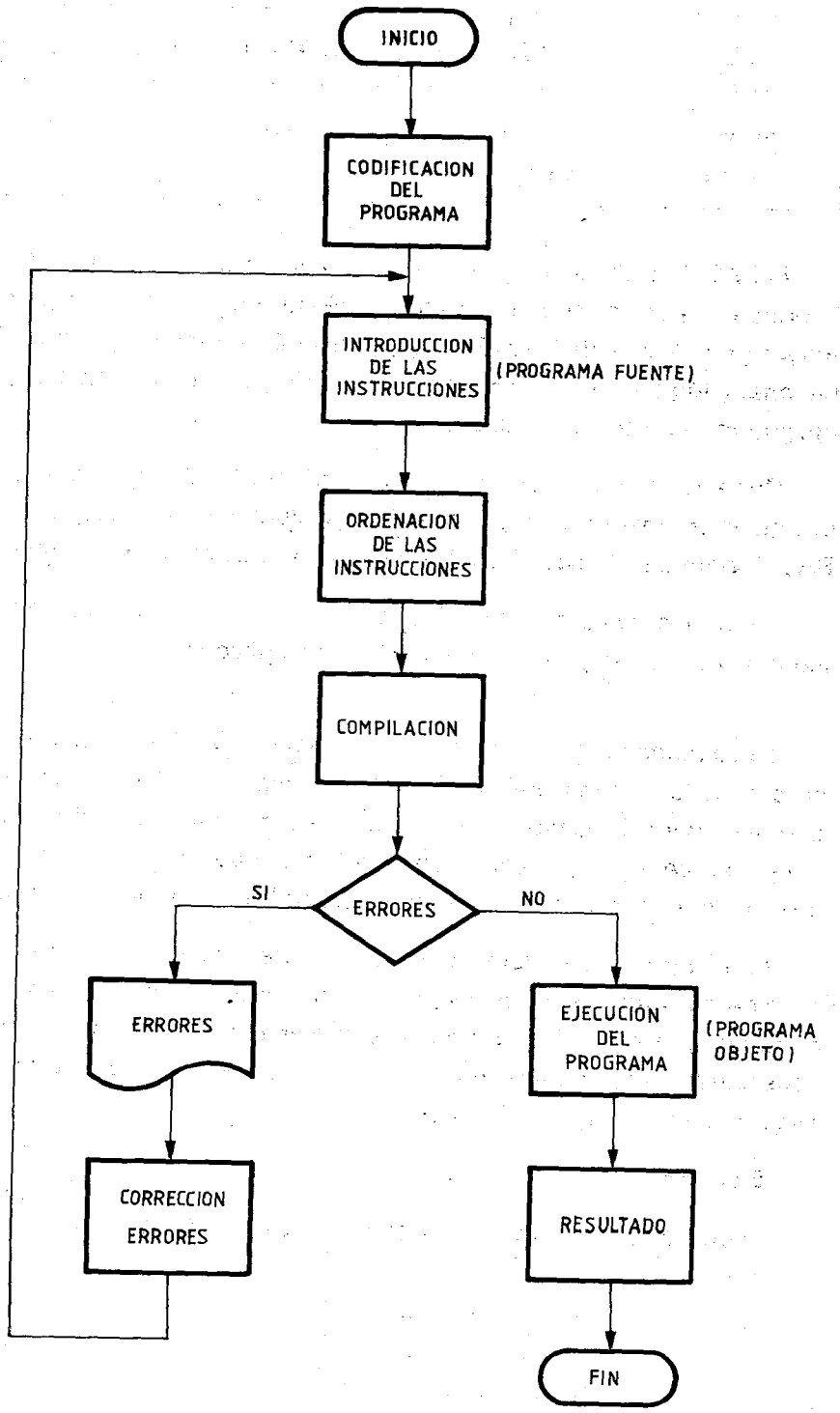

INICIO
CODIFICACION DEL PROGRAMA
INTRODUCCION DE LAS INSTRUCCIONES
(PROGRAMA FUENTE)
ORDENACION DE LAS INSTRUCCIONES
COMPILACION
ERRORES
SI
NO
ERRORES
CORRECCION ERRORES
EJECUCION DEL PROGRAMA
(PROGRAMA OBJETO)
RESULTADO
FIN

Fig. 1.1

El programa queda ampliado de esta manera:

```
720 .....
730 .....
740 .....
744 _____
747 _____
750 .....
760 .....
```

Pero si se hubieran codificado las etiquetas en orden consecutivo, no sería posible la intercalación. Por ejemplo:

```
 72 .....
 73 .....
 74 .....
→
 75 .....
 76 .....
```

En este caso hay que volver a codificar todo el programa de nuevo, *reetiquetando* las instrucciones:

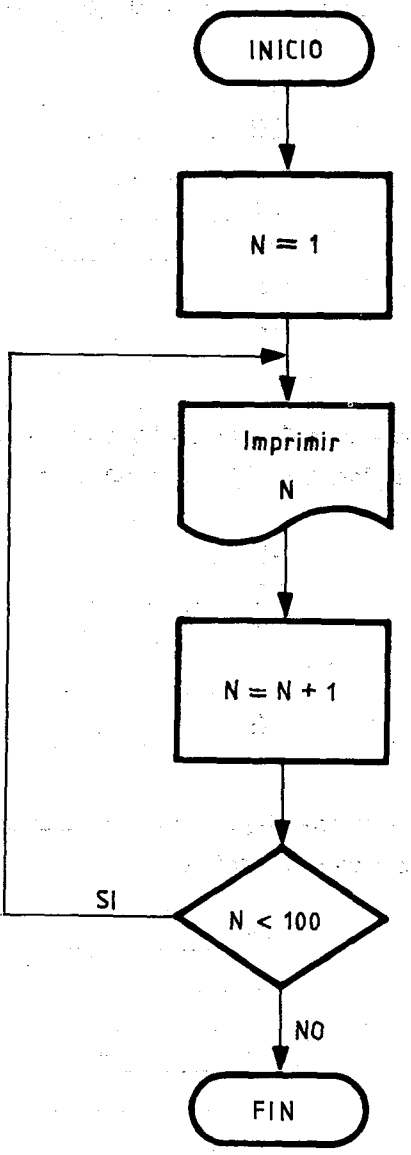
```
72 .....
73 .....
74 .....
75 _____
76 _____
77 .....
78 .....
```

y como existen instrucciones que hacen referencia, mediante la etiqueta, a otras instrucciones, se deberá repasar todo el programa. En este ejemplo la tarea de intercalar dos instrucciones, es muy laboriosa y además el programador puede equivocarse al modificar una etiqueta, con lo que se obtendrán resultados imprevisibles.

Para facilitar las posibles modificaciones de un programa, el BASIC permite introducir las instrucciones en cualquier orden, encargándose el compilador de ordenarlas según la etiqueta codificada delante de cada instrucción.

Ejemplo:

Sea el organigrama siguiente:



Las instrucciones del programa correspondiente a este organigrama se pueden introducir en cualquier orden; por ejemplo:

```
30  IMPRIMIR N
50  SI N < 100 IR A 30
20  N = 1
60  FIN
10  INICIO
40  N = N + 1
```

Y el compilador las ordena así:

```
10  INICIO
20  N = 1
30  IMPRIMIR N
40  N = N +
50  SI N < 100 IR A 30
60  FIN
```

Si se desea intercalar la instrucción IMPRIMIR N * 105 entre las instrucciones 20 y 30 se introduce al principio con la etiqueta 25, por ejemplo, ó a continuación del último renglón:

```
30  ESCRIBIR N
50  SI N < 100 IR A 30
20  N = 1
60  FIN
10  INICIO
40  N = N + 1
25  IMPRIMIR N * 105
```

Y el compilador ordenará las instrucciones así:

```
10  INICIO
20  N = 1
25  IMPRIMIR N * 105
30  ESCRIBIR N
40  N = N + 1
50  SI N < 100 IR A 30
60  FIN
```

EJERCICIOS

1. El BASIC es un lenguaje de:

   a) Alto nivel.
   b) Bajo nivel.
   c) Nivel medio.

2. Las constantes pueden ser de dos tipos Num Alf

3. ¿Qué tres caracteres especiales son considerados alfabéticos en BASIC?
   @ $ #

4. La precisión puede ser:

   a) Simple o doble.
   b) Compuesta o normal.
   c) Variable o fija.

5. Las constantes alfanuméricas se definen entre caracteres limitadores

6. ¿Cuáles son los caracteres limitadores del BASIC? comillas simples o dobles

7. La jerarquía de las operaciones establece que entre una suma y una multiplicación se ejecutará:

   a) Primero la suma y luego la multiplicación.
   b) Primero la multiplicación y luego la suma.
   c) La situada más a la izquierda.

8. Los operadores pueden ser de relacion o aritmeticas

9. Las expresiones aritméticas están compuestas por operadores aritmeticos constantes variables

10. ¿En qué orden han de introducirse las instrucciones de un programa BASIC dentro del ordenador? En el que se quiera

11. ¿Con qué deben separarse dos operadores aritméticos que se escriben juntos? ( )

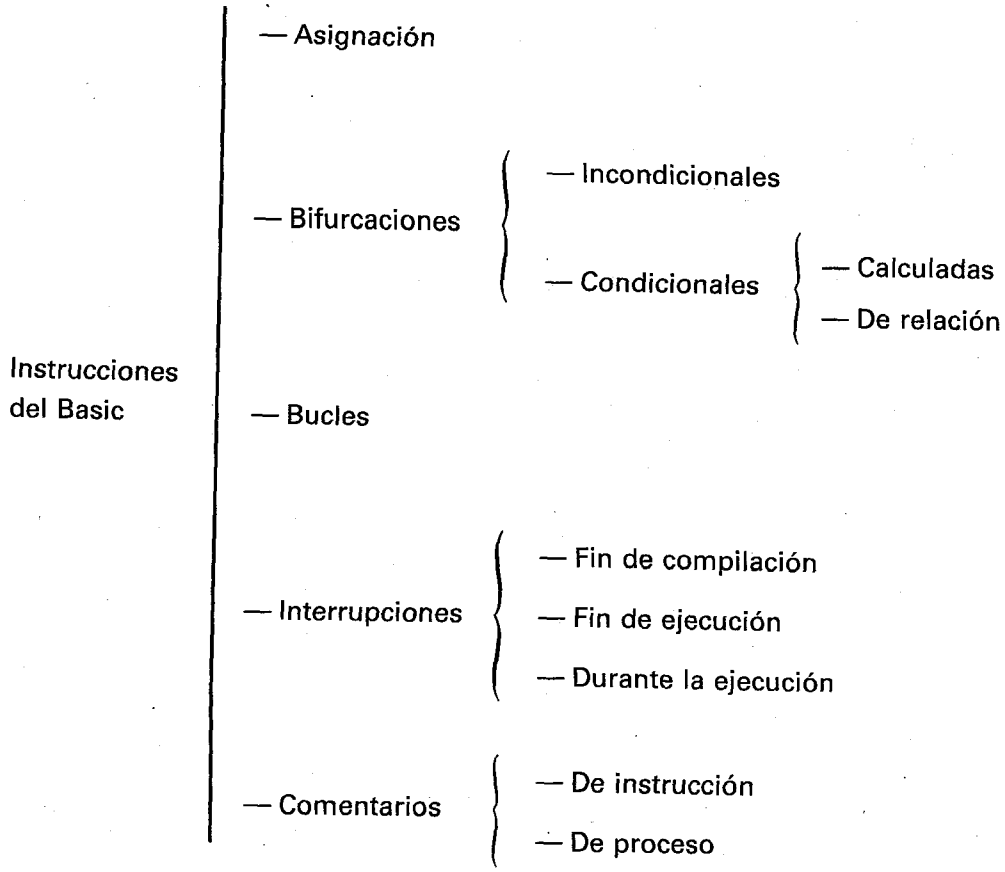
12. ¿Cuántos dígitos deben tener como máximo un número de línea?

    a) Cinco.
    b) Siete.
    c) Tres.

2.

INSTRUCCIONES DEL BASIC

CUADRO RESUMEN DE LA LECCION



Las instrucciones que componen el lenguaje de programación BASIC se clasifican en los siguientes grupos:

- Asignaciones
- Bifurcaciones
- Bucles
- Interrupciones
- Comentarios
- Entrada/salida
- Funciones
- Subrutinas
- Series
- Matrices

Se muestran a continuación las palabras (*palabras clave*) utilizadas en las instrucciones del lenguaje BASIC, junto a su significado.

Palabras clave	*Ejemplos*	*Significado*
CLOSE	9400 CLOSE 3	Cierra el fichero número 3.
DATA	500 DATA 1,2, NOMBRE	Crea un bloque de datos formado por las constantes 1,2 NOMBRE
DEF	40 DEF FNA (X1) = X1 * 2	Define una función con el nombre FNA que devuelve el doble del valor que recibe.

Palabras clave	*Ejemplos*	*Significado*
DIM	30 DIM A (3,5)	Define una matriz de nombre A de 3 filas y 5 columnas.
END	9010 END	Termina la compilación del programa.
FILE	70 FILE DATOS, 30	Crea un fichero con el nombre DATOS y le asigna 30 pistas.
FNEND	95 FNEND	Indica el final de definición de una función.
FOR	130 FOR N = 1 TO 34 STEP 2	Ejecuta un bucle variando el valor de la variable N desde 1 hasta 34 sumando cada vez 2 a la variable N.
GET	300 GET 3: A1, A2	Lee del fichero número 3 un registro que contiene dos datos numéricos y los asigna a las variables A1 y A2.
GO TO	50 GO TO 180	«Ir al número de línea 180». Se bifurca a la instrucción de la línea 180.
GO TO ... ON	70 GO TO 100, 200 ON X1	Si el valor de la variable X1 es 1, se bifurca a la instrucción de la línea 100. Si el valor de X1 es 2, se bifurca a la instrucción de la línea 200. Para cualquier otro valor de X1, se continúa en secuencia.
GOSUB	840 GOSUB 2500	Bifurcar a la subrutina que empieza en el número de línea 2500.
GOSUB ... ON	920 GOSUB 1180, 1250 ON X2	Bifurcar a la subrutina que empieza en la línea 1180 si el valor de X2 es 1, y a la línea 1250 si el valor de X2 es 2. Para cualquier otro valor de X2 se continúa en secuencia.
IF ... THEN	90 IF A < 3 THEN 340	Si el valor de la variable A es inferior a 3, entonces bifurcar a la instrucción de la línea 340.

Palabras clave	*Ejemplos*	*Significado*
IF ... THEN	90 IF B = 4 THEN B1 = B2	Si el valor de la variable B es 4, entonces asignar a la variable B1 el valor de la variable B2.
IF ... THEN	70 IF A = 1 AND B = 2 THEN 90	Si el valor de la variable A es 1 y el valor de la variable B es 2, entonces bifurcar a la línea 90.
IF ... THEN	70 IF A = 1 OR B = 2 THEN 90	Si el valor de la variable A es 1 o el valor de la variable B es 2, entonces bifurcar a la línea 90.
INPUT	430 INPUT R1, X, D2	Pide la introducción de tres valores numéricos por el terminal de entrada y los asigna a las variables R1, X y D2.
LET	20 LET J = J + 1	A la variable J se le asigna el valor de J + 1 (es decir, se suma 1 a J).
NEXT	180 NEXT N	Indica el final de un bucle que varía la variable N.
NL	570 NL	El terminal de salida avanza una línea.
OPEN	200 OPEN 3, "FICH-5", INPUT	Prepara el fichero secuencial FICH-5 para lectura y le asigna el número de referencia 3.
OPEN	200 OPEN 3, "FICH-5", OUTPUT	Prepara el fichero secuencial FICH-5 para grabarle registros y le asigna el número 3.
OPEN	200 OPEN 3, "FICH-5", APPEND	Prepara el fichero secuencial FICH-5 para añadirle más registros y le asigna el número 3.

Palabras clave	*Ejemplos*	*Significado*
OPEN	200 OPEN 2, "FICH-4", UPDATE	Prepara el fichero aleatorio FICH-4 para la modificación de sus registros y le asigna el número 2.
OPEN	200 OPEN 2, "FICH-4", INQUIRY	Prepara el fichero aleatorio FICH-4 para lectura y le asigna el número 2.
OPEN	200 OPEN 4, "FICH-9", INPUT, LINE	Prepara el fichero secuencial FICH-9 para lectura, incluyendo como dato el número de línea de los registros.
PAUSE	580 PAUSE DESEA CONTINUAR?	Interrumpe la ejecución del programa y saca el mensaje «¿Desea continuar?».
PRINT	120 PRINT "A ="; A	Escribe en el terminal de salida la constante alfanumérica A = y a continuación escribe el valor de la variable A.
PRINT ... USING	580 PRINT A, B, C USING 10	Escribe en el terminal de salida los valores de las variables A, B y C en la forma que indica la instrucción 10.
PUT	980 PUT 3: A3, A4, A5	Graba en el fichero número 3 un registro con los tres datos numéricos de las variables A3, A4 y A5.
READ	350 READ B7, C4, A$	Almacena en las variables B7, C4 y A$ datos de un bloque de datos.
REM	70 REM ENTRADA DE DATOS	Comentario que explica el por qué de las instrucciones que encabeza.
RESET	1020 RESET 3:	Pone el puntero del fichero número 3 apuntando el primer registro.

Palabras clave	*Ejemplos*	*Significado*
RESTORE	80 RESTORE $	Devuelve el puntero del bloque de datos alfanuméricos a la primera posición.
RETURN	2900 RETURN	Bifurca a la instrucción siguiente a la que llamó a la subrutina.
SET	2040 SET 3: 180	Sitúa el puntero del fichero número 3 apuntando al registro número 180.
STOP	9000 STOP	Terminar la ejecución del programa.
TAB	750 PRINT TAB (5), B	Pone cinco blancos antes de imprimir el valor de la variable B.

Otras palabras clave utilizadas comúnmente en BASIC son las siguientes:

Palabras clave	*Ejemplos*	*Significado*
ERR	800 IF ERR THEN 1020	En lugar de cancelar el programa debido a algún error, bifurcar a la línea 1020.
OVERFLOW	800 IF OVERFLOW THEN 1020	Si aparece un error por sobrepasar la capacidad máxima de almacenamiento de ordenador en alguna operación aritmética, bifurcar a la línea 1020.
UNDERFLOW	50 ON UNDERFLOW GO TO 200	Si aparece un error por sobrepasar la capacidad mínima de almacenamiento del ordenador en alguna operación aritmética, bifurcar a la línea 200.

Palabras clave	*Ejemplos*	*Significado*
ZERODIVIDE	50 ON ZERODIVIDE GO TO 200	Si aparece un error debido al intento de dividir por cero en alguna operación aritmética, bifurcar a la línea 200.
EOF	800 IF EOF THEN 1020	Cuando se encuentre la marca de fin de fichero en una operación de entrada/salida, bifurcar a la línea número 1020.
MAT	750 MAT LET A = B * C	Indica que A, B y C son matrices.

Las palabras clave que indican *funciones* son las siguientes:

Palabras clave	*Ejemplos*	*Significado*
LTW	50 LET A = LTW (B)	Asigna a la variable A el valor del logaritmo en base 2 del valor almacenado en la variable B.
LGT	50 LET A = LGT (B)	Asigna a la variable A el valor del logaritmo decimal del valor almacenado en la variable B.
SQR	50 LET A = SQR (B)	Asigna a la variable A el valor de la raíz cuadrada del número almacenado en la variable B.
INT	50 LET A = INT (B)	Asigna a la variable A el valor ~~de la raíz cuadrada~~ del número almacenado en la variable B sin parte decimal (truncada).
SGN	50 LET A = SGN (B)	Asigna a la variable A el signo del valor almacenado en la variable B.

C = INT (A/B)

Palabras clave	*Ejemplos*	*Significado*
CPU	50 LET A = CPU	Asigna a la variable A el tiempo que lleva ejecutándose un programa en segundos.
CLK	50 LET A = CLK	Asigna a la variable A la hora del reloj del ordenador en segundos.
DAY	50 LET A = DAY	Asigna a la variable A la fecha (*día*).
ABS	80 LET A = ABS (A1)	Asigna a la variable A el valor absoluto de la variable A1.
SIN	40 LET A = SIN (B)	Asigna a la variable A el valor del seno trigonométrico del ángulo en radianes almacenado en la variable B.
COS	40 LET A = COS (B)	Asigna a la variable A el valor del coseno trigonométrico del ángulo en radianes almacenado en la variable B.
TAN	40 LET A = TAN (B)	Asigna a la variable A el valor de la tangente trigonométrica del ángulo en radianes almacenado en la variable B.
COT	40 LET A = COT (B)	Asigna a la variable A el valor de la cotangente trigonométrica del ángulo en radianes almacenado en la variable B.
SEC	40 LET A = SEC (B)	Asigna a la variable A el valor de la secante trigonométrica del ángulo en radianes almacenado en la variable B.
CSC	40 LET A = CSC (B)	Asigna a la variable A el valor de la cosecante trigonométrica del ángulo en radianes almacenado en la variable B.
ASN	40 LET A = ASN (B)	Asigna a la variable A el valor del arco seno del ángulo en radianes almacenado en la variable B.

Palabras clave	*Ejemplos*	*Significado*
ACS	40 LET A = ACS (B)	Asigna a la variable A el valor del arco coseno del ángulo en radianes almacenado en la variable B.
ATN	40 LET A = ATN (B)	Asigna a la variable A el valor del arco tangente del ángulo en radianes almacenado en la variable B.
HSN	40 LET A = HSN (B)	Asigna a la variable A el valor del seno hiperbólico del ángulo en radianes almacenado en la variable B.
HCS	40 LET A = HCS (B)	Asigna a la variable A el valor del coseno hiperbólico del ángulo en radianes almacenado en la variable B.
HTN	40 LET A = HTN (B)	Asigna a la variable A el valor de la tangente hiperbólica del ángulo en radianes almacenado en la variable B.
DEG	40 LET A = DEG (B)	Asigna a la variable A el ángulo en grados del ángulo en radianes almacenado en la variable B.
RAD	40 LET A = RAD (B)	Asigna a la variable A el ángulo en radianes del ángulo en grados almacenado en la variable B.
EXP	50 LET A = EXP (B)	Asigna a la variable A el valor del exponente neperiano del valor almacenado en la variable B.
LOG	50 LET A = LOG (B)	Asigna a la variable A el valor del logaritmo neperiano del valor almacenado en la variable B.
ZER	370 MAT LET A = ZER	Pone el valor cero en todos los elementos de la matriz A.

Palabras clave	*Ejemplos*	*Significado*
CON	380 MAT LET B = CON	Pone a unos todos los elementos de la matriz B.
STR	800 LET Y$ = STR (Z$, 7, 2)	Asigna a la variable Y$ los caracteres séptimo y octavo de la variable Z$.
VAL	810 LET X = VAL (X$)	Asigna a la variable X el valor numérico de los caracteres de la variable X$.
IDN	390 MAT LET C = IDN	Pone a unos los elementos de la diagonal principal y a ceros los elementos restantes de la matriz C.
RND	550 LET A3 = RND	Asigna a la variable A3 un valor aleatorio.
TRN	600 MAT LET D = TRN (E)	Asigna a la matriz D la matriz traspuesta de la matriz E..
INV	610 MAT LET F = INV (G)	Asigna a la matriz F la matriz inversa de la matriz G.

Y por último, existen otras palabras clave utilizadas por el ordenador llamadas «comandos». Por ejemplo:

Palabras clave	*Ejemplos*	*Significado*
ENTER	ENTER BASICL	Indica al compilador que el programa debe compilarse en Doble Precisión.
GOODBYE	GOODBYE	Fin de la utilización de un terminal.
LIST	LIST	Proporciona un listado del programa que hay en la memoria interna del ordenador.

Palabras clave	*Ejemplos*	*Significado*
NEW	NEW	Vacía la memoria interna del ordenador.
READY	READY	El ordenador informa al operador a través de terminal de salida que está esperando.
RETURN	RETURN	Final de un dato.
RUN	RUN	Compilar y ejecutar el programa almacenado en la memoria interna del ordenador.
SAVE	SAVE "FIC-7"	Graba un fichero con el nombre FIC-7.

2.1. ASIGNACION

Para asignar valores numéricos o alfanuméricos a una o más variables se utiliza la instrucción LET. Las instrucciones LET también sirven para definir las variables que utiliza el programa.

La forma de representar esta instrucción (*formato*) es:

$$\textbf{LET } V_1, V_2, V_3, \ldots, V_n = X$$

donde:

LET = Asignar.
V_i = Nombres de variables.
X = Una constante, una variable o una expresión aritmética.

Ejemplo 1:

Asignar el valor 1 a las variables numéricas A, B y C.

Una de las formas de la codificación es la siguiente:

```
50  LET A = 1
60  LET B = 1
70  LET C = 1
```

Es decir, cada asignación se codifica en una línea con un número de línea distinto.

Estas tres instrucciones indican al ordenador que debe poner el valor 1 en los campos A, B y C.

La palabra LET puede omitirse, quedando el mismo significado:

```
50  A = 1
60  B = 1
70  C = 1
```

Estas asignaciones pueden codificarse todas en una línea formando una sola instrucción:

```
50  LET A, B, C = 1
```

O también, omitiendo la palabra clave LET:

```
50  A, B, C = 1
```

Existen otras formas de codificar estas asignaciones, por ejemplo en dos líneas:

```
50  A, B = 1
60  C = 1
```

Cada uno de estos grupos de instrucciones logran el mismo objetivo.

En la fase de compilación se reservan campos de memoria para las variables V_i y se asocian dichos campos a los nombres que tiene cada variable. De esta manera es como el ordenador define las variables utilizadas en un programa.

Cuando se ejecuta una instrucción de asignación, se transfiere a cada una de las variables (situadas a la izquierda del signo «=») el valor del elemento situado a la derecha del signo «=».

A la izquierda del signo «=» se pueden especificar tantas variables como sea necesario. Si aparece más de una, se utiliza el carácter coma (,) para separar las variables entre sí.

Nunca existirán dos campos de memoria (pertenecientes a un mismo programa) referenciados con el mismo nombre. Por lo tanto, si el compilador encuentra otra instrucción de asignación con un nombre que ya ha sido asignado a un campo de memoria con anterioridad, ignorará dicha orden.

En una instrucción de asignación, los valores situados a la izquierda del signo «=» así como los situados a su derecha, deberán ser del mismo tipo, no estando permitidos los siguientes puntos:

— Mezclar variables numéricas con variables alfanuméricas.
— Asignar constantes numéricas a variables alfanuméricas.
— Asignar constantes alfanuméricas a variables numéricas.

Ejemplo 2:

— Asignar la constante SUMA a la variable alfanumérica A$.

Puede codificarse la palabra clave LET y utilizarse el carácter delimitador *comillas* (").

```
90  LET A$ = "SUMA"
```

Esta instrucción también puede codificarse utilizando el carácter delimitador *comillas simples* (').

```
90  LET A$ = 'SUMA'
```

Otra forma de codificar esta instrucción es omitiendo la palabra clave LET:

```
90  A$ = "SUMA"
```

O también:

```
90  A$ = 'SUMA'
```

2.2. BIFURCACIONES

Un programa en lenguaje BASIC se ejecuta secuencialmente. Pero hay situaciones en las cuales es necesario transferir el control del procesador a otra parte del programa (alterando de esta manera la secuencia de ejecución). Las instrucciones que realizan este cometido se llaman instrucciones de *bifurcación* y se clasifican en:

— Incondicionales.
— Condicionales.

2.2.1. Bifurcaciones incondicionales

El formato de la instrucción que bifurca siempre (incondicionalmente) hacia otra instrucción es:

GO TO n

siendo:

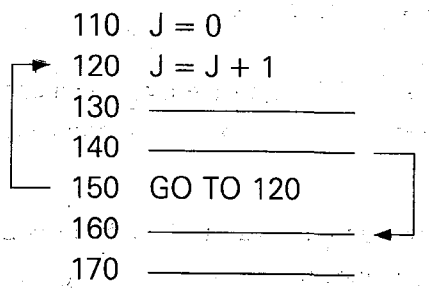
n = El número de línea de la instrucción a donde se bifurca.
GO TO = Ir hacia.

Por ejemplo:

```
10 GO TO 40
```

Esta instrucción transfiere el control del programa a la instrucción con el número de línea 40, continuando a partir de dicho número de línea con la ejecución del programa y en orden secuencial.

Supongamos el siguiente programa:



En este ejemplo, se ejecutan todas las instrucciones en orden secuencial hasta llegar a la línea 140, la cual contiene una bifurcación a la línea 160. Si esta bifurcación se realiza, la siguiente línea a ejecutar es la 160 y a partir de ella se continuará con la ejecución secuencial del programa. Pero si dicha bifurcación no se produce, la siguiente línea a ejecutar será el 150, con lo que se transferirá el control del procesador a la línea 120 (en donde se incrementa en 1 el valor de J).

2.2.2. Bifurcaciones condicionales

Existen ocasiones en las que es necesario transferir el control del procesador a una instrucción del programa siempre y cuando se cumpla una determinada condición.

Las bifurcaciones condicionales se dividen en dos tipos:

— Calculadas.
— De relación.

Bifurcaciones calculadas

El formato de una instrucción de bifurcación calculada es:

GO TO n – 1, n – 2, n – 3, ..., n – i **ON** X

donde:

GO TO = Ir hacia
ON = Según
n – i = Números de línea dentro del programa
X = Variable numérica o expresión aritmética

Esta instrucción, llamada comúnmente *GO TO calculada,* transfiere el control del procesador a la n – 1 si el valor de X es 1, a la n – 2 si dicho valor es 2, a la n – 3 si X vale 3.

Si el valor de X es inferior a 1 o superior a la cantidad de números de línea, se ejecuta la instrucción siguiente en secuencia.

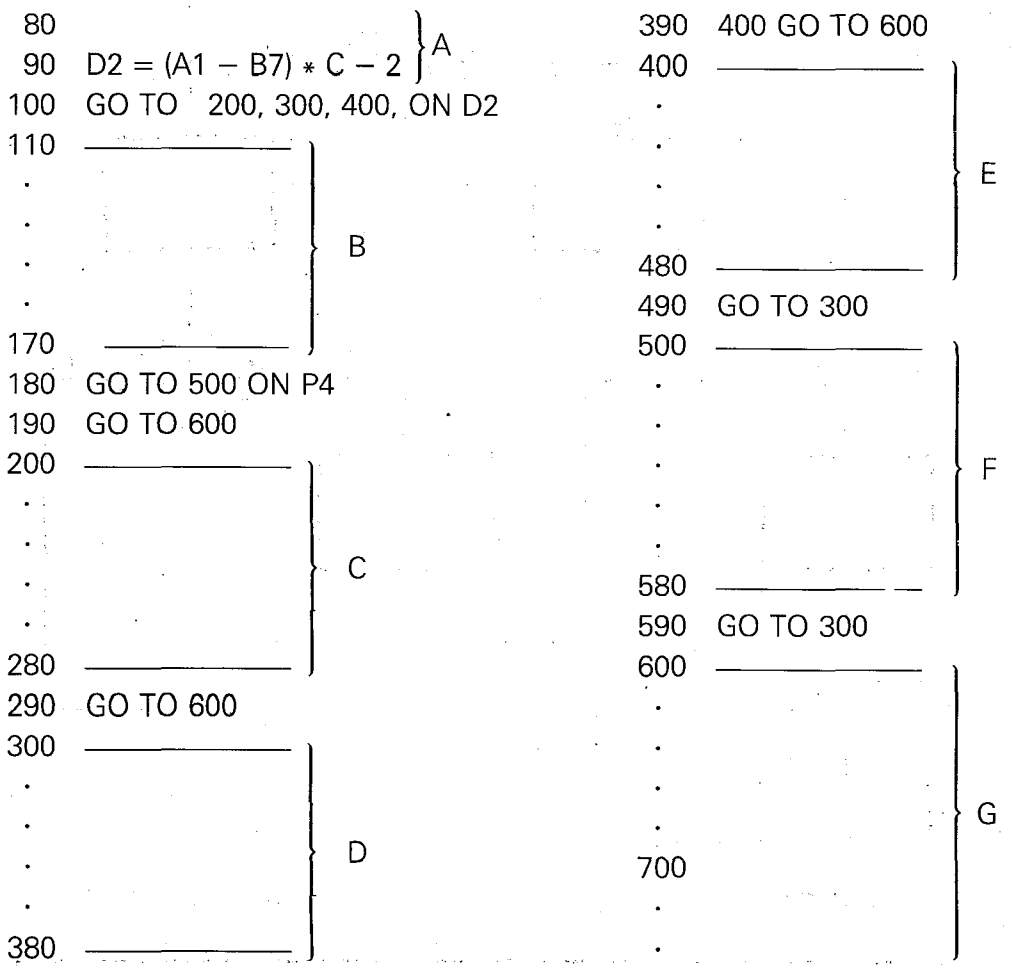
En el caso de que el valor X tenga decimales, dichos decimales serán truncados (es decir, se elimina la parte decimal del número).

Si el campo X es una expresión aritmética, deberá ir encerrada entre paréntesis.

Ejemplo:

Supongamos un fragmento de un programa dado. El organigrama correspondiente a este fragmento del programa se presenta en la figura 2.1.

Una de las múltiples maneras de codificación es la siguiente:



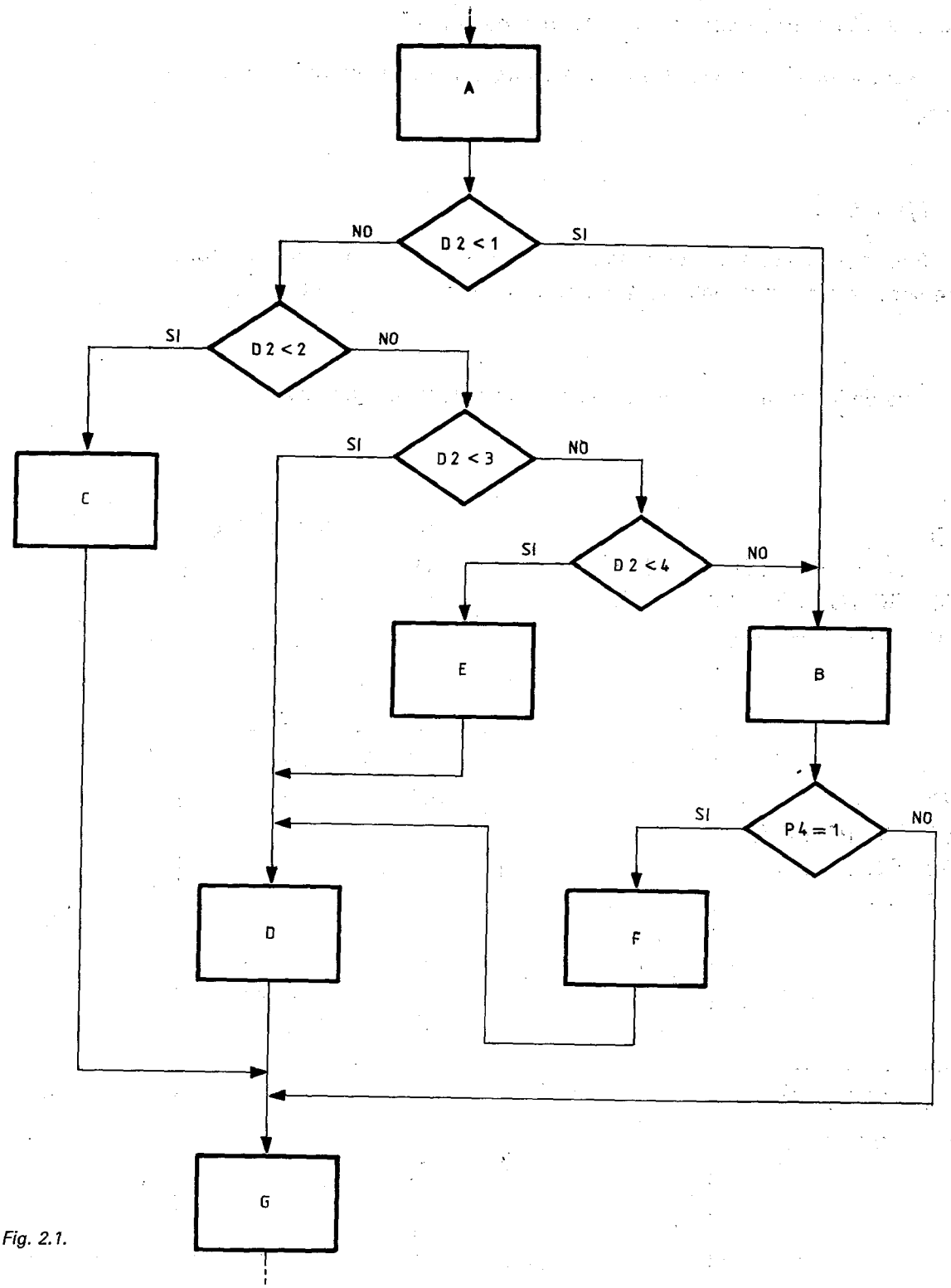


Fig. 2.1.

Este es el orden de ejecución que seguirá esta parte del programa según los distintos valores de las variables A1, B7, C, D2 y P4:

Variables					
A1	*B7*	*C*	*D2*	*P4*	*Secuencia del programa*
2	1	3	1	17	A – C – G
4	2	2	2	24	A – D – G
3	–2	1	3	–0.5	A – E – D – G
5	2	2	4	1.2	A – B – F – D – G
1	2	3	–5	3	A – B – G

También pueden codificarse las instrucciones 90 y 100 en una sola del siguiente modo:

```
 80  ————————————
100  GO TO 200, 300, 400 ON [(A1 – B7) * C – 2]
110  ————————————
```

Bifurcaciones de relación

El formato de las instrucciones de bifurcación condicionadas por una relación es:

$$\text{IF } X_1 \text{ R } X_2 \left\{\begin{matrix}\text{AND}\\\text{OR}\end{matrix}\right\} X_3 \text{ R } X_4 \left\{\begin{matrix}\text{AND}\\\text{OR}\end{matrix}\right\} \ldots \left\{\begin{matrix}\text{THEN}\\\text{GO TO}\end{matrix}\right\} n$$

siendo:

X_i = Constantes, variables o expresiones aritméticas.
R = Operador de relación.
n = Número de línea.
IF = Si condicional
AND = Y
OR = O
THEN = Entonces
GO TO = Ir hacia

Esta instrucción bifurca a n si son ciertas las relaciones (X_1 R X_2), (X_3 R X_4), ... y la bifurcación es ignorada en caso contrario.

Ejemplo 1:

Volviendo al ejemplo visto en el apartado de las instrucciones de bifurcación incondicionales, supongamos que se bifurca desde el renglón 140 al 160 cuando se cumpla la condición:

$$(A * B) \uparrow H > 57.28$$

Con lo que resulta la siguiente codificación:

```
110  J = 0
120  J = J + 1
130  ————————————
140  IF (A * B) ↑ H > 57.28 THEN 160
150  GO TO 120
160  ————————————
170  ————————————
```

La instrucción 140 también podía haber sido codificada así:

```
140  IF (A * B) ↑ H > 57.28 GO TO 160
```

Ejemplo 2:

Codificar de nuevo el programa que corresponde al organigrama de bifurcaciones calculadas utilizando esta vez instrucciones de bifurcación por relación.

```
.
.
.                                  } A
80  ————————————
90  D2 = (A1 − B7) * C − 2
100 IF D2 < 1 THEN 110
101 IF D2 < 2 THEN 200
102 IF D2 < 3 THEN 300
103 IF D2 < 4 THEN 400
110 ————————————
.
.                                  } B
.
180 IF P4 = 1 THEN 500

190 GO TO 600
200 ————————————
.
.                                  } C
.
.
280 ————————————
290 GO TO 600
300 ————————————
.
.                                  } D
.
Etcétera
```

El bloque de instrucciones 90 a la 103 también podía haber sido codificado de la siguiente forma:

```
 80  ————————————
 90  IF (A1 – B7) * C – 2 < 1 THEN 110
100  IF (A1 – B7) * C – 2 < 2 THEN 200
101  IF (A1 – B7) * C – 2 < 3 THEN 300
102  IF (A1 – B7) * C – 2 < 4 THEN 400
110  ————————————
```

Sin embargo esta última codificación introduce pérdidas de tiempo para el programador y para el ordenador. Para cada instrucción de bifurcación, el ordenador debe realizar el cálculo de la expresión

$$(A1 - B7) * C - 2$$

En un programa sencillo, este segundo factor no se aprecia (dada la rapidez de cálculo de la unidad aritmético/lógica), pero en un programa muy complicado, la pérdida de tiempo que introduce un programa mal pensado tiene repercusiones de tipo comercial (eficacia, coste, rentabilidad, rapidez, etc.).

La palabra clave AND indica que deben cumplirse las dos condiciones entre las que se encuentra codificada para que se ejecute la bifurcación.

Ejemplo 3

La codificación:

```
490  ————————————
500  IF A = 1 AND B = 2 THEN 1000
510  ————————————
```

es equivalente a:

```
490  ————————————
500  IF A ≠ 1 THEN 510
505  IF B = 2 THEN 1000
510  ————————————
```

La palabra clave OR indica que basta que se cumpla una de las dos condiciones entre las que está codificada para que se ejecute la bifurcación.

Ejemplo 4

La codificación:

```
490 ————————————
500 IF A = 1 OR B = 2 THEN 1000
510 ————————————
```

es equivalente a:

```
490 ————————————
500 IF A = 1 THEN 1000
505 IF B = 2 THEN 1000
510 ————————————
```

2.3. BUCLES

Un bucle consta de una serie de instrucciones que deben ejecutarse un determinado número de veces dentro de un programa de forma reiterativa.

Un bucle está definido por las instrucciones FOR y NEXT, cuyo formato es:

```
n – 1  FOR V = X1 TO X2 STEP X3
          .
          .
n – 2  NEXT V
```

donde:

n – 1 y n – 2 = Números de líneas
FOR = Para
TO = Hasta
STEP = Dar pasos
NEXT = Siguiente
V = Variable numérica

X_1, X_2 y X_3 = Constante numérica, variable numérica o expresión aritmética.

El funcionamiento de un bucle es el siguiente:

1.° La variable V es inicializada con el valor de X_1.

2.° Se ejecutan las instrucciones pertenecientes al bucle.

3.° Antes de ejecutar la instrucción NEXT, la variable V se incrementa con el valor de X_3.

4.° Cuando el procesador ejecuta la instrucción NEXT, pueden darse dos casos: *a)* Si el valor de V es superior al valor de X_2, se ejecuta la instrucción siguiente de NEXT. *b)* Si el valor de V no supera X_2, se transferirá el control a la siguiente instrucción en secuencia de la instrucción FOR.

Si no se especifica el parámetro STEP X_3, el ordenador asume automáticamente que el valor del incremento es la unidad.

Las reglas que han de seguirse en la construcción de un bucle son:

— En el bucle pueden consultarse los valores de V, X_1, X_2 o X_3, pero no pueden ser modificados.

— La entrada a un bucle, ha de ser siempre a través de la instrucción FOR.

— Si un bucle contiene a otro, las instrucciones del segundo bucle deben estar incluidas dentro del primero.

— Un bucle puede contener en su interior un máximo de 14 bucles.

— Pueden emplearse las mismas variables en dos bucles siempre que no estén contenidos uno dentro del otro.

— Se puede salir en cualquier momento desde un bucle hacia cualquier otra parte del programa sin necesidad de *completar* el bucle. Esta posibilidad de salida se denomina *salida extraordinaria* (se llama *salida ordinaria* cuando el bucle ha sido completado).

— Es posible salir de un bucle y volver a entrar en él en cualquier instrucción posterior a la utilizada, siempre que no se modifiquen los parámetros del bucle y la entrada se efectúe en una instrucción posterior a la de salida.

Ejemplo 1:

Supongamos la siguiente parte de un determinado programa:

```
500  K2 = 1
510  ————————————
  .                 }
  .                 }   A
  .                 }
610  ————————————
620  K2 = K2 + K1
625  IF K2 < = K3 THEN 510
630  ————————————
```

En esta codificación, el programador construye el bucle mediante instrucciones de asignación y de bifurcación.

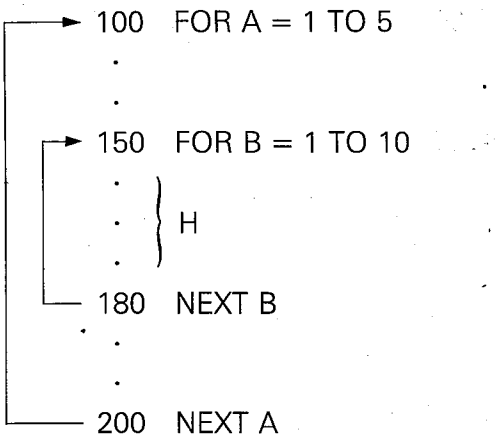
Otra codificación equivalente sería:

```
500  FOR K2 = 1 TO K3 STEP K1
510  ————————————
  .                 }
  .                 }   A
  .                 }
610  ————————————
620  NEXT K2
630  ————————————
```

En esta otra codificación, el programador utiliza las instrucciones FOR y NEXT para construir el mismo bucle. Ambas codificaciones significan lo mismo, pero la segunda codificación es más cómoda para el programador.

Ejemplo 2:

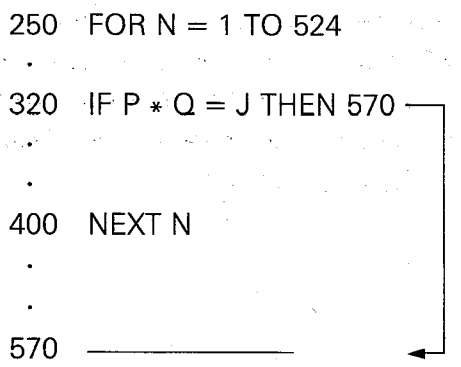
Un bucle dentro de otro bucle:



En este ejemplo, el bloque de instruciones H se ejecuta diez veces para cada uno de los cinco valores de A, por lo tanto el bloque H se ejecuta un total de cincuenta veces.

Ejemplo 3:

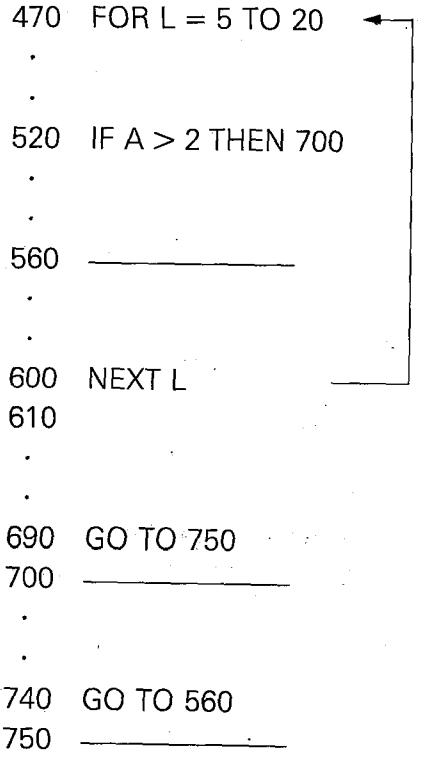
Bucle con salida extraordinaria:



Cuando se cumple la condición P * Q = J en la instrucción 320, el procesador ejecutará una salida extraordinaria hacia la instrucción 570.

Ejemplo 4:

Regreso a un bucle:



En este ejemplo, la secuencia será la siguiente:

Si el valor de A es superior a 2 en la instrucción 520, el procesador bifurcará a la instrucción 700, continuará ejecutando en secuencia hasta la instrucción 740 y regresará a la instrucción 560 dentro del bucle. Una vez completado el bucle, seguirá en secuencia a partir de la instrucción 610 hasta la 690, saltando hacia la 750 (la 740 bifurca a una instrucción perteneciente al bucle y esto no está permitido cuando el bucle ha sido completado).

2.4. INTERRUPCIONES

Una interrupción en un programa es la detención momentánea o definitiva, de la ejecución de las instrucciones de dicho programa.

Las causas que motivan una interrupción son debidas a:

A) Un error cometido por el ordenador o cualquier unidad periférica motivado por el mal estado físico de un soporte de información, o por el desajuste de alguno de los múltiples y complicados circuitos internos, porque la temperatura ambiente es superior a la soportable por el ordenador, etc.

Este tipo de interrupciones provoca la inmediata cancelación del programa en ejecución debido a la incapacidad del ordenador para continuar con la ejecución de la tarea que estuviera realizando.

Sin embargo, la probabilidad de una interrupción de este tipo es mínima gracias al grado de desarrollo que poseen los ordenadores actuales.

B) El tipo de interrupción más común se debe a los errores de programación, bien porque las instrucciones del programa no estén codificadas de acuerdo a las reglas gramaticales del lenguaje (con lo cual el programa es rechazado por el compilador), o bien porque la lógica que sigue el programa no es la adecuada para el problema tratado.

C) También existen interrupciones debidas a errores no previstos en la lógica del programa como son la introducción incorrecta de datos por parte del operador, el intento de añadir nuevos datos a un fichero que no dispone de suficiente espacio para su almacenamiento, ordenar la lectura de un dato en un fichero cuando ya no existen más datos disponibles, etc.

Estas interrupciones también provocan la cancelación del programa. Para evitarlas, el programador debe prever tanto los errores humanos como los errores lógicos del modo siguiente:

— Cuando el programa necesita la introducción de datos a través del terminal, codificar inmediatamente después de la instrucción de entrada, las instrucciones necesarias para comprobar la validez de los datos suministrados por el operador.

— Prever los errores en la ejecución de las operaciones aritméticas.

D) Finalmente existe un tipo de interrupciones provocadas por el programador cuya finalidad estudiaremos en este capítulo. Estas interrupciones son:

— De compilación.
— De ejecución.
— De espera.

2.4.1. Interrupciones de compilación

Cuando se introduce un programa en lenguaje BASIC en el ordenador, el primer paso consiste en compilar el programa. El ordenador necesita una indicación de final de compilación. Dicha indicación se logra con la instrucción END, cuyo formato es:

```
END c
```

siendo:

END = Fin
c = Comentario

Una vez introducidas todas las instrucciones de un programa, el compilador las va traduciendo una a una al lenguaje interno de la máquina. Si el compilador detecta un error de sintaxis en una instrucción, interrumpe momentáneamente la compilación, activa un código de error interno para evitar que el programa pueda ejecutarse, da un mensaje explicativo del tipo de error detectado y continúa interpretando las instrucciones.

Cuando el compilador encuentra la instrucción END, entiende que ha alcanzado el final del programa. Interrumpe de forma definitiva la compilación y si el código de error de compilación está activado, el programa no entra en la fase de ejecución.

Por tanto, la instrucción END debe ser la última instrucción dentro de un programa, ya que de haber más instrucciones a continuación serán ignoradas por el compilador. De manera que sólo puede codificarse una instrucción END en cada programa.

El comentario de la instrucción END aparecerá en el terminal del usuario cuando el compilador interrumpa su trabajo. Este comentario es optativo.

2.4.2. Interrupciones de ejecución

En la fase de ejecución, es necesario indicar al procesador cuándo debe finalizar la ejecución del programa. Esto se logra mediante la instrucción STOP, cuyo formato es:

STOP c

en donde:

STOP = Parar
c = Comentario

Cuando el procesador encuentra una instrucción de este tipo, interrumpe definitivamente la ejecución del programa, destruye todas las definiciones de las constantes y el contenido de las variables, borra el programa y prepara la memoria central para poder iniciar la ejecución de otro trabajo.

Dentro de un programa pueden codificarse tantas instrucciones STOP como sean necesarias.

El comentario c es opcional y puede emplearse para indicar al operador en dónde y por qué se ha dado por finalizada la ejecución del programa.

Ejemplo:

```
 190  IF A ≠ Ø THEN 11Ø
1100  STOP EL VALOR DE LA VARIABLE 'A' ES INCORRECTO
1110  C = B/A
1540  STOP PROGRAMA FINALIZADO CORRECTAMENTE
1580  END
```

Después de efectuar una serie de cálculos en el bloque B, se comprueba que la variable A no sea nula, en cuyo caso la instrucción 1110 no podría ser ejecutada por el ordenador ya que no es posible dividir por cero.

2.4.3. Interrupciones de espera

Existen casos en los que es necesario interrumpir momentáneamente la ejecución de un programa. Esto se logra mediante la instrucción PAUSE. Su formato es:

PAUSE c

siendo:

PAUSE = Esperar
c = Comentario

Cuando el procesador ejecuta una instrucción de este tipo, detiene la ejecución del programa e imprime el siguiente mensaje por la terminal del usuario:

PAUSE AT LINE n c

siendo:

n = Número de línea de la instrucción PAUSE ejecutada.
c = El comentario codificado en dicha instrucción.

A continuación el ordenador queda a la espera de que el procesador ordene continuar con la secuencia de ejecución.

Mediante la instrucción PAUSE se puede avisar al operador de la acción que debe realizar y que el programa se mantenga a la espera hasta que no la haya finalizado.

Ejemplo:

Supongamos un programa que calcula la nómina de cada empleado de una determinada empresa:

```
10   PRINT "YA ESTOY CALCULANDO LA NOMINA"
.
.    } A
.
8000 PAUSE PONGA EL PAPEL C - 2 EN LA IMPRESORA
.
.    } B
.
9000 STOP FIN DE LA NOMINA
9010 END NO HAY ERRORES EN EL PROGRAMA DE NOMINA
```

En este programa, primero se ejecuta el bloque de instrucciones A, donde se calcula la nómina correspondiente a cada empleado. Una vez finalizados los cálculos, se imprimen las nóminas en los formularios C – 2 (estos formularios

deben entregarse al Ministerio de Trabajo para su posterior inspección). Pero antes de la impresión, que se realiza en el bloque de instrucciones B, se interrumpe la ejecución con la instrucción 8000. Cuando el usuario haya preparado la impresora con los formularios C – 2, dará la orden de que continúe la ejecución del programa, con lo que se ejecutará el bloque B y el programa finalizará.

En el terminal del operador aparecerán los siguientes comentarios:

Al finalizar la compilación:

```
NO HAY ERRORES EN EL PROGRAMA DE NOMINA
```

Al empezar la ejecución:

```
NO HAY ERRORES EN EL PROGRAMA DE NOMINA
YA ESTOY CALCULANDO LA NOMINA
```

Al ejecutar la instrucción 8000:

```
NO HAY ERRORES EN EL PROGRAMA DE NOMINA
YA ESTOY CALCULANDO LA NOMINA
PAUSE AT LINE 80000 PONGA EL PAPEL C-2 EN LA IMPRESORA
```

Al finalizar la ejecución:

```
NO HAY ERRORES EN EL PROGRAMA DE NOMINA
YA ESTOY CALCULANDO LA NOMINA
PAUSE AT LINE 80000 PONGA EL PAPEL C-2 EN LA IMPRESORA
FIN DE LA NOMINA
```

2.5. COMENTARIOS

Es conveniente que la lógica de todo programa pueda ser seguida leyendo los comentarios explicativos de cada uno de los pasos.

Una de las razones es la posible necesidad de modificar un programa después de algún tiempo, con lo que será imposible recordar cuál era la finalidad de las distintas instrucciones, qué conceptos contenían las variables, qué tarea realizaban cada una de la distintas funciones, etc. Por ello los comentarios son de gran utilidad para una lectura posterior del programa.

Los comentarios también son útiles a la hora de crear un programa que realiza varias tareas, con el fin de saber qué grupo concreto de instrucciones cumplen una determinada finalidad. Por tanto, dentro de un programa en BASIC pueden codificarse tantos comentarios como sea preciso.

En la fase de compilación, los comentarios son ignorados por el compilador por lo que no tienen ninguna repercusión sobre el Lenguaje Objeto resultante (siempre y cuando hayan sido codificados de acuerdo a las reglas gramaticales del lenguaje BASIC).

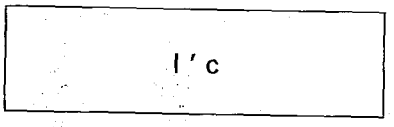
Como se ve, la finalidad de los comentarios es explicar el Lenguaje Fuente del programa.

Los comentarios se clasifican en:

— De instrucción.
— De proceso.

2.5.1. Comentarios de instrucción

Los comentarios a las instrucciones se codifican de la siguiente manera:



siendo:

I = Instrucción a comentar.
' = Apóstrofe.
c = Texto del comentario.

Las reglas de codificación de los comentarios son:

— Los comentarios de instrucción van situados en el mismo renglón de la instrucción a comentar.

— Siempre después de la instrucción, y en la columna que se considere más conveniente, se codificará el texto que se considere más oportuno.

— El texto del comentario debe empezar siempre con el carácter apóstrofe.

— El límite del texto del comentario es el final del renglón.

Ejemplo:

```
280 X1 = ( - B + SQR (B ↑ 2 - 4 * A * C)) / (2 * A)     ' CALCULA RAIZ 1
290 X2 = ( - B - SQR (B ↑ 2 - 4 * A * C)) / (2 * A)     ' CALCULA RAIZ 2
```

2.5.2. Comentarios de proceso

Los comentarios de proceso se codifican de la siguiente manera:

REM c

donde:

REM = Comentario.
c = Texto del comentario.

Una manera optativa para aumentar la legibilidad de estos comentarios, es intercalar renglones en blanco antes y después de las instrucciones REM para que destaquen dentro del contexto general del lenguaje fuente codificado.

```
10  REM ESTE PROGRAMA CALCULA LA ENERGIA CINETICA
20  REM
30  REM     A1 = MASA EN GRAMOS
40  REM     A2 = VELOCIDAD EN CM/SEG.
50  REM     A3 = ENERGIA CINETICA
60  REM
70  ————————————————
```

EJERCICIOS

1. Para finalizar la fase de compilación de un programa se utiliza la instrucción:

   a) PAUSE.
   b) END.
   c) STOP.

2. ¿Cuántas instrucciones STOP puede tener un programa?

   a) Una.
   b) Dos.
   c) Varias.

3. Después de ejecutar una instrucción PAUSE un programa continúa ejecutándose cuando:

   a) Lo ordena el operador.
   b) Cuando pasa un cierto tiempo.
   c) No continúa ejecutándose, porque es una instrucción definitiva.

4. La instrucción END, en un programa, debe codificarse:

   a) Siempre al principio.
   b) Siempre al final.
   c) En cualquier posición.

5. ¿Cuál de estas instrucciones provoca una interrupción momentánea?

   a) END.
   b) STOP.
   c) PAUSE.

6. El comentario de una instrucción STOP:

   a) Aparece en el terminal del operador cuando es ejecutada.
   b) Sirve para explicar la lógica del programa.
   c) La instrucción STOP no admite comentario.

7. Las instrucciones FOR y NEXT definen:

   a) Una bifurcación.
   b) Un bucle.
   c) Un comentario.

8. Los comentarios a las instrucciones:

   a) Se codifican en cualquier parte del programa.
   b) Van precedidos del carácter apóstrofe (').
   c) Pueden continuarse en la siguiente instrucción.

9. En la fase de compilación la instrucción LET:

   a) Asigna valores a las variables definidas.
   b) Asigna campos de memoria a las variables definidas.
   c) Pone las variables numéricas a cero y las alfanuméricas a blancos.

10. En la instrucción PAUSE el comentario sirve para .....

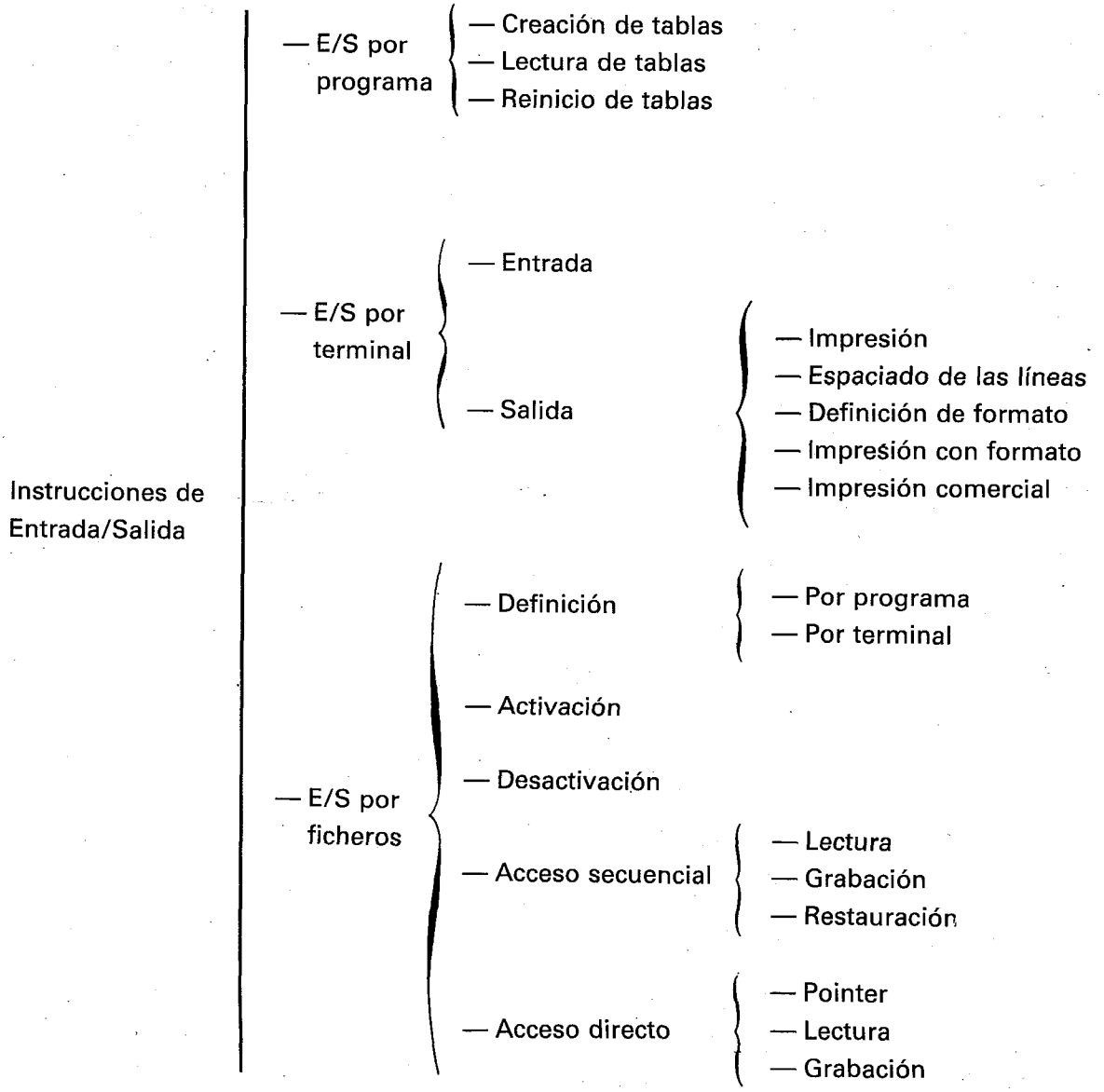
11. La instrucción STOP interrumpe la fase de .....

12. Las bifurcaciones pueden ser de dos tipos: .....

3.

INSTRUCCIONES DE ENTRADA/SALIDA

CUADRO RESUMEN DE LA LECCION



Las instrucciones de Entrada/Salida son las que permiten introducir los datos del problema o dar a conocer los resultados de los cálculos. Esta comunicación puede ser con:

— El usuario.
— Las Unidades de memoria auxiliar.

La comunicación con el usuario se efectúa mediante un terminal. Dicho terminal debe tener las características adecuadas al tipo de comunicación mantenida. Veamos tres tipos de terminales:

a) Terminales cuya finalidad es *introducir* datos al ordenador. Algunos de estos terminales son: Lectora de papel perforado, teclado, lectora de fichas perforadas.

b) Terminales cuya finalidad es *conocer* los resultados. Como por ejemplo: Impresora, pantalla, plotter.

c) Terminales que permiten *responder* a las preguntas del ordenador, como es el caso de: Consolas, pantallas con teclado.

Las unidades de memoria auxiliar permiten almacenar de manera provisional los resultados intermedios que son demasiado voluminosos como para estar contenidos en la Memoria Central del ordenador. Algunos de estos periféricos son: Disco magnético, cinta magnética, tambor magnético.

Para que un programa no se halle *incomunicado,* se han de conocer los siguientes requisitos:

— El sentido de la operación (entrada o salida).
— Las variables implicadas en el diálogo y el orden en que se transmiten.
— Modo de presentación de dichas variables (cifras significativas, número de variables por línea, situación del punto decimal, etc.).

Los dos primeros requisitos se especifican en la misma instrucción de entrada/salida y el tercero (que es optativo) viene definido por otra instrucción a la que hará refererencia la instrucción de entrada/salida.

Las instrucciones de entrada/salida de datos se clasifican, según el medio utilizado, en tres tipos:

— E/S por programa.
— E/S por terminal.
— E/S por fichero.

3.1. E/S POR PROGRAMA

Las instrucciones de entrada/salida por programa se utilizan para la creación y consulta de datos creados en el propio programa. Este tipo de instrucciones pueden ser utilizadas para:

— Creación de tablas.
— Lectura de tablas.
— Reinicio de tablas.

3.1.1. Creación de tablas

La instrucción que permite crear una tabla de datos dentro de un programa es la instrucción DATA, cuyo formato es:

DATA $C_1, C_2, ..., C_n$

siendo:

DATA = Palabra clave (*datos*)
C_i = Constantes

Con esta instrucción se crea una tabla de valores que están definidos por las constantes que componen la instrucción. Dichas constantes pueden ser tanto de tipo numérico como alfanumérico.

Las reglas de codificación de la instrucción DATA son:

— Las constantes deben ir separadas por comas (excepto la última constante que no lleva coma).

— Los valores definidos deben ser constantes o expresiones aritméticas.

— No es necesario que estén definidas entre comillas las constantes alfanuméricas cuyo primer carácter sea distinto al carácter blanco ( ) y además no contenga el carácter coma (,). Las constantes alfanuméricas que no cumplan alguna de estas condiciones deben ir definidas necesariamente entre comillas (simples o dobles).

— Una instrucción DATA no puede ocupar más de una línea.

— Todas las instrucciones DATA forman dos bloques internos, uno de constantes numéricas y otro de constantes alfanuméricas.

— Las instrucciones DATA pueden apareçer en cualquier parte del programa. Sin embargo, es recomendable colocarlas todas juntas al principio o al final del programa, para lograr una mayor legibilidad del mismo.

Ejemplo:

```
210  DATA 3, – 2, 11, 'AB'
220  DATA 'CD', – 8, 0, 10
```

Estas dos instrucciones pueden codificarse como una sola:

```
210  DATA 3, – 2, 11, AB, CD, – 8, 0, 10
```

El bloque de constantes generados con cualquiera de estas dos codificaciones es el que muestra la figura 3.1.

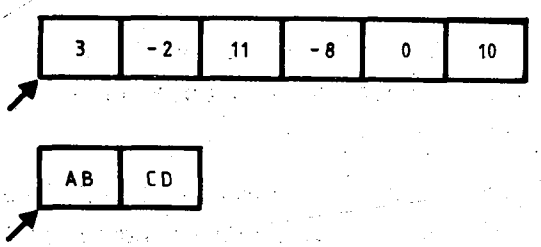


Fig. 3.1.

3.1.2. Lectura de una tabla

Para la lectura de una tabla se utiliza la instrucción READ cuyo formato es el siguiente:

READ $V_1, V_2, ..., V_n$

siendo:

READ = Palabra clave (*leer*).
V_i = Lista de variables.

Esta instrucción asigna a cada una de las variables especificadas, una constante definida en una instrucción DATA.

Las reglas de lectura son:

— Las variables numéricas deben corresponderse con constantes numéricas.

— Las variables alfanuméricas deben corresponderse con constantes alfanuméricas.

— Deben existir tantas constantes definidas en la instrucción DATA como variables tenga la instrucción READ.

— Las variables especificadas en la instrucción READ irán separadas por comas.

— No es necesario que a cada instrucción READ le corresponda una instrucción DATA.

— La primera instrucción READ ejecutada, leerá los datos a partir de la primera posición del bloque definido por las instrucciones DATA hasta completar sus variables.

— Las sucesivas instrucciones READ ejecutadas, leerán los datos a partir de la posición siguiente a la última constante transferida a la instrucción READ ejecutada.

Ejemplo:

```
 40  READ A, B, C
 50  READ P$, S$
  .
  .
  .
 90  READ D, E, F, R$
  .
  .
  .
210  DATA 3, – 2, 11, AB, CD, – 8, 0, 10, EF, 24
220  DATA 17, 'P,L'
```

Antes de ejecutar la instrucción 40, el bloque definido por la instrucción 210 está como muestra la fig. 3.2.

Una vez ejecutada la instrucción 40 se tiene (Fig. 3.3):

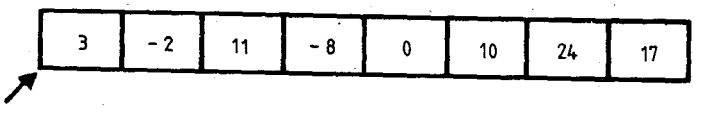
Variable	*Valor asignado*
A	3
B	– 2
C	11

Después de ejecutar la instrucción READ de la línea número 50 (Fig. 3.4):

Variable	*Valor asignado*
P$	AB
Q$	CD

Después de ejecutar la instrucción 90 (Fig. 3.5):

Variable	*Valor asignado*
D	– 8
E	0
F	10
R$	EF



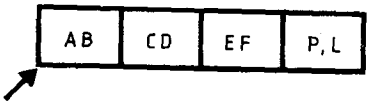
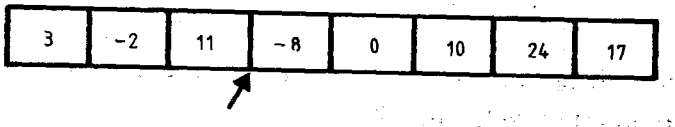


Fig. 3.2.



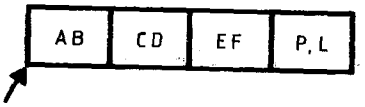
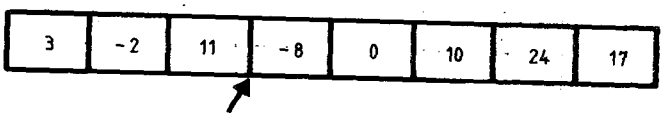


Fig. 3.3.



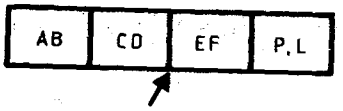
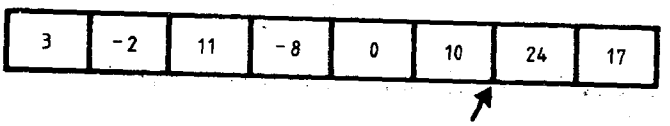


Fig. 3.4.



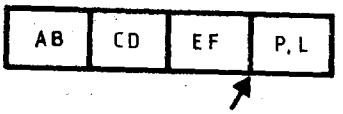
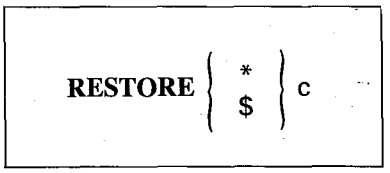


Fig. 3.5.

3.1.3. Reinicio de tablas

La instrucción que reinicia una tabla es la instrucción RESTORE, cuyo formato es:



donde:

RESTORE = Palabra clave que significa *restaurar*
c = Comentario optativo

Esta instrucción provoca que las constantes (a transmitir a las variables de una instrucción READ) sean transmitidas de nuevo a partir de la primera constante definida en el bloque generado por las instrucciones DATA. Es decir, el *pointer* (indicador interno que señala la posición a partir de la cual deben transmitirse los datos) es llevado a la primera posición del bloque generado por las instrucciones DATA.

Al codificar:

```
140  RESTORE *
```

sólo se inicializará el pointer correspondiente a las constantes numéricas.

Al codificar:

```
140  RESTORE $
```

sólo se inicializará el pointer correspondiente a las constantes alfanuméricas.

Al codificar:

```
140  RESTORE
```

se inicializarán los dos pointers.

En el comentario (optativo) pueden explicarse los motivos por los que se utiliza esta instrucción.

Ejemplo:

```
 40  READ A, B, C, A$
 50  READ D, B$, C$
 60  RESTORE *
 70  READ E, F, D$
 80  RESTORE $
 90  READ E$, F$, G, H
100  RESTORE
 .
 .
 .
500  DATA 1, 2, A, 3, B, C, 4, 5, 6, 7, 8; E, F, G, 9, H
```

Antes de ejecutar la instrucción 40 (Fig. 3.6):

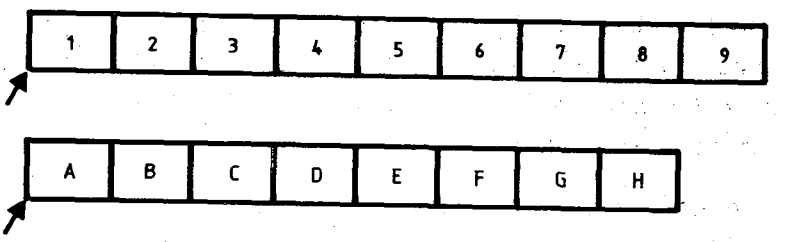


Fig. 3.6.

Después de ejecutar la instrucción 40 (Fig. 3.7):

Variable	*Valor*
A	1
B	2
C	3
A$	A

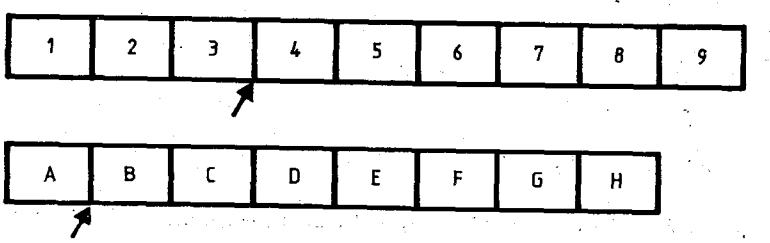


Fig. 3.7.

Después de ejecutar la instrucción 50 (Fig. 3.8):

Variable	*Valor*
D	4
B$	B
C$	C

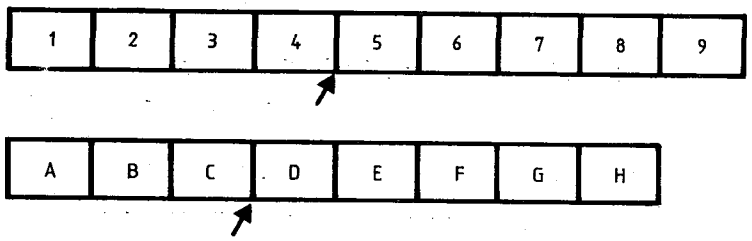


Fig. 3.8.

Después de ejecutar la instrucción 60 (Fig. 3.9):

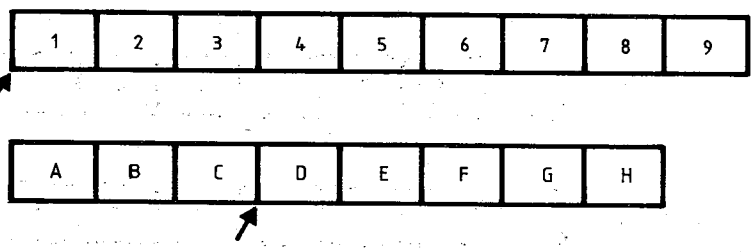


Fig. 3.9.

Después de ejecutar la instrucción 70 (Fig. 3.10):

Variable	*Valor*
E	1
F	2
D$	D

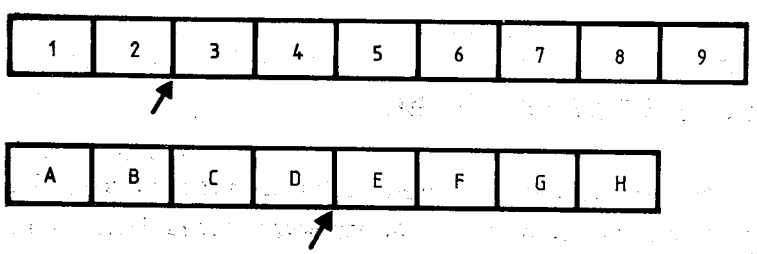


Fig. 3.10.

Después de ejecutar la instrucción 80 (Fig. 3.11):

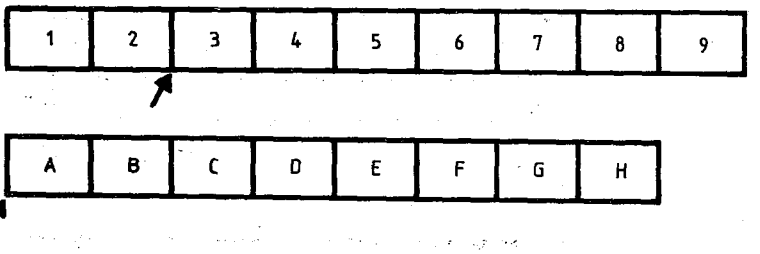


Fig. 3.11.

Después de ejecutar la instrucción 90 (Fig. 3.12):

Variable	*Valor*
E$	A
F$	B
G	3
H	4

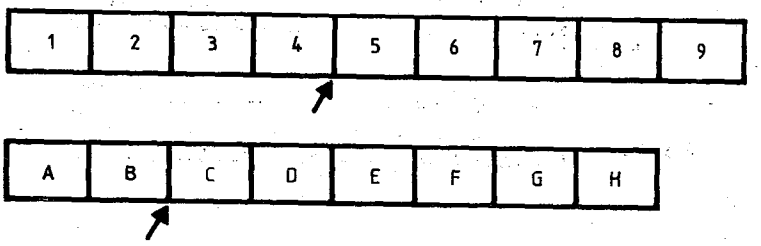


Fig. 3.12.

Después de ejecutar la instrucción 100 (Fig. 3.13):

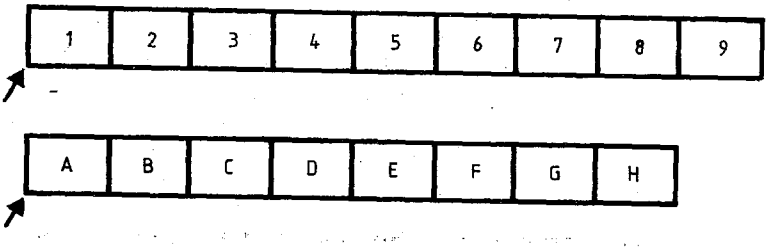


Fig. 3.13.

3.2. E/S POR TERMINAL

Las instrucciones de entrada/salida a través de un terminal sirven para solicitar al usuario los datos de un problema, comunicarle cuáles son los resultados obtenidos, preguntar qué camino seguir dentro de un programa, etc.

3.2.1. Entrada por terminal

La instrucción que solicita datos al usuario es la intrucción INPUT, cuyo formato es:

```
INPUT V1, V2, ..., Vn
```

siendo:

INPUT = Entrar
V_i = Lista de variables

La instrucción INPUT se utiliza para introducir datos (numéricos o alfanuméricos) en la lista de variables especificadas.

Cuando el control del procesador encuentra esta instrucción, transmite un carácter al terminal y detiene la ejecución del programa hasta que han sido transmitidas constantes suficientes para inicializar todas las variables especificadas en la lista. Después continúa con la ejecución del programa.

Las constantes transmitidas por el usuario deben corresponder en número y tipo con la lista de variables de la instrucción INPUT.

Las constantes deben separarse por comas y pueden ser numéricas (si van destinadas a una variable numérica) o alfanuméricas (si van destinadas a una variable alfanumérica).

— Las constantes alfanuméricas deben ir delimitadas entre comillas (simples o dobles) si empiezan por un carácter blanco o si contienen comas, en caso contrario pueden especificarse sin los caracteres limitadores.

— Si el usuario conoce de antemano los datos que pedirán las instrucciones INPUT de un programa, los puede introducir antes de que el programa los precise, transmitiendo en primer lugar el carácter % y a continuación la lista de constantes. Con esto evita que se interrumpa momentáneamente la ejecución del programa.

— Con la instrucción INPUT el ordenador almacena las constantes introducidas en un *buffer* (memoria intermedia) y sólo pedirá más datos al operador cuando las distintas instrucciones INPUT hayan vaciado dicho buffer.

Ejemplo:

```
 90  INPUT A, B
100  C = A + B
```

si el operador transmite 1, 2 el valor de las variables A, B y C será:

	Variables		
Secuencia	A	B	C
90	1	2	?
100	1	2	3

3.2.2. Salida por terminal

Las salidas por terminal sirven para proporcionar los resultados de la ejecución de un programa y para mantener informado al usuario de los pasos que sigue el programa.

Impresión

La instrucción que imprime los datos y los mensajes de un programa es la instrucción PRINT, cuyo formato es:

```
PRINT x
```

donde:

PRINT = Imprimir

x = Lista de constantes y/o de variables y/o expresiones aritméticas.

Esta instrucción transmite al terminal de salida asignado al ordenador, las constantes, el valor de las variables y el resultado de las expresiones aritméticas codificadas en la instrucción PRINT.

Sus reglas son:

— Las constantes alfanuméricas deben encerrarse entre comillas (simples o dobles).

— Si una cantidad numérica sobrepasa los ocho dígitos se redondea a seis dígitos y se imprime en notación exponencial.

— Si, al codificar los datos en la instrucción PRINT, son separados por comas, la línea de impresión se divide en cinco zonas de igual longitud y se imprime un valor en cada una de ellas.

— Si están separados por un punto y coma (;), cada dato ocupará la longitud que posea (con lo que se puede imprimir más de cinco datos en cada línea del terminal de salida).

— Si los valores de las constantes y variables codificadas supera en longitud una línea de impresión, la impresión continúa en la siguiente línea.

— Si a continuación del último dato definido se coloca una coma (,), los datos de salida de la siguiente instrucción PRINT se imprimirán en las zonas disponibles dentro de la misma línea.

— Si después del último dato definido se coloca un punto y coma (;), los siguientes datos de otra instrucción PRINT se imprimirán inmediatamente a continuación de los ya impresos.

— Cada instrucción PRINT realiza la impresión en una nueva línea si la última instrucción PRINT no indicaba lo contrario.

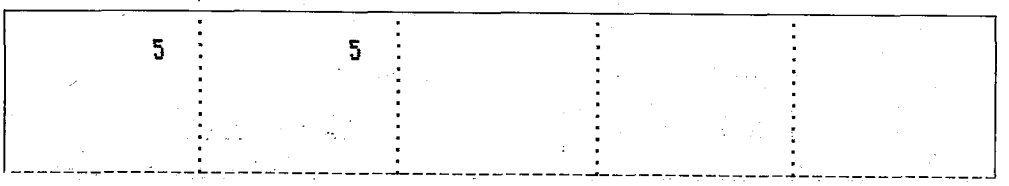
Es usual codificar antes de una instrucción INPUT, una instrucción PRINT que informe al operador cuáles son los datos solicitados.

Ejemplos:

— Con la siguiente codificación:

```
470  A, B = 5
480  PRINT A, B
```

Se obtiene en el terminal:



Sea, por ejemplo, un programa que calcula el área de un círculo.

```
10  PRINT 'ESTE PROGRAMA CALCULA EL AREA DE ';
20  PRINT 'UN CIRCULO'
30  PRINT 'RADIO DEL CIRCULO?'
40  INPUT R
50  PRINT 'SU AREA ES = '; (& PI * R ↑ 2)
60  PRINT 'DESEA CALCULAR OTRO?'
70  INPUT A$
80  IF A$ = 'SI' THEN 30
90  PRINT 'FIN DEL PROCESO'
100 STOP
110 END
```

Con lo que se obtiene:

```
ESTE PROGRAMA CALCULA EL AREA DE UN CIRCULO
RADIO DEL CIRCULO?
? 3
SU AREA ES = 28.27443
DESEA CALCULAR OTRO?
? SI
RADIO DEL CIRCULO?
? 2.7
SU AREA ES = 22.9022
DESEA CALCULAR OTRO?
? NO
FIN DEL PROCESO
```

Espaciado de campos

Existe una palabra clave, TAB, que sirve para intercalar espacios entre los datos a imprimir. Su formato es:

TAB (K)

siendo;

TAB = Tabular
K = Constante numérica

En el anterior formato, el valor de K son los blancos que hay que intercalar.

Ejemplo:

Con las instrucciones:

```
580  PRINT "A"; TAB (11); "B"
590  PRINT
600  PRINT TAB (10); "ABC"
```

se obtiene el listado:

```
A          B

          ABC
```

Espaciado de líneas

Para lograr una mayor legibilidad de los resultados impresos, es conveniente separar los bloques de resultados mediante líneas en blanco.

Esto se logra con cualquiera de estas dos instrucciones:

```
NL
PRINT
```

NL = Línea nueva
PRINT = Imprimir

La primera de ellas significa *línea nueva* (del inglés *New-Line*). Al ser ejecutada, hará avanzar una línea el carro de la impresora, continuándose la impresión en la columna que indicaba la última instrucción PRINT ejecutada.

La instrucción PRINT trabaja como si se imprimiera una línea de blancos y la siguiente impresión de datos empieza a partir de la posición 1 en la hoja de la impresora.

Si la última instrucción PRINT con datos de salida ejecutada indicaba el tipo de *impresión continua* (acabando en coma o en punto y coma), dicho tipo queda anulado y la siguiente instrucción de salida de datos imprimirá a partir de la primera posición de la siguiente línea.

Ejemplos:

Sea el siguiente programa:

```
10  PRINT "ESTE";
20  NL
30  PRINT "LISTADO";
40  NL
50  PRINT "UTILIZA";
60  NL
70  PRINT "LA INSTRUCCION";
80  NL
90  PRINT " " "NL" " "
100 STOP
110 END
```

Obteniendo:

```
ESTE
    LISTADO
          UTILIZA
                LA INSTRUCCION
                          "NL"
```

Sea ahora el siguiente:

```
10  PRINT 'ESTE PROGRAMA CALCULA EL PERI-
    METRO';
20  PRINT 'Y EL AREA DE UNA CIRCUNFERENCIA'
30  PRINT
40  PRINT 'RADIO?'
50  INPUT R
60  PRINT 'PERIMETRO', 'AREA'
```

```
 70  PRINT (&PI * R * 2), (&PI * R ↑ 2)
 80  PRINT
 90  PRINT 'DESEA OTRO?'
100  INPUT A$
110  IF A$ = 'SI' THEN 30
120  PRINT
130  PRINT 'FIN DEL PROCESO'
140  STOP
150  END
```

Con este programa se obtiene el siguiente listado por el terminal de salida:

```
ESTE PROGRAMA CALCULA EL PERIMETRO Y EL AREA DE UNA CIRCUNFE
RENCIA


RADIO?
? 4
PERIMETRO       AREA
25.1327         50.2655

DESEA OTRO?
? SI

RADIO?
? 5.2
PERIMETRO       AREA
32.6726         84.9487

DESEA OTRO?
? NO

FIN DEL PROCESO
```

Definición de formato

Hasta ahora, la manera en que eran impresos los datos de salida, venía dada por el formato standard del ordenador (si eran más de ocho, truncaba a seis y pasaba a notación exponencial).

Definiendo dentro del programa un formato de salida, se puede especificar la estructura que se quiere de la información en el terminal.

La definición de formato se logra con:

$$: C_1\ S_1\ C_2\ S_2\ \ldots\ C_n\ S_n$$

donde:

: = Carácter que hace las funciones de palabra clave.
C_i = Serie de caracteres.
S_i = Especificación de conversión.

Sus reglas son:

— Esta instrucción no es una instrucción ejecutable, es el modelo o patrón que debe seguir el ordenador cuando transmite los datos al terminal de salida.

— Los formatos pueden ir en cualquier parte del programa, pero se recomienda codificarlos todos juntos al principio o al final del programa.

— Un mismo formato puede ser empleado por varias instrucciones de salida.

— Los distintos C_i son constantes alfanuméricas formadas por cualquier tipo a excepción del carácter #. Sirven para dar una mayor legibilidad a los datos de salida.

— Los S_i definen la especificación de conversión deseada de acuerdo a:

— Datos numéricos:
  * Por cada dígito significativo deseado, se codifica un carácter #.
  * Un punto para indicar la parte decimal.
  * En caso de que el valor transmitido tenga más dígitos decimales que caracteres # codificados después del punto, dicha parte decimal será redondeada automáticamente para ajustarse a la especificación de conversión deseada.
  * En el caso contrario, menos dígitos decimales significativos de especificación de conversión decimal deseada, se completará automáticamente con ceros.

* Si no se codifica parte decimal en la especificación de conversión, y dicha parte decimal existe, será truncada automáticamente.
* Si se quiere el dato de salida en notación exponencial, se codificarán cuatro signos de admiración (!!!!) a continuación del último carácter #.
* Si el primer carácter de la especificación de conversión es el carácter +, el valor se imprimirá precedido siempre del signo (+ si es positivo y − si es negativo).
* Si dicho cáracter es el carácter − (o ni el carácter + ni el carácter −) el valor se imprimirá precedido de un blanco si es positivo y precedido del signo − si es negativo.
* Si el dato a imprimir tiene un valor numérico superior al que permite la especificación de conversión, como indicación de error se imprimirán asteriscos.

— Datos alfanuméricos:
* Por cada elemento de especificación se imprime un carácter.

Impresión con formato

Para imprimir resultados utilizando un formato de salida, se utiliza la instrucción: PRINT USING, cuyo formato de codificación es:

PRINT USING n, V_1, V_2, ..., V_n

donde:

PRINT = Imprimir
USING = Usando
n = n.° de líneas que contiene el formato.
V_i = Lista de variables.

Reglas de funcionamiento:

— Cuando el procesador ejecuta una instrucción de este tipo, busca el n.° de línea n y va explorando la definición de formato de izquierda a derecha, asignado a cada variable V_i una especificación de conversión.

— Deben existir por lo menos tantas especificaciones de conversión en la definición del formato de salida, como variables tenga la instrucción de salida que llama a dicho formato.

— Si dos V_i están separadas por punto y coma (;), el carro de la impresora, permanecerá en la última posición de impresión.

— Si una variable está precedida por una coma (,), antes de ser impresa, el cursor del carro retrocederá a la primera posición de impresión.

— Si la definición de formato es demasiado larga, se abandona cuando se completa la lista de V_i.

— Si después de la última V_i se codifica una coma o un punto y coma, la impresión sigue las mismas reglas ya explicadas en el apartado sobre IMPRESION para estos casos.

— Si la definición de formato es demasiado corta (existen más variables en la instrucción PRINT USING que especificaciones de conversión en la definición de formato empleado), la definición de formato vuelve a ser explorada a partir de la primera especificación de conversión (con objeto de que no quede nunca ninguna variable sin imprimir).

Algunos ejemplos son:

```
 10  : EL VALOR DE "A" ES # # #
 20  : EL VALOR DE "A" ES – # # # . #
 30  : EL VALOR DE "A" ES + # # # # . # # # # #
 40  : EL VALOR DE "A" ES # . # # # # !!!!

500  A = 50.123458
510  PRINT USING 10,A
520  PRINT USING 20,A
530  PRINT USING 30,A
540  PRINT USING 40,A
```

se obtendrá en el terminal:

```
EL VALOR DE 'A' ES  50
EL VALOR DE 'A' ES  50.1
EL VALOR DE 'A' ES +50.12346
EL VALOR DE 'A' ES  5.0123E+01
```

Con este programa

```
10  : (A = #  B = #  C = #)
20  A = 1
30  B = 2
40  C = 3
50  D = 4
60  E = 5
70  F = 6
80  PRINT USING 10; A; B; C;
90  PRINT USING 10; A; B; C
100 PRINT USING 10; A
110 PRINT USING 10; A;
120 PRINT USING 10; A; B; C
130 PRINT USING 10; A; B; C, D; E; F
140 PRINT USING 10; A; B; C; D;
150 PRINT USING 10; A; B; C; D
```

se obtendrá:

```
(A=1    B=2    C=3)(A=1    B=2    C=3)
(A=1    B=
(A=1    B=(A=1    B=2    C=3)
(A=1    B=2    C=3)
(A=4    B=5    C=6)
(A=1    B=2    C=3)(A=4    B=(A=1    B=2    C=3)(A=1    B=
```

Impresión comercial

La principal utilidad de las salidas de datos con formato, es que la impresión de los resultados pueda hacerse siguiendo las normas de tipo comercial. Dichas normas tratan de evitar la posterior falsificación de los datos impresos. Y son:

— Signo más (+) o menos (−) a la izquierda del valor impreso.

— Un signo menos (−) o los caracteres CR (crédito) a la derecha del valor impreso.

— Ceros o asteriscos a la izquierda del valor impreso en sustitución de los espacios en blanco.

— Un carácter $ a la izquierda del valor impreso.

— Inserción de separadores de miles (comas).

La especificación de conversión típica de una impresión comercial es:

— En primer lugar el carácter más (+).

— A continuación combinaciones de estos caracteres:
  $ = Se imprime el carácter $ entre el signo y el primer dígito.
  & = Este carácter sustituye al carácter # de la impresión con formato no comercial.
  * = Los ceros no significativos a la izquierda de este carácter se sustituyen por asteriscos.
  Ø = Se imprimen los ceros no significativos de la izquierda.
  B = Si se sustituye el último carácter & por B, y el valor a imprimir es nulo, todo el campo se deja en blanco.
  . = Posición del punto decimal.

— La especificación de conversión comercial se comporta como si los caracteres &, *, 0 y B fueran el carácter # de una especificación de conversión no comercial, pero con las características especiales ya aplicadas.

— Las opciones de signo a la izquierda y a la derecha del valor impreso, son incompatibles entre sí, (si se utiliza una, no puede utilizarse la otra).

— Si no se especifica ninguna de las dos opciones, y el valor impreso es negativo, aparecerá un signo − a la izquierda del primer dígito o en sustitución del primer *, 0 ó &.

— Cuando el signo menos esté situado entre dos especificaciones de conversión consecutivas, afectará siempre que sea posible a la que antecede.

— Si el valor a imprimir es mayor que la longitud de la especificación de conversión, se imprimen asteriscos.

— Cuando se imprimen variables o constantes alfanuméricas, todos los elementos de la especificación se sustituyen por caracteres.

— En cualquier parte de la especificación de conversión pueden especificarse comas, las cuales serán impresas en la posición que ocupan.

Ejemplos:

— La especificación de conversión

```
&&&-&&&-&&&&
```

es interpretada como:

```
&&&-    &&&-    &&&&
```

y no como:

```
&&&    -&&&    -&&&&
```

— Con el siguiente programa:

```
 10 : &&&.&&-  $0&&        $&&   +$&.&&&&  &,&&&-      0&&&&-
 20 : *&.B-    +$&&&.&&&   $&&B  -&&.&&&&&  &&&        +$&&&
 30 : &&&&.&   *&&&&&&&&   *&&   &.&&CR     *&&&&,&&&  &&&&.&CR
 40 : &&.&CR,, +0,&&&.&B,, &&&,, +*&&.&&&,, 0&&&&&.&&,, *&&&&.B
 50 A = -10.10
 60 B = 123.45
 70 C = 0
 80 D = 0.024
 90 E = 1000
100 F = -100
110 PRINT USING 10; A; B; C; D; E; F
120 PRINT USING 20; A; B; C; D; E; F
130 PRINT USING 30; A; B; C; D; E; F
140 PRINT USING 40; A; B; C; D; E; F
150 STOP
160 END
```

Se obtiene esta impresión:

```
 10.10-   $123          $0    +$0.0240   1,000      001-
10.1-     +$123.450           0.02400    ***        -$1
 -10.1    *****123     **0    0.02       ****1,000  0100CR
10.1CR,,  +0,123.45,,   0,,   ***0.024,, 001000.00,, -$1.0
```

3.3. FICHEROS

La estructuración lógica de una unidad periférica de memoria es, en orden descendente:

— Ficheros
— Registros
— Campos

Se denomina *Fichero* a cualquier conjunto de datos agrupados en una unidad periférica de memoria (disco, cinta magnética, tambor, etc.).

Un *Registro* es cada una de las distintas agrupaciones de datos dentro de un Fichero.

Un *Campo* es cada uno de los datos de un Registro.

3.3.1. Definición de ficheros

Todo fichero que sea utilizado por un programa, debe estar definido previamente antes de ser utilizado para que el ordenador conozca en qué zona de la memoria periférica están almacenados los registros que consulta el programa, o bien dónde debe ubicar los nuevos registros creados por el programa.

Si al crear nuevos registros, el espacio sobrante en el fichero se agota durante la ejecución del programa, el ordenador emite automáticamente un mensaje avisando al usuario de que el fichero está lleno para que aumente su tamaño.

Un fichero puede tratarse de forma *secuencial* o de forma *directa* (aleatoria). En el primer caso (acceso secuencial), los registros son introducidos o extraídos uno a continuación de otro, de forma secuencial, de manera que para tratar un registro determinado es preciso tratar primero todos los que le anteceden. En acceso directo, los registros son introducidos *(grabados)* o extraídos *(leídos)* en el orden que especifica el programa.

Definición por programa

Para definir un fichero desde un programa, debe asignársele un nombre y el espacio de almacenamiento requerido. Entonces, el ordenador buscará una zona en la memoria periférica que esté libre para ubicar allí el fichero.

Cuando el programador quiera trabajar con dicho fichero, indicará el nombre con el que fue definido, siendo el propio ordenador el encargado de recordar dónde ubicó el fichero que tenía asignado dicho nombre.

La instrucción para definir un fichero es la instrucción FILE, cuyo formato es:

FILE N, n, D

siendo:

FILE = Fichero.
N = Nombre de fichero.
n = Unidades de memoria.
D = Palabra clave opcional.

Si en la definición del fichero no se incluye la palabra clave opcional D, el fichero será organizado para tratamiento secuencial, pero si es incluida, el tratamiento será de tipo aleatorio.

Las reglas de composición del nombre de un fichero (tipo de caracteres permitidos y longitud del nombre) varían según el modelo del ordenador (por lo que se recomienda consultar el manual que facilita el fabricante).

En general, se admite cualquier tipo de caracteres (incluidos los blancos).

Además el nombre no puede exceder de 6 caracteres y el primer carácter debe ser alfabético.

Para saber cuántas pistas necesita un fichero se calcula mediante la fórmula:

$$\frac{R \times (K_1 + C_1 + C_2 + K_2 + \ldots + C_n)}{P} = n$$

siendo:

R = Número de registros que contiene el fichero.
C_i = Longitud de cada uno de los campos que forman el fichero.
P = Número de octetos por pista.
K_i = 2 octetos de control que anteceden a un mismo tipo de datos. Indica el tipo y el número de datos.
n = Número de pistas necesarias.

Es conveniente indicar más registros de los necesarios ya que, si posteriormente se necesita ampliar el fichero, se evitarán problemas de falta de espacio.

La longitud de cada campo C_i es:

— 4 octetos para cada variable o constante numérica en simple precisión.
— 8 octetos para cada variable o constante numérica en doble precisión.
— 18 octetos para cada variable o constante alfanumérica.

Ejemplo:

Definir un fichero de tipo secuencial, de nombre FICH-1 con una longitud de pista de 3440 octetos y una capacidad de 100 registros, trabajando en simple precisión, cuya descomposición es:

A, B, C, A$, D, B$, C$

La longitud de cada registro será:

— 2 octetos del factor K
— 4 octetos para el campo numérico A
— 4 octetos para el campo numérico B
— 4 octetos para el campo numérico C

— 2 octetos del factor K
— 18 octetos para el campo alfanumérico A$

— 2 octetos del factor K
— 4 octetos para el campo numérico D

— 2 octetos del factor K
— 18 octetos para el campo alfa numérico B$
— 18 octetos para el campo alfa numérico C$

por tanto, el número de octetos ocupados por cada registro es de 78.

Como el fichero tendrá un mínimo de 100 registros, harán falta:

$$100 \times 78 = 7800 \text{ octetos}$$

las pistas necesarias son:

$$\frac{7800}{3440} = 2{,}3$$

como la cantidad de pistas ha de ser un número entero, se codificará:

```
10 FILE FICH-1, 3
```

Para saber el número de registros que admite un fichero se tiene la siguiente expresión:

$$\frac{n \times P}{L} = R$$

siendo:

n = Número de pistas del fichero
P = Longitud de cada pista (en octetos)
L = Longitud de cada registro (en octetos)
R = Número máximo de registros del fichero

En el ejemplo anterior habrá espacio para:

$$\frac{3 \times 3440}{78} = 132 \text{ registros}$$

Si no se especifica el número de pistas necesarias, se reservan automáticamente 50 pistas para el fichero.

Ejemplos:

— Al codificar:

```
10 FILE FICH-1
```

y la longitud por registro es de 78 octetos, habrá espacio para

$$\frac{50 \times 3440}{78} = 2205 \text{ registros}$$

— Definir un fichero de tipo aleatorio de nombre FICH-2 con capacidad para 2000 registros con una longitud de 80 octetos cada uno.

Las pistas necesarias son:

$$\frac{2000 \times 80}{3440} = 46{,}5$$

por tanto, se codificará:

```
10 FILE FICH-2, 47, D
```

y habrá capacidad para 2021 registros.

Si se desea tener más margen, se puede codificar:

```
10 FILE FICH-2, 50, D
```

o simplemente:

```
10 FILE FICH-2, D
```

con lo que habrá capacidad para 2150 registros de 80 octetos cada uno.

Una vez esté definido el fichero, se irá llenando (creando) y consultando desde el programa.

Definición por terminal

Para definir manualmente un fichero, se procede desde el terminal del usuario de manera similar a la escritura de un programa en BASIC.

Las reglas son:

— Cada línea tendrá un número de línea (1 a 5 dígitos) a continuación un blanco y, finalmente, los campos del registro deben estar separados entre sí por comas (,).

— En este tipo de definición no se admiten comentarios.

— Las constantes alfanuméricas que no contengan comas ni espacios en blanco pueden definirse sin caracteres limitadores.

— Al mismo tiempo que se define manualmente (por terminal) un fichero, se están creando los registros que contiene.

— Un fichero definido manualmente se denomina *fichero fuente* y sólo puede ser consultado.

— Una vez definido el fichero, se le asigna un nombre y se protegen los registros mediante el siguiente comando:

```
SAVE nombre-del-fichero
```

Cuando el operador teclea SAVE, el ordenador contesta:

```
READY
```

con lo que da por finalizada la definición y creación del fichero.

Ejemplo:

— Definir y crear manualmente un fichero de nombre FICH-9 de dos registros con los siguientes campos:

1.er registro = «UNO», «DOS», 100, 12.34E-5
2.º registro = «TRES», «CUATRO», 10.2, 100

En el terminal se codifica:

```
10  UNO, DOS, 100, 12.34E-5
20  TRES, CUATRO, 10.2, 100
30  END
```

Para almacenar el fichero, se tecleará a través del terminal:

```
SAVE «FICH-9»
```

a lo que contestará:

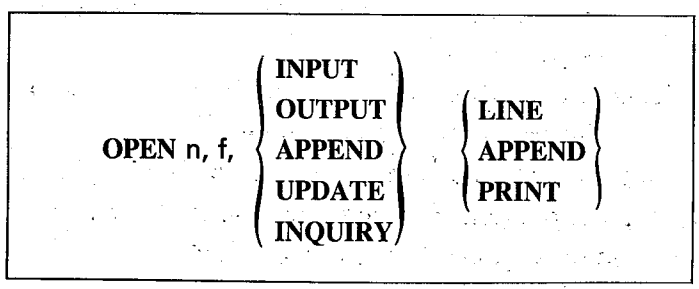
```
READY
```

quedando a la espera de las órdenes del operador.

3.3.2. Activación de un fichero

Activar un fichero consiste en comunicarle al ordenador qué fichero se desea y que lo prepare para ser leído o grabado.

La instrucción para activar un fichero es la instrucción OPEN, cuyo formato es:



siendo:

OPEN = Abrir
n = Expresión aritmética o constante numérica o variable numérica
f = Constante o variable alfanumérica
INPUT = Entrada
OUTPUT = Salida
APPEND = Añadir
UPDATE = Actualizar
INQUIRY = Preguntar
LINE = Línea
PRINT = Imprimir

Esta instrucción asigna el valor entero de n (en caso de existir decimales se truncan) al fichero cuyo nombre es el indicado por la constante o la variable alfanumérica f.

Debe especificarse una de las cinco primeras opciones para que el ordenador prepare el fichero de acuerdo al uso que se quiere hacer de él.

Dichas opciones son:

— *Para ficheros de tipo secuencial*

INPUT: Indica que solamente se leerán los registros del fichero activado.
OUTPUT: Indica que solamente se grabarán registros en el fichero activado.
APPEND: Indica que los registros que se graben sean situados a continuación de los ya existentes.

— *Para ficheros de tipo aleativo.*

UPDATE: Indica que se leerán registros para introducirles modificaciones o que se grabarán nuevos registros.
INQUIRY: Indica que sólo se leerán los registros del fichero.

La finalidad de las palabras clave opcionales es:

LINE: Indica que los números de línea (llamados también *CLAVE*, del inglés *Key*) correspondientes a cada registro, deben leerse como un campo más del registro.
APPEND: Misma función que la palabra clave de mismo nombre ya visto como una de las cinco opciones obligatorias.
PRINT: Indica que el fichero en el que se van a grabar los registros debe estructurarse de manera que pueda ser utilizado desde un programa escrito en el lenguaje de programación PL/1.

Cada vez que se ejecuta una instrucción OPEN, el puntero de registros del fichero se posiciona en el primer registro; excepto con la opción APPEND con la que se sitúa siempre al final del último registro.

Ejemplos:

— Abrir el fichero (suele decirse *abrir* en lugar de *activar*) secuencial que se definió como FICH-1 y prepararlo con el número 2 para grabar registros:

```
70 OPEN 2, "FICH-1", OUTPUT
```

— Abrir el fichero secuencial FICH-2 con el número que tenga la variable B4 y prepararlo para leer sus registros.

```
80 OPEN B4, "FICH-2", INPUT
```

— Abrir el fichero secuencial cuyo nombre está en la variable A$ con el número resultante de restar tres a la variable A1 y prepararlo para añadir nuevos registros.

```
90 OPEN (A1-3), A$, APPEND
```

— Abrir el fichero aleatorio FICH-3 con el número 4 para poder modificar sus registros y que puedan ser tratados posteriormente desde un programa en PL/1.

```
40 OPEN 4, "FICH-3", UPDATE, PRINT.
```

3.3.3. Desactivación de un fichero

Para desactivar un fichero (suele decirse *cerrar*) y guardarlo de nuevo en la biblioteca del usuario, se utiliza la instrucción CLOSE cuyo formato es:

CLOSE $n_1, n_2, ..., n_n$

CLOSE = Cerrar.

n_i = Constante numérica o variable numérica o expresión aritmética.

Mientras un programa procesa un fichero, éste debe permanecer abierto y en caso de ser cerrado y abierto de nuevo el tratamiento continuaría por el primer registro otra vez.

Una instrucción CLOSE cerrará tantos ficheros como n_i se codifiquen.

Cuando se ejecuta la última instrucción de un programa (STOP), todos los ficheros son desactivados automáticamente, aunque no se incluya la instrucción CLOSE.

Ejemplos:

— Un programa que define y crea el fichero secuencial ART-1 con el número 1:

```
  10  FILE ART-1, 30
  20  OPEN 1, "ART-1", OUTPUT
   .
   .
   .
9000  CLOSE 1
9010  STOP
9020  END
```

— Un programa que consulta el fichero EMPL con el n.º 1 y define y crea el fichero SUELDO con el n.º 2.

```
  10  FILE SUELDO
  20  OPEN 2, "SUELDO", OUTPUT
  30  OPEN 1, "EMPL", INPUT
   .
   .
   .
2300  CLOSE 1,2
2310  STOP
2320  END
```

3.3.4. Acceso secuencial

En un fichero organizado secuencialmente, los registros se leen o se graban uno tras otro, siguiendo un orden establecido.

Lectura en acceso secuencial

Para leer los registros ya existentes en un fichero, se utiliza la instrucción GET cuyo formato es:

GET n: V_1, V_2, V_3, ..., V_n

siendo:

GET = Obtener
n = Constante numérica o variable numérica o expresión aritmética
V_i = Variables numéricas o alfanuméricas.

Cada vez que el procesador ejecuta una instrucción de este tipo, busca el siguiente registro en el fichero y transfiere cada uno de los campos del registro a las variables definidas en la instrucción GET.

Las normas de esta instrucción son:

— Si el valor de n tiene decimales, se truncan.

— Si se omite la codificación de n, así como los dos puntos, el ordenador asume automáticamente que la referencia es al fichero número 1.

— Las variables V_i definidas deben coincidir, en tipo, con los campos del registro.

— Deben existir tantas variables como campos tenga el registro. Por tanto, antes de leer los registros de un fichero, debe conocerse cuál es su estructuración interna.

— Si en la instrucción OPEN se especificó la opción LINE, debe tenerse en cuenta que la primera variable V_i definida contendrá el número de línea del registro que acaba de leerse.

Ejemplo:

Sea el siguiente programa

```
10 OPEN 1, "ALMA", INPUT, LINE
20 FOR N = a TO 1000
```

```
30  GET A, A$, B, C
40  IF A$≠ 'TORNILLO' THEN 60
50  PRINT USING 80, A, B, C
60  NEXT N
70  STOP
80  :'TORNILLO '####' DIAMETRO = '##·##' LONGITUD = '###·##'
90  END
```

Se obtiene el listado de todos los tipos de tornillos que posee el almacén (se supone que el almacén posee mil artículos).

Se considera oportuno que el lector diseñe los registros y haga funcionar el programa para que logre familiarizarse con el tratamiento de los ficheros.

Grabación en acceso secuencial

La instrucción que graba registros secuencialmente en un fichero es la instrucción PUT, cuyo formato es:

PUT n: X_1, X_2, X_3, ..., X_n

donde:

PUT = poner
n = constante numérica o variable numérica o expresiones aritméticas.
X_i = constantes numéricas o variables numéricas o expresiones aritméticas.

Cada ejecución de la instrucción PUT crea un nuevo registro en un fichero.

El fichero que utiliza esta instrucción debe estar abierto y preparado para grabación (ver instrucciones OPEN, OUTPUT, GET).

Ejemplos:

Hacer un programa para definir y crear un fichero de nombre ALMA para grabar los artículos en stock del almacén de una empresa.

Cada registro contendrá los siguientes campos:

— Código del artículo (6 dígitos)
— Descripción del artículo (18 caracteres alfanuméricos)
— Unidades en stock (4 dígitos)
— Precio por unidad (7 dígitos: 5 enteros y 2 decimales)

Se necesita definir suficiente espacio en el fichero para un máximo de 15.000 artículos.

El organigrama del programa se muestra en la figura 3.14.

Los pasos para la confección del programa son:

— ¿Simple precisión o doble precisión? El campo que más dígitos necesita no excede de 8 dígitos (precio por unidad, son 7 dígitos), por lo tanto, se trabajará en simple precisión (cada constante o variable numérica ocupará 4 octetos).

— En primer lugar hay que calcular los octetos que ocupa un registro:

2 octetos para control
4 octetos para el código del artículo
2 octetos para control
18 octetos para la descripción del artículo
2 octetos para control
4 octetos para las unidades en stock
4 octetos para el precio por unidad

Por lo tanto, cada registro ocupará 36 octetos.

— Ahora debe calcularse el número de pistas que necesita el fichero (se considera que 1 pista = 3440 octetos):

$$\frac{15000 \times 36}{3440} = 156{,}97674 \Rightarrow 157$$

redondea a 157 (el número de pistas ha de ser un número entero)

$$\frac{157 \times 3440}{36} = 15002{,}22 \Rightarrow 15002$$

el fichero admitirá 15002 registros (los decimales se desprecian siempre)

El número de referencia del fichero puede ser el 4.

Con lo cual, se conocen los datos necesarios para definir y abrir el fichero.

```
FILE ALMA, 157
OPEN 4, "ALMA", OUTPUT
```

Si la introducción de los datos es:

```
INPUT C, D$, U, P
```

La instrucción de grabación será:

```
PUT 4: C, D$, U, P
```

Entonces, se puede codificar el siguiente programa:

```
 10  FILE ALMA, 157
 20  OPEN 4, "ALMA", OUTPUT
 30  INPUT C, D$, U, P
 40  PUT 4: C, D$, U, P
 50  PRINT "DESEA OTRO ARTICULO?"
 60  INPUT A$
 70  IF A$ = 'SI' THEN 30
 80  CLOSE 4
 90  STOP
100  END
```

Este programa funciona correctamente, pero de cara al operador es deficiente, ya que no dice el orden de introducción de los datos y no permite modificar un error cometido por equivocación. Además, debe contestarse SI para continuar con la grabación de los artículos.

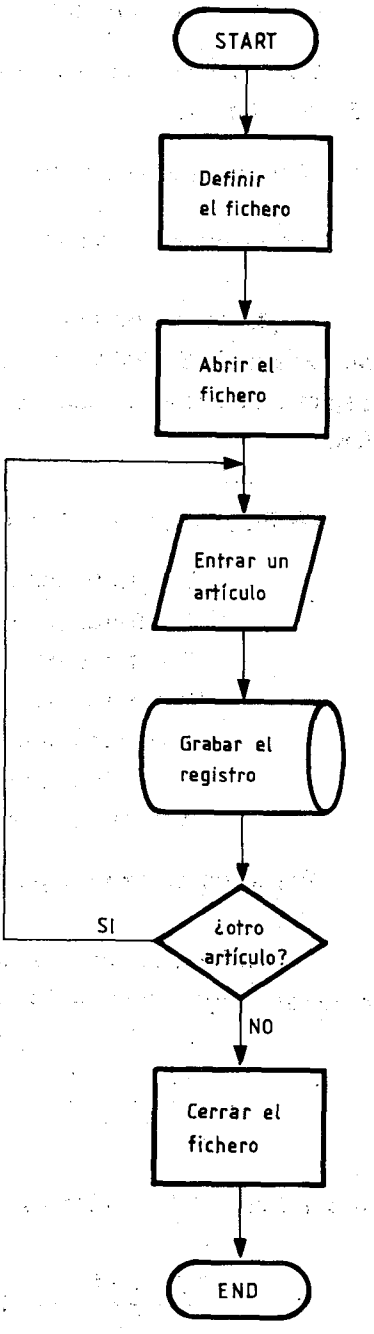


Fig. 3.14. ▶

El organigrama (modificado) que tiene en cuenta los inconvenientes enumerados anteriormente, es el que presenta la figura 3.15.

El programa quedará codificado de esta manera:

```
 10  FILE ALMA, 157
 20  PRINT "CREACION DEL FICHERO DEL ALMACEN"
 30  FOR N = 1 TO 31
 40  PRINT "-";
 50  NEXT N
 60  PRINT "-"
 70  PRINT
 80  PRINT "INTRODUZCA LOS DATOS EN EL SIGUIENTE ORDEN:"
 90  PRINT "CODIGO (6), DESCRIPCION (18), UNIDADES (4), PRECIO (5.2), CLAVE"
100  PRINT
110  PRINT "CLAVE: S = CORRECTO Y DESEO INTRODUCIR OTRO ARTICULO"
120  PRINT TAB (7), "F = CORRECTO Y NO DESEO INTRODUCIR OTRO ARTICULO"
130  PRINT TAB (7), "N = LA INTRODUCCION NO ES CORRECTA"
140  PRINT
150  OPEN 4, "ALMA", OUTPUT
160  PRINT "**** YA PUEDE EMPEZAR ****"
170  PRINT
180  INPUT C, D$, U, P, C$
190  IF C$ = 'N' THEN 170
200  IF C$ = 'S' THEN 250
210  IF C$ = 'F' THEN 250
220  PRINT "**CLAVE INCORRECTA, REPITALA POR FAVOR**"
230  INPUT C$
240  GO TO 190
250  PUT 4: C, D$, U, P
260  IF C$ = 'S' THEN 180
270  CLOSE 4
280  PRINT
290  PRINT "**** FIN ****"
300  STOP
310  END
```

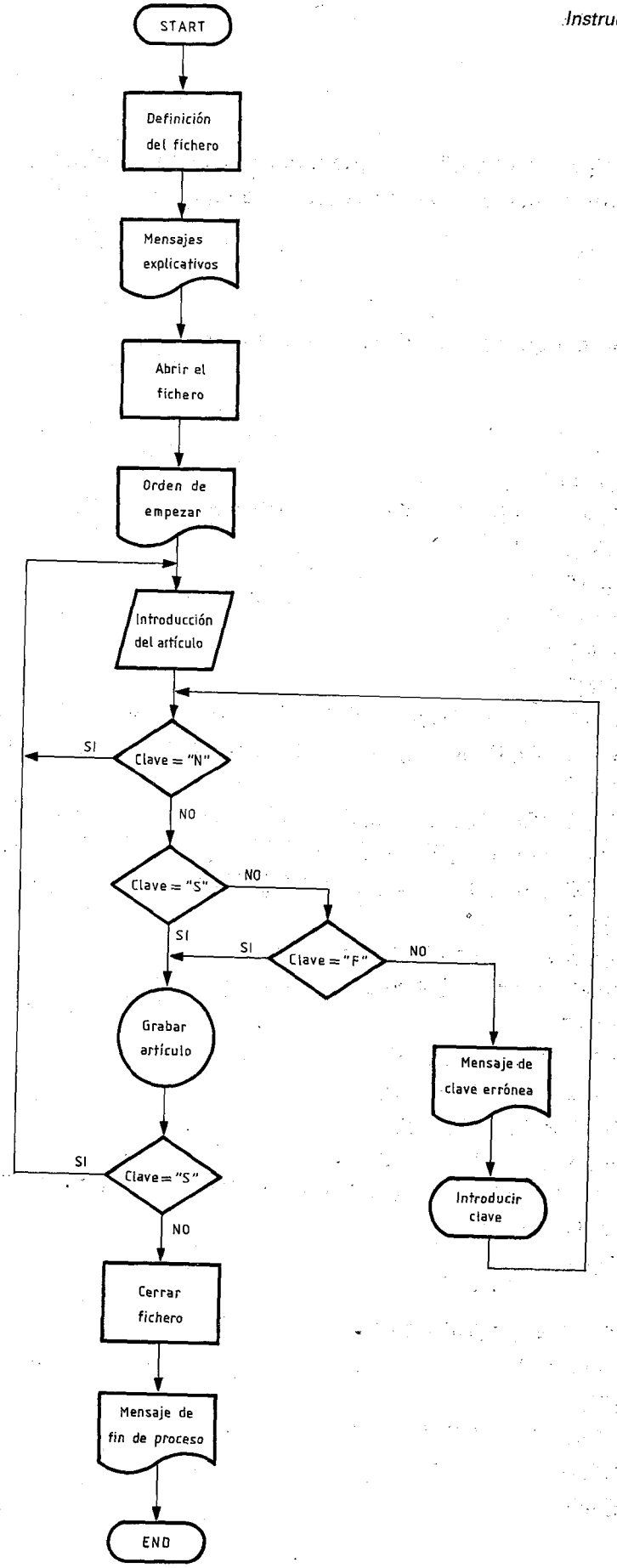

START
Definición del fichero
Mensajes explicativos
Abrir el fichero
Orden de empezar
Introducción del artículo
Clave = "N"
SI
NO
Clave = "S"
NO
SI
SI
Clave = "F"
NO
Grabar artículo
Mensaje de clave errónea
Introducir clave
Clave = "S"
SI
NO
Cerrar fichero
Mensaje de fin de proceso
END

Fig. 3.15.

Con el anterior programa se obtiene el siguiente listado:

```
      CREACION DEL FICHERO DE ALMACEN
      -------------------------------

      INTRODUZCA LOS DATOS EN EL SIGUIENTE ORDEN:
      CODIGO(6), DESCRIPCION(18), UNIDADES(4), PRECIO(5.2), CLAVE

      CLAVE: S = CORRECTO Y DESEO INTRODUCIR OTRO ARTICULO
             F = CORRECTO Y NO DESEO INTRODUCIR OTRO ARTICULO
             N = LA INTRODUCCION NO ES CORRECTA

      **** YA PUEDE EMPEZAR ****

      ? 174092,TORNILLO,527,2.35,S
      ? 174095,TORNILLO,19,50,S
      ? 274092,TUERCA,527,1.75,S
      ? 274092,TUERCA,19,24.10,N

      ? 274095,TUERCA,19,24.10,S
      ? 374092,ARANDELA,527,0.15,S
      ? 374095,ARANDELA,19,2.95,K
      ** CLAVE INCORRECTA, REPITALA POR FAVOR **
      ? S

      ? 954143,MESA,2,7535,S
      ? 954144,MESA,7,8990,P
      ** CLAVE INCORRECTA, REPITALA POR FAVOR **
      ? N

      ? 9541444,MESA,7,8995,F

      **** FIN ****
```

Restauración de ficheros de acceso secuencial

En algunos casos es necesario que el pointer del fichero apunte de nuevo al primer registro. Esto se consigue con la instrucción RESET cuyo formato es:

RESET n_1, n_2, n_3, ..., n_n

en donde:

RESET = Volver a poner

n_i = Constantes numéricas o variables numéricas o expresiones aritméticas.

El pointer de un fichero puede restaurarse (volver a apuntar al primer registro) en cualquier parte del programa. Después de restaurarse el puntero de un fichero, cualquier lectura o grabación, incluso si el fichero se activó con la opción APPEND, tendrá lugar a partir del primer registro.

Ejemplo:

Siguiendo con el mismo almacén del ejemplo anterior, se desea ahora obtener varios listados de las *altas del almacén* una vez finalizada la entrada de artículos en el fichero.

El organigrama se muestra en la figura 3.16.

El programa será:

```
 10  FILE ALMACEN, 157
 20  J = 0
 30  PRINT "CREACION DEL FICHERO DEL ALMACEN"
 40  FOR N = 1 TO 31
 50  PRINT "-";
 60  NEXT N
 70  PRINT "-"
 80  PRINT
 90  PRINT "INTRODUZCA LOS DATOS EN EL SIGUIENTE ORDEN:"
100  PRINT "CODIGO (6), DESCRIPCION (18), UNIDADES (4), PRECIO (5.2), CLAVE"
110  PRINT
120  PRINT "CLAVE: S = CORRECTO Y DESEO OTRA INSTRUCCION"
130  PRINT "CLAVE: F = CORRECTO Y NO DESEO OTRA INSTRUCCION"
140  PRINT "CLAVE: N = LA INTRODUCCION NO ES CORRECTA"
150  PRINT
160  OPEN 4, "ALMACEN", OUTPUT
170  PRINT "**** YA PUEDE EMPEZAR ****"
180  PRINT
```

```
190  INPUT C, D$, U, P, C$
200  IF C$ = 'N' THEN 180
210  IF C$ = 'S' THEN 260
220  IF C$ = 'P' THEN 260
230  PRINT "**CLAVE INCORRECTA, POR FAVOR REPITALA**"
240  INPUT C$
250  GO TO 200
260  J = J + 1
270  PUT 4: C, D$, U, P,
280  IF C$ = 'S' THEN 190
290  PRINT
300  PRINT "*CUANTOS LISTADOS QUIERE?*"
310  INPUT K
320  FOR N = 1 TO K
330  PRINT "ARTICULOS DEL ALMACEN"
340  FOR L = 1 TO 20
350  PRINT "-";
360  NEXT L
370  PRINT "-"
380  PRINT
390  PRINT "CODIGO DESCRIPCION UNIDADES PRECIO"
400  FOR L = 1 TO J
410  GET 4: C, D$, U, P
420  PRINT USING 490, C, D$, U, P
430  NEXT L
440  RESET 4
450  NEXT N
460  PRINT
470  CLOSE 4
480  STOP
490  : ######.####################.####...#####.##
500  END
```

3.3.5. Acceso directo

La eficacia y el ahorro de tiempo que se consigue con el acceso directo (o acceso aleatorio) se hace notoria cuando solamente han de procesarse determinados registros dentro de un fichero de gran volumen.

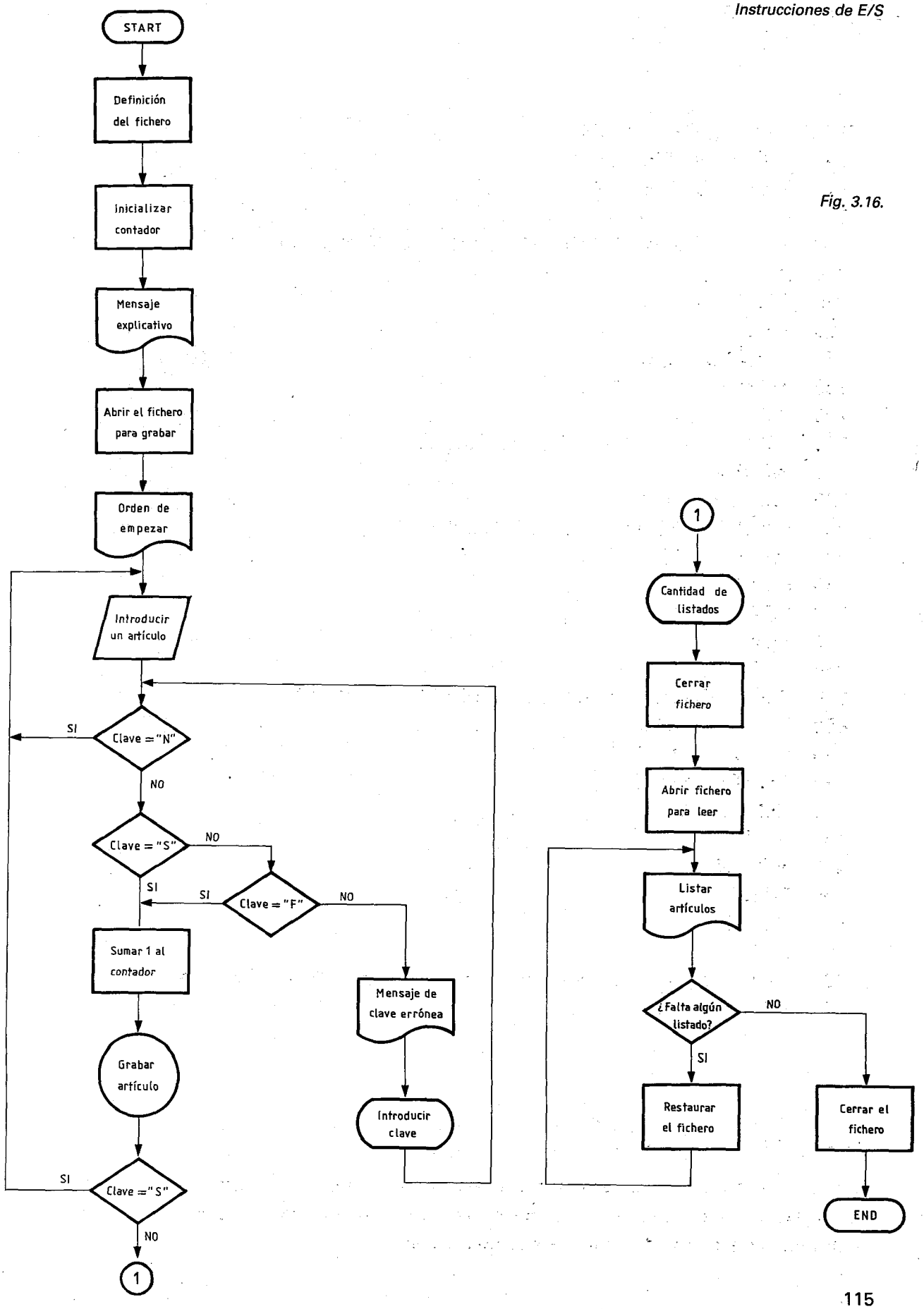

START
Definición del fichero
Inicializar contador
Mensaje explicativo
Abrir el fichero para grabar
Orden de empezar
Introducir un artículo
Clave = "N"
SI
NO
Clave = "S"
NO
SI
Clave = "F"
SI
NO
Sumar 1 al contador
Mensaje de clave errónea
Grabar artículo
Introducir clave
Clave = "S"
SI
NO
1
1
Cantidad de listados
Cerrar fichero
Abrir fichero para leer
Listar artículos
¿Falta algún listado?
NO
SI
Restaurar el fichero
Cerrar el fichero
END

Fig. 3.16.

Pointer

Mientras que en un fichero de tipo secuencial, el *pointer* (puntero) avanza de un registro al siguiente registro por cada instrucción de lectura o de grabación, en los ficheros de acceso directo, es el programador quien determina la posición del pointer. Esto se logra con la instrucción SET, cuyo formato es:

SET n: { X / EOF }

en donde:

SET = Hacer
n = Constante numérica o variable numérica o expresión aritmética.
X = Constante numérica o variable numérica o expresión aritmética
EOF = Fin de fichero (End Of File)

El campo n es el número de referencia del fichero al que se le posiciona el pointer. Si su valor es 1, puede omitirse junto con los dos puntos.

El valor que especifica el campo X es la posición del registro al que apuntará el pointer.

El valor de X no puede ser mayor que el número de registros del fichero. Además, el valor de X no puede ser cero, negativo ni tener parte decimal.

La opción EOF permite poner el puntero al final del archivo, de forma que cualquier instrucción PUT ejecutada posteriormente añadirá nuevos registros al fichero. Su finalidad es la misma que la opción APPEND en la instrucción OPEN, pero evitando la necesidad de cerrar el fichero y volverlo a abrir.

Ejemplos:

— Situar el puntero del fichero número 3 apuntando al registro 1092:

```
740 SET 3: 1092
```

— Situar el puntero del fichero cuyo número de referencia está almacenado en la variable A7, apuntando al registro definido por la suma de las variables A6 y B4:

```
640 SET A7: (A6 + B4)
```

— Situar el puntero del fichero número 17 apuntando a la posición siguiente al último registro.

```
440 SET 17: EOF
```

Lectura en acceso directo

La lectura de los registros de un fichero organizado de forma aleatoria, se logra mediante la combinación de las instrucciones SET y GET.

Con la instrucción SET, se sitúa el puntero en el registro que se desea leer. En caso de no codificarse esta instrucción, los registros del fichero aleatorio serán leídos secuencialmetne.

Ejemplos:

— Leer el registro 19 del fichero 6

```
750 SET 6: 19
760 GET 6: A, B, A$, C
```

— Leer los registros pares del fichero aleatorio número 2. Su nombre es FICH y posee 500 registros:

```
10 OPEN 2, "FICH", INQUIRY
 .
640  J = 0
650  J = J + 2
660  SET 2 : J
670  GET 2 : A, B, C, A$, B$, C$
 .
940  IF J < 500 THEN 650
```

o también

```
640  FOR N = 2 TO 500 STEP 2
650  SET 2 : N
660  GET 2 : A, B, C, A$, B$, C$
 .
930  NEXT N
```

Grabación en acceso directo

La grabación de registros dentro de un fichero aleatorio, se realiza mediante la combinación de las instrucciones SET y PUT.

Con la instrucción SET se sitúa el puntero en la posición en la que se desea grabar el registro. En el caso de que no se codifique esta instrucción los registros del fichero aleatorios serán grabados secuencialmente.

Ejemplos:

— Grabar en la posición 82 del fichero aleatorio FICH (número 8) los campos A4, −3.17, B$, A5:

```
 10  OPEN 8, "FICH", UPDATE
  .
  .
840  SET 8 : 82
850  PUT 8 : A4, −3.17, B$, A5
```

— Grabar secuencialmente, a partir de la posición 43 del fichero aleatorio EMP (número 1), cien registros con el contenido de A1, A2 y A3:

```
 10  OPEN 1, "EMP", UPDATE
  .
  .
580  FOR N = 43 TO 143
  .
  .
640  SET N
650  PUT A1, A2, A3
660  NEXT N
```

o también:

```
580  SET 43
590  FOR N = 1 TO 100
  .
  .
650  PUT A1, A2, A3
660  NEXT N
```

— Hacer un programa para cambiar la constante ACERO por COBRE del segundo campo del registro número 547 del fichero aleatorio MA (número 5).

```
 10  OPEN 5, "MA", UPDATE
 20  SET 5 : 547
 30  GET 5 : A, A$, B, C, D, E
 40  IF A$#'ACERO' THEN 80
 50  PRINT 'ERROR'
 60  CLOSE 5
 70  STOP
 80  A$ = 'COBRE'
 90  SET 5 : 547
100  PUT 5 : A, A$, B, C, D, E
110  GO TO 60
120  END
```

EJERCICIOS

1. Un registro está formado por:

   a) Ficheros.
   b) Campos.
   c) Variables.

2. Una instrucción de salida de fichero permite:

   a) Leer registros.
   b) Grabar registros.
   c) Imprimir registros.

3. La impresión se realiza con la instrucción:

   a) PUT.
   b) READ.
   c) PRINT.

4. En una instrucción DATA pueden codificarse:

   a) Sólo constantes.
   b) Sólo constantes alfanuméricas.
   c) Sólo constantes numéricas.

5. En la impresión comercial, el carácter &:

   a) Sustituye el carácter # de la impresión no comercial.
   b) Imprime los ceros no significativos.
   c) Imprime el signo de un dato numérico.

6. Las instrucciones DATA crean:

   a) Un bloque de constantes.
   b) Dos bloques, uno numérico y otro alfanumérico.
   c) Dos bloques, uno numérico y otro alfabético.

7. La impresión con formato se realiza con la instrucción:

   a) PRINT.
   b) PRINT USING.
   c) PUT.

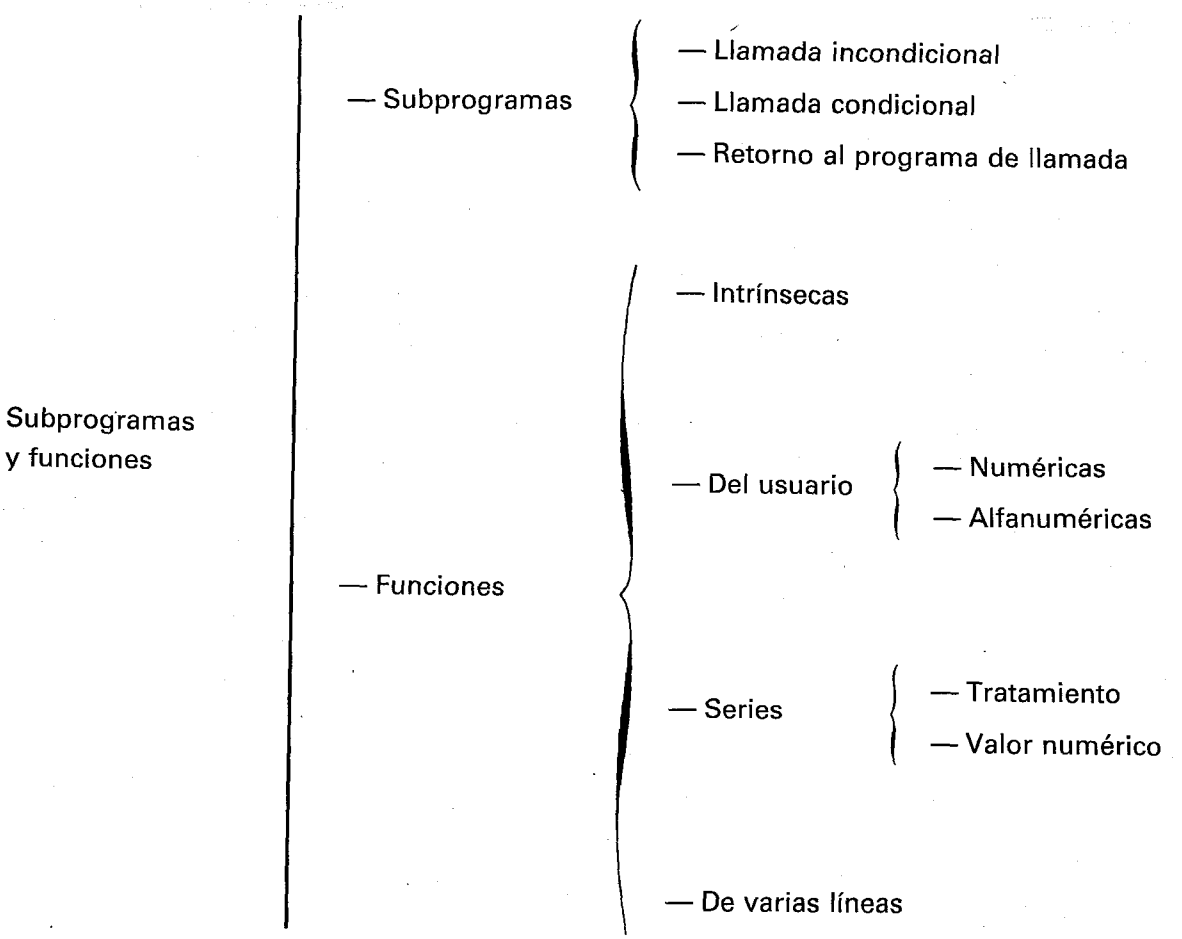
8. La instrucción NL equivale a la instrucción:

   a) PRINT
   b) PRINT " ";
   c) Ninguna de las anteriores

4.

SUBPROGRAMAS Y FUNCIONES

CUADRO RESUMEN DE LA LECCION



4.1. SUBPROGRAMAS

Es posible que un determinado bloque de instrucciones se repita varias veces dentro de un programa. En este caso, se codifica dicho bloque de instrucciones como un *subprograma (subrutina)* y, cada vez que deba ejecutarse, se hará una referencia *(llamada)* a dicho subprograma, con lo que el procesador detendrá momentáneamente la ejecución del programa principal que realiza la llamada, bifurcará al subprograma y una vez finalizado éste, continuará con la secuencia normal de ejecución del programa principal.

Con la utilización de los subprogramas se pueden obtener una serie de ventajas, como por ejemplo:

— Ahorrar tiempo de codificación, ya que el bloque de instrucciones que se ejecutan varias veces dentro de un programa sólo ha de codificarse una vez.

— Ahorrar tiempo de compilación puesto que dicho bloque de instrucciones solamente ha de ser compilado una vez.

Un programa principal puede llamar a todos los subprogramas que se considere necesario, y cada uno de estos subprogramas puede llamar a su vez a tantos subprogramas como sea preciso (excepto a sí mismo).

La ejecución del subprograma puede comenzar por cualquier instrucción, es decir, por la que indique el programa principal. Una vez finalizada la ejecución de un subprograma, el control del procesador retorna a la instrucción siguiente a la de llamada en el programa principal.

Las instrucciones de llamada a un Subprograma se clasifican en:

— Incondicionales.
— Condicionales.

4.1.1. Llamada incondicional

La instrucción de llamada incondicional a un subprograma es la instrucción GOSUB, cuyo formato es:

```
GOSUB n
```

en donde:

GOSUB = palabra clave
n = número de línea del subprograma

Siempre que se ejecute una instrucción de este tipo, el procesador bifurcará la secuencia de ejecución hacia la instrucción cuyo número de línea sea n, y no se devolverá el control al programa principal hasta que no encuentre la instrucción de retorno.

Ejemplo (Fig. 4.1.):

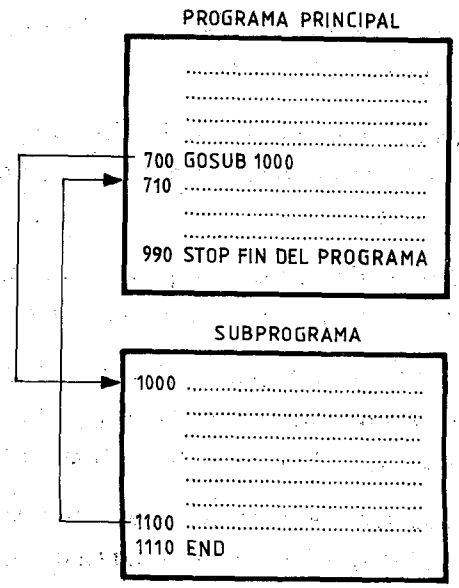


Fig. 4.1. ▶

4.1.2. Llamada condicional

El formato de este tipo de instrucciones es:

GOSUB n_1, n_2, n_3, ..., ON X

en donde:

GOSUB y ON = palabras clave
n_i = números de líneas de subprograma
X = variable numérica o expresión aritmética.

Con esta instrucción se bifurca a la línea n_i según el valor que tenga el campo X. Si dicho valor es 1, bifurca al n_1, si es 2, al n_2, etc.

Si el valor de X contiene decimales son truncados automáticamente.

Si el valor de X es menor que uno, o mayor que la cantidad de números de línea especificados, la instrucción es ignorada.

Al igual que las llamadas incondicionales, el control del procesador no retornará a la siguiente instrucción en secuencia a la instrucción de llamada condicional ejecutada, hasta que no encuentre la instrucción de retorno.

Ejemplo: (Fig. 4.2.):

En el instante de ejecutar la instrucción 300 y dependiendo de los valores de las variables A y B se obtiene una de estas secuencias:

A	B	*Valor resultado*	*Secuencia*
5	−4	(= 1)	A − C − B
−2	4	(= 2)	A − D − B
3	0	(= 3)	A − H − C − I − B
1	3	(= 4)	A − G − B
3	2	(= 5)	A − E − H − C − I − F − B
7	4	(= 11)	A − B
−1	−5	(= −6)	A − B

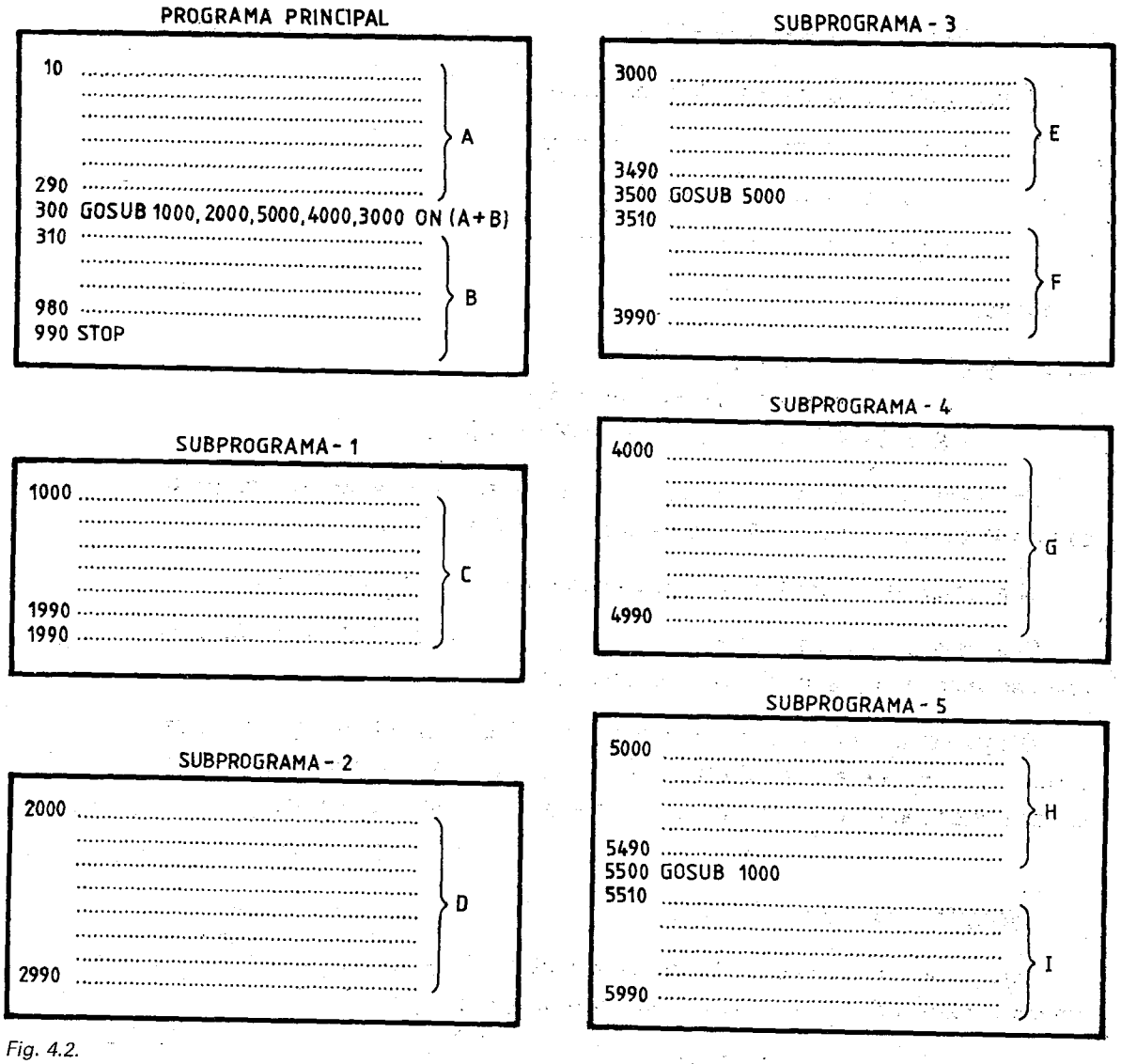


Fig. 4.2.

4.1.3. Retorno al programa de llamada

La instrucción que provoca el retorno desde la subrutina (subprograma) al programa principal es la instrucción RETURN, cuyo formato es:

```
RETURN
```

Para ver el funcionamiento de esta instrucción, es necesario saber qué es una *pila* y cómo funciona dentro del ordenador.

Una *pila* es una zona de memoria (a la cual sólo puede acceder el procesador) destinada a almacenar direcciones del programa en ejecución. Existen dos tipos de pilas:

— FIFO
— LIFO

La primera de ellas (First In First Out = el primero que entra es el primero que sale) no se comentará ya que su explicación sale del contenido de este apartado.

La segunda de ellas (Last In First Out = el último que entra es el primero que sale), es el tipo de pila empleado por el procesador para almacenar las sucesivas direcciones de retorno dentro de un programa.

Ejemplo:

```
10  REM PROGRAMA EJEMPLO
20  REM
30  PRINT "PROGRAMA PRINCIPAL"
40  GOSUB 120
50  GOSUB 220
60  GOSUB 170
70  GOSUB 280
80  STOP FIN DEL EJEMPLO
90  REM
100 REM SUBRUTINA PEPE
110 REM
120 PRINT 'CORTESIA DE "PABLO"'
130 RETURN
140 REM
150 REM SUBRUTINA JUAN
160 REM
170 PRINT 'CORTESIA DE "JUAN"'
180 RETURN
190 REM
200 REM SUBRUTINA LUIS
210 REM
220 PRINT 'CORTESIA DE "LUIS"'
230 GOSUB 170
240 RETURN
250 REM
260 REM SUBRUTINA JAVIER
270 REM
280 PRINT 'CORTESIA DE "JAVIER"'
290 GOSUB 120
300 RETURN
310 REM
320 END
```

En la página siguiente se presenta, en forma de tabla, parte de los sucesivos pasos de ejecución de este programa. En cada paso se indica qué es lo que aparecería en el terminal, cuál sería la siguiente instrucción a ejecutar y cuáles serían las direcciones de retorno almacenadas en la pila de Returns. Cada vez que se ejecutase una instrucción Return, el proceso seguiría con la ejecución de la última instrucción almacenada en dicha pila.

	En el terminal	Siguiente instrucción a ejecutar	Pila de Returns
• instrucción 30:	PROGRAMA PRINCIPAL	40	
• instrucción 40:	PROGRAMA PRINCIPAL	120	↓ 50
• instrucción 120:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO"	130	50
• instrucción 130:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO"	50 ←	
• instrucción 50:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO"	220	↓ 60
• instrucción 220:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO" CORTESIA DE "LUIS"	230	60
• instrucción 230:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO" CORTESIA DE "LUIS"	170	↓ 240 60
• instrucción 170:	PROGRAMA PRINCIPAL CORTESIA DE "PABLO" CORTESIA DE "LUIS" CORTESIA DE "JUAN"	180	240 60

4.2. FUNCIONES

Una *función* define un determinado cálculo del que se tendrá la necesidad de realizar un considerable número de veces dentro de un programa.

Para no tener que codificar cada uno de dichos cálculos tantas veces como hiciera falta, el lenguaje BASIC permite definirlos una sola vez si se especifica que dichos cálculos son una función.

Las funciones se llaman mediante el nombre que las referencia y cerrando entre paréntesis los datos que se transmiten a la función.

Al encontrar una referencia, el procesador busca la función, ejecuta los cálculos que determine, utilizando los datos transmitidos y devuelve el resultado correspondiente.

4.2.1. Funciones intrínsecas

Existen una serie de cálculos muy complicados cuya utilización es muy común.

Como ayuda al programador, el lenguaje BASIC tiene ya definidos internamente estos cálculos para que puedan ser solicitados cuando sean necesarios, ahorrando al programador la necesidad de crearlos.

A estos cálculos ya definidos internamente, se les denomina *funciones intrínsecas.* Son las siguientes:

Función	*Significado*
SIN (x)	seno trigonométrico (x en radianes)
COS (x)	coseno trigonométrico (x en radianes)
TAN (x)	tangente trigonométrica (x en radianes)
COT (x)	cotangente trigonométrica (x en radianes)
SEC (x)	secante trigonométrica (x en radianes)
CSC (x)	cosecante trigonométrica (x en radianes)
ASN (x)	arco del seno (ángulo en radianes)
ACS (x)	arco del coseno (ángulo en radianes)
ATN (x)	arco de la tangente (ángulo en radianes)

Función	*Significado*
HSN (x)	seno hiperbólico
HCS (x)	coseno hiperbólico
HTN (x)	tangente hiperbólica
DEG (x)	conversión de radianes a grados
RAD (x)	conversión de grados a radianes
EXP (x)	exponencial en base *e*
ABS (x)	valor absoluto
LOG (x)	logaritmo neperiano
LTW (x)	logaritmo en base 2
LGT (x)	logaritmo decimal
SQR (x)	raíz cuadrada
INT (x)	truncamiento
CPU	proporciona el tiempo (en segundos) empleado por la Unidad Central de Proceso desde que el programa empezó a ejecutarse hasta el instante en que se encuentra esta función.
CLK	proporciona la hora, en segundos.
DAY	proporciona la fecha con el siguiente formato a a m m d d por ejemplo, el 5 de junio de 1960 aparecería así: 6 0 0 6 0 5
SGN (x)	proporciona el signo de x.

si	*devuelve*
$x > 0$	SGN (x) = +1
$x = 0$	SGN (x) =0
$x < 0$	SGN (x) = −1

RND ..... proporciona un número aleatorio entre 0 y 1.

- si x = 0, la primera vez que se llama a esta función en un programa, el sistema proporciona una constante aleatoria inicial y a partir de este momento, cada número aleatorio se genera a partir del anterior.

Función	Significado
	• si X > 0 el primer número aleatorio se genera a partir de X. • si X < 0 el primer número aleatorio se genera a partir de la hora del día.
RND	equivalente a RND(0)
ERM	proporciona el código del error que cancela un programa.
ERL	proporciona el número de línea de la instrucción que generó un código de error.
CHR (x)	proporciona el carácter correspondiente al código ASCII especificado.
ASC (x)	proporciona el código ASCII correspondiente al carácter especificado.
DET	determina si una matriz posee inversa.
LOC (x)	proporciona la posición del puntero dentro de un fichero.
LOF (x)	proporciona la alocación del último registro de un fichero.
TAB (x)	posiciona el cabezal de la impresora.

Ejemplos:

— Hallar la raíz cuadrada del valor contenido en la variable A8.

```
340 P1 = SQR (A8)
```

si A8 = 12.35 entonces P1 = 3.5142567

— Hallar el seno trigonométrico del ángulo contenido en A

```
280 B7 = SIN (A)
```

si A = 2 entonces B7 = 0.90929743

— Hallar el equivalente en radianes de un ángulo en grados:

```
540 A2 = RAD (CS)
```

si C5 = 90 entonces A2 = 1.5707963

4.2.2. Funciones del usuario

Cuando el programador tenga necesidad de definir un cálculo dentro de su programa, debido a que dicho cálculo no está contemplado en las funciones intrínsecas, lo hará con la instrucción DEF.

Aunque las funciones del usuario pueden ser definidas en cualquier parte del programa, es más correcto el definirlas todas juntas y al principio o al final del mismo.

Funciones del usuario numéricas

Las funciones numéricas que define el programador deberán tener este formato:

DEF FNx $(x_1, x_2, ..., x_n) = E$

siendo

DEF = palabra clave
FNx = *nombre que se da a la función*
X_i = argumentos de la función
E = expresión aritmética

Las reglas de construcción de una función numérica del usuario son las siguientes:

— El nombre de la función está constituido por tres caracteres, siendo los dos primeros las letras FN y el tercero, cualquier carácter alfabético (A, ..., Z, @, $, #).

— Dentro de un programa pueden definirse un máximo de 29 funciones numéricas.

— Una función numérica puede tener tantos argumentos como sea necesario.

— Los argumentos de definición de una función numérica pueden ser tanto variables numéricas como alfanuméricas.

— Una función numérica puede hacer referencia a otras funciones, pero nunca a sí misma.

— Cuando se haga referencia a una función numérica, los argumentos transmitidos y los argumentos que la definen, deben coincidir en número, tipo y posición. Pero no es necesario que tengan el mismo nombre, ya que los argumentos de la definición solamente son útiles durante el tiempo de compilación, (que es cuando el compilador define la función). Por esto, a los argumentos de la definición se les llama *argumentos mudos* (del inglés *DUMMY*).

— Una función numérica retorna siempre un valor numérico, independientemente del tipo de argumentos mudos que la definan.

Ejemplo 1:

— Se necesita hacer una función para calcular el área de un triángulo rectángulo.

$$\text{Area} = \frac{\text{Base} \times \text{altura}}{2}$$

La codificación es la siguiente:

```
10   DEF FNA (A,B) = A*B/2
.
.
.
120  J = FNA (P,Q)
```

En tiempo de ejecución, el procesador se dispone a ejecutar la instrucción etiquetada como 120, buscará la definición que hizo el compilador de FNA y pondrá el valor de la variable P en todos los campos de la expresión numérica de la definición de función que tengan el argumento mudo A y hará lo mismo con el valor de la variable Q con el argumento mudo B.

Una vez ejecutada la función, asignará a la variable J el resultado del cálculo efectuado.

Ejemplo 2:

— Hacer una función que calcule el radio de un círculo sabiendo el valor del área.

$$\text{Radio} = \sqrt{\frac{\text{Area}}{\pi}}$$

La codificación podría ser:

```
10  DEF FNX (A) = SQR (A/&PI)
```

Funciones alfanuméricas

La definición de una función alfanumérica se realiza según el siguiente formato:

```
DEF FNx$ (A) = C
```

siendo:

DEF = palabra clave.
FNx$ = nombre de la función.
A = argumentos mudos.
C = expresión.

Las reglas de definición de las funciones seriales son las mismas que las de las funciones numéricas, excepto que:

— El valor retornado será siempre una constante alfanumérica.

— El nombre de la función se construye con los caracteres FN seguidos de un carácter alfabético (A, ... Z,@,#$) y del carácter $.

Como aplicación práctica de las funciones seriales se expondrán dos ejemplos:

Ejemplo 1:

— Encontrar una letra del abecedario sabiendo su número de orden:

```
10   DEF FNA$(A) = STR('ABCDE ..... YZ',A,1)
.
.
.
690  A$ = FNA$(P1)
```

si al ejecutar la instrucción 690: P1 = 4 entonces A$ = D

Ejemplo 2:

— Extraer de la serie de caracteres almacenados en la variable A$, N caracteres a partir del carácter cuyo número de orden es B2:

```
10   DEF FNK$(A,A$,B) = STR(A$,A,B)
.
.
.
500  J$ = FNK$ (B2, A$, N)
```

si: A$ = LARRIBA MORALES
N = 7
B2 = 9

entonces:

J$ = MORALES

4.2.3. Series

Una serie es un conjunto de caracteres numéricos, alfabéticos o especiales, dentro de una constante alfanumérica o de una variable alfanumérica con una longitud máxima de 256 caracteres (el blanco también es un carácter).

Tratamiento de las series

La instrucción para hacer referencia a una serie tiene el siguiente formato:

STR (V_1, X_1, X_2)

donde:

STR = palabra clave
V_1 = constante o variable de tipo alfanumérico
X_1 y X_2 = constantes o variables de tipo numérico o expresiones aritméticas.

Esta instrucción permite hacer referencia a unos caracteres determinados dentro de una serie.

Sus reglas son:

— X_1 y X_2 deben tener valores comprendidos entre 1 y 18.

— X_1 indica la posición del primer carácter a referenciar.

— X_2 indica el número de caracteres a referenciar a partir de la posición determinada por X_1.

— En el caso de que el resultado de X_1 o X_2 tenga parte decimal, dicha parte será truncada.

Ejemplo 1:

A partir de la variable Y$, crear una subserie y almacenarla en la variable alfanumérica Z$.

Siendo: Y$ = ABCDEFGHIJK

mediante la instrucción:

```
440  LET Z$ = STR (Y$, 2, 3)
```

se obtendrá: Z$ = BCD

Ejemplo 2:

Enlazar la serie B$ a la serie A$,

Siendo: A$ = ABC
B$ = DEFG

mediante la instrucción:

```
270  STR (A$, 4, 4) = B$
```

se obtendrá: A$ = ABCDEFG

Valor numérico

Para hallar el valor numérico de una serie se utiliza la instrucción VAL, cuyo formato es:

LET V_1 = **VAL** (V_2)

siendo:

LET = palabra clave opcional
VAL = palabra clave
V_1 = variable numérica.
V_2 = variable alfanumérica o referencia STR

Esta instrucción halla el valor numérico de los caracteres especificados dentro de una serie y almacena su valor en una variable numérica.

Las reglas de esta instrucción son:

— Si no se especifica lo contrario, la evaluación empieza por el primer carácter de la izquierda.

— La evaluación finaliza tanto al encontrar un carácter coma (,), como al llegar al final de la serie, o porque se haya especificado cuántos caracteres se quería evaluar y no han ocurrido ninguno de los dos motivos anteriores que determinan el final de la evaluación.

— Los caracteres blancos ( ) no son tenidos en cuenta.

La manera de extraer el valor numérico de unos determinados caracteres es combinando la instrucción VAL y la instrucción STR de la siguiente manera:

LET V_1 = **VAL** [**STR** (V_2, X_1, X_2)]

La principal utilidad de esta combinación es pasar unos dígitos que están almacenados en formato alfanumérico a formato numérico para poder realizar cálculos con ellos.

Algunos ejemplos para clasificar los conceptos expuestos son:

— Si: A$ = −1234E2

codificando:

```
530  A = VAL (A$)
```

se obtendrá: A = −123400

— Si A$ = ABC123 DEF456

codificando:

```
940  A = VAL (STR(A$, 5, 2))
```

se tendrá que: A = 23

— Si: A$ = ABCD123.4E-1EF

con:

```
730  A = VAL [STR (A$, 5, 8)]
```

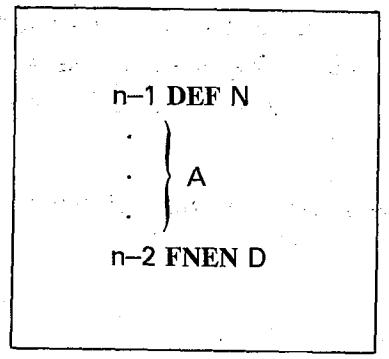
se obtendrá: A = 12.34

4.2.4. Funciones de varias líneas

Existen cálculos matemáticos cuya complejidad es tal que no se pueden llevar a cabo utilizando solamente un renglón, esto es más evidente cuando las

fórmulas aritméticas son muy largas o cuando deben contemplarse operaciones de bifurcación condicional, haciéndose imprescindible el uso de varios renglones para definir una sola función.

La manera de definir este tipo de funciones es mediante el formato:



siendo:

n–1 y n–2 = número de línea
DEF y FNEND = palabras clave
N = nombre de la función y argumentos
A = bloque de instrucciones

Las reglas de composición son:

— Una función de renglón puede tener tantos argumentos mudos como sean precisos.

— Este tipo de funciones retornarán un solo valor resultado, que será numérico si es del tipo numérico y un solo valor alfanumérico si es del tipo alfanumérico.

— La determinación del tipo al que pertenece la función (numérica o alfanumérica) viene dada por las reglas de definición del nombre de una función.

— Entre las instrucciones de definición (DEF) y de fin de definición (FNEND), pueden incluirse tantas instrucciones de cualquier tipo como sea necesario (lecturas, bifurcaciones, cálculos, llamadas a otras funciones, bucles, ... etc.).

— Los argumentos mudos pueden ser numéricos y alfanuméricos, debiendo coincidir los argumentos transmitidos en número, tipo y posición.

— No está permitido bifurcar desde una instrucción dentro de la definición de función a ninguna instrucción exterior.

— La última instrucción ejecutada en una función de varios renglones antes de volver a la secuencia del programa, debe ser una instrucción que asigne el valor que se quiere retornar al nombre de la función.

— Las diversas instrucciones de una función de varios renglones pueden utilizar variables que no estén definidas como argumentos mudos, con lo que al ser evaluada dicha función, se procesarán como variables pertenecientes al programa.

A continuación se muestran unos ejemplos prácticos sobre el uso de las funciones de varios renglones:

Ejemplo 1:

La función que aquí se muestra, cumple el propósito de determinar el mayor de entre dos números:

```
10  DEF FNA (A,B)
20  IF A >= B THEN 50
30  FNA = B
40  GO TO 60
50  FNA = A
60  FNEND
```

su funcionamiento lógico es el siguiente:

10 — Los argumentos mudos A y B adquieren el valor suministrado por la instrucción de llamada a esta función.

20 — Compara A con B y si resulta que A no es menor que B:

50 — Asigna a la función el valor de A.

60 — El control del procesador retorna a la instrucción de llamada.

30 — Si en la comparación resulta que A es menor que B, se asigna a la función el valor de B.

60 — El control del procesador retorna a la instrucción de llamada.

La instrucción de llamada podría ser:

```
580  J = FNA (X1, X2)
```

con lo que la variable J adquirirá el valor de la variable X1 o de la variable X2 dependiendo de cuál de ellas sea mayor que la otra.

Ejemplo 2:

Hallar la mitad del mayor de entre tres números:

Se puede utilizar la misma función FNA de la siguiente manera:

```
610  K = FNA (FNA(X1, X2), X3) / 2
```

Ejemplo 3:

De igual manera se puede hallar el mayor de entre cuatro números:

```
720  L = FNA (FNA(X1,X2), FNA(X3,X4))
```

Ejemplo 4:

Hallar la suma de los números comprendidos entre dos números L y N dados.

```
10  DEF FNX(A,B)
20  S = 0
30  FOR J = A TO B
40  S = S + J
50  NEXT J
60  FNX = S
70  FNEND
.
.
.
180 ————————
200 P4 = FNX (L,N)
210 ————————
```

EJERCICIOS

1. La instrucción de retorno al programa de llamada es .....

2. Un subprograma puede:

   a) Llamar a otro subprograma.
   b) Llamarse a sí mismo.
   c) Llamar al programa principal.

3. La instrucción para llamar a un subprograma es la instrucción .....

4. ¿Cuántos tipos de instrucciones de llamada existen?

   a) Uno.
   b) Dos.
   c) Tres.

5. ¿Qué significa LIFO? .....

6. Se puede entrar en un subprograma.

   a) Sólo por su primera instrucción.
   b) Por cualquier instrucción.
   c) Sólo por una instrucción ejecutable.

7. Se puede salir de un subprograma.

   a) Por cualquier instrucción.
   b) Sólo por la última instrucción.
   c) Por una instrucción de retorno.

8. La instrucción GOSUB, es una instrucción:

   a) De bifurcación.
   b) De control.
   c) Que no admite comentario.

9. A cada instrucción GOSUB ejecutada debe corresponderle una instrucción .....

10. La instrucción VAL:

    a) Obtiene el valor numérico de una serie.
    b) Define una serie.
    c) Extrae caracteres de una serie.

11. Las funciones intrínsecas:

    a) Las crea el programador.
    b) Son funciones alfanuméricas.
    c) Ya están definidas dentro del ordenador.

12. Una Función del Usuario:

    a) Sólo puede ocupar un renglón.
    b) No posee argumentos.
    c) Puede ocupar uno o varios renglones.

5.

MATRICES

CUADRO RESUMEN DE LA LECCION

Matrices	— Definición de una matriz — Asignación de una matriz — Suma de matrices — Resta de matrices — Producto de matrices — Producto de una matriz por un escalar — Matriz traspuesta — Matriz inversa — Inicialización — Almacenamiento interno — Matrices alfanuméricas — Lectura por programa — Lectura por terminal — Impresión de matrices — Lectura de un fichero — Grabación de un fichero

Una matriz de dimensiones *p* y *n* está formada por *p* filas y *n* columnas de constantes numéricas o alfanuméricas. Su representación general es:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots\ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots\ldots & a_{2n} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ a_{p1} & a_{p2} & \ldots\ldots\ldots & a_{pn} \end{bmatrix}$$

Cada a_{ij} de una matriz se llama *elemento*. Su posición se indica por dos subíndices numéricos, el primero indica la fila y el segundo indica la columna.

Es una matriz, por ejemplo:

$$A = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & -1 & 0 & 2 \\ 1 & 2 & 1 & 0 \end{bmatrix}$$

en donde:

$a_{11} = a_{24} = a_{32} = 2$
$a_{14} = a_{31} = a_{33} = 1$
$a_{22} = -1$
$a_{12} = a_{13} = a_{21} = a_{23} = a_{34} = 0$

La *diagonal principal* de una matriz está formada por los elementos a_{ij} de tal manera que i = j.

Se llama *matriz unidad* a aquella matriz en la que todos sus elementos son 0 excepto los elementos de la diagonal principal que son 1.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se dice que dos matrices A y B son *equidimensionales* cuando tienen el mismo número de filas y el mismo número de columnas. En caso contrario son *incongruentes*.

Ejemplo:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 0 \end{bmatrix} \quad C = \begin{bmatrix} -3 & 2 & 7 \\ -4 & 3 & -3 \end{bmatrix}$$

Las matrices A y C son equidimensionales y las matrices A y B son incongruentes entre sí.

Dos matrices son *iguales* cuando son equidimensionales y $a_{ij} = b_{ij}$ para todo $i = 1, 2, 3, \ldots p$ y todo $j = 1, 2, 3, \ldots n$.

Ejemplo:

$$E = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad G = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

Las matrices E y F son iguales, pero las matrices F y G aunque son equidimensionales, no son iguales.

Se dice que dos matrices son *conformables* cuando el número de columnas de una de ellas es igual al número de filas de la otra.

Ejemplo:

$$H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad I = \begin{bmatrix} 7 & 8 \\ 9 & 0 \\ -1 & -2 \end{bmatrix}$$

Una matriz es *idéntica* cuando tiene el mismo número de filas y de columnas.

Ejemplo:

$$J = \begin{bmatrix} 9 & 3 & 5 \\ 4 & 8 & 1 \\ 6 & 2 & 7 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 4 & 7 & 3 \\ 8 & 6 & 2 & 9 \\ 5 & -3 & 0 & -1 \end{bmatrix}$$

La matriz J es idéntica, pero no así la matriz L.

Las matrices son utilizadas en cálculos matemáticos de cierta complejidad, por lo que se recomienda consultar un libro especializado en *cálculo matricial* al lector que tenga un especial interés en la resolución de problemas de tipo matricial. A lo largo de este capítulo, se estudia el modo de trabajar con matrices en lenguaje BASIC sin entrar en sus aplicaciones matemáticas.

5.1. DEFINICION DE UNA MATRIZ

La instrucción para definir una matriz (o *variable compuesta*), es DIM, cuyo formato es:

DIM V_1 (K_1, K_2) [, V_2 (K_3, K_4), ..., V_n (K_{n-1}, K_n)]

en donde:

DIM = Dimensión
V_i = Nombre de la Matriz
K_i, K_j = Subíndices (filas y columnas)

Las reglas para la definición de una Matriz son las siguientes:

— Dentro de una instrucción DIM pueden definirse tantas matrices como sea necesario.

— Los subíndices son constantes enteras sin signo. El primer subíndice corresponde al número de filas, y el segundo al número de columnas.

— Si el número de columnas de la matriz es la unidad el subíndice que indica las columnas puede omitirse.

— Las instrucciones DIM deben preceder en el programa a cualquier instrucción que utilice una matriz.

— Un programa en BASIC puede tener un máximo de 29 matrices numéricas (ya que V_i debe ser un carácter alfabético: A, ..., Z, @, $ ó #.

— Una matriz puede definirse de manera implícita haciendo referencia a uno de sus elementos con la instrucción de asignación LET, si no ha sido definida previamente con la instrucción DIM. En este caso, el ordenador asumirá que la dimensión de la matriz es (10,1) si sólo se especifica un subíndice y (10,10) si se especifican ambos subíndices.

Algunos ejemplos de definición de matrices son:

```
10  DIM A(5,3), B(2,6)
```

y se tiene:

$$A = \begin{bmatrix} a_{11} & a_{12} & 13 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \end{bmatrix} \qquad B = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \end{bmatrix}$$

```
10  DIM C(4), D(1,4)
```

y se tiene:

$$C = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \qquad D = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \end{bmatrix}$$

5.2. ASIGNACION DE UNA MATRIZ

La asignación de una matriz sigue las reglas de la instrucción LET. En cualquier asignación de una matriz debe codificarse su nombre seguido de los subíndices entre paréntesis.

Los subíndices pueden ser constantes numéricas, variables numéricas, expresiones aritméticas o un elemento de una matriz numérica. Si el valor de un subíndice posee decimales, serán truncados.

Ejemplos:

— La matriz: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

con la instrucción: 350 C4 = A(2,2) se tiene: C4 = 5

— La matriz: $B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & -1 & -2 \end{bmatrix}$

con la instrucción: 320 B(2,3) = −5 se tiene: $B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & -5 & 8 \\ 9 & 0 & -1 & -2 \end{bmatrix}$

— Las matrices: $C = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$ $D = \begin{bmatrix} 9 & 0 & -1 \\ -2 & -3 & -4 \\ -5 & -6 & -7 \end{bmatrix}$

con la instrucción: 330 D(2,3) = C(2,3) se tiene: $D = \begin{bmatrix} 9 & 0 & -1 \\ -2 & -3 & 7 \\ -5 & -6 & -7 \end{bmatrix}$

— Las matrices: $E = \begin{bmatrix} 9 & 8 & 7 & 4 \end{bmatrix}$ $F = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

con la instrucción: 340 F(1,1) = E(1,2) + F(2,2) se tiene: $F = \begin{bmatrix} 12 & 2 \\ 3 & 4 \end{bmatrix}$

Supongamos, ahora el siguiente programa:

```
10  DIM A(3,4)
20  L = 0
30  FOR I = 1 TO 3
40  FOR J = 1 TO 4
50  A (I,J) = L
60  L = L + 2
70  NEXT J
80  NEXT I
90  STOP
100 END
```

después de ejecutar este programa se tiene: $A = \begin{bmatrix} 0 & 2 & 4 & 6 \\ 8 & 10 & 12 & 14 \\ 16 & 18 & 20 & 22 \end{bmatrix}$

5.3. SUMA DE MATRICES

La instrucción para sumar matrices posee el siguiente formato:

MAT $m_1 = m_2 + m_3$

en donde:

MAT = Matriz
m_1, m_2, m_3 = nombres de las matrices

Las reglas de la suma de matrices son:

— Las matrices han de ser equidimensionales.

— La suma se efectúa sustituyendo cada elemento en la matriz m_1 por la suma de los elementos correspondientes a las matrices m_2 y m_3.

Sean las matrices: $B = \begin{bmatrix} 6 & 4 & 1 \\ 3 & 2 & 5 \end{bmatrix}$ $C = \begin{bmatrix} 1 & 6 & 3 \\ 5 & 2 & 4 \end{bmatrix}$

— Si se codifica:

```
40  ————————————
50  MAT A = B + C
60  ————————————
```

Al ejecutarse la instrucción 50 se tiene que: $A = \begin{bmatrix} 7 & 10 & 4 \\ 8 & 4 & 9 \end{bmatrix}$

— La codificación anterior es equivalente a la siguiente:

```
40  ————————————
50  FOR I = 1 TO 2
51  FOR J = 1 TO 3
52  A(I,J) = B(I,J) + C(I,J)
53  NEXT J
54  NEXT I
60  ————————————
```

5.4. RESTA DE MATRICES

La instrucción que efectúa la resta de matrices posee el siguiente formato:

MAT $m_1 = m_2 - m_3$

en donde:

m_1, m_2, m_3 = nombres de las matrices.

Ejemplos:

Sean las matrices A, B y C con los elementos siguientes:

$$A = \begin{bmatrix} 5 & 4 & 3 \\ 9 & 8 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 4 & 5 \\ 8 & 7 & 9 \end{bmatrix} \quad C = \begin{bmatrix} 7 & 3 & 9 \\ 4 & 8 & 5 \end{bmatrix}$$

codificando:

```
810 ————————————
820 MAT C = A – B
830 ————————————
```

y se obtiene: $C = \begin{bmatrix} 2 & 0 & -2 \\ 1 & 1 & -2 \end{bmatrix}$

Se obtendría el mismo resultado codificando las instrucciones siguientes:

```
810 ————————————
820 FOR I = 1 TO 2
821 FOR J = 1 TO 3
822 C(I,J) = A(I,J) – B(I,J)
823 NEXT J
824 NEXT I
830 ————————————
```

5.5. PRODUCTO DE MATRICES

La instrucción que permite multiplicar dos matrices, posee el formato siguiente:

MAT $m_1 = m_2 * m_3$

en donde:

MAT = palabra clave
m_1, m_2, m_3 = nombres de las matrices

Sus reglas son:

— Dos matrices pueden multiplicarse cuando el número de columnas de la primera matriz es igual al número de filas de la segunda matriz (han de ser *conformables*).

— No se obtiene el mismo resultado multiplicando la matriz A por la matriz B que multiplicando la matriz B por la matriz A. Es decir, $A * B \neq B * A$.

– La dimensión de la matriz resultado son:
  - El número de filas es igual al número de filas de la primera matriz.
  - El número de columnas es igual al número de columnas de la segunda matriz.

— Cada elemento de la matriz resultado se obtiene sumando los productos obtenidos al multiplicar cada elemento de la fila correspondiente de la primera matriz por cada elemento en la columna correspondiente de la segunda matriz.

Ejemplo:

Sean las matrices A y B: $A = \begin{bmatrix} 0 & -1 & -2 \\ 7 & -1 & 0 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 4 \\ 0 & 0 \\ -2 & 1 \end{bmatrix}$

— Codificando:

```
850  MAT C = A * B
```

se obtiene:

$$C_{11} = (0 \times 2) + (-1 \times 0) + [-2 \times (-2)] = 4$$
$$C_{12} = (0 \times 4) + (-1 \times 0) + (-2 \times 1) = -2$$
$$C_{21} = (7 \times 2) + (-1 \times 0) + (0 \times -2) = 14$$
$$C_{22} = (7 \times 4) + (-1 \times 0) + (0 \times 1) = 28$$

entonces: $C = \begin{bmatrix} 4 & -2 \\ 14 & 28 \end{bmatrix}$

— Pero si se codifica:

```
890  MAT C = B * A.
```

entonces se obtiene $C = \begin{bmatrix} 28 & -6 & -4 \\ 0 & 0 & 0 \\ 7 & 1 & 4 \end{bmatrix}$

5.6. PRODUCTO DE UNA MATRIZ POR UN ESCALAR

El formato de la instrucción que permite multiplicar una matriz por un escalar es:

MAT $m_1 = (X) * m_2$

en donde:

m_1, m_2 = nombres de matrices

X = constante numérica, variable numérica o expresión aritmética. La X representa al escalar.

Sus reglas son:

— Un escalar es un número. Sin embargo se considera oportuno mantener esta denominación dentro de este apartado, ya que este tipo de multiplicación de una matriz, se corresponde con la notación matemática del *Cálculo Vectorial* y *Escalar.*

— La matriz resultado debe ser equidimensional con la matriz multiplicando.

— Los elementos de la matriz resultado son el producto del elemento correspondiente en la matriz multiplicado por el escalar.

Ejemplo:

Sea la matriz: $E = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & -1 & -4 & -3 \\ 6 & -2 & -6 & -5 \end{bmatrix}$

codificando: 920 MAT E = (2) * E

se obtiene $E = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & -2 & -8 & -6 \\ 12 & -4 & -12 & -10 \end{bmatrix}$

5.7. MATRIZ TRASPUESTA

Una matriz A es la traspuesta de otra matriz B cuando las filas de A son las columnas de B y viceversa.

Si la matriz traspuesta de A es la matriz B, entonces: $a_{ij} = b_{ji}$

Para encontrar la matriz traspuesta de una matriz dada, se utiliza la función interna TRN, cuyo formato es:

MAT m_1 = **TRN** (m_2)

en donde:

m_1, m_2 = nombre de matrices
TRN = función interna

Las matrices m_1 y m_2 han de ser *idénticas.* Además, las matrices m_1 y m_2 han de ser dos matrices con distinto nombre.

Ejemplo:

Sea la matriz:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

codificando:

```
10   DIM A(3,3), B(3,3)
.
.
.
500  MAT B = TRN(A)
```

se obtiene:

$$B = \begin{bmatrix} 1 & 8 & 7 \\ 2 & 9 & 6 \\ 3 & 4 & 5 \end{bmatrix}$$

5.8. MATRIZ INVERSA

Para invertir una matriz se utiliza la función INV, cuyo formato es:

MAT m_1 = **IVN** (m_2)

en donde:

m_1, m_2 = nombres de matrices
INV = función interna

Las matrices m_1 y m_2 han de ser *idénticas* (el número de filas debe ser igual al número de columnas en cada una de ellas) y las dos matrices han de ser equidimensionales.

Por otro lado, las matrices m_1 y m_2 han de ser dos matrices con distinto nombre.

Ejemplo:

Sea la matriz siguiente:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 2 & 0 \\ -3 & -1 & 1 \end{bmatrix}$$

codificando:

```
10   DIM A(3,3), B(3,3)
.
.
.
500  MAT B = INV (A)
```

se obtiene

$$B = \begin{bmatrix} 1 & -1 & -1 \\ -2 & 2.5 & 2 \\ 1 & -0.5 & 0 \end{bmatrix}$$

5.9. INICIALIZACION

La instrucción para inicializar una matriz tiene el formato:

$$\mathbf{MAT}\ m_1 = \left\{ \begin{matrix} \mathbf{ZER} \\ \mathbf{CON} \\ \mathbf{IDN} \end{matrix} \right\} (n_1, n_2)$$

en donde:

MAT, ZER, CON e IDN = palabras clave
n_1, n_2 = subíndices optativos

Opción **ZER:**

$$\mathbf{MAT}\ m_1 = \mathbf{ZER}\ (n_1, n_2)$$

Con la opción ZER se ponen a cero todos los elementos de la matriz.

Opción **CON:**

$$\mathbf{MAT}\ m_1 = \mathbf{CON}\ (n_1, n_2)$$

Con la opción CON se hacen igual a 1 todos los elementos de la matriz.

Opción **IDN:**

$$\mathbf{MAT}\ m_1 = \mathbf{IDN}\ (n_1, n_2)$$

Con la opción IDN se ponen a 1 todos los elementos de la diagonal principal y a 0 los elementos restantes *(matriz unidad).*

Los subíndices n_1 y n_2 son optativos. Si se especifican, la matriz m_1 cambia sus dimensiones anteriores por las aquí especificadas (la matriz se *redimensiona*).

El valor de cada subíndice n_i no puede superar el valor del subíndice correspondiente en la instrucción DIM.

Ejemplo:

— Con el programa:

```
10  DIM A(4,4)
20  MAT A = ZER
30  MAT A = ZER (3,3)
40  MAT A = CON
50  MAT A = IDN (4,4)
```

Se obtendrá:

- instrucción 20

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- instrucción 30

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- instrucción 40

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- instrucción 50

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.10. ALMACENAMIENTO EN LA M.C

Los elementos de una matriz se almacenan consecutivamente en memoria.

Sea la matriz:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

en la memoria se almacena:

A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) A(2,3) A(3,3)

5.11. MATRICES ALFANUMERICAS

Cada elemento de una matriz alfanumérica es una constante alfanumérica (de 18 caracteres).

Por ejemplo, sea la matriz A$. Con las instrucciones:

```
10  DIM A$ (S)
20  INPUT STR [A$(1), 1,36]
```

si el operador teclea:

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890

se obtendrá:

A$(1) = ABCDEFGHIJKLMNOPQR
A$(2) = STUVWXYZ1234567890

Si los elementos 1 y 2 de la matriz A$ y los elementos 7 y 8 de la matriz B$ son:

B$(7) = * * * * * * * * * * * * * * * * * *
B$(8) = * * * * * * * * * * * * * * * * * *
A$(1) = ABCDEFGHIJKLMNOPQR
A$(2) = STUVWXYZ1234567890

con la instrucción:

```
580  STR [A$(1),11,18] = STR [B$(7),3,18]
```

se obtendrá:

A$(1) = * * * * * * * * * * CDEFGHIJ
A$(2) = KLMNOPQRST * * * * * * * *

5.12. LECTURA POR PROGRAMA

La instrucción para leer una matriz tiene el formato:

MAT READ m_1 (n_1, n_2), m_2 (n_3, n_4),...

donde:

m_i = nombre de la matriz
n_i = subíndices opcionales

Si se codifican los subíndices, la matriz será redimensionada a los valores especificados.

Ejemplo:

```
10  DIM A (3,3)
20  MAT READ A
30  MAT READ A (2,3)
40  DATA 1,2,3,4,5,6,7,8,9,0
```

- con la instrucción 20

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

- con la instrucción 30

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

5.13. LECTURA POR TERMINAL

Para entrar una matriz desde el terminal, se utilizará la instrucción INPUT con el siguiente formato:

MAT INPUT m_1 (n_1, n_2), m_2 (n_3, n_4),...

en donde:

m_i = nombre de la matriz
n_i = subíndices opcionales

Si se codifican los subíndices, la matriz será redimensionada a los valores especificados.

Ejemplo:

Sean las instrucciones

```
10  DIM A (3,3)
20  INPUT A
30  INPUT A (3,2)
```

Si el operador teclea: % 1,2,3,4,5,6,7,8,9,0,1,2,3,4,5 se obtendrá:

- con la instrucción 20:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

- con la instrucción 30:

$$A = \begin{bmatrix} 0 & 3 \\ 1 & 4 \\ 2 & 5 \end{bmatrix}$$

5.14. IMPRESION DE MATRICES

Las matrices se imprimen por filas mediante la instrucción MAT PRINT, cuyo formato es:

MAT PRINT $m_1, m_2, ..., n$

donde:

m_i = nombre de la matriz

Ejemplo:

Sean las matrices A, B y C

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

codificando la instrucción:

```
7420  MAT PRINT A, B, C
```

se obtendrá en el terminal de salida:

```
 1        2        3
 4        5        6
 7        8        9

 1        2
 3        4
 5        6

 1        2        3
 4        5        6
```

5.15. LECTURA DE UN FICHERO

Para leer una matriz en un fichero se utilizará la instrucción MAT GET con el siguiente formato:

MAT GET m_1 (n_1, n_2), m_2 (n_3, n_4) ...

siendo:

m_i = nombre de la matriz
n_i = subíndices opcionales

Si se codifican los subíndices, la matriz será redimensionada a los valores especificados.

Sean las instrucciones

```
 10  DIM A(5,7)
120  MAT GET A(2,2)
```

si el registro contiene:

1	2	3	4

con la instrucción 120 se obtendrá: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

5.16. GRABACION DE UN FICHERO

La grabación de matrices en un fichero se realiza mediante la instrucción MAT PUT, cuyo formato es:

MAT PUT m_1, m_2, ...

en donde:

m_i = nombre de matriz

Sigue las mismas reglas que la instrucción PUT.

Ejemplo:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 0 \end{bmatrix}$$

codificando:

```
9880  MAT PUT A,B
```

se grabará el registro:

1	2	3	4	5	6	7	8	9	0

EJERCICIOS

1. Dos matrices son equidimensionales cuando:

   a) Tienen las mismas dimensiones.
   b) Una de ellas es la inversa de la otra.
   c) Sus elementos son iguales.

2. Al multiplicar la matriz A por la matriz B

   a) Se obtiene un resultado distinto de multiplicar la matriz B por la matriz A.
   b) Se obtiene un escalar.
   c) Se obtiene la matriz idéntica.

3. Con la función IDN los elementos de una matriz se inicializan

   a) A cero.
   b) A uno.
   c) Todos a cero excepto los elementos de la diagonal principal que se inicializan a uno.

4. La traspuesta de una matriz es el resultado de:

   a) Multiplicarla por un escalar.
   b) Cambiar las filas por las columnas y viceversa.
   c) Multiplicarla por la matriz unidad.

5. Para sumar dos matrices A y B:

   a) Se suman los elementos a_{ij} con los elementos b_{ij}.
   b) Se suman los elementos a_{ij} con los elementos b_{ji}.
   c) Dos matrices no se pueden sumar.

6. Para leer una matriz por programa se utiliza la instrucción:

   a) MAT GET.
   b) READ.
   c) MAT READ.

7. Al multiplicar la matriz A por un escalar, la matriz resultado:

   a) Es equidimensional con la matriz A.
   b) No puede tener el mismo nombre que la matriz A.
   c) Será incongruente con la matriz A.

8. Para leer una matriz desde el terminal del operador:

   a) Se utiliza la instrucción MAT INPUT.
   b) Se solicitan los elementos uno a uno.
   c) Una matriz no puede ser entrada desde el terminal.

9. La inversa de una matriz se obtiene:

   a) Con la función INV.
   b) Con la instrucción INV.
   c) Redimensionando la matriz.

10. El nombre de una matriz numérica se forma con:

   a) Un carácter alfabético y con un carácter numérico.
   b) Un cáracter alfabético y con un carácter especial.
   c) Un carácter alfabético.

6.

ERRORES DE EJECUCION

CUADRO RESUMEN DE LA LECCION

Errores de Ejecución	— Cancelación de un programa. — Tratamiento de un fichero. — Final de fichero. — Cálculo aritmético. — Cancelación de una condición de error.

Todo error que aparezca durante la fase de ejecución de un programa provoca la interrupción definitiva de su ejecución si no ha sido previsto por el programador.

Existen dos instrucciones para la detección de errores, IF y ON... GO TO cuyos formatos son:

```
IF x { THEN } n
     { GO TO }
```

```
ON x GO TO n
```

siendo

x = condición de error.
m = número de línea.

Con estas instrucciones se bifurca hacia el número de línea n cuando se detecta la condición de error especificada.

La instrucción ON... GO TO puede codificarse en cualquier parte del programa ya que su condición de error (x) permanece a lo largo de la ejecución del mismo.

La condición de error de la instrucción IF sólo se detecta cuando el procesador ejecuta la instrucción.

Cuando aparece un error en un programa, el procesador *activa* las funciones ERM (código del tipo de error) y ERL (número de línea de la instrucción en

la que ha tenido lugar el error) para que el programador pueda consultarlas y saber cuál es el error y en dónde ha surgido.

Los códigos asociados a cada tipo de error deben consultarse en el manual del fabricante del ordenador, ya que varían según el modelo y la marca.

6.1. CANCELACION DE UN PROGRAMA

La condición de error que activa el procesador cuando debe cancelar un programa en ejecución (interrupción definitiva) es:

ERR

Por lo tanto al codificar:

```
  10  ON ERR GO TO 5020
  .
  .
5010  STOP
5020  PRINT "*TERMINACION ANORMAL*"
5030  PRINT "*ERROR:"; ERM;
5040  PRINT "EN LA INSTRUCCION:"; ERL; "*"
  .
  .   } A
  .
7000  END
```

El procesador, en lugar de cancelar el programa en la instrucción 3180, por ejemplo, aparecerá en el terminal del operador el siguiente mensaje: y ejecutará el bloque de instrucciones A.

```
* TERMINACION ANORMAL *
* ERROR: 17 EN LA INSTRUCCION: 3180 *
```

El programador consultará en el manual del fabricante el error cuyo código es el 17 y revisará su programa para detectar por qué falta la instrucción 3180.

6.2. TRATAMIENTO DE UN FICHERO

La condición de error que activa el procesador cuando detecta un error en el tratamiento de un fichero es:



siendo:

ERR = condición de error.
u = número del fichero.

Si se codifica:

```
 10   ON ERR 3 GO TO 600
 20   OPEN 3, "FICH", OUTPUT
 .
 .
100   PUT 3: A1, A2, A3
 .
 .
590   STOP
600   PRINT "*ERROR:"; ERM, "EN EL FICHERO FICH*"
 .  )
 .  )
 .  ) A
 .  )
 .  )
6200   END
```

Si al ejecutar la instrucción 100, aparece un error de grabación en el fichero 3 (y el código del error es el 4), en el terminal aparecerá el aviso de ERROR y la indicación del número de código correspondiente, pasándose a ejecutar el bloque de instrucciones A.

6.3. FINAL DE FICHERO

La condición de error del final de fichero es:

EOF

siendo:

EOF = condición de error (End Of File: fin de fichero).

Esta condición de error se utiliza cuando se desconoce el número de registros que contiene un fichero.

Ejemplo:

```
 10  OPEN 1, "ARTICULOS", INQUIRY, LINE
 20  PRINT "CODIGO", "DESCRIPCION",
 30  PRINT "P. COSTE", "P. VENTA", "EXISTENCIAS"
 40  GET A1, A$, A2, A3, A4
 50  IF EOF THEN 80
 60  PRINT A1, A$, A2, A3, A4
 70  GO TO 40
 80  ————————
  .
  .
  .
900  END
```

Este programa irá leyendo secuencialmente los registros del fichero aleatorio de ARTICULOS y los imprimirá.

Cuando no queden más registros para leer, (es decir, cuando se alcance el final del fichero) la condición de error EOF será cierta y el procesador, en lugar de cancelar el programa, ejecutará la instrucción 50 bifurcando hacia la instrucción 80.

Si el número de campos de un registro es menor que el número de variables de una instrucción GET o MAT GET y se ha especificado la condición de error EOF para dicho fichero, las variables sobrantes toman el valor cero (si son numéricas) o espacios en blanco (si son alfanuméricas).

6.4. CALCULO ARITMETICO

Existen tres condiciones de error para detectar las anomalías que puedan surgir al realizar un cálculo aritmético, evitar la cancelación del programa y tomar las medidas oportunas:

OVERFLOW
UNDERFLOW
ZERODIVIDE

Con la condición de error OVERFLOW se puede detectar si se está sobrepasando el límite del valor máximo admisible por el ordenador.

Ejemplo:

— Siendo el valor máximo admisible por el ordenador 10^{+50}, al ejecutarse el programa:

```
.
.
.
700   A = 12E40
710   B = 34E20
720   C = A + B
770   IF OVERFLOW THEN 9000
.
.
.
9000  ————————————
.
.
.
10000 END
```

aparecerá la condición de OVERFLOW en la instrucción 720 y en lugar de cancelarse el programa, se bifurcará a la instrucción 9000.

Con la condición de error UNDERFLOW se puede detectar si se está sobrepasando el límite del valor mínimo admisible por el ordenador.

Ejemplo:

— Siendo el valor mínimo admisible por el ordenador 10^{-50}, al ejecutarse el programa:

```
.
.
.
800   A = 1E-35
810   B = 1E-25
820   C = A/B
830   IF UNDERFLOW THEN 9300
.
.
9300  ————————
.
.
10000 END
```

aparecerá la condición de UNDERFLOW en la instrucción 820 y en lugar de cancelarse el programa, se bifurcará a la instrucción 9300.

Con la condición de error ZERODIVIDE se detecta el intento de dividir por cero.

Ejemplo:

```
10    ON ZERODIVIDE GO TO 500
20    A = 3
30    B = 50
40    C = B/(A-3)
.
.
500   ————————
.
.
1000  END
```

Al ejecutarse la instrucción 40, la condición de ZERODIVIDE es cierta, por lo que se bifurca hacia la instrucción 500 en lugar de cancelarse el programa.

6.5. CANCELACION DE UNA CONDICION DE ERROR

La instrucción que cancela el efecto de una condición de error es la instrucción REVERT, cuyo formato es:

```
REVERT x
```

siendo:

REVERT = volver a empezar.
x = condición de error.

Cuando se ejecuta una instrucción de este tipo, cancela la instrucción ON condición de-error GO TO etq.–1 que posea la condición de error x.

Ejemplo:

— En este programa

```
 10  A = 2
 20  B = 0
 30  C = A/B
 40  PRINT C
 50  REVERT ZERODIVIDE
 60  D = A/B
 70  ZERODIVIDE THEN 80
 90  E = A/B
100  PRINTE E
1100  STOP
1110  ON ZERODIVIDE GO TO 50
1120  END
```

Los pasos del procesador son:

— Instrucción 30:

Se da la condición de error ZERODIVIDE, pero como la instrucción 1110 permanece, se bifurca hacia la instrucción 50.

— Instrucción 50:

Se anula la instrucción 1110.

— Instrucción 60:

Se da la condición de error ZERODIVIDE, pero la instrucción 70 dice que en caso de ZERODIVIDE en la instrucción anterior, hay que bifurcar en la instrucción 90.

— Instrucción 90:

Se da la condición de error ZERODIVIDE. Como la instrucción 1110 fue cancelada (en la instrucción 50) y la siguiente instrucción no detecta esta condición de error, el programa es cancelado.

EJERCICIOS

1. Un error de ejecución provoca:

   a) Una bifurcación dentro del programa.
   b) La cancelación del programa.
   c) La parada momentánea de la ejecución del programa.

2. Un error de ejecución:

   a) No puede ser detectado.
   b) Es un error debido al mal funcionamiento del terminal.
   c) Activa una condición de error.

3. Las funciones ERM y ERL permiten conocer (respectivamente):

   a) El nombre de la variable que generó el error y el valor que contiene.
   b) La instrucción que lo generó y el código del error.
   c) El código del error y la instrucción que lo generó.

4. Una condición de error:

   a) Permite detectar un error en la ejecución.
   b) Puede codificarse en cualquier parte del programa.
   c) Es una palabra clave opcional.

5. La condición de error EOF detecta:

   a) El final del programa.
   b) El final de un fichero.
   c) El final de un registro.

6. La condición de error ZERODIVIDE:

   a) Sólo puede emplearse en Doble Precisión.
   b) Sólo puede emplearse en Simple Precisión.
   c) Puede emplearse en Simple y Doble División.

7. La condición de error UNDERFLOW detecta:

   a) El valor numérico máximo del ordenador.
   b) El desbordamiento del valor numérico máximo del ordenador.
   c) El intento de dividir por cero.

8. La instrucción ON condición-de-error GO TO etiqueta:

   a) Siempre permanece activa en un programa.
   b) Puede codificarse en cualquier parte del programa.
   c) No es una instrucción ejecutable.

9. La instrucción IF ERR THEN 2000:

   a) Es distinta de la instrucción IF ERR GO TO 2000.
   b) Puede codificarse en cualquier programa.
   c) Detecta el mismo error que la instrucción ON ERR GO TO 190.

10. La instrucción REVERT:

    a) Cancela una condición de error.
    b) Puede codificarse en cualquier parte del programa.
    c) Corrige una condición de error.

11. La condición de error ERR 3:

    a) Detecta un error en el tratamiento del fichero número 3.
    b) Activa el código de error número 3.
    c) Detecta el final del archivo número 3.

12. ¿Qué instrucción está codificada incorrectamente?

    a) 290 IF ERM = 8 THEN 5200.
    b) 290 IF ERM THEN 5200.
    c) 290 IF ERR THEN 5200.

13. Para detectar el intento de dividir por cero, se utiliza la condición de error .....

14. ¿Qué instrucción está codificada correctamente?

    a) 350 REVERT EOF 1.
    b) 350 REVERT EOF.
    c) 350 REVERT ERL.

15. Si se especifica la condición de error EOF y el número de campos numéricos de un registro es menor que el número de variables numéricas de una instrucción GET ó MAT GET, las variables numéricas restantes .....
    .....

7.

PROGRAMAS RESUELTOS EN BASIC

PROBLEMA 1

Planteamiento:

Ordenar de menor a mayor una cantidad de números no superior a cien.

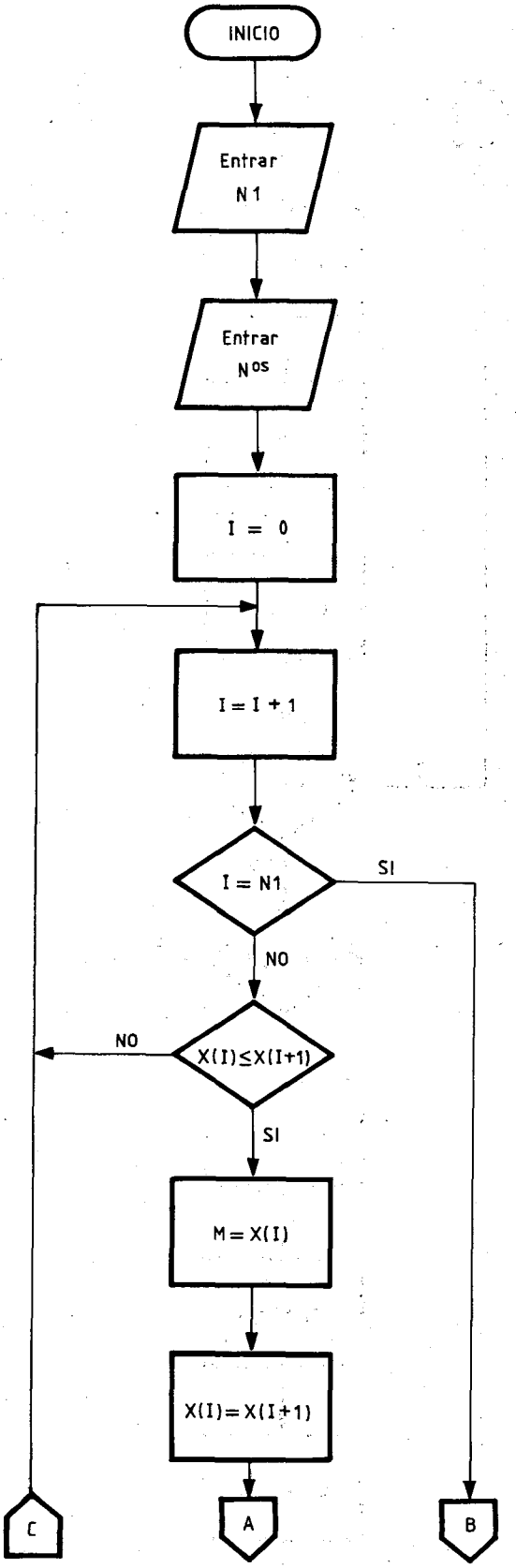
Datos necesarios:

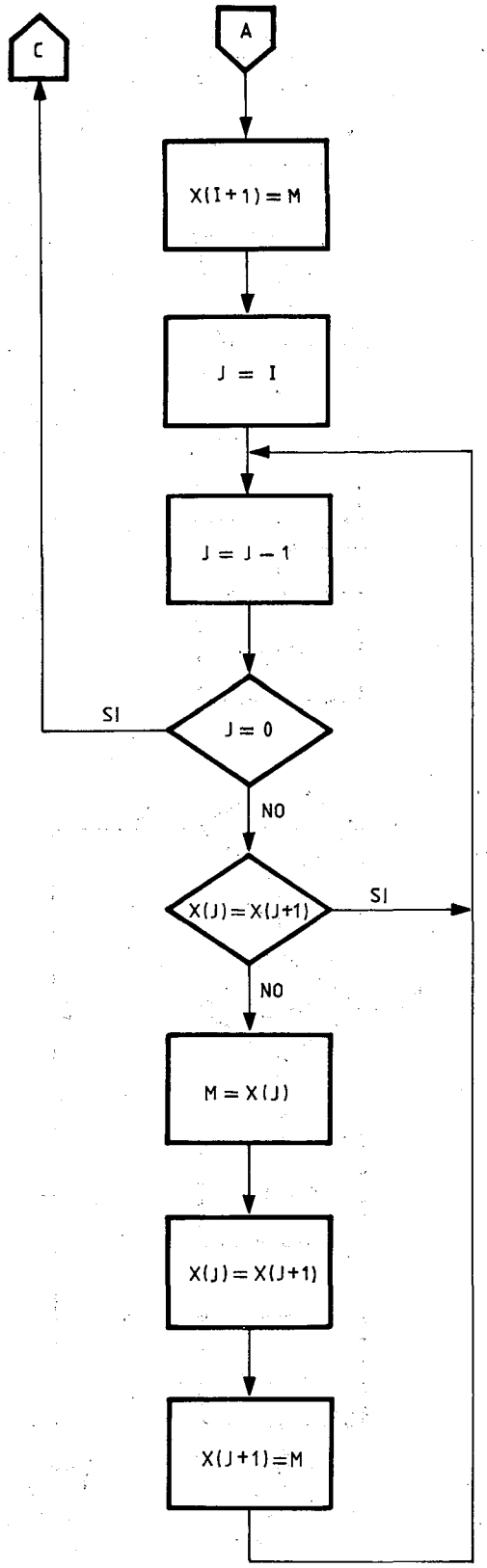
— Cantidad de números.

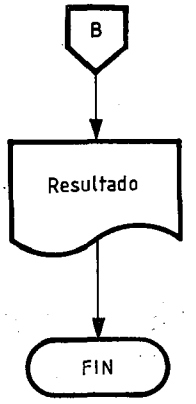
— Cada uno de los números.

Resultado:

Los números ordenados en orden creciente.







Programa

```
 10  DIM X(100)
 20  INPUT N1
 30  FOR N = 1 TO N1
 35  INPUT X(N)
 39  NEXT N
 40  I = O
 50  I = I + 1
 60  IF I = N1 THEN 150
 70  IF X(I) ≤ X(I + 1) THEN 50
 80  M = X(I)
 85  X(I) = X(I + 1)
 89  X(I + 1) = M
 90  J = I
100  J = J – 1
110  IF J = 0 THEN 50
120  IF X(J) ≤ X(J + 1) THEN 100
130  M = X(J)
135  X(J) = X(J + 1)
139  X(J + 1) = M
140  GO TO 100
150  FOR N = 1 TO N1
155  PRINT X(N);
159  NEXT N
160  STOP
170  END
```

PROBLEMA 2

Planteamiento:

¿Cuánto debe cobrar cada empleado de una empresa?

Las horas extras se pagan doble.

Datos necesarios:

— Código del trabajador.

— Precio de la hora.

— Horas trabajadas.

Resultado:

— Nombre del trabajador.

— Importe correspondiente.

Descripción:

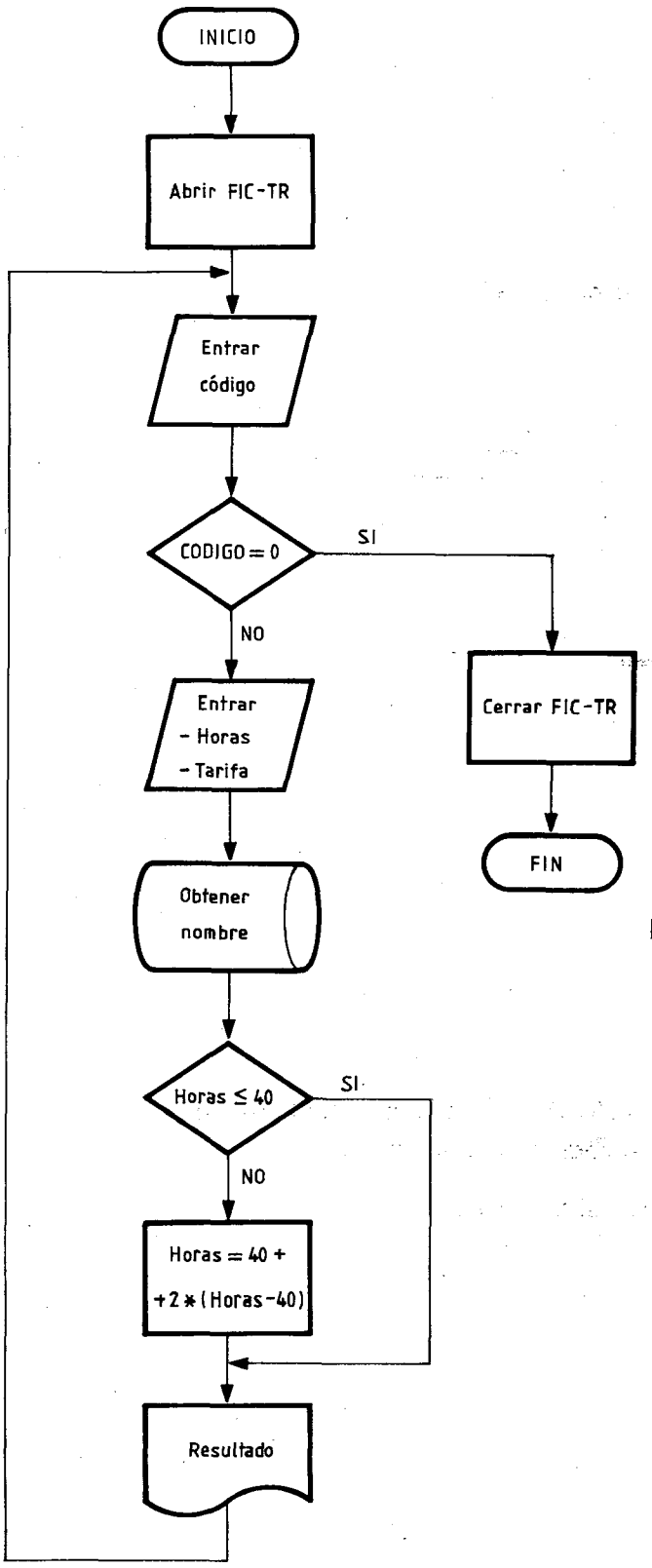
Cada registro del fichero de acceso directo FIC-TR es un trabajador. El número de línea del registro se corresponde con el código del trabajador.

El primer campo del registro es el nombre del trabajador.

Fin del proceso:

Código del trabajador = 0

◄ Organigrama



Programa:

```
 10  OPEN 3, "FIC-TR", INPUT
 20  PRINT "CODIGO";
 30  INPUT C
 40  IF C ≠ 0 THEN 70
 50  CLOSE 3
 60  STOP
 70  PRINT "HORAS, TARIFA";
 80  INPUT H, T
 90  SET 3:C
100  GET 3:N$
110  IF H ≤ 40 THEN 120
110  H = 40 + 2 *(H − 40)
120  PRINT "IMPORTE ="; H * T)
130  GO TO 20
140  END
```

PROBLEMA 3

Planteamiento:

Escribir los n primeros números de la serie de Fibonacci.

Datos necesarios:

Cantidad de números (n).

Resultado:

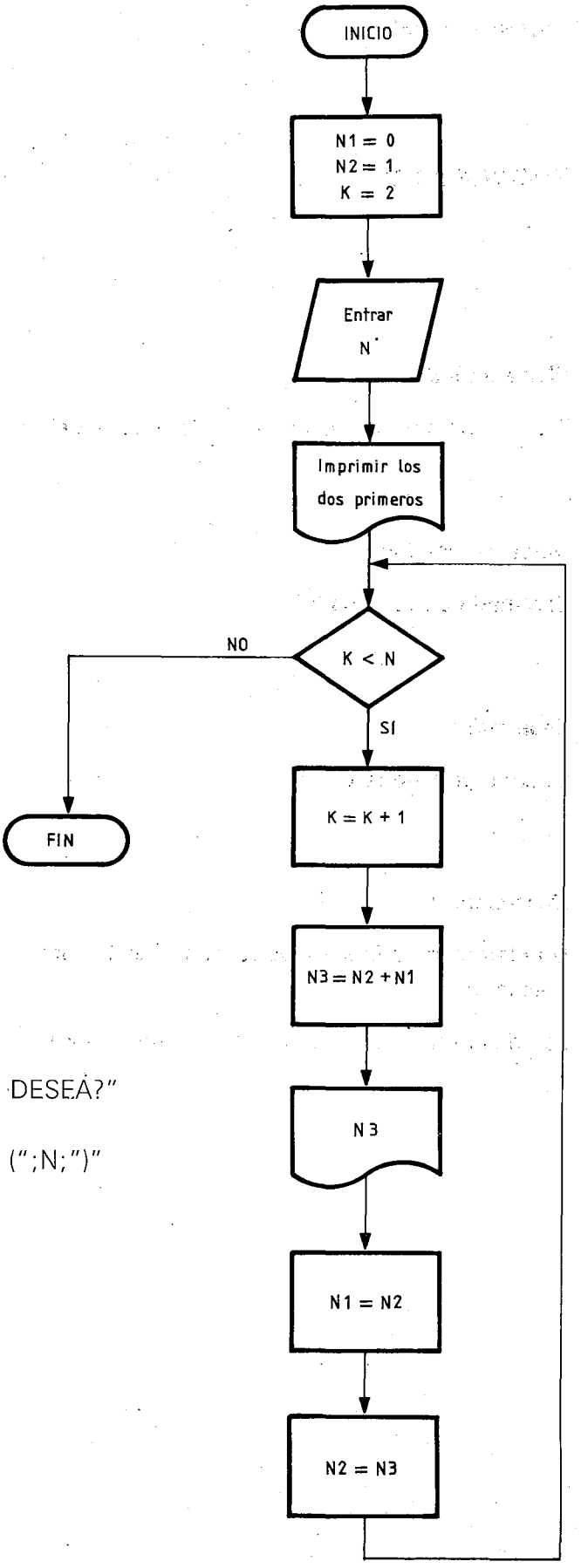
La serie de Fibonacci.

Descripción:

Un elemento de la serie de Fibonacci es la suma de los dos elementos que le preceden.

Los dos primeros elementos son siempre 0 y 1.

Organigrama ►



Programa:

```
 10  K = 2
 20  N1 = 0
 30  N2 = 1
 40  PRINT "CUANTOS NUMEROS DESEA?"
 50  INPUT N
 60  PRINT "SERIE DE FIBONACCI (";N;")"
 70  PRINT "0";"1";
 80  IF K < N THEN 100
 85  PRINT "FIN DE LA SERIE"
 90  STOP
100  K = K + 1
110  N3 = N2 + N1
120  PRINT N3;
130  N1 = N2
140  N2 = N3
150  GO TO 80
160  END
```

PROBLEMA 4

Planteamiento:

Calcular la altura a la que asciende un líquido por un tubo capilar, debido a su tensión superficial, mediante la ley de fusión.

Datos necesarios:

— Radio del tubo capilar (cm.)

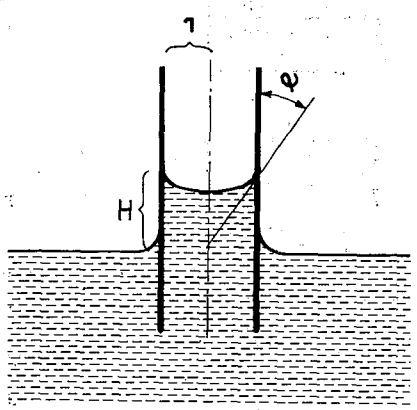
— Masa específica del líquido (gr/cm^3)

— Tensión superficial del líquido ($dinas/cm^2$).

— Angulo de menisco (grados)

Resultados:

Altura en centímetros.



Fórmula:

$$H = \frac{2 \times \sigma \times \cos \varphi}{r \times \rho \times g}$$

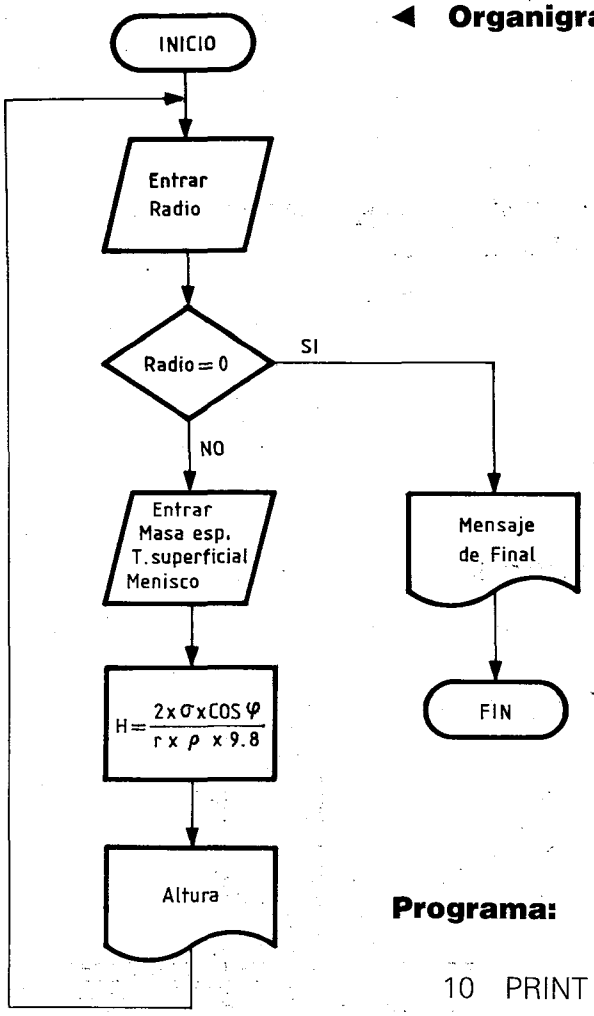
siendo:

H = altura
σ = tensión superficial
φ = ángulo del menisco
r = radio del tubo capilar
ρ = masa específica
g = aceleración de la gravedad (= 9.8)

Fin del proceso:

Radio del tubo capilar = 0.

◄ Organigrama



Programa:

```
10  PRINT "RADIO DEL CAPILAR";
20  INPUT R
30  IF R > 0 THEN 60
40  PRINT "* * FIN * *"
50  STOP
60  PRINT "MASA ESPECIFICA";
70  INPUT M
80  PRINT "TENSION SUPERFICIAL";
90  INPUT T
100 PRINT "ANGULO DEL MENISCO";
110 INPUT A
120 H = 2 * T * COS[RAD(A)]/(R * M * 9.8)
130 PRINT "ALTURA ="; H
140 PRINT
150 GO TO 10
160 END
```

PROBLEMA 5

Planteamiento:

Calcular la potencia eléctrica de una presa hidráulica.

Datos necesarios:

— Caudal de agua (litros/sg.)

— Altura del salto (metros)

— Rendimiento del generador (%)

Resultado:

Potencia en Kilowatios.

Fórmula:

$$P = \frac{C \times \rho \times g \times h \times R \times 0.735}{7500}$$

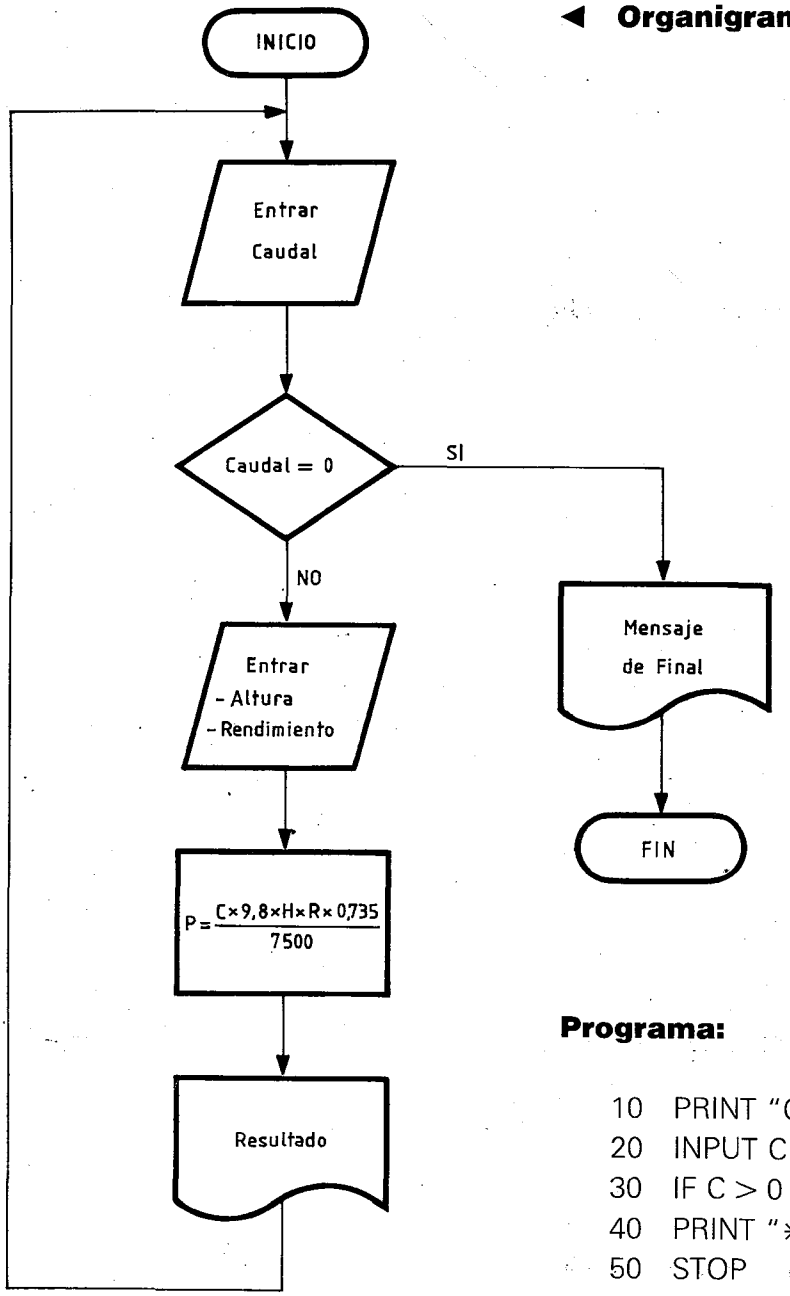
siendo:

P = potencia
C = caudal
ρ = masa específica (masa específica del agua = 1)
g = aceleración de la gravedad (= 9.8)
h = altura del salto
R = rendimiento del generador

Fin del proceso:

Caudal = 0

◄ Organigrama



Programa:

```
 10  PRINT "CAUDAL";
 20  INPUT C
 30  IF C > 0 THEN 60
 40  PRINT "* * FIN * *"
 50  STOP
 60  PRINT "ALTURA";
 70  INPUT H
 80  PRINT "RENDIMIENTO";
 90  INPUT R
100  P = C * 9.8 * H * R * 0.735/7500
110  PRINT "POTENCIA=";P;"KW"
120  GO TO 10
130  END
```

SOLUCIONES DE LOS EJERCICIOS

1. Elementos de BASIC

1. a)
2. Numéricas y alfanuméricas.
3. @, $, #.
4. a)
5. Caracteres limitadores.
6. Las comillas simples y las comillas dobles.
7. b).
8. De relación y aritméticos.
9. Constantes, variables y operadores aritméticos.
10. Pueden introducirse en cualquier orden.
11. Con paréntesis.
12. a).

2. Instrucciones del BASIC

1. b); 2. c); 3. a); 4. b); 5. c); 6. a); 7. b); 8. b); 9. b).
10. Informar al operador del motivo de la interrupción.
11. Ejecución.
12. Incondicionales y condicionales.

3. Instrucciones de ENTRADA/SALIDA

1. b); 2. b); 3. c); 4. a); 5. a): 6. b); 7. b); 8. c).

4. Subprogramas y Funciones

1. RETURN; 2. a); 3. GOSUB; 4. b); 5. Last In First Out (el último que entra es el primero en salir); 6. b); 7. c); 8. a); 9. RETURN; 10. a); 11. c); 12. c).

5. Matrices

1. a); 2. a); 3. c); 4. b); 5. a); 6. c); 7. a); 8. b); 9. a); 10. c);

6. Errores de ejecución

1. b); 2. c); 3. c); 4. a); 5. b); 6. c); 7. b); 8. b); 9. c); 10. a); 11. a); 12. b); 13. ZERODIVIDE; 14. a); 15. Son inicializadas a cero.

Indice General

Contenido 7
Introducción 9

1. ELEMENTOS DEL BASIC

Cuadro resumen 13
1.1. Constantes 14
1.2.1. Constantes numéricas 14
1.1.2. Constantes alfanuméricas 17
1.1.3. Precisión 18
— Simple precisión 19
— Doble precisión 20
1.2. Variables 21
1.2.1. Variables numéricas 21
1.2.2. Variables alfanuméricas 22
1.3. Operadores 22
1.3.1. Operadores de relación 23
1.3.2. Operadores aritméticos 24
1.4. Expresiones aritméticas 25
1.5. Jerarquía de las operaciones en las expresiones aritméticas 25
1.6. Codificación de un programa BASIC 27

Ejercicios 33

2. INSTRUCCIONES DEL BASIC

Cuadro resumen 37
2.1. Asignación 47
2.2. Bifurcaciones 50
2.2.1. Bifurcaciones incondicionales 50
2.2.2. Bifurcaciones condicionales 51
— Bifurcaciones calculadas 51
— Bifurcaciones de relación 54
2.3. Bucles 57
2.4. Interrupciones 61
2.4.1. Interrupciones de compilación 63
2.4.2. Interrupciones de ejecución 63
2.4.3. Interrupciones de espera 64
2.5. Comentarios 67
2.5.1. Comentarios de instrucción 67
2.5.2. Comentarios de proceso 68

Ejercicios 69

3. INSTRUCCIONES DE ENTRADA/SALIDA

Cuadro resumen 73
3.1. E/S por programa 75
3.1.1. Creación de tablas 75
3.1.2. Lectura de una tabla 77
3.1.3. Reinicio de tablas 80
3.2. E/S por terminal 83
3.2.1. Entrada por terminal 84
3.2.2. Salida por terminal 85
— Impresión 85
— Espaciado de campos 87
— Espaciado de líneas 88
— Definición de formato 90
— Impresión con formato 92
— Impresión comercial 94
3.3. Ficheros 97
3.3.1. Definición de ficheros 97
— Definición por programa 97
— Definición por terminal 101

3.3.2. Activación de un fichero 102
3.3.3. Desactivación de un fichero 104
3.3.4. Acceso secuencial 105
— Lectura en acceso secuencial 106
— Grabación en acceso secuencial 107
— Restauración de ficheros de acceso secuencial 112
3.3.5. Acceso directo 114
— Pointer 116
— Lectura en acceso directo 117
— Grabación en acceso directo 118

Ejercicios 119

4. SUBPROGRAMAS Y FUNCIONES

Cuadro resumen 123
4.1. Subprogramas 124
4.1.1. Llamada incondicional 125
4.1.2. Llamada condicional 126
4.1.3. Retorno al programa de llamada 127
4.2. Funciones 130
4.2.1. Funciones intrínsecas 130
4.2.2. Funciones del usuario 133
— Funciones del usuario numéricas 133
— Funciones alfanuméricas 135
4.2.3. Series 136
— Tratamiento de las series 137
— Valor numérico 138
4.2.4. Funciones de varias líneas 139

Ejercicios 143

5. MATRICES

Cuadro resumen 147
5.1. Definición de una matriz 150
5.2. Asignación de una matriz 151
5.3. Suma de matrices 153
5.4. Resta de matrices 154
5.5. Producto de matrices 155

5.6. Producto de una matriz por un escalar 157
5.7. Matriz traspuesta 158
5.8. Matriz inversa 159
5.9. Inicialización 160
5.10. Almacenamiento en la M.C. 161
5.11. Matrices alfanuméricas 162
5.12. Lectura por programa 162
5.13. Lectura por terminal 163
5.14. Impresión de matrices 164
5.15. Lectura de un fichero 165
5.16. Grabación de un fichero 166

Ejercicios 167

6. ERRORES DE EJECUCION

Cuadro resumen 171
6.1. Cancelación de un programa 173
6.2. Tratamiento de un fichero 174
6.3. Final de fichero 175
6.4. Cálculo aritmético 176
6.5. Cancelación de una condición de error 178

Ejercicios 179

7. PROGRAMAS RESUELTOS EN BASIC

1. Ordenación de números en orden decreciente 184
2. Cálculo del sueldo de un empleado 186
3. Serie de Fibonacci 188
4. Cálculo de la capilaridad de un líquido 190
5. Cálculo de la potencia eléctrica de una presa hidráulica 192

Soluciones de los ejercicios 195

En este volumen se hace una exposición clara y explícita de los fundamentos del lenguaje BASIC, así como una extensa exposición de los posibles casos de aplicación en diversos campos del Proceso de datos.

Todos los capítulos poseen un determinado número de ejercicios al final, que ayudan al lector a una autocomprobación en el estudio.

Se ha hecho gran hincapié en la correspondencia entre la descripción teórica y los casos concretos, a modo de ejemplo o ejercicios, como aplicación práctica. Dichos ejercicios tienen un gran porcentaje de aplicación en la realidad, a la vez que complementan el perfecto estudio de la materia tratada.

El dibujo de la cubierta es cortesía de GENERAL DE COMPUTADORES, S.A.