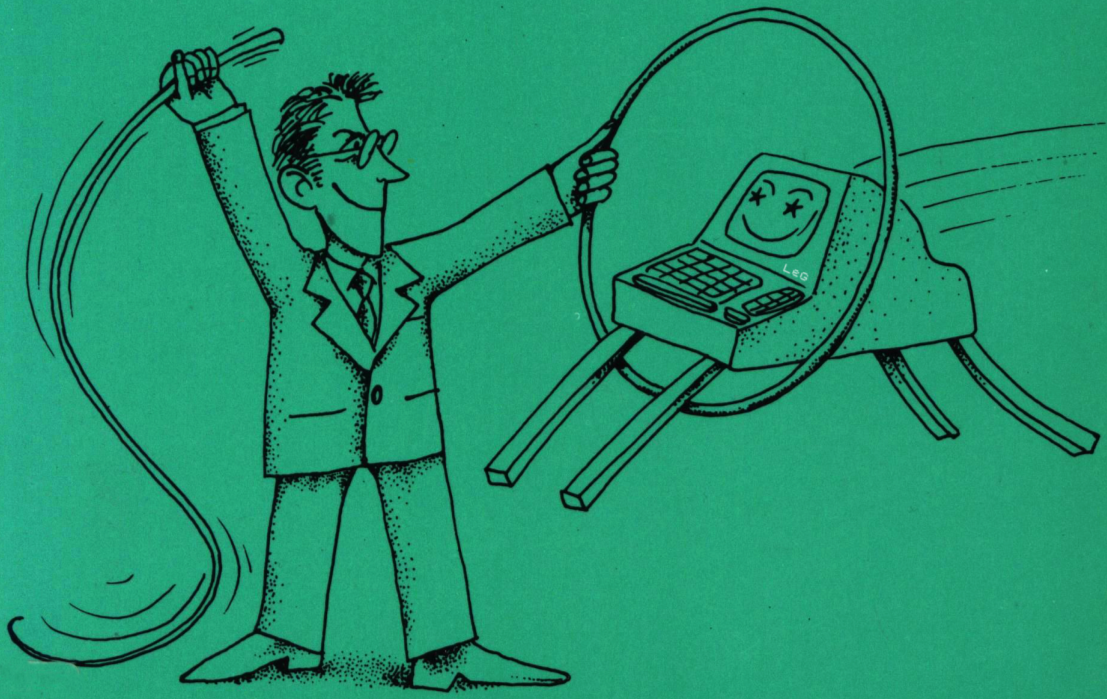
RICARDO AGUADO–MUÑOZ
AGUSTIN BLANCO
JAVIER ZABALA
RICARDO ZAMARREÑO

BASIC BASICO

CURSO DE PROGRAMACION



SEPTIMA EDICION

De los mismos autores:

PROGRAMAS COMENTADOS DE BASIC BASICO

Ciento ocho problemas de programación explicados y comentados.

Cincuenta de ellos responden a los ejercicios propuestos en el libro BASIC BASICO CURSO DE PROGRAMACION.

Contiene:

Juegos.

Simulaciones.

Gráficos y color.

Cálculo numérico.

BASIC
BASICO

CURSO DE PROGRAMACION

RICARDO AGUADO–MUÑOZ
AGUSTIN BLANCO
JAVIER ZABALA
RICARDO ZAMARREÑO

AUTORES EDITORES:

RICARDO AGUADO-MUÑOZ
General Cabrera, 23
MADRID-20
Tfno: 91-270 91 50

AGUSTIN BLANCO
Avda. del Generalísimo, 26 - 7º D.
ALCORCON (MADRID)
Tfno: 91-611 32 43

JAVIER ZABALA
Vitrubio, 3
MADRID-6
Tfno: 91-262 49 25

RICARDO ZAMARREÑO
Bristol, 10 - 9º izda.
MADRID-28
Tfno: 91-256 23 67

DISTRIBUYE:

GRUPO DISTRIBUIDOR EDITORIAL, S. A.
Don Ramón de la Cruz, 67
MADRID-1
Tfno: 91 - 401 12 00

Editan:

I.S.B.N.: 84-300-7294-2
Depósito legal: M. 6.663-1984
Imprime: Hijos de E. Minuesa, S. L. Ronda de Toledo, 24. Madrid-5

INDICE

Página

RECONOCIMIENTOS

A nuestro amigo y director Ricardo Sánchez Ortíz de Urbina, sin cuyo decidido apoyo no hubiera sido posible escribir este libro como fruto de una experiencia real.

A nuestro amigo y compañero Saltés, a cuya generosa genialidad se deben la portada y las ilustraciones de los capítulos.

A nuestros amigos y compañeros del Seminario de Matemáticas, cuyas aportaciones críticas han contribuido a mejorar el original.

Los autores

Presentación

El billete de avión o de tren que reservamos con días de antelación para un viaje soñado; el sueldo con el que procuramos llegar a fin de mes, calculado en una nómina; las amenazas de Hacienda con sus medios, para que seamos veraces a la hora de declarar nuestros impuestos; la precisión del disparo de un cohete que hunde un barco; el acceso a un banco de datos, gigantesca biblioteca, que podré consultar sin moverme de mi casa, apretando las teclas de un microordenador doméstico, que me ha costado poco más que un televisor en color; la música de "Tornasol" que Luis de Pablo ha escrito en París; la perspectiva axonométrica de un edificio histórico que empieza a girar en la pantalla ante la mirada ávida del estudiante de arquitectura; la contabilidad y el recuento de miles de artículos del pequeño negocio de una ferretería; la exploración de un posible tumor cerebral en un gran hospital; la jurisprudencia acumulada, cuyos precedentes necesita el abogado para defender a un cliente... Estas y otras miles de aplicaciones de los ordenadores son ejemplos que se van multiplicando cada día que pasa.

Y no son sólo los míticos cerebros electrónicos, mastodontes electromecánicos o electrónicos que costaban millones de dólares, los únicos aparatos capaces de solucionar tales problemas. Ahora hay microordenadores que están al alcance de todos los bolsillos. Utilizan programas que se venden o alquilan con la facilidad con que compramos una cassette de la melodía que nos apetece. Estos pequeños ordenadores están desmitificando la Informática, haciéndola asequible a cualquier persona.

Siempre ha habido resistencia al cambio que supone cualquier progreso relacionado con los medios de producción. Recordemos la resistencia que ofrecieron los obreros frente a la introducción de las máquinas, al comienzo de la Revolución Industrial del siglo XIX. Sin embargo, los avances tecnológicos que no ponen en peligro el puesto de trabajo del hombre han sido fácilmente asumidos; a veces con verdadero fervor, aun a sabiendas de que se corrían otros peligros no menores. Pensemos en el cine, la radio y la televisión.

La manera de incorporar racionalmente los riesgos del progreso es conocer las nuevas técnicas. En nuestro caso se trata de aprender un nuevo lenguaje. Hace unos años los expertos en Informática estaban obligados a aprender un lenguaje muy próximo al lenguaje-máquina, el lenguaje utilizado por el propio ordenador. La tarea era abrumadora: cada ordenador estaba construido por un fabricante, tenía un lenguaje propio que no compartía con los demás ordenadores, y que además resultaba minucioso y complejo, muy alejado del lenguaje natural. Ahora existen ***lenguajes de programación*** *de alto nivel que nos permiten comunicarnos con el ordenador, pero que están más próximos a nuestro* ***lenguaje natural.*** *Al ordenador le comunicamos un* ***programa*** *escrito en un lenguaje de programación y el ordenador lo traduce al lenguaje-máquina propio de su estructura.*

De entre todos los lenguajes de programación hay uno muy sencillo y relativamente potente a la vez. Está muy extendido, lo utilizan la totalidad de los microordenadores, y es el internacionalmente más aceptado para introducir a la Informática a los estudiantes de Enseñanza Secundaria. Es el lenguaje BASIC, que en versiones más modernas se extiende cada vez más al mundo de la Empresa. Diseñado hace veinte años en Darmouth por Kemeny y Kurtz, el lenguaje BASIC es un lenguaje de programación que nos permite acercarnos cómodamente a este mundo misterioso. Permitirá que hablemos con el ordenador.

El libro que el lector tiene en las manos es un ***Curso de Programación en lenguaje BASIC.*** *Siguiendo sus capítulos, haciendo sus ejercicios cuidadosamente secuenciados, el lector aprenderá el lenguaje BASIC y además aprenderá a* ***programar,*** *a diseñar programas, algoritmos operativos para analizar problemas, tratarlos y reducirlos a una secuencia de instrucciones que la máquina realizará a velocidades increibles, dándonos la solución buscada. Es pues un curso de lenguaje BASIC y un curso de PROGRAMACION.*

Es además un libro que ha brotado de la realidad, de la experiencia. Sus autores son profesionales de la Enseñanza que han llevado a cabo una experiencia piloto: enseñar Informática a alumnos de Bachillerato. Trabajando en equipo en un Centro Oficial, el Instituto Piloto de Bachillerato "Cardenal Herrera Oria", han conseguido elaborar un método asequible y válido para aprender a programar en el lenguaje BASIC.

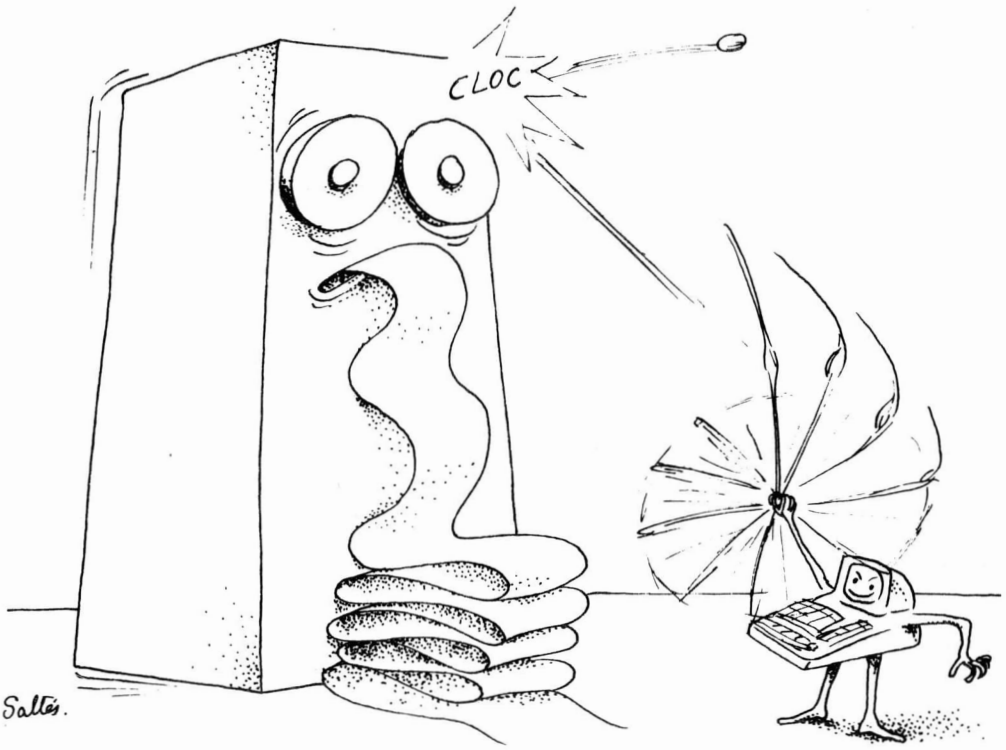
El libro está dirigido a los alumnos de Enseñanza Media, a los profesionales y científicos cuya formación tiene una laguna que les bloquea el acceso a numerosas investigaciones actuales en todos los campos de la actividad humana, y a todos los que quieran familiarizarse con un lenguaje que en un futuro muy próximo resultará imprescindible, sea cual sea la profesión del lector.

Hay pocos textos escritos en castellano con esta pretensión y ninguno, creo, que logre su objetivo con la claridad pedagógica y la altura técnica con las que éste lo hace. El libro que el lector está hojeando no viene a llenar una laguna, como se suele decir, sino a irrigar un desierto de desconfianza y recelo que impide a nuestra sociedad aceptar, aun deseándolo, el reto del futuro.

Ricardo Sánchez Ortíz de Urbina
Director del Instituto Piloto
"Cardenal Herrera Oria" de Madrid

0

LOS ORDENADORES



0.1 HISTORIA

Podemos dividir los ordenadores en dos tipos: digitales y analógicos, según que traten información que varía de forma discreta o de forma continua.

Aquí vamos a referirnos exclusivamente a ordenadores digitales. Aunque esto parece una restricción importante, hay que recordar que la naturaleza a nivel microscópico varía de forma discontinua, y si muchas magnitudes las detectamos como continuas es debido a las limitaciones de nuestros aparatos de observación.

El tratamiento de las magnitudes discretas mediante instrumentos es muy antiguo. El primer aparato conocido es el ábaco, usado en el mundo mediterráneo, en la China de Confucio y en las civilizaciones precolombinas. Prueba de su eficacia para el cálculo es el hecho de que en formas más evolucionadas se sigue utilizando en muchos países de Oriente.

Pero tenían que pasar bastantes siglos hasta que aparecieran los primeros ingenios mecánicos para el tratamiento de cantidades discretas.

Una de las dificultades que existían para realizar cálculos de un modo sencillo era debida a los sistemas de numeración empleados. Un hallazgo trascendental fue la invención por matemáticos hindúes del sistema de representación decimal que llegó a Europa a través de los árabes.

A partir del Renacimiento, con el desarrollo del comercio y la banca, la navegación y la astronomía, se creó un ambiente favorable a las innovaciones tecnológicas. De esta época datan los primeros relojes y los muñecos de los relojes de Iglesia, que realizan siempre los mismos gestos. Pueden ser considerados máquinas de programa interior único.

En el siglo XVII, Pascal construye la primera calculadora mecánica para sumar. Consistía en una serie de ruedas dentadas numeradas de cero a nueve. Cuando en una rueda se pasaba del nueve al cero, después de dar una vuelta completa, se producía el giro de una posición en la rueda situada a su izquierda; era igualmente, una máquina con programa interior único. Leibniz, treinta años después, ideó otra máquina que realizaba las cuatro operaciones aritméticas. No obstante, estas máquinas no pueden considerarse automáticas porque precisan la intervención constante del operador para realizar las maniobras que exige cada operación.

El siguiente paso, de gran importancia en la mecanización del cálculo, fue dado por Charles Babbage (1792–1871) con su máquina de diferencias, ideada en principio para mejorar las tablas de logaritmos de su época. El proyecto fue abandonado por otro mucho más ambicioso, la máquina analítica, capaz de realizar cualquier operación matemática, sin precisar intervención humana durante el proceso de cálculo. Esta máquina tampoco llegó a terminarse por su complejidad mecánica, muy superior a las posibilidades de la época.

Si bien las ideas de Babbage no llegaron a materializarse de forma definitiva, su contribución es decisiva, ya que los ordenadores actuales responden a un esquema análogo al de la máquina analítica.

En su diseño, la máquina constaba de cinco unidades básicas: 1) *Unidad de entrada,* para introducir datos e instrucciones; 2) *Memoria,* donde se almacenaban datos y resultados intermedios; 3) *Unidad de control,* para regular la secuencia de ejecución de las operaciones; 4) *Unidad Aritmético-Lógica,* que efectúa las operaciones; 5) *Unidad de salida,* encargada de comunicar al exterior los resultados.

Babbage adoptó el sistema de las tarjetas perforadas de Jacquard (mecánico francés, 1752–1834, que inventó a comienzos del siglo XIX, un telar automático, en el que se introducían programas mediante tarjetas perforadas), para introducir en la máquina, las instrucciones del programa y los datos del problema.

La máquina de Babbage y el telar de Jacquard se diferencian de los ingenios anteriores en que son autómatas con programa exterior.

Las calculadoras mecánicas y la máquina de Babbage, constituyen la prehistoria de los instrumentos de cálculo. A lo largo del siglo XIX el gran desarrollo industrial provoca el consecuente aumento de la complejidad organizativa de la sociedad. Surge el problema de tratar grandes masas de información.

Herman Hollerith (1860–1929), funcionario de la Oficina de Censos de los Estados Unidos, ante la necesidad de mecanizar los datos del censo de 1890, y conociendo el sistema de tarjetas perforadas de Jacquard tuvo la idea de representar la respuesta SI a una pregunta, mediante una perforación en un lugar de la tarjeta y la respuesta NO sin la perforación. Había nacido la codificación binaria de la información, perfectamente apta para ser representada por soportes físicos con sólo dos estados.

Todos los logros obtenidos hasta este momento, finales del siglo XIX y comienzos del XX, unidos a las condiciones sociales existentes en el primer tercio del siglo XX, (concentraciones financieras, mayor complejidad de los aparatos administrativos estatales y empresariales, necesidad de tecnología militar más avanzada, etc.) hacen posible el nacimiento del ordenador. Con él entramos en la Historia Antigua de los aparatos de cálculo.

Howard Aiken (1900–1973), profesor de la Universidad de Harvard, inspirándose en las ideas de Babbage, diseñó y terminó en 1944 el primer ordenador de la Historia, el MARK I, a base de elementos electromecánicos. Constituía la primera realización práctica de las ideas de Babbage.

La máquina de Aiken era muy lenta (invertía en sumar dos números dos

décimas de segundo, y en una multiplicación de diez dígitos tres segundos, entre otras razones, por la inercia de sus partes móviles.

Un gran avance desde el punto de vista tecnológico, supuso la construcción entre 1943 y 1946 del ENIAC, primera calculadora electrónica. Todas las operaciones, excepto las de entrada y salida, se realizaban mediante circuitos electrónicos a base de válvulas de vacío. Disponía de 18.000 tubos de vacío y cada vez que se ponía en funcionamiento, las luces de la ciudad de Filadelfia experimentaban un brusco descenso. En cuanto a la velocidad era muchísimo más rápido que el MARK I. Empleaba 0,003 segundos en realizar una multiplicación de dos números de diez dígitos; pero presentaba un gran inconveniente porque cada problema exigía una configuración de los circuitos.

En 1946 J. Von Neuman (1903–1957) enunció los principios de funcionamiento de un ordenador en el que no hubiese que modificar los circuitos internos para cada programa. En las máquinas anteriores la memoria almacenaba los datos y resultados intermedios. La idea de Von Neumann era que la memoria de la calculadora almacenara además el programa. Esta posibilidad daba a la máquina de Von Neumann un carácter universal, hasta el punto de que algunos autores consideran el año 1946 como año de nacimiento de la Informática.

El primer ordenador comercial construido con estas ideas apareció en el mercado en 1951; se trataba del UNIVAC I. Posteriormente, también en la década de los cincuenta, se construyeron las series 600 y 700 de I.B.M. Los ordenadores de esta década forman lo que se suele llamar la primera generación: utilizan válvulas de vacío y pueden ejecutar unas mil instrucciones por segundo. Estaban orientados a los campos científico y militar.

Con el descubrimiento del transistor en el año 1956 y su incorporación a los ordenadores en 1960, sustituyendo a las válvulas de vacío, comienza la segunda generación (1960–1965). El transistor por ser más pequeño, más barato y de fácil producción en grandes cantidades, permite construir ordenadores más pequeños, más potentes, más económicos y veloces (10^6 instrucciones por segundo). Se inicia así la producción en serie. Los ordenadores se impusieron rápidamente en el mundo de los negocios.

Pronto se vió que más interesante que manipular transistores individuales conectados entre sí, era reunirlos en una sola pastilla de silicio, mediante técnicas especiales. Así surgían los circuitos integrados, y la tercera generación en el año 1965. Con estos circuitos se ha conseguido mayor velocidad de cálculo y mayor potencia. Al ampliarse de modo espectacular las aplicaciones de los or-

denadores, se ha concedido gran importancia al desarrollo de lenguajes, para facilitar cada vez más el uso de estos aparatos.

Ejemplares típicos de esta generación son las series 360 y 370 de I.B.M.

Posteriormente la noción de generación se ha desvanecido un poco. Los ordenadores actuales se construyen a base de circuitos integrados con una gran densidad de componentes (hasta más de 100.000 transistores en un solo chip). Se habla a veces de la cuarta generación.

Las técnicas de integración han alcanzado tal desarrollo que permitieron comercializar en 1970 el primer *microprocesador,* que consiste en esencia en la unidad central del ordenador construida sobre un circuito integrado. Nacen los microordenadores, cuyo bajo precio, bajo consumo y gran velocidad de cálculo, han extendido su utilización de un modo inimaginable hace quince años.

0.2 ESTRUCTURA DE UN ORDENADOR

Todo ordenador consta de dos partes fundamentales: la *Unidad Central de Proceso* que designaremos por U.C.P. (en inglés C.P.U.), en la que se acumula la información y se procede a su tratamiento, y las unidades periféricas o, simplemente, periféricos (de entrada y de salida) que permiten la comunicación del ordenador con el exterior.

0.2.1 UNIDAD CENTRAL DE PROCESO

Se considera dividida en tres unidades:

a) *Memoria central*

b) *Unidad aritmético-lógica,* designada por U.A.L.

c) *Unidad de control*

Podemos representar el ordenador por el esquema elemental de la Fig. 0.1.

En la Memoria se almacenan los datos y las instrucciones que forman el programa, así como resultados intermedios y finales del problema.

La Unidad de Control extrae de la Memoria las instrucciones, las descodifica e interpreta, enviando a la U.A.L. las órdenes oportunas para que ejecute las operaciones indicadas.

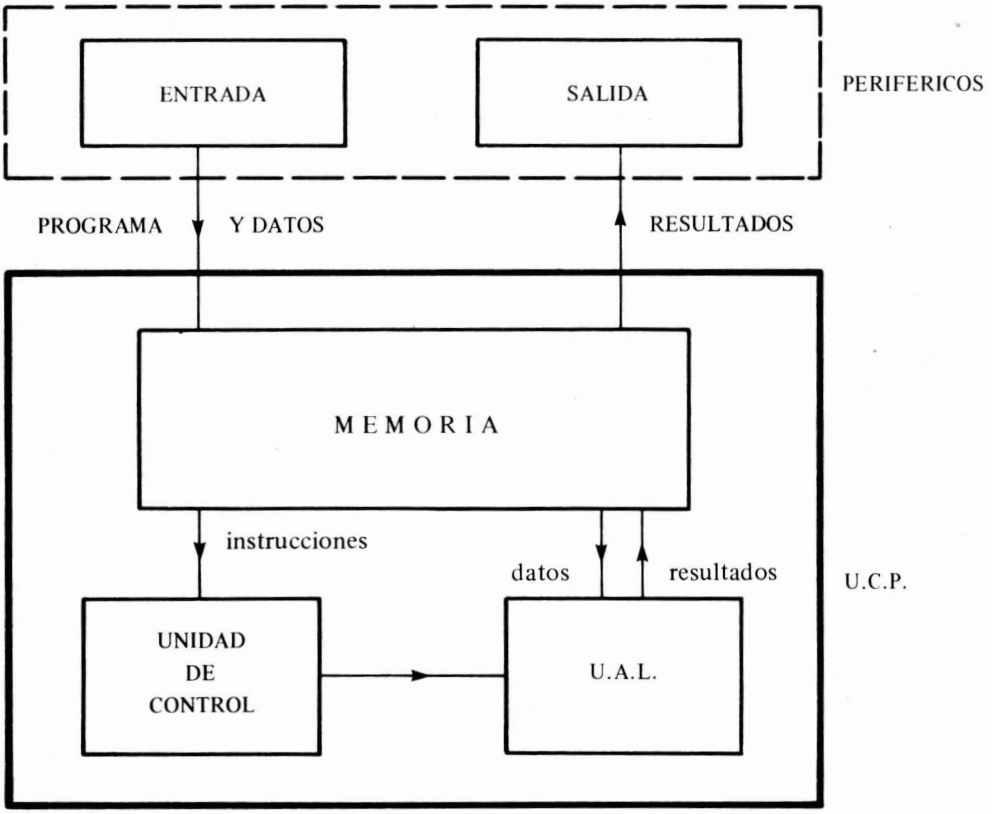


Fig. 0.1

En la U.A.L. se realizan los cálculos y comparaciones, para lo cual se reciben de la Memoria los datos necesarios para efectuar con ellos las operaciones indicadas por la Unidad de Control. Por último se transfieren a la Memoria los resultados de dichas operaciones.

Vamos a dedicar un poco más de atención a la Memoria. Podemos considerar la Memoria como un casillero de correos formado por un conjunto de casillas o celdas elementales. Cada casilla tiene un número que la identifica y que es su *dirección.* El contenido de cada celda puede ser un dato o una instrucción.

En el dibujo, la Memoria dispone de 24 celdas; y en la celda 18 está contenido el número 1324.

Para el trasiego de la información entre la Memoria y la Unidad de Control y U.A.L. aquélla dispone de una serie de órganos o registros auxiliares.

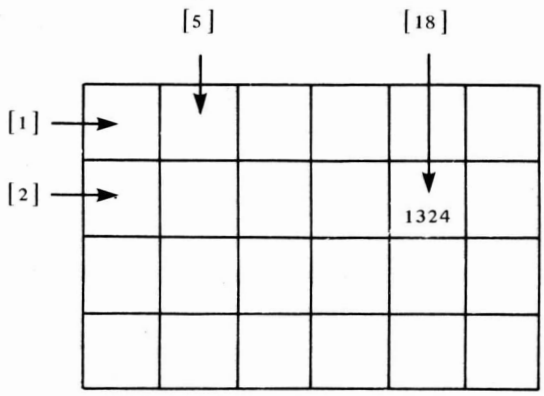


Fig. 0.2

Una propiedad importante de la Memoria es que cuando una información se registra en una celda, desaparece la información que contenía anteriormente dicha celda. Por el contrario, la información permanece cuando la envía a las otras unidades. Manda un duplicado de su contenido.

Las Memorias se caracterizan por una serie de propiedades cualitativas y cuantitativas como son:

a) *Tamaño de la celda*

Hemos dicho anteriormente que el ordenador trata la información codificada en forma binaria, que podemos representar por los números 0 y 1, a los que se denomina bits. El bit es la unidad elemental de información y toma uno de esos dos valores. Surge la siguiente pregunta: cada celda de Memoria ¿cuántos bits contiene? Los tipos más corrientes de celda contienen 4, 6, 8, 12, 16, 24 y 32 bits. El número de bits de cada celda se llama tamaño de celda.

En las celdas de 8 bits, (se llaman octetos y bytes) por ejemplo, como cada bit puede tomar sólo dos valores (0 ó 1), puede haber $2^8 = 256$ estados diferentes,

desde 00000000 hasta 11111111

Con esas 256 posibilidades se pueden almacenar 256 caracteres: las letras del alfabeto, los diez dígitos decimales, caracteres gráficos, etc.

b) *Capacidad de la memoria*

Es el número de casillas que contiene. Se suele expresar en múltiplos de

$K = 2^{10} = 1024$. Así, una Memoria de 32 K bytes tiene 32 x 1024 = 32.768 posiciones o celdas, aunque por comodidad K se toma como 1000 y hablamos de 32.000 direcciones.

c) Tiempo de acceso a la información

Es el tiempo transcurrido entre el instante en que se ordena acceder a una posición de Memoria para obtener una información, y el instante en que la información está disponible. Los tiempos se miden en microsegundos ($1\,\mu s = 10^{-6}$ seg.) y en nanosegundos (1 ns $= 10^{-9}$ seg.). Si el tiempo de acceso no depende más que de la dirección, se dice que es de *acceso libre, directo* o *aleatorio,* y si es necesario leer todas las direcciones anteriores a ella, se habla de *acceso secuencial.*

En un ordenador sería deseable disponer de una Memoria Central muy grande y rápida. Como esto resulta complejo y costoso, se ha optado por disponer de una *Memoria Central,* rápida y con capacidad de almacenamiento relativamente limitada y una *Memoria Auxiliar,* de gran capacidad, y tiempo de acceso grande. Memorias Auxiliares son las cintas, tambores y discos magnéticos.

0.2.2 SOPORTES DE INFORMACION. UNIDADES PERIFERICAS

La Unidad Central de Proceso tiene que comunicar con el exterior, tanto para recibir información como para suministrarla. Ya dijimos que los órganos encargados de esta misión se denominan unidades periféricas o simplemente periféricos. La *Unidad de Entrada* debe leer la información en un soporte adecuado; los soportes habituales que nosotros utilizamos para intercambiar información como libros, discos y otros, en general no sirven para nuestra relación con el ordenador, por lo que se han ideado diversos soportes y los correspondientes periféricos especializados en cada tipo de soporte. Los soportes más utilizados son:

Tarjetas perforadas
Cintas perforadas
Cintas magnéticas
Tambor magnético
Disco magnético
Diskette o Floppy disk

Los principales tipos de Periféricos son:

a) *Lectora-perforadora de tarjetas*

El soporte más antiguo es la tarjeta perforada. El modelo habitual es una cartulina de 187 x 82,5 mm que consta de 80 columnas y 12 filas. Un carácter es representado mediante una determinada combinación de perforaciones situadas en una columna, según el código de Hollerith.

El periférico de entrada correspondiente a este soporte es la lectora de tarjetas, en la que por procedimientos mecánicos u ópticos, se detecta la existencia o no de agujeros. La velocidad de las lectoras oscila entre 200 y 2.000 tarjetas por minuto.

Si el ordenador comunica los resultados mediante tarjetas, el periférico de salida es la perforadora de tarjetas. La velocidad de perforación es menor que la de lectura.

b) *Lectora-perforadora de cintas*

El soporte consiste en una cinta continua en la que se perforan agujeros, de forma que cada carácter viene representado por una combinación de perforaciones en sentido vertical.

Los periféricos correspondientes son la lectora de cintas y la perforadora de cintas.

c) *Lector-grabador de cinta magnética*

La cinta magnética es un soporte similar a la cinta de un magnetófono.

La idea básica es la misma que la de la cinta de papel, pero en lugar de perforaciones hay puntos magnetizados. Del mismo modo que en la cinta, cada carácter está expresado por un grupo de señales magnéticas en el sentido transversal de la cinta.

La capacidad de la cinta contenida en un carrete de 20 cm de diámetro y 2 ó 3 de anchura es de varios millones de caracteres.

El periférico lector-grabador consiste en uno o varios pares de cabezales, uno de lectura y otro de grabación. La cabeza de lectura tiene la ventaja de ser unas 50 veces más rápida que el lector de tarjetas.

d) *Tambor magnético. Lectora-grabadora de tambor magnético*

La cinta magnética es un soporte de acceso secuencial; para acceder a un dato hay que leer todos los anteriores a él. Para subsanar este inconveniente se creó el tambor magnético, que consiste en un cilindro con una capa de sustancia magnética. La información está contenida en pistas circulares paralelas de su superficie. El tambor gira continuamente sobre su eje, y una serie de cabezas de lectura y grabación, una para cada pista, explora dichas pistas, consiguiéndose una mayor velocidad de lectura.

e) *Lectora-grabadora de discos magnéticos*

El tambor magnético almacena menos información que la cinta, por lo que se ideó el disco magnético. La información se almacena en circunferencias concéntricas llamadas pistas, situadas en las dos caras del disco.

Generalmente se utilizan paquetes de discos de cinco o más unidades. Cada disco lleva su correspondiente cabeza lectora-grabadora, que puede situarse mediante un brazo sobre la pista deseada.

El disco resuelve el problema de la capacidad de almacenamiento del tambor, con la contrapartida de una menor velocidad de lectura.

Una modalidad del disco magnético es el diskette o floppy disk, de un tamaño aproximado al de un microsurco de 45 r.p.m.

Los tambores y discos admiten acceso directo y secuencial. Estas unidades magnéticas pueden funcionar también como memorias auxiliares.

f) *Impresora*

Es uno de los órganos más comunes para la salida de resultados de un ordenador. Similares a máquinas de escribir, pero con carro fijo y dispositivo portacaracteres móvil, que puede adoptar forma de barra, tambor, etc.

La velocidad de impresión oscila entre 200 y 300 líneas por minuto (y cada línea de 40 a 130 caracteres).

g) *Consola*

Este periférico facilita la comunicación entre el ordenador y el operador. El modelo más corriente dispone de un teclado y una impresora o un teclado

y una pantalla de rayos catódicos. Los caracteres tecleados o enviados por el ordenador aparecen sobre la pantalla. Por su comodidad la pantalla de T.V. está cada vez más extendida.

Por último, citemos las lectoras de marcas ópticas, donde la codificación se hace a base de la presencia o ausencia de marcas oscuras, y las lectoras ópticas de caracteres que pueden leer documentos, siempre que estén escritos con caracteres producidos por impresoras especiales.

0.3 LENGUAJES DE PROGRAMACION

El ordenador es una máquina que sólo puede realizar operaciones muy simples (sumar, comparar, etc.), pero a gran velocidad. Para que el ordenador ejecute los cálculos más o menos complicados de un problema, es necesario darle una detallada secuencia de instrucciones a seguir. El conjunto de estas instrucciones forma el programa.

En los primeros tiempos de la informática, había que almacenar el programa en la memoria utilizando el único lenguaje que entiende el ordenador: el lenguaje máquina, en el cual cada instrucción es una ristra de ceros y unos.

Veamos, por ejemplo, cómo podría ser representada la instrucción de sumar dos números en este lenguaje de máquina.

Si dividimos, imaginariamente, la celda de memoria en cuatro partes iguales, de izquierda a derecha, cada grupo de bits podría presentar:

- Código de la operación (en nuestro caso, código de la suma).
- Dirección de la celda de memoria en la que se encuentra el primer sumando.
- Dirección de la celda de memoria en la que se encuentra el segundo sumando.
- Dirección de la celda de memoria en la que se almacenará el resultado.

0101	0111	1001	1101

Una celda cargada con la anterior información nos dice que se suma (el código de la suma nos lo da el grupo 0101) el contenido de la celda de direc-

ción 0111, con el de la celda 1001 y que el resultado se almacena en la celda de dirección 1101. Por comodidad se puede representar en el sistema decimal

5	7	9	13

Esta es una instrucción de tres direcciones. También se podría representar mediante una sola dirección, lo que supone la existencia de un registro especial llamado acumulador.

5	134

En este caso la instrucción ordena sumar al contenido del acumulador el contenido de la celda de dirección 134, y dejar el resultado en el acumulador.

Como puede suponerse, la programación, incluso de problemas sencillos, en este lenguaje es sumamente laboriosa; además de requerir por parte del programador el conocimiento de la "geografía" de la memoria.

Para subsanar estos inconvenientes, y a medida que se ampliaban las posibilidades de aplicación de los ordenadores, surgieron nuevos lenguajes destinados a facilitar la relación del usuario con la máquina.

En una primera etapa se sustituyó el código binario de instrucción por grupos de letras mnemotécnicos, lo que dio lugar a un lenguaje llamado *Ensamblador.* Así, el código de la operación suma se puede representar por SUM, etc.

Pero las instrucciones así escritas guardan aún demasiada relación con el lenguaje máquina, por lo que fue necesario desarrollar lenguajes de programación parecidos a los lenguajes humanos, a los que se llamó *lenguajes de alto nivel,* distinguiéndolos de los lenguajes tipo Ensamblador o de *bajo nivel.* Los lenguajes de alto nivel son lenguajes universales; el usuario sólo ha de tener en cuenta el problema a resolver y no la estructura interna de cada ordenador.

0.3.1 INTERPRETES Y COMPILADORES

Con el empleo de estos lenguajes aparecería un nuevo problema. El ordenador sólo entiende el lenguaje máquina y es necesario dotarle de unos programas traductores que pasen los lenguajes de bajo o alto nivel a lenguaje máquina. Se llama programa fuente al programa escrito en lenguaje evolucionado y programa objeto al que resulta de traducirlo al lenguaje máquina.

Hay tres tipos de traductores:

a) *Ensambladores*

Se emplean para traducir programas en ensamblador. Toda instrucción del programa fuente da lugar a una sóla instrucción del programa objeto.

b) *Compiladores*

Se emplean para traducir programas escritos en lenguajes de alto nivel. Una instrucción del programa fuente se traduce en varias del programa objeto.

c) *Intérpretes*

Tienen los mismos fines y características que los compiladores. La diferencia entre ellos es que el programa compilador traduce todo el programa que posteriormente se ejecutará, y el programa intérprete traduce una instrucción y no pasa a la siguiente hasta que la anterior es ejecutada.

0.3.2 BREVES CONSIDERACIONES SOBRE LOS LENGUAJES DE ALTO NIVEL

Estos lenguajes presentan innegables ventajas, por su potencia y su relativa facilidad de manejo, y algunos inconvenientes, debido a la necesidad de mayor memoria central para almacenar el programa traductor. Por otra parte, el programa tarda más tiempo en ejecutarse y ocupa más memoria que el programa escrito directamente en lenguaje máquina. Esta última objeción puede plantear, en ocasiones, la conveniencia de utilizar lenguaje máquina en aquellos problemas que exijan una gran velocidad de ejecución, como puede ser el caso de la reserva de billetes de ferrocarril o avión, control de naves espaciales, contabilidad bancaria, etc.

Por último, para que se observe lo tedioso de un programa escrito en lenguaje máquina, tratemos un sencillo problema: el cálculo del máximo común divisor de dos número por el algoritmo de Euclides.

Empleamos el lenguaje máquina y el ensamblador que a continuación se proponen:

Lenguaje convencional	Lenguaje máquina (se pondría en sistema binario)	Lenguaje ensamblador
Poner A en N	1,A,N	PON,A,N
Llevar el valor de M a N	2,M,N	MOV,M,N
Ir a N	3,N	IR,N
Sumar lo que hay en M con lo que hay en N y dejar el resultado en P	4,M,N,P	SUM,M,N,P
Restar...	5,M,N,P	RES,M,N,P
Multiplicar...	6,M,N,P	MUL,M,N,P
Dividir...	7,M,N,P	DIV,M,N,P
Si lo que hay en M es igual a lo que hay en N, ir a P	8,M,N,P	COM,M,N,P
Tomar la parte entera de M y dejarla en M	9,M	INT,M
Escribir lo que hay en M	10,M	ES,M
FIN	11	FIN

Cuando digamos "Poner A en N", queremos decir "Poner el dato A en la celda de memoria cuya dirección es N"; y "Sumar lo que hay en M..." es la forma abreviada de expresar "Sumar lo que hay en la celda de memoria cuya dirección es M, con lo que hay en la celda de memoria cuya dirección es N, y dejar el resultado en la celda de dirección P".

El cálculo del m.c.d., en el lenguaje máquina propuesto tendría la forma:

Lenguaje convencional	Lenguaje máquina
1) Poner A en 20	1,A,20
2) Poner B en 21	1, B, 21
3) Poner 0 en 22	1, 0, 22
4) Dividir lo que hay en 20 por lo que hay en 21 y poner el resultado en 23	7, 20, 21, 23
5) Tomamos la parte entera de lo que hay en 23 y lo dejamos en 23	9, 23
6) Multiplicar lo que hay en 21 por lo que hay en 23 y dejar el resultado en 23	6, 21, 23, 23
7) Restar de lo que hay en 20 lo que hay en 23 y poner el resultado en 23	5, 20, 23, 23
8) Si lo que hay en 23 es igual que lo que hay en 22 ir a 12	8, 23, 22, 12
9) Llevar el valor de 21 a 20	2, 21, 20
10) Llevar el valor de 23 a 21	2, 23, 21
11) Ir a 4	3, 4
12) Escribir lo que hay en 21	10, 21
13) FIN	11

Con un lenguaje de alto nivel como BASIC, el programa presenta una gran sencillez debido a la potencia de las instrucciones, como tendrás ocasión de comprobar en las páginas siguientes.

1

BALBUCEANDO BASIC

1.1 INTRODUCCION

Mucha gente cree que los ordenadores son máquinas muy inteligentes, y que resuelven todos los problemas. Pero ¿qué sucede cuando te pones delante de un ordenador y lo enciendes? Que la máquina saca un mensaje, más bien raro, en la pantalla y no hace nada, sólo espera.

El ordenador no hace nada si no se le pide correctamente. La forma de pedirle algo, de decirle lo que queremos, se llama *programa.* Lo que es sorprendente es la velocidad con que ejecuta los programas, la capacidad de cálculo que

tiene; pero carece de inteligencia. Hay que darle todo muy detallado, y si tú te equivocas él se equivoca.

Y ¿cómo se le mete un programa? ¿Entenderá igual a un chino que a un español? Se han ideado unos cuantos LENGUAJES de programación. Aquí vamos a estudiar el BASIC, que es el más universal y el más apropiado para comenzar.

Una pequeña dificultad es que el LENGUAJE BASIC ofrece ligeras modificaciones en las distintas marcas comerciales y según los modelos de ordenadores. Trataremos de enseñar las versiones más extendidas haciendo algunas observaciones sobre estas posibles variantes. Por ejemplo: vamos a suponer desde ahora que el ordenador con el que trabajas consta de un teclado, como una máquina de escribir, y una pantalla de televisión. Si en vez de pantalla tuvieras impresora las cosas cambian un poco, pero muy poco, y te resultará fácil adaptar estas lecciones a tu caso personal.

1.2 PRIMEROS PASOS

Vamos a iniciarnos en el lenguaje de programación BASIC. Empezaremos escribiendo un pequeño programa:

```
1Ø PRINT 2*3.1416*5
2Ø END
```
(1)

Este programa se compone de dos *líneas,* la 1Ø y la 2Ø, y en cada línea tiene una *instrucción* o *sentencia.* Cada línea comienza por un número natural que indica al ordenador el orden en que las tiene que ejecutar.

El programa (1) es muy sencillo; sirve para calcular la longitud de una circunferencia de radio 5.

El asterisco (*) es el símbolo que reserva el BASIC para la multiplicación, y que no debes confundir con . ó x.

Observa que 3.1416 es el valor aproximado de π, y que la coma decimal(,) es sustituida por el punto (.), siguiendo la notación anglosajona.

La cifra cero se escribe Ø para distinguirla de la letra O.

PRINT es la instrucción que imprime el resultado de la operación que se indicaba a continuación. Print = Imprimir (Sacar en pantalla).

END indica que el programa se ha terminado. End = Fin.

Te habrá sorprendido que la numeración de las líneas sea 1Ø, 2Ø en vez de 1, 2. Podíamos haber puesto 1, 2; puedes hacer la prueba y el programa funciona igual. Pero es frecuente que en los programas haya olvidos, y que tengamos que intercalar varias líneas entre dos ya escritas. Y así entre la 1Ø y la 2Ø queda lugar para las 11, 12, ..., 19. Los buenos programadores numeran siempre de 1Ø en 1Ø para no pillarse los dedos si hay que intercalar líneas.

Se pueden escribir las líneas salteadas:

2Ø . . .
1Ø . . .
4Ø . . .
3Ø . . .

ya que la máquina las ordena inmediatamente según el número de línea.

1.3 TECLA RETURN

Al teclear una línea en el ordenador hay que terminarla siempre pulsando la tecla [RETURN]. Ya que es necesario indicarle de alguna forma a la máquina que has terminado la línea, que no vas a seguir añadiendo operaciones. Return = Retorno: equivale al retorno del carro de una máquina de escribir. Tú das por terminada una línea y el ordenador la mete entonces de golpe en su memoria. Nota: algunos ordenadores llaman a esta tecla [END LINE]. (End line = = Fin de línea).

Así pues, para comunicar el programa (1) al ordenador tecleamos:

1Ø PRINT 2*3.1416*5 [RETURN]
■ (cursor)

Los ordenadores suelen tener un *cursor*, o señal luminosa, que indica en qué lugar se imprimirá la próxima letra o número que escribas. Con la tecla [RETURN] el cursor ha pasado, de estar a la derecha del 5, a estar al principio de la línea siguiente; en la posición adecuada para poder escribir la segunda línea:

2Ø END [RETURN]
■ (cursor)

Una vez que hemos comunicado el programa al ordenador, su ejecución se realiza formando la palabra RUN y pulsando la tecla [RETURN]. Run = Correr. En la línea siguiente aparecerá el resultado: 31.416. La pantalla quedará así:

```
1Ø PRINT 2*3.1416*5
2Ø END
RUN
31.416
```

Recuerda esto: el ordenador no se da por enterado de una instrucción hasta que no se pulsa la tecla [RETURN].

En los programas que siguen nosotros no escribiremos ya esta tecla, lo damos por supuesto, pero tú la tendrás que pulsar cada vez que dictes una instrucción al ordenador.

1.4 ASIGNACION: LET

El programa (1) calcula únicamente la longitud de una circunferencia de radio 5. Para hallar la longitud de circunferencias de otros radios hay que modificar la línea 1Ø, y esto parece pesado. Vamos a mejorar el programa, con objeto de que se adecúe al cálculo de longitudes de circunferencias para distintos radios. Una primera modificación puede ser ésta:

```
1Ø LET R=5
2Ø LET L=2*3.1416*R                    (2)
3Ø PRINT L
4Ø END
```

Las líneas 1Ø y 2Ø contienen sentencias de *asignación,* para lo que se utiliza la instrucción LET. (Let = Sea). En la línea 1Ø a la variable R se le asigna el valor 5. En la 2Ø la variable L toma el valor 31.416, resultado de multiplicar 2*3.1416*5. Al ejecutar la línea 3Ø, el ordenador sacará en pantalla el valor de L, es decir, el número 31.416.

Al programar se puede prescindir de la palabra LET y teclear sencillamente:

```
1Ø R=5
2Ø L=2*3.1416*R
```

con el mismo significado que en (2). Es decir, LET es opcional, y los programadores no suelen utilizarlo.

En el programa (2) para hallar longitudes de circunferencias de radios distintos hay que corregir la línea 1Ø.

1.5 MODIFICACION DE UN PROGRAMA

Seguimos con el programa (2) sobre longitud de circunferencias. Si queremos calcular la longitud de una circunferencia de radio 8 tendremos que modificar el programa, cambiando la línea 1Ø por:

```
1Ø LET R=8
```

Si todavía teníamos el programa (2) en el ordenador la modificación se efectúa escribiendo la nueva línea:

1Ø LET R=8 (y pulsar [RETURN]).

La antigua línea 1Ø queda sustituida inmediatamente por la que acabamos de escribir.

El comando LIST (List = Listar) permite obtener en la pantalla el listado del programa que tiene actualmente el ordenador. Así pues, escribiendo LIST (sin olvidar [RETURN]) en la pantalla aparece el nuevo programa:

```
1Ø LET R=8
2Ø LET L=2*3.1416*R                    (3)
3Ø PRINT L
4Ø END
```

y comprobamos que, en efecto, la nueva instrucción ha sido incluida en el programa. Si ahora pulsamos RUN, la pantalla queda así:

```
RUN
5Ø.2656
```

La longitud de la circunferencia de radio 8 es 50.2656.

Cuando lo que te interesa no es modificar un programa, sino escribir un

programa nuevo, debes empezar por borrar el antiguo. El *comando* NEW borra el programa de la memoria. Antes de teclear alguno de los programas de los ejercicios, o los del capítulo próximo, debes escribir NEW, o apagar y encender el ordenador.

1.6 ENTRADA DE DATOS: INPUT

Los programas (2) y (3), desde un punto de vista práctico, no son muy interesantes, puesto que hay que modificarlos cada vez que se cambia de radio. Los hemos puesto con fines didácticos al objeto de introducir algunos conceptos del BASIC.

Para una persona que tenga que calcular longitudes de muchas circunferencias, en función de los diferentes radios, es interesante disponer de un programa que lo haga para cualquier radio. Es evidente que al ejecutar este programa el ordenador nos deberá preguntar cuál es el valor del radio; nosotros deberemos introducir este valor desde el teclado. Todo esto se logra con la instrucción INPUT (Input = Entrada).

```
1Ø  INPUT  R
2Ø  LET  L=2*3.1416*R                    (4)
3Ø  PRINT  L
4Ø  END
```

Cuando corramos este programa por medio del comando RUN, el ordenador encontrará la instrucción o sentencia INPUT y se parará mostrando una interrogación (?). Espera que le demos el valor de R. Si deseamos que calcule L para el valor R = 8 bastará escribir 8 [RETURN], e inmediatamente aparecerá el resultado en la línea siguiente:

```
RUN
? 8
5Ø.2656
```

Si volvemos a ejecutar el programa, en pantalla tendremos:

```
RUN
?
```

Y el ordenador queda dispuesto para recibir un nuevo valor de R.

1.7 QUE ES UNA VARIABLE

Una *variable* es una "palabra" que comienza necesariamente por una letra, y que puede ir seguida de otras letras o de cifras. Por ejemplo:

Son variables: R, RADIO, RAD, R1, RAD23, AA, AMOR
No son variables: 2A, TO+, ØLA, 1 23

Tampoco son variables las palabras utilizadas por el BASIC, como por ejemplo: PRINT, END, LET, RUN, INPUT, LIST; ni tampoco palabras que las contienen como: ALETA, OFENDE, LETRA, SENDA, etc.

Muchos ordenadores sólo distinguen los dos primeros caracteres de las variables; es decir, para estos ordenadores dos variables que tienen los dos primeros caracteres iguales, no se diferencian. Así RAD y RADIO representan la misma variable; mientras que R, RAD, AR, y R1 son variables distintas.

OBSERVACION: Algunos ordenadores sólo admiten como variables una letra, o una letra seguida de un número, como A, B, C, A2, B7, CØ, etc.

1.8 COMO DEBEMOS IMAGINAR UNA VARIABLE

Debemos imaginar una variable como una caja capaz de "guardar" un número. Un número se "guarda" en una variable por medio de una asignación LET o de una entrada INPUT.

```
1Ø  LET  A=5
```

Con esta instrucción la variable A toma el valor 5.

```
1Ø  LET  A=5
2Ø  LET  B=A
```

Ahora en la línea 2Ø la variable B toma el valor de A, es decir 5; sin que por ello se varíe el contenido de A. Es como si se pasara una copia del contenido de A a B.

Obsérvese que el "paso" del contenido se realiza de derecha a izquierda: en la línea 2Ø, B toma el valor de A (y no A el de B). Cuando queremos asignar un valor a una variable, ésta ha de situarse necesariamente a la izquierda del sig-

no igual (=). A la derecha del signo igual (=) puede haber un número, una variable o una expresión aritmética. Observa el siguiente programa:

```
1Ø  LET  N=3
2Ø  LET  N=N + 1
3Ø  PRINT  N
4Ø  END
```

En la línea 1Ø, N toma el valor 3. En la línea 2Ø, N toma el valor N+1 = = 3+1 = 4. Al correr el programa aparecerá en la pantalla:

```
RUN
4
```

Fíjate que en matemáticas la expresión N = N+1 no tiene sentido. Sin embargo en Informática sí, porque primero se calcula el valor a la derecha del signo igual (=) en la línea 2Ø (N+1=3 + 1 = 4), y después se asigna ese valor a la variable que está a la izquierda de igual (=), haciendo en este caso N= 4.

```
1Ø  LET  B=7       (B toma el valor 7)
2Ø  LET  A=3+B     (A toma el valor 1Ø; + es el signo de la adición)
3Ø  LET  B=A       (B toma el valor 1Ø)
4Ø  LET  N=3       (N toma el valor 3)
5Ø  LET  N=N+1     (N toma el valor 4)
6Ø  LET  C=3*7     (C toma el valor 21)
7Ø  LET  9=D       (Sentencia incorrecta. Error de sintaxis. 9 no es nombre de una variable)
8Ø  LET  A+B=F     (Sentencia incorrecta por igual motivo)
```

Si en un programa aparece una variable a la que no se ha asignado previamente un valor, se entiende que su valor es Ø.

Al estudiar la instrucción INPUT has visto que es otra forma de "guardar" un número en la "caja" de la variable.

RESUMEN

En el siguiente cuadro recordamos los conceptos e instrucciones del lenguaje de programación BASIC que, por el momento, tienes que conocer. Si no

estás seguro de algún concepto o de cómo funciona una instrucción, búscalo en el texto antes de continuar. Si lo comprendes todo ya puedes empezar a balbucear BASIC, a lo que te ayudarán los ejercicios de la página siguiente.

RECUERDA	
Programa	PRINT
Línea	END
Numeración de las líneas	RUN (comando)
Instrucción o sentencia	LET A=B
Comando	LIST (comando)
*	INPUT
+	NEW (comando)
. (punto decimal)	
Ø y 0	
RETURN	
=	
Asignación	
N=N+1	
Variable. Nombres de variables.	
Palabra reservada	
Sustitucion de una sentencia por otra	
Cursor	

EJERCICIOS

1.1) Todas las sentencias BASIC que siguen son incorrectas. ¿Por qué?

```
1Ø LET A=2,5
2Ø IMPUT R
3Ø LET 27=B
4Ø LET C=2.50
5Ø LET 2*F=T
6Ø LET RUN=8
7Ø LET A=B.
```

1.2) ¿Son correctas las siguientes proposiciones?

```
1Ø LET A=5
2Ø LET N=A+1
3Ø LET M=2*A
4Ø LET P=2M+4P
5Ø LET Q=2*H
```

1.3) Escribe un programa BASIC que calcule áreas de círculos.

1.4) Escribe un programa que dé al mismo tiempo la longitud de la circunferencia y el área del círculo.

1.5) Explica qué propósito tiene el siguiente programa:

```
1Ø INPUT R
2Ø LET A=4*3.1416*R*R
3Ø PRINT R
4Ø PRINT A
5Ø END
```

1.6) Decir si el siguiente programa está correctamente escrito:

```
1Ø IMPUT R
2Ø LET V=4*3.1416*R*R*R/3
3Ø PRINT V
4Ø END
```

1.7) Decir el objeto del siguiente programa, y si está correctamente escrito:

```
1Ø INPUT R
2Ø INPUT H
3Ø LET V=3.1416*R*R*H/3
4Ø LIST
5Ø PRINT
6Ø END
```

1.8) ¿Qué indica la instrucción END en un programa?

1.9) Escribe un programa usando dos instrucciones LET.

1.10) ¿Qué ocurre al oprimir la tecla [RETURN] ?

1.11) ¿Qué instrucción se usa para introducir datos en un programa?

1.12) ¿Qué hace falta para ejecutar el siguiente programa?

```
1Ø PRINT 7+5
2Ø END
```

1.13) Escribe un programa utilizando las instrucciones PRINT, INPUT y LET.

1.14) ¿Qué indica la siguiente línea? 2Ø LET X=Y
¿Y la siguiente? 3Ø LET V=3

1.15) La instrucción INPUT ¿qué papel desempeña en un programa?

1.16) Escribe un programa que calcule el área de un rectángulo de lados A y B.

1.17) El alcance de un proyectil, en función de su velocidad y del ángulo de salida, es

$$L = \frac{V_0^2}{g} \text{ sen } (2\alpha)$$

Que en BASIC se escribe: L=VØ↑2*SIN(2*A)/9.8Ø665

Haz un programa que, dándole los valores de Vo y α, dé el alcance.

OBSERVACION: La función SIN trabaja con radianes, por lo que, como

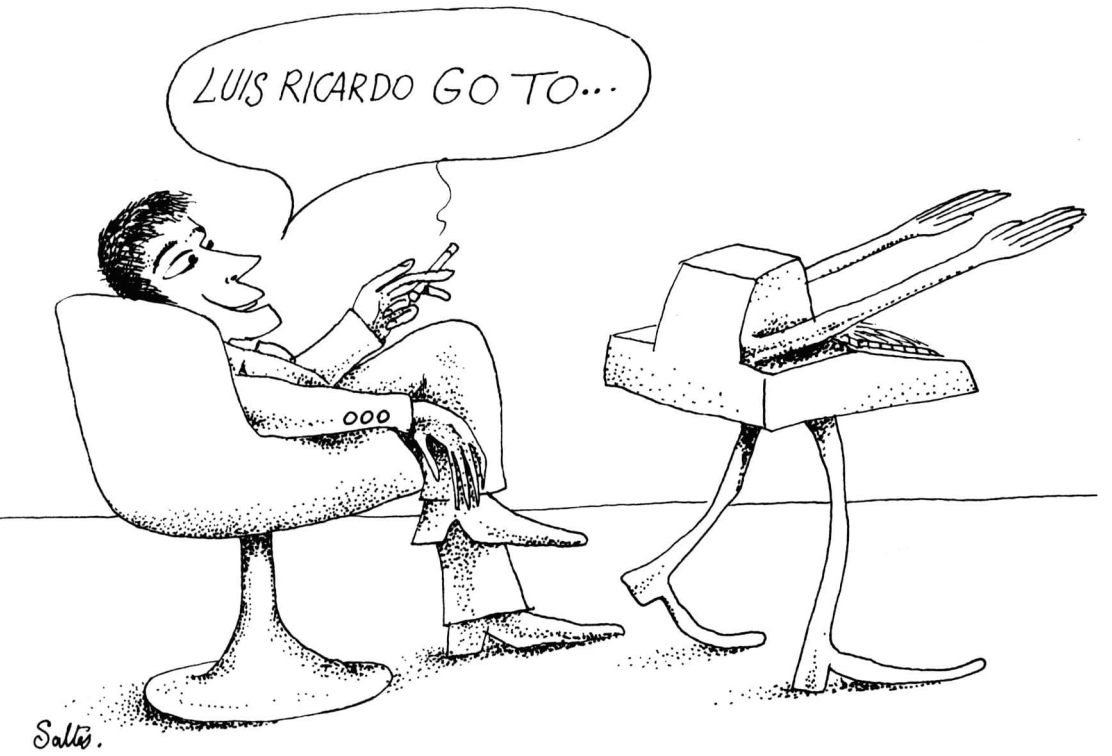
queremos introducir el ángulo en grados, antes de calcular el SIN debes pasar el ángulo a radianes.

1.18) Un ciclista se pone a bajar sin frenos un puerto de montaña. Si suponemos que el aire tampoco le frena, la velocidad que alcanza es $v = \sqrt{2gh}$ aproximadamente, siendo h el desnivel. Haz un programa que, según el desnivel, dé la velocidad alcanzada. Interesa al final pasar el resultado de m/seg a Km/h. Utiliza la expresión:

V=(2*9.8Ø665*H)↑(1/2)

2

EL LENGUAJE BASIC



2.1 B.A.S.I.C.

La palabra BASIC viene de la expresión inglesa "Beginner's All-purpose Symbolic Instruction Code": código de instrucción simbólico de uso múltiple para principiantes.

El BASIC es un lenguaje que permite al hombre comunicarse con el ordenador. Fue desarrollado en la década de los sesenta por los profesores John G. Kemeny y Thomas E. Kurtz de los EE.UU. Utiliza palabras inglesas y expresiones matemáticas.

El BASIC, como todo lenguaje, utiliza un alfabeto formado por los siguientes *caracteres:*

Letras: A, B, C, ..., Z (¡siempre mayúsculas!)

Cifras: Ø, 1, 2, ..., 9

Caracteres especiales:

Operadores aritméticos: +, −, *, /, ↑ (adición, sustracción, multiplicación, división y exponenciación).

Otros caracteres: (,), =, <, >, $, etc.

Todos los caracteres se introducen en el ordenador desde un teclado semejante al de una máquina de escribir.

2.2 LAS FORMULAS

Entre las expresiones más importantes que podemos escribir con los caracteres BASIC, están las *fórmulas.*

Una fórmula es una expresión formada por *constantes* y *variables,* ligadas por los operadores aritméticos, siguiendo unas reglas parecidas a las del cálculo usual.

Las fórmulas se computan con los valores actuales de las variables; por tanto una fórmula representa siempre un número concreto.

En la expresión C=2+A*B, el segundo miembro, 2+A*B, es una fórmula. Si A vale 5 y B vale 6, el valor de C, como resultado de operar la fórmula, es 32.

He aquí algunas fórmulas matemáticas y su correspondiente traducción al BASIC:

FORMULA MATEMATICA	FORMULA BASIC
$a\,b+c$	A*B+C
$(a+b)\,c$	(A+B)*C
$a_1 + a_2 + a_3$	A1 + A2 + A3
$3a - bc$	3*A − B*C
$\frac{a+b}{c}$	(A + B)/C
$a + \frac{b}{c}$	A + B/C
a^2	A↑2
a^3	A↑3
$a^2 + b^2$	A↑2+B↑2
$\sqrt{a}$	A↑Ø.5

Observa que el lenguaje BASIC no permite subíndices ni superíndices. Todos los caracteres de una expresión tienen que ir en la misma línea.

Al transcribir al lenguaje BASIC las anteriores fórmulas hemos utilizado algunas reglas que vamos a poner de manifiesto en el apartado siguiente.

2.3 JERARQUIA DE LOS OPERADORES ARITMETICOS

El orden de ejecución de los operadores aritméticos en una fórmula BASIC es el siguiente:

Primero se ejecutan todas las exponenciaciones; después se ejecutan todas las multiplicaciones y divisiones; por último se ejecutan todas las adiciones y sustracciones.

Podemos clasificar los operadores, según la prioridad de ejecución, en tres niveles:

Primer nivel: ↑

Segundo nivel: *, /

Tercer nivel: +, –

No hay orden de prelación dentro de un mismo nivel. Las operaciones de un mismo nivel se ejecutan en las fórmulas de izquierda a derecha.

Veamos algunos ejemplos:

Ejemplo 1:

Consideremos la fórmula:

$$5*6+2*3\uparrow 2-5\uparrow 2 \quad (2)$$

Numeremos las operaciones según el orden de ejecución:

```
[3] [5] [4] [1] [6] [2]
 ↓   ↓   ↓   ↓   ↓   ↓
5 * 6 + 2 * 3 ↑ 2 – 5 ↑ 2
```

[1]	5*6+2*9–5↑2	Se ejecutan primero las exponenciaciones empezando
[2]	5*6+2*9–25	por la de la izquierda.
[3]	3Ø+2*9–25	Después los productos y cocientes, empezando también
[4]	3Ø+18–25	por el más a la izquierda.
[5]	48–25	Por último las sumas y restas, también de izquierda a
[6]	23	derecha.

Ejemplo 2:

Sea ahora la fórmula:

$$12/3*2 \quad (3)$$

Como los operadores / y * son del mismo nivel, se ejecutan de izquierda a derecha; esto es: primero la división y luego la multiplicación; resultando sucesivamente:

12/3*2 (3)
4*2
8

2.4 USO DE PARENTESIS

Si en la fórmula (3) queremos que 2 esté en el denominador, dividendo, deberíamos ponerlo entre paréntesis:

$$12/(3*2) \quad (4)$$

El ordenador antes que nada, antes que las exponenciaciones incluso, realiza las operaciones de los paréntesis:

$$\begin{matrix} & [2] & [1] \\ & \downarrow & \downarrow \\ 12 & / & (3*2) \end{matrix}$$

[1] 12/6
[2] 2 (resultado distinto que el de la fórmula (3)).

Luego para romper el orden de ejecución según los niveles señalados en el apartado anterior, hay que utilizar paréntesis. Como en matemáticas, se pueden poner unos paréntesis dentro de otros.

Para computar una fórmula con varios niveles de paréntesis, se empieza ejecutando las operaciones de los paréntesis más interiores, sucesivamente, hasta llegar a computar todos los niveles de paréntesis. Dentro de cada paréntesis las operaciones se ejecutan siguiendo las reglas del apartado 2.3.

Ejemplo 3:

$$\begin{matrix} & [1] & [3] & [5] & & [2] & & [4] & & [6] \\ & \downarrow & \downarrow & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ (& (2 & + & 1) & \uparrow & 3 & - & (3 & + & 2) * 4) \uparrow 2 \end{matrix} \quad (5)$$

El resultado de computar esta fórmula es 49.

Recuerda que siempre que se abre un paréntesis hay que cerrarlo más adelante: han de cerrarse el mismo número de paréntesis que se abren.

Algunas máquinas tienen en su teclado corchetes, que pueden utilizarse en la presentación de resultados, etc., pero que no pueden utilizarse en las fórmulas, no tienen el sentido matemático que tienen los paréntesis.

Un número negativo sólo se puede elevar a una potencia entera. Más explícitamente: la potencia A↑B se computa siempre que A sea positivo, o bien cuando A es negativo y B entero. Por ejemplo, la fórmula $(-1.25)\uparrow 3$ es correcta: pero la fórmula $(-8)\uparrow(1/3)$, que equivale a $\sqrt[3]{-8}$, aunque matemáticamente es igual a -2, no puede ser calculada por el ordenador.

2.5 ESCRITURA DE NUMEROS O CONSTANTES

En BASIC se pueden escribir números hasta con nueve cifras (*), un punto decimal y un signo (+ ó −). Si no figura el signo, se supone que es el signo +. Así por ejemplo, se puede escribir:

+123456789
123.Ø56789
−333333333

Siguiendo la notación anglosajona, si delante del punto decimal va el cero, éste se puede suprimir. Podemos escribir .25 en lugar de Ø.25.

Existe otra notación, la exponencial, que permite escribir números muy grandes o muy pequeños. El número:

$3ØØØØØØØØØØØØ = 3 \cdot 10^{12}$

se escribe en BASIC así: 3E12

El número $Ø.ØØØØØØØØ4 = 4 \cdot 10^{-9}$

se escribe 4E−9

Observa que la letra E sustituye a la expresión "x 10 elevado a".

Delante de la E tiene que haber siempre un número, aunque sea sólo el 1. Así 10^{15} se escribe 1E15, y no E15, que daría error.

El número que sigue a la letra E en la notación exponencial ha de ser entero, (sin punto decimal), y con signo, aunque el signo + se puede suprimir. La notación exponencial equivale a mover el punto decimal a derecha o izquierda, el número de veces que señale E.

Es posible escribir una constante BASIC de varias formas. A continuación damos varios ejemplos:

3	+3	3.
325.4	3.254E+2	3.254E2
Ø.Ø126	1.26E−2	12.6E−3
−4	−4.	−Ø.4E1

(*) Este número puede variar en algunos casos

El valor absoluto de una constante BASIC varía desde 10^{-38} hasta 10^{38}, aunque este recorrido puede cambiar de unos ordenadores a otros. Por supuesto que aunque el 0 no está incluido en el recorrido anterior, también es una constante BASIC.

Al tratar de las variables no hablamos de las variables enteras, y lo hacemos ahora. Si una variable no va a tomar más que valores enteros, conviene especificarlo porque utiliza la tercera parte de memoria que las variables reales. Se designa con el carácter % a continuación del nombre de la variable. Por ejemplo A% , RAD% , BA7% , etc. Pero tienen una restricción adicional y es que sus valores han de estar comprendidos entre −32768 y +32767, recorrido mucho más pequeño que el de las variables reales.

RECUERDA

Caracteres
Operadores aritméticos: +, −, *, /, ↑
Teclado
Fórmula
Constante
Computación de fórmulas
Jerarquía de operadores
Niveles de operadores
Variables enteras
Ejecución de operadores del mismo nivel
Uso de paréntesis
Niveles de paréntesis
Potencias de base negativa
Escritura de constantes
Supresión del cero
Notación exponencial
Varias formas de escribir una constante

EJERCICIOS

2.1) ¿Cómo se expresa el número 2^6 en lenguaje BASIC?

2.2) Sabiendo que las operaciones aritméticas se efectúan en un orden determinado, ¿cómo se efectuarían las siguientes operaciones dentro del ordenador?

3+4+5/3 7/3/2 4+5*2 6/3↑4

2.3) ¿Qué operación aritmética tiene prioridad en el lenguaje BASIC, * ó /? Y en general, dentro de todos los operadores aritméticos, ¿cuál se ejecuta en primer lugar?

2.4) ¿Qué papel desempeñan los paréntesis en una fórmula BASIC?

2.5) Indica el orden en que la computadora realiza los cálculos:

```
1Ø PRINT 8+3*4-7
2Ø PRINT 8↑4+5*9
3Ø PRINT (8/3)*(4-7)
4Ø PRINT (8-7)/5-9
```

2.6) Transcribir al lenguaje BASIC las siguientes fórmulas:

a) $2a^3 + a^2 + ab$

b) $\sqrt{a}+\sqrt{b}$

c) $\sqrt[3]{x}$

d) $\dfrac{(3a^2 + b)^3}{7 + 6c}$

e) $\dfrac{1}{\sqrt[3]{bc^2}}$

f) $\dfrac{x(x-1)}{2} - \dfrac{y(x-2)}{3}$

2.7) Escribir una instrucción LET que contenga la expresión:

$$x^3 + y^3 - \frac{225}{12}$$

2.8) Escribir un programa en que se utilice la fórmula:

$$S = \left(\frac{2x + 3}{5}\right)^2$$

2.9) Transcribe al lenguaje matemático las siguientes fórmulas BASIC:

a) A/B+C/D
b) A/(B+C)/D
c) (5*T)↑.5
d) A↑(2*B)
e) A+B/C*D−A*B↑3
f) (A↑3+A↑2+A)↑.5
g) A↑B↑C
h) A+A−A*A/A↑A

2.10) Escribir una sentencia LET correspondiente a cada una de las siguientes igualdades:

a) $s = \frac{1}{2} gt^2$ (espacio en caída libre de un cuerpo)

b) $V = \frac{1}{3} \pi r^2 h$ (volumen del cono)

c) $V = \frac{4}{3} \pi r^3$ (volumen de la esfera)

d) $S = 4\pi r^2$ (superficie de la esfera)

e) $E = \frac{1}{2} mv^2$ (energía cinética de un cuerpo)

2.11) Escribir un programa para computar la siguiente fórmula utilizando dos variables intermedias o auxiliares:

$$p = \left(\frac{3q}{2a + b^2} + \frac{r}{3a + b} \right)^{3/2}$$

2.12) Escribir un programa para computar

a) $\dfrac{1{,}48 \times 4{,}2 + (9 - 19{,}47) \times 5}{(3{,}25 + 10{,}76 + 6{,}25) \times 12{,}43}$

b) $\left(9 - 7{,}5 \times \sqrt{29{,}6} \right)^2$

2.13) Escribir un programa que calcule el valor del polinomio:

$x^4 + 3x^3 - x^2 + x + 1$

a) Para x = 7,023
b) Para x = −7,023
c) Para cualquier x

2.14) ¿Qué dificultad se puede presentar al ejecutar un programa con la siguiente proposición LET?

```
15 LET X=(Z-2*Y)↑.5
```

2.15) ¿Qué error estamos cometiendo en la siguiente instrucción LET?

```
5 LET A = [(3+X)/5+Y]↑2
```

2.16) Escribe en lenguaje BASIC los siguientes números:

a) mil millones
b) 0,000000000027
c) $3,1735 \times 10^{13}$
d) −3,1234567890
e) 0,19
f) $-2,49 \times 10^{3}$
g) $12,49 \times 10^{3}$
h) 17×10^{20}

2.17) Todos los números que siguen están mal escritos en BASIC. Identifica los errores:

a) Ø.76−E2
b) 1.ØØØ.ØØØ
c) 17E+53
d) E−3
e) 123456789Ø
f) Ø,27
g) −3.2.5
h) 2E.5

2.18) ¿Qué indica en notación ordinaria la expresión que en BASIC se escribe 3.6ØØE+9?

2.19) Deduce una fórmula para convertir grados centígrados en Fahrenheit y escribe el programa correspondiente.

2.20) Calcular la cantidad de tela necesaria para hacer una tienda de campaña de forma cónica si la altura es de 5 m y el radio de la base 3 m .Hacer la programación para H y R variables.

2.21) Un depósito de butano tiene forma esférica y su altura es de 20 m. Calcular su capacidad en litros.

2.22) Escribir un programa en BASIC para hallar la hipotenusa de un triángulo rectángulo.

3

CLARIDAD DE EXPRESION

3.1 COMENTARIOS Y OBSERVACIONES

En este capítulo vamos a ver algunas cuestiones que contribuyen a mejorar la claridad de los programas y la presentación de los resultados.

Con mucha frecuencia se manejan programas elaborados por otras personas. Si no te explican su funcionamiento, antes de utilizar un programa tendrás que estudiar su listado, y esto puede ser laborioso. Incluso tus propios programas, si ya ha pasado tiempo, te puede costar reconocerlos. Por esto es conve-

niente poner en el listado una observación que te recuerde la finalidad del programa, una especie de título.

Además, si el programa es largo y complicado, es interesante meter entre líneas algún comentario que lo explique. Por ejemplo: para corregir o mejorar un programa es necesario entenderlo, saber cómo funciona; y con observaciones al margen se facilita mucho este trabajo.

Pensando en esto se ha habilitado la instrucción REM, abreviatura de la palabra inglesa REMARK (nota, comentario, observación). Cuando en una instrucción, tras el número de línea, el ordenador se encuentra la palabra clave REM, la máquina ignora esa línea.

Veamos un ejemplo: supongamos que queremos disponer de un programa que nos dé el volumen de un cilindro en función del radio de la base y de la altura. Recuerda que la fórmula matemática es $V = \pi r^2 h$. Este puede ser un programa:

```
1Ø REM  VOLUMEN DE UN CILINDRO
2Ø INPUT  R
3Ø INPUT  H
4Ø LET  V=3.1416*R↑2*H
5Ø PRINT  V
6Ø END
```

La línea 1Ø es un comentario, cuya única misión es identificar el programa.

Nosotros usaremos siempre la sentencia REM para poner el título. Por tanto, todos los programas que escribas en adelante han de comenzar por una línea REM (para el título) y terminar con una instrucción END (para indicar el final). Puedes también escribir sentencias REM en medio de un programa para explicarlo.

3.2 PRESENTACION DE LOS RESULTADOS

Si ejecutas el programa anterior, dando al radio el valor 1Ø y a la altura el valor 3, en la pantalla sale:

```
? 1Ø
? 3
942.48
```

Como conoces el programa, sabes que el número 942,48 corresponde al volumen. Pero estaría mejor que en la pantalla apareciera una identificación del resultado, sobre todo cuando un programa da varios resultados, como en el ejemplo siguiente:

```
1Ø REM  VOLUMEN Y AREA TOTAL CILINDRO
2Ø INPUT  R
3Ø INPUT  H
4Ø LET  PI=3.1416
5Ø LET  V=PI*R↑2*H
6Ø LET  AT=2*PI*R*H+2*PI*R↑2
7Ø PRINT  V
8Ø PRINT  AT
9Ø END
```

Este programa da el volumen y el área total de un cilindro. Al correr el programa, en la pantalla aparece:

```
? 1Ø
? 3
942.48
816.816
```

Y quizá no sepas a qué variable corresponde cada número. Si no te acuerdas del programa tendrás que listarlo y ver el orden de aparición de las instrucciones PRINT, con la consiguiente molestia y pérdida de tiempo.

Para clarificar los resultados puedes etiquetarlos utilizando la instrucción:

```
PRINT "                "
```

que hace aparecer en la pantalla cualquier símbolo, o cadena de caracteres, que sitúes entre las comillas. Así, sustituyendo en el programa anterior las líneas 7Ø y 8Ø por:

```
7Ø PRINT "VOLUMEN=";V
8Ø PRINT "AREA TOTAL=";AT
```

obtienes un programa que, para R = 1Ø y H = 3, imprime como resultado:

```
VOLUMEN=     942.48                    (1)
AREA TOTAL=     816.816
```

Recuerda: cuando una instrucción PRINT tiene detrás algo entre comillas, el ordenador lo reproduce íntegro; se podría decir que el ordenador "dibuja" el entrecomillado.

Observa que en las dos líneas PRINT del último programa aparece un punto y coma (;). El papel que juega este signo lo veremos más adelante.

Normalmente el ordenador imprime un resultado en la línea inmediatamente siguiente a la de la impresión anterior. Por eso los resultados (1) aparecen en líneas contiguas, demasiado juntos. Para que los resultados salgan separados puedes intercalar en el programa una línea PRINT sin nada detrás:

```
75 PRINT
```

Ya sabes que para intercalar una línea no hay que reescribir todo el programa, sino únicamente la línea que se desea intercalar, ¡y pulsar RETURN!
Si ahora listas el programa, la última parte quedará así:

```
•
•
•
7Ø PRINT "VOLUMEN =";V
75 PRINT
8Ø PRINT "AREA TOTAL =";AT
9Ø END
```

Al llegar a la línea 7Ø el ordenador imprime: VOLUMEN = 942.48, y salta la línea siguiente para imprimir lo indicado en 75, pero como aquí no hay nada, "imprime" una línea en blanco, y pasa a la línea 8Ø. Si R = 1Ø, como hasta ahora, y H = 3, en la pantalla aparece:

```
VOLUMEN =    942.48
                    ←——— línea en blanco
AREA TOTAL =   816.816
```

3.3 UTILIZACION DE LA COMA (,) Y DEL PUNTO Y COMA (;) EN PRINT

Ya conoces varias posibilidades de la instrucción PRINT:

a) PRINT X — imprime el valor de la variable X

b) PRINT 7*3↑2 imprime 63

c) PRINT "BUEN PROGRAMADOR" imprime BUEN PROGRAMADOR

d) PRINT "imprime" una línea en blanco

Veremos ahora cómo pueden combinarse estas cuatro posibilidades, y cómo pueden ordenarse los resultados en la pantalla de forma clara mediante la coma (,) y el punto y coma (;).

La pantalla del ordenador tiene una especie de cuadrícula invisible, en la que se van colocando los caracteres que indican las instrucciones PRINT. La distribución de la cuadrícula depende de cada marca y modelo; vamos a suponer que el nuestro tiene 25 líneas y 40 espacios en cada línea. Además la pantalla está dividida en zonas verticales, que también dependen del aparato. Supondremos que nuestros 40 espacios están divididos en 4 zonas, que tendrán por tanto 10 espacios por zona. Lo entenderás mejor viendo la figura 3.1.

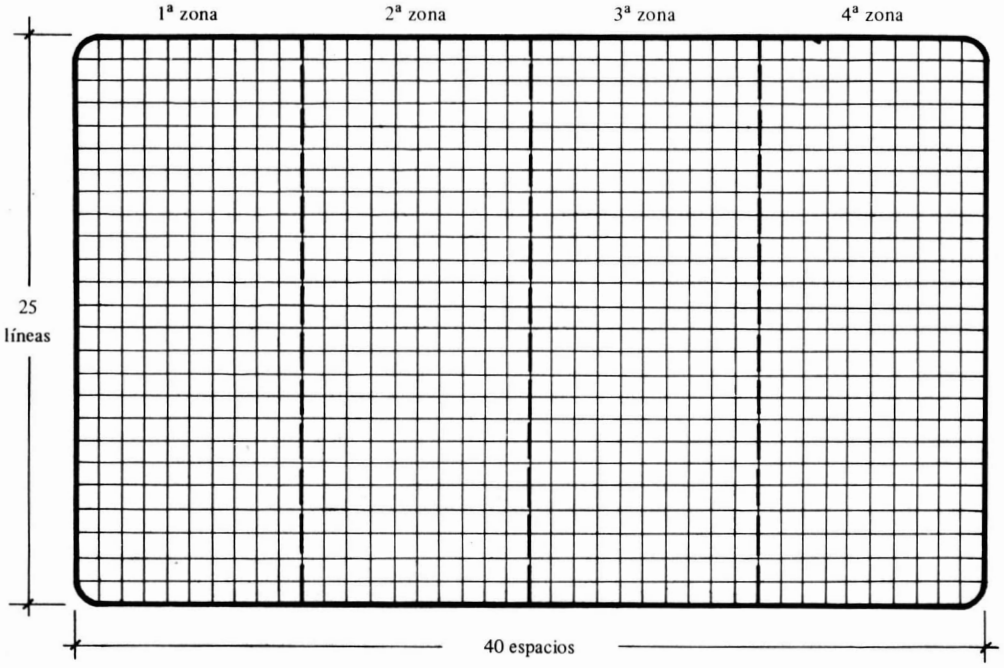


Fig. 3.1

El uso de las zonas se regula mediante la coma (,). Veámoslo con un ejemplo sencillo. Observa la salida del siguiente programa:

```
1Ø PRINT  1, 2, -3, 4, 5, -6
2Ø END
```

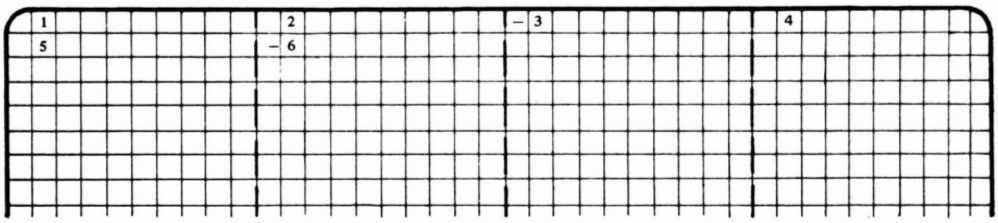


Fig. 3.2

Puedes ver que cada valor se ha escrito en una zona distinta. Podrías imaginar que la coma "dice": *la próxima impresión se efectuará en la zona siguiente a la zona ya utilizada.* (Y si se han agotado las cuatro zonas de una línea, la coma ordena imprimir en la primera zona de la línea siguiente).

En la figura 3.2 se ve cómo es tratado el signo: va en el primer lugar de la zona; cuando es positivo no aparece y deja entonces un espacio en blanco.

La coma (,) cumple la misma función si se sitúa al final de una instrucción PRINT, ordenando entonces que la próxima instrucción PRINT que se ejecute, no imprima en la línea siguiente, sino en la zona siguiente. Lo verás mejor con otro programa:

```
1Ø PRINT  1, 2, -3,   (hay coma al final)
2Ø PRINT  4, 5        (no hay coma al final)
3Ø PRINT  -6
4Ø END
```

que produce la siguiente salida:

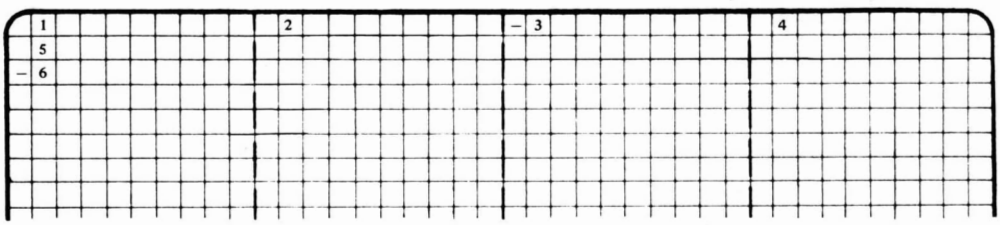


Fig. 3.3

Imagínate que en el programa (1), sobre el volumen y área total del cilindro, modificamos las instrucciones de impresión por las siguientes:

```
7Ø PRINT "VOLUMEN =", V
75 PRINT
8Ø PRINT "AREA TOTAL =", AT
```

Con los valores que hemos utilizado antes para R y H la salida queda así:

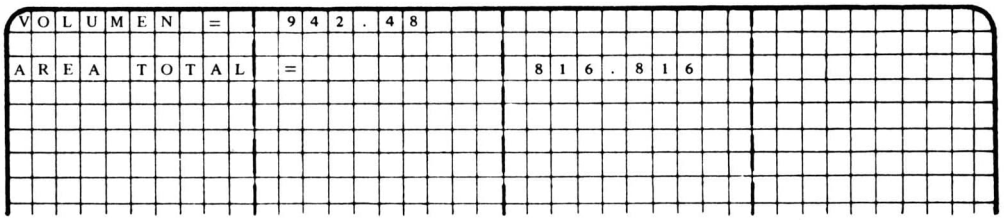


Fig. 3.4

Donde el último número ha pasado a la tercera zona, pues el entrecomillado ha invadido la segunda.

Veamos ahora la utilización del punto y coma (;), que viene a "decir": *la siguiente impresión se hace inmediatamente a continuación.* Esto cuando se trata de cadenas o entrecomillados, pues cuando se trata de números el punto y coma (;) deja un espacio en blanco para mayor claridad, además del correspondiente al signo. Mediante un programa sencillo lo entenderás mejor:

```
1Ø PRINT 1;2;-3;4;5;-6;7;8;-9
2Ø END
```

que produce la siguiente presentación en pantalla:

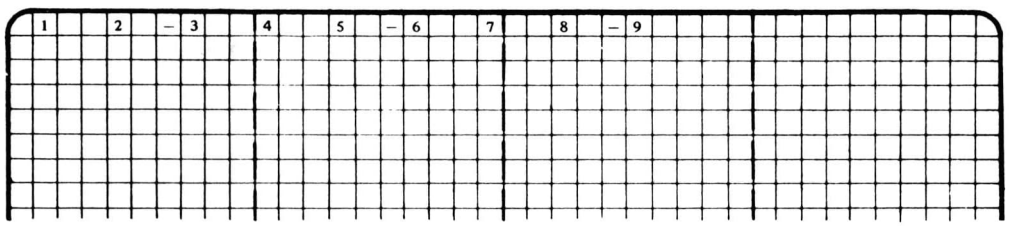


Fig. 3.5

Como en el caso de la coma, el punto y coma del final de una línea actúa sobre la siguiente instrucción PRINT, haciendo que no imprima en la línea siguiente sino a continuación, como lo muestra el siguiente programa:

```
1Ø PRINT 1;2;-3;   (hay (;) al final)
2Ø PRINT 4;5       (no hay (;) al final)
3Ø PRINT -6;7;     (hay (;) al final)
4Ø PRINT 8         (no hay (;) al final)
5Ø PRINT -9
6Ø END
```

Cuya salida es:

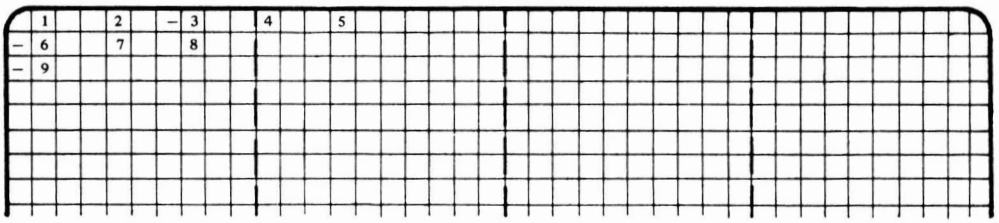


Fig. 3.6

Como último ejemplo, si en el programa (1) del apartado anterior cambiamos las líneas 7Ø y 8Ø por la línea:

```
7Ø PRINT "VOLUMEN =";V,"AREA TOTAL =";T
```

tendremos en la pantalla una salida más elegante:

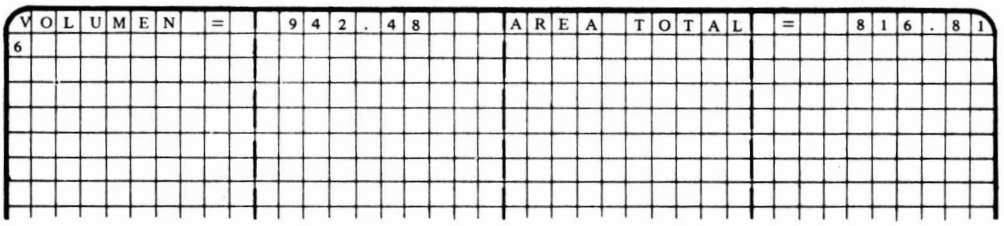


Fig. 3.7

Fíjate en las posibilidades que ofrece la combinación de comas y puntos y comas. Aunque en este caso deberíamos haber acortado un espacio, para que no apareciera ese "6" en la segunda línea. Podríamos poner "AREA TOTAL =" suprimiendo ese espacio en blanco para que quepa 816.816 en la línea.

3.4 CLARIDAD EN LOS DATOS

Hemos visto que cuando el ordenador necesita datos, nos los pide mostrando en la pantalla un signo de interrogación, provocado por la ejecución de una instrucción INPUT. Pero sería conveniente que nos los pidiera llamándolos por su nombre. Esto se puede lograr etiquetando los datos, como vamos a ver:

```
1Ø  REM  VOLUMEN Y AREA TOTAL CILINDRO
15  PRINT  "RADIO=";   (atención al ;)
2Ø  INPUT  R
25  PRINT  "ALTURA=";   (atención al ;)
3Ø  INPUT  H
etc.
```

Al ejecutar este programa aparece en la pantalla la leyenda:

```
RADIO =?
```

que es consecuencia de la ejecución de las líneas 15 y 2Ø, ya que la línea 15 hace imprimir RADIO =, pero como hay punto y coma, la siguiente impresión se hace a continuación.

Cuando demos el valor del radio aparecerá la leyenda siguiente:

```
ALTURA =?
```

Hay una manera más sintética de producir los mensajes anteriores: se trata de fundir las líneas 15 y 2Ø en una sola, lo mismo que las líneas 25 y 3Ø:

```
1Ø  REM VOLUMEN Y AREA TOTAL CILINDRO
2Ø  INPUT  "RADIO=";R
3Ø  INPUT  "ALTURA=";H
etc.
```

Como ves, la instrucción INPUT admite un mensaje inicial de identificación de las variables, separado por un punto y coma.

Además te puede interesar meter unas cuantas variables seguidas, y el INPUT también recoge esta posibilidad, como se ve en el siguiente programa:

```
1Ø REM  MEDIA DE 3 NUMEROS
2Ø INPUT  N1, N2, N3
3Ø LET  M=(N1+N2+N3)/3
4Ø PRINT  "LA MEDIA ES" ; M
5Ø END
```

Como en los ejemplos anteriores la línea 2Ø puede ser mejorada con un mensaje previo:

```
2Ø INPUT  "DECIR LOS 3 NUMEROS" ; N1, N2, N3
```

Al correr este programa y llegar a la línea 2Ø aparece en pantalla:

```
DECIR LOS 3 NUMEROS?
```

y el ordenador espera que le des los tres valores que quieras. Habrás visto en el programa que las variables van separadas entre sí por comas. Pues bien, en la ejecución, los datos que introduzcas tras la interrogación también se separan por comas. Por ejemplo:

```
DECIR LOS 3 NUMEROS? 6.5,4.2,5.5   ( RETURN )
LA MEDIA ES    5.4
```

Observa el papel de la coma y del punto y coma en la instrucción INPUT.

3.5 CADENAS DE CARACTERES Y VARIABLES DE CADENA

El lenguaje BASIC no sólo maneja números, sino también *cadenas de caracteres,* que son sucesiones de signos gráficos cualesquiera, que se encierran entre comillas para saber dónde empieza y dónde acaba la cadena.

Ejemplos de cadenas son:

"HOLA", "AGENTE ØØ7", "JOSE PEREZ GARCIA", "12345", "A=", "A + B", "(", ")".

Esto no debe parecerte nuevo pues lo hemos utilizado alguna vez. En el programa (1) de la pregunta 3.2 etiquetamos los resultados con entrecomillados: "VOLUMEN=" y "AREA TOTAL="; y en la pregunta 3.4 hicimos lo mismo para la instrucción INPUT con los mensajes "RADIO =" y "ALTURA =".

Los espacios se consideran en BASIC como caracteres. Así la cadena:

"JOSEPEREZGARCIA"

es diferente de la cadena:

"JOSE PEREZ GARCIA"

Es claro que las cadenas carecen de significado numérico y que no hay que confundirlas con los nombres de las variables numéricas. Una "operación" que admiten las cadenas es la adición, entendida como yuxtaposición.

"MONTE" + "VIDEO" = "MONTEVIDEO"
"HOLA" + " " + AGENTE ØØ7" = "HOLA AGENTE ØØ7"

La cadena "12345" no tiene significado numérico, y no hay que confundirla con la constante numérica 12345.

Las cadenas de caracteres son una de las posibilidades más interesantes del lenguaje BASIC, pues permiten etiquetar los datos y los resultados, como ya hemos visto, y tratar textos en castellano. La longitud máxima de una cadena es 255 caracteres.

Lo mismo que los números se "guardan" en variables, las cadenas se guardan en *variables de cadena* (o *variables alfanuméricas),* cuyos nombres siguen las mismas reglas de formación que aquéllos, pero siempre acaban con el signo $ (dólar). Así, por ejemplo:

A$, AB$, NOMBRE$, A7$

son variables de cadena.

Las asignaciones, las impresiones, las tomas de datos, etc., se hacen de la

misma manera que para las variables numéricas. Veámoslo con un pequeño ejemplo, que no tiene otro objeto que adiestrarnos en el manejo de las cadenas:

```
1Ø REM  NOMBRES Y APELLIDOS
2Ø LET  A$="ALUMNO:"  (asignación)
3Ø INPUT  N$          (pide el nombre)
4Ø INPUT  A1$         (pide el primer apellido)
5Ø INPUT  A2$         (pide el segundo apellido)
6Ø PRINT  A$          (imprime ALUMNO:)
7Ø PRINT              (deja línea en blanco)
8Ø PRINT  N$,A1$,A2$  (imprime el nombre y los dos apellidos)
9Ø END
```

Si al ejecutar el programa escribimos:

```
? PEDRO
? FERNANDEZ
? RAMIREZ
```

tendremos como respuesta las líneas:

```
ALUMNO:

PEDRO          FERNANDEZ  RAMIREZ
```

Al igual que veíamos con los datos numéricos, también existe la posibilidad de etiquetar los datos alfanuméricos y saber en todo momento cuál es el dato que nos pide el ordenador:

```
1Ø REM  NOMBRES Y APELLIDOS
2Ø LET  A$="ALUMNO:"
3Ø INPUT  "NOMBRE";N$
4Ø INPUT  "PRIMER APELLIDO";A1$
5Ø INPUT  "SEGUNDO APELLIDO";A2$
6Ø PRINT  A$
7Ø PRINT
8Ø PRINT  N$,A1$,A2$
9Ø END
```

Corriendo el programa podemos tener:

```
NOMBRE? PEDRO
PRIMER APELLIDO? FERNANDEZ
SEGUNDO APELLIDO? RAMIREZ
ALUMNO:

PEDRO          FERNANDEZ  RAMIREZ
```

Con una sola línea INPUT también existe la posibilidad de tomar varios datos alfanuméricos. Por ejemplo, la línea:

```
3Ø INPUT N$,A1$,A2$
```

sirve para tomar el nombre y los dos apellidos:

```
? PEDRO,FERNANDEZ,RAMIREZ
```

Todavía más: con una sola línea INPUT se pueden tomar datos numéricos y alfanuméricos. Así, la línea 4Ø sirve para tomar el nombre de un alumno y la nota de un examen:

```
4Ø INPUT N$,X
```

Cuando salga la interrogación correspondiente escribiremos:

```
? JOSE RAMIREZ PEREZ,7
```

La variable N$ tomará el nombre completo del alumno y la variable numérica X tomará el número 7.

Como advertencia final diremos que no se puede etiquetar por separado cada uno de los datos de una misma línea INPUT, tanto si los datos son todos numéricos, como si son alfanuméricos o mezclados. Por ejemplo, es incorrecta la expresión:

```
4Ø INPUT "NOMBRE";N$,"NOTA";X
```

Son correctas las siguientes líneas:

```
4Ø INPUT "NOMBRE";N$
45 INPUT "NOTA";X
```

y también es correcta la línea:

```
4Ø INPUT "NOMBRE Y NOTA";N$,X
```

RECUERDA	
REM	Cadenas de caracteres
Zonas de la pantalla	Variables de cadena (A$)
PRINT (sin nada detrás)	
PRINT A	PRINT A$
PRINT A,	PRINT A$,
PRINT A;	PRINT A$;
PRINT A,B,C	PRINT A$, B$, C$
PRINT A;B;C	PRINT A$;B$;C$
PRINT "RADIO=";	PRINT "NOMBRE";
INPUT R	INPUT N$
INPUT "RADIO=";R	INPUT "NOMBRE";N$
INPUT N$,X	INPUT "NOMBRE Y NOTA";N$,X

EJERCICIOS

3.1) Todos los programas que diseñes desde ahora deberán comenzar por una sentencia REM y acabar por una END. Explica el papel que cumplen ambas.

3.2) ¿Cómo podemos insertar el comentario

CALCULO DEL AREA DE UN ROMBO

en un programa?

3.3) La proposición REM ¿es ejecutable por el ordenador?

3.4) Titula el siguiente programa mediante una línea REM:

```
2Ø INPUT R
3Ø A=3.1416*R↑2
4Ø PRINT A
5Ø END
```

3.5) Modifica la línea 4Ø del programa anterior para que en la impresión del resultado aparezca la leyenda AREA =.

3.6) El siguiente programa calcula la hipotenusa de un triángulo rectángulo dando los valores de los catetos. Pon título al programa y etiquetas a la entrada de datos y a la presentación de resultados:

```
2Ø INPUT C1,C2
3Ø H=(C1↑2+C2↑2)↑.5
4Ø PRINT H
5Ø END
```

3.7) Escribe una sentencia REM para el programa siguiente:

```
2Ø INPUT "RADIO=";R
3Ø L=2*3.1416*R
4Ø PRINT "LONGITUD=";L
5Ø END
```

3.8) Escribe un programa que se adapte a esta proposición:

```
1Ø REM AREA DE UN TRIANGULO
```

3.9) ¿Cómo podemos imprimir los valores de A, B y C en un sólo renglón y lo más cerca posible? ¿Cómo lograremos que los valores de la instrucción siguiente se impriman en la misma línea? Pon un ejemplo.

3.10) ¿Qué se puede hacer para que dos líneas de impresión no salgan juntas?

3.11) ¿Qué efecto produce un punto y coma al final de una línea PRINT? ¿Y una coma?

3.12) Decir qué imprime el programa:

```
1Ø REM DIFERENCIAS DE IMPRESION
2Ø LET A=5
3Ø PRINT A
4Ø PRINT "A"
5Ø END
```

3.13) ¿Qué imprime el siguiente programa?

```
 4Ø PRINT 3+2
 8Ø PRINT "3+2"
12Ø END
```

3.14) Explica qué imprimen estas líneas:

```
7Ø PRINT 3,2
8Ø PRINT
9Ø PRINT 3;2
```

3.15) ¿Qué resultado nos proporciona este programa?

```
1Ø REM EJERCICIO
2Ø PRINT A,B,C,D
3Ø END
```

3.16) Muestra cómo quedará la pantalla después de correr este programa:

```
1Ø REM EJERCICIO DE IMPRESION
2Ø PRINT 3,4
```

```
3Ø PRINT 5;6
4Ø PRINT
5Ø PRINT 7;
6Ø PRINT 8
7Ø END
```

3.17) ¿Qué sale en la pantalla cuando el ordenador ejecuta este programa?

```
1Ø REM TABLA DE VALORES
2Ø PRINT "X", "F(X)"
3Ø LET X=X+1
4Ø PRINT X,X*X
5Ø X=X+1
6Ø PRINT X,X*X
7Ø END
```

3.18) ¿Qué imprime este programa?

```
1Ø REM SUMA
2Ø PRINT "2+3=";2+3
3Ø END
```

3.19) Programa el ejercicio anterior para dos sumandos cualesquiera X e Y mediante las instrucciones INPUT correspondientes. Piensa bien la "suma" de cadenas que será necesaria para escribir el principio de la línea 2Ø.

3.20) Escribe un programa que imprima en la pantalla:

a)
```
1       2       3       4
5       6       7       8
```

b)
```
1 2 3 4 5 6 7 8
```

3.21) Simula una pantalla en papel cuadriculado y representa los resultados del siguiente programa:

```
1Ø PRINT "L1";"MADRID"
2Ø PRINT "L1"
3Ø PRINT "MADRID"
4Ø PRINT "L1","MADRID"
5Ø END
```

3.22) ¿Qué papel juega la línea 45?

```
•
•
•
4Ø PRINT A
45 PRINT
5Ø PRINT B
•
•
```

3.23) ¿Qué imprime la siguiente línea?

```
12Ø PRINT " ";
```

3.24) ¿Cómo podemos hacer aparecer en pantalla estas expresiones?

```
A =      ?
B = ?
```

3.25) ¿Qué resultado proporciona este programa?

```
1Ø REM EJERCICIO
2Ø LET A$="123"
3Ø LET B$="456"
4Ø PRINT A$+B$
5Ø END
```

3.26) ¿Qué hace el programa siguiente?

```
1Ø REM COLLAGE
2Ø LET A$="NICO"
3Ø LET B$="LAS"
4Ø LET C$=" "
5Ø PRINT A$+B$,B$+C$+A$
6Ø END
```

3.27) En pantalla hemos escrito:

```
1Ø LET B=A↑2
```

```
2Ø END
15 PRINT "B=";B
5  REM CUADRADO DE UN NUMERO
7  INPUT A
6  PRINT "A=";
```

¿Qué hay que hacer para que aparezcan las líneas ordenadas? Al correr el programa, ¿cómo se desarrolla?

3.28) Hacer un programa para calcular la nota media de 5 alumnos.

3.29) Una instrucción INPUT, ¿puede asignar valores a más de una variable? ¿Qué harías para lograrlo?

3.30) ¿Para qué sirve la línea siguiente?

```
3Ø INPUT X,Y,Z
```

3.31) ¿Qué misiones pueden tener la coma y el punto y coma en un programa en BASIC?

3.32) Diseña un programa como el del ejercicio 8 (área de un triángulo) pero de forma que la base y la altura puedan tomar cualquier valor (INPUT) y etiquetando los datos y el resultado.

3.33) Programa el cálculo de la longitud de una circunferencia y del área de un círculo, poniéndole título y etiquetando los datos y los resultados.

3.34) En una gasolinera necesitan un depósito cilíndrico. Quieren saber el volumen en función del radio y de la altura, y su precio con los siguientes datos: el espesor de la chapa de acero es 5 mm; la densidad del acero es 7,87 kg/dm^3 y el acero cuesta 180 ptas./kg. Hazles un programa que calcule la capacidad (volumen) y el precio.

3.35) Utilizando variables de cadena haz un programa que pregunte al que lo use cómo se llama, y luego le pregunte por su salud, la familia, y le hable del tiempo.

3.36) Escribir lo que queda en la pantalla después de ejecutar el programa:

```
1Ø  REM  ESCRITURA DE UN SOBRE
2Ø  N$="JOSE PEREZ GARCIA"
3Ø  C$="GRAN VIA 391"
4Ø  P$="MADRID 47"
5Ø  PRINT  "SR. ";
6Ø  PRINT  N$
7Ø  PRINT  C$
8Ø  PRINT
9Ø  PRINT  P$
1ØØ  END
```

3.37) Haz un programa semejante al anterior, pero que sirva para cualquier persona y dirección, utilizando INPUT.

3.38) Haz un programa que tome una palabra y la decline según la primera declinación latina.

3.39) Programa el "adulador cibernético" que, tras preguntar el nombre, diga muchas alabanzas. (Han de ser alabanzas un tanto impersonales; que sirvan para ambos sexos, para cualquier edad, etc.).

3.40) Pedro Hernández vive en la Plaza del Diamante, 48 - 6º. Su cuenta corriente tiene un saldo de 32.416 ptas. Haz un programa que imprima su nombre, dirección y el saldo; que después le pregunte cuánto dinero quiere ingresar (+) o sacar (−); para terminar imprimiendo el nuevo saldo.

3.41) Haz un programa que asigne las siguientes cadenas a las variables de cadena que se indican:

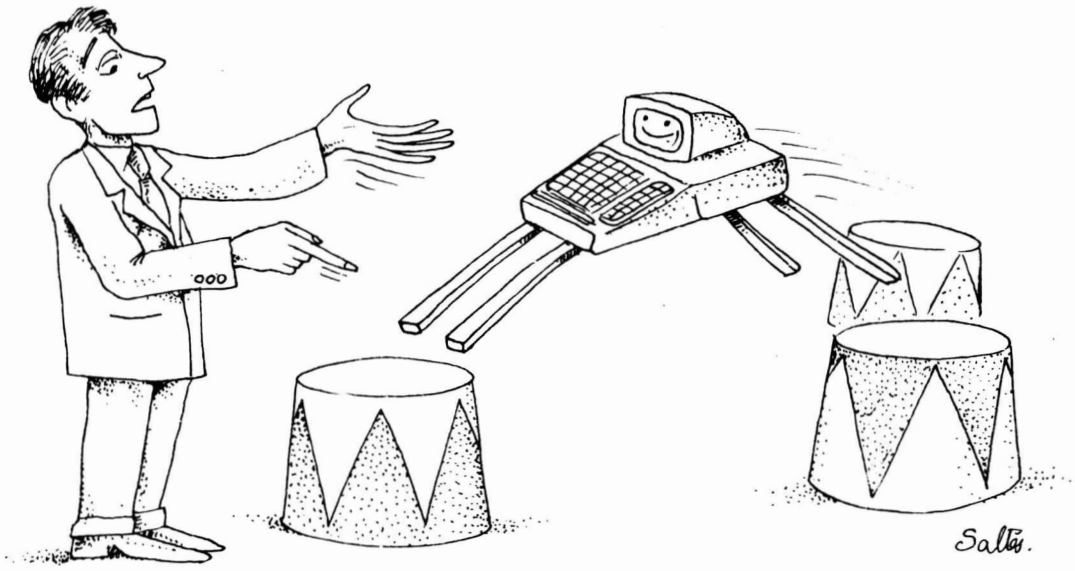
```
A1$=UNOS HILAN LA FAMA          A2$=OTROS CARDAN LA LANA
B1$=PIENSA EL LADRON            B2$=QUE TODOS SON DE SU
                                    CONDICION
C1$=PERRO LADRADOR              C2$=POCO MORDEDOR
D1$=OBRAS SON AMORES            D2$=Y NO BUENAS RAZONES
E1$=DE CASTA LE VIENE AL        E2$=TENER EL RABO LARGO
    GALGO
```

F1$=QUIEN A BUEN ARBOL SE ARRIMA
F2$=BUENA SOMBRA LE COBIJA
G1$=A BUEN ENTENDEDOR
G2$=POCAS PALABRAS BASTAN

Que el programa tenga dos salidas: primero una ordenada para que salgan los refranes correctamente escritos; después otra desordenada, por ejemplo: A1$+E2$; B1$+F2$, C1$+B2$, D1$+G2$, E1$+C2$, F1$+A2$, G1$+D2$.

4

TRANSFERENCIAS DE CONTROL



4.1 LA INSTRUCCION GO TO

Las instrucciones en el lenguaje BASIC se ejecutan normalmente según el orden creciente de sus números de línea. Pero a veces es necesario alterar este orden normal de ejecución y "saltar" a una línea posterior o anterior. Este es el papel de la instrucción GO TO (ir a).

Si en un programa en BASIC aparecen las líneas:

```
•
•
•
8Ø  GO  TO  2Ø
9Ø  LET  A=7
1ØØ  PRINT  A
•
•
```

Al llegar a la línea 8Ø se le ordena a la máquina que "salte" a la línea 2Ø. De momento las líneas 9Ø, 1ØØ, etc., no se ejecutan. Esto se llama en términos técnicos una *transferencia de control.*

Con esta nueva sentencia vamos a mejorar el programa (4) del primer capítulo. Se trataba de calcular la longitud de circunferencias según el valor que se le diera al radio. El programa terminaba al calcular la longitud de una circunferencia.

Si queremos calcular más longitudes, hay que volver a teclear RUN, y al funcionar de nuevo el programa nos pide otra vez un radio y nos da la longitud. Para calcular muchas longitudes de circunferencia nos conviene evitar el tener que teclear cada vez el comando RUN.

PROGRAMA 1.4 ANTIGUO

```
1Ø  INPUT  R
2Ø  LET  L=2*3.1416*R
3Ø  PRINT  L
4Ø  END
```

PROGRAMA 1.4 MEJORADO

```
 5  REM  CALCULO DE LONGITUDES
    DE CIRCUNFERENCIAS
1Ø  INPUT "CUAL ES EL RADIO";R
2Ø  LET  L=2*3.1416*R
3Ø  PRINT "LA LONGITUD ES"; L
35  PRINT
4Ø  GO  TO 1Ø
5Ø  END
```

La línea 35 es una línea en blanco entre dos cálculos sucesivos. Si corremos el programa mejorado aparecerá en la pantalla:

```
RUN
CUAL ES EL RADIO?
```

Si queremos el cálculo para el valor 5, tecleamos 5 RETURN y sale en la pantalla:

```
RUN
CUAL ES EL RADIO? 5
LA LONGITUD ES 31.416

CUAL ES EL RADIO?
```

Puedes ver que tras imprimir el resultado (que corresponde a la línea 3Ø del programa) deja una línea en blanco, la correspondiente a 35 PRINT, y vuelve a la línea 1Ø por orden de la instrucción 4Ø pidiendo otro valor del radio. Al teclear por ejemplo 12 la pantalla queda así:

```
RUN
CUAL ES EL RADIO? 5
LA LONGITUD ES 31.416

CUAL ES EL RADIO? 12
LA LONGITUD ES 75.3984

CUAL ES EL RADIO?
```

Y así sucesivamente. Comprueba en el microordenador la notable ventaja en rapidez del programa mejorado. En este caso la línea 5Ø END no se ejecuta nunca, pero en un programa en BASIC se debe incluir para indicar la terminación del programa.

4.2 LA INSTRUCCION IF-THEN

Los ordenadores no sólo son capaces de ejecutar una serie de cálculos a gran velocidad, sino que además tienen capacidad para tomar *decisiones lógicas.* Veamos un ejemplo:

```
•
•
3Ø  IF  X > 3  THEN  7Ø
4Ø  PRINT  X
•
•
```

Al llegar a la línea 3Ø el ordenador compara el valor de la variable X con 3. Si es X > 3, entonces la ejecución salta a la línea 7Ø; en caso contrario el programa sigue su curso normal en la línea siguiente, en nuestro caso la 4Ø.

IF.. THEN... equivale a *Si... Entonces... Si No, Continuar.* Mira el siguiente ejemplo:

```
1Ø REM  COMPRA DE DIVISAS
2Ø INPUT  "A CUANTO ESTA EL DOLAR";D
3Ø INPUT  "A CUANTO ESTA EL FRANCO SUIZO";FS
4Ø INPUT  "CUANTAS PESETAS QUIERES CAMBIAR";X
5Ø PRINT  "QUIERES COMPRAR DOLARES (1) O FRANCOS SUI-
   ZOS (2)"
6Ø INPUT  "(CONTESTA 1 O 2)";A
7Ø IF A=2 THEN 11Ø
8Ø LET N=X/D
9Ø PRINT X; "PESETAS SON AL CAMBIO";N;"DOLARES"
1ØØ GO TO 13Ø
11Ø LET  N=X/FS
12Ø PRINT  X; "PESETAS SON AL CAMBIO";N;"FRANCOS SUIZOS"
13Ø END
```

Las primeras líneas del programa te preguntan el cambio actual de las divisas y el dinero que quieres cambiar, mediante instrucciones INPUT que ya manejas bien.

Después, mediante otro INPUT, se te pregunta si quieres dólares o francos suizos. Como el mensaje era bastante largo se ha puesto parte en un PRINT en la línea 5Ø y parte como etiqueta del INPUT de la línea 6Ø. Fíjate que has de contestar 1 ó 2, que será el valor que tomará A.

En la línea 7Ø viene la decisión lógica: Si A=2, es decir, si deseas francos suizos, el cambio ha de calcularse a partir de la línea 11Ø. En esa línea se divide tu dinero (X) por el cambio del franco suizo (FS) y resulta la cantidad de francos suizos que te corresponde; se imprime el resultado en la línea 12Ø, y se termina el programa.

```
A CUANTO ESTA EL DOLAR? 9Ø
A CUANTO ESTA EL FRANCO SUIZO? 5Ø
CUANTAS PESETAS QUIERES CAMBIAR? 63ØØØ
QUIERES COMPRAR DOLARES (1)  O  FRANCOS SUIZOS (2)
```

(CONTESTA 1 O 2)? 2
63ØØØ PESETAS SON AL CAMBIO 126Ø FRANCOS SUIZOS

Arriba ves el desarrollo del programa en el que hemos respondido a las interrogaciones (INPUT) con diversos valores, y a la última con un 2.

Pero si deseabas dólares habrás tecleado 1, y como A entonces no vale 2, la condición del IF no se cumple y la ejecución continúa en la línea 8Ø, donde dividiendo tu dinero (X) por el cambio del dólar (D) se obtienen los dólares que te darán a cambio. Después, en la línea 9Ø, se imprime el resultado.

Ahora bien, impreso el resultado por la línea 9Ø se terminó el cálculo que te interesaba, y ponemos un GO TO que envíe el control al final del programa. De lo contrario se ejecutarían a continuación las líneas correspondientes a los francos suizos, lo que en este momento no te interesa.

Si deseas dólares y contestas con un 1 al INPUT correspondiente la pantalla queda así:

A CUANTO ESTA EL DOLAR? 9Ø
A CUANTO ESTA EL FRANCO SUIZO? 5Ø
CUANTAS PESETAS QUIERES CAMBIAR? 63ØØØ
QUIERES COMPRAR DOLARES (1) O FRANCOS SUIZOS (2)
(CONTESTA 1 O 2)? 1
63ØØØ PESETAS SON AL CAMBIO 7ØØ DOLARES

Este es el estado en que queda la pantalla si en vez de 2 tecleas un 1. Si el programa no tuviera a continuación la línea 1ØØ GO TO 13Ø aparecería debajo de lo anterior:

63ØØØ PESETAS SON AL CAMBIO 126Ø FRANCOS SUIZOS

lo que estropea la claridad, pues deseábamos comprar dólares esta vez.

La instrucción IF-THEN es una bifurcación condicional; hay transferencia de control (salto a otra línea) sólo si la condición del IF se cumple; a diferencia de la instrucción GO TO que es una bifurcación incondicional, inexorable. En la mayoría de los ordenadores la instrucción IF-THEN se puede escribir también así: IF–GO TO (Si–ir a) que te ayudará a distinguir los dos tipos de bifurcaciones: con o sin condición.

4.3 OPERADORES LOGICOS

Modifiquemos algunas líneas del programa (1):

```
5Ø PRINT "QUIERES COMPRAR DOLARES O FRANCOS SUIZOS"
6Ø INPUT "ESCRIBIR LA MONEDA CON TODAS LAS LETRAS";A$
7Ø IF A$="FRANCOS SUIZOS" THEN 11Ø
```

El programa es sustancialmente idéntico. Ha variado la contestación, que ya no es un número sino el nombre de las divisas, y la condición de la instrucción IF-THEN, que ahora compara una variable de cadena con una cadena.

Han salido ya tres ejemplos de condiciones de la instrucción IF-THEN:

```
X > 3
A=2
A$="FRANCOS SUIZOS"
```

Vamos a estudiar de forma sistemática las condiciones que puede encerrar esta instrucción. Las condiciones se caracterizan por dos términos de comparación a derecha e izquierda de un símbolo que es el operador lógico.

SIMBOLO U OPERADOR	SIGNIFICADO
=	es igual a
<	es menor que
>	es mayor que
<=	es menor o igual que
>=	es mayor o igual que
<>	es distinto de

Te será fácil recordar estos símbolos, muy parecidos a los que has utilizado en Matemáticas. Algunas pequeñas variantes se deben a que los ordenadores no tienen los signos $\leqslant$, $\geqslant$ ni $\neq$ y ha de escribirse en su lugar <=, >= ó <>, con cuidado de no alterar el orden.

A ambos lados del operador puede haber no sólo números y variables sino también expresiones o fórmulas. Son válidas por ejemplo las siguientes líneas:

```
4Ø IF (X+Y)/2 <>(X-Y)/2 THEN 71Ø
4Ø IF XØ-5< XØ↑2+5*A THEN 71Ø
4Ø IF (AZ↑2-X)/(5*A)<=AZ+A↑2 THEN 71Ø
```

Hasta aquí la instrucción IF-THEN tal como fue ideada por los creadores del BASIC en los años sesenta. Pero el lenguaje BASIC ha incorporado una serie de mejoras que, sin alterar su diseño inicial, lo han potenciado notablemente. La instrucción IF-THEN es quizá la que más se ha enriquecido con nuevas posibilidades. Citaremos sólo dos, cuya implantación está muy generalizada, son casi universales.

Veámoslo con un ejemplo: ¿qué sucede en el programa de la compra de divisas si se comete un pequeño error al teclear la divisa que se desea? Imagínate que has dado las siguientes respuestas al ordenador:

```
A CUANTO ESTA EL DOLAR? 9Ø
A CUANTO ESTA EL FRANCO SUIZO? 5Ø
CUANTAS PESETAS QUIERES CAMBIAR? 63ØØØ
QUIERES COMPRAR DOLARES O FRANCOS SUIZOS
ESCRIBIR LA MONEDA CON TODAS LAS LETRAS? FRACOS SUIZOS
                                    (Habiendo omitido la N)
```

Entonces A$="FRACOS SUIZOS" ≠ "FRANCOS SUIZOS", luego la condición del IF no se cumple, pasa a ejecutarse la línea 8Ø, y con gran asombro por tu parte verás aparecer en la pantalla:

```
63ØØØ PESETAS SON AL CAMBIO 7ØØ DOLARES
```

El pequeño error al teclear ha producido una respuesta desconcertante, ha calculado la otra divisa. Pero hay una forma de evitar estos errores, intercalando por ejemplo las siguientes líneas:

```
62 IF A$="DOLARES" OR A$="FRANCOS SUIZOS" THEN 7Ø
64 PRINT "HAS COMETIDO ALGUN ERROR AL TECLEAR"
66 PRINT "REPITE DE NUEVO POR FAVOR"
68 GO TO 5Ø
```

La condición del IF de la línea 62 dice que si has tecleado DOLARES ó FRANCOS SUIZOS, el programa continúa en la línea 7Ø como antes. Si has tecleado otra cosa, hay algún error, y te vuelve a formular la pregunta de las líneas 5Ø y 6Ø. Como ves el operador OR (equivalente a O) liga dos condiciones, y basta que se cumpla la primera o la segunda para que se verifique el salto o transferencia de control.

De forma similar se podía haber escrito:

```
62 IF A$<>"DOLARES" AND A$ <>"FRANCOS SUIZOS" THEN 65
64 GO TO 7Ø
65 PRINT "HAS COMETIDO ALGUN ERROR AL TECLEAR"
67 PRINT "REPITE DE NUEVO POR FAVOR"
69 GO TO 5Ø
```

Se tiene el mismo efecto pero ahora utilizando AND (y). Si A$ ≠ "DOLARES" y además A$ ≠ "FRANCOS SUIZOS", es decir si hay un error al teclear, la ejecución continúa en la línea 65 con el mismo mensaje que en el ejemplo anterior. Como ves el operador AND liga dos condiciones, y sólo hay salto o transferencia de control si se cumplen las dos. Basta que una no se cumpla, por ejemplo que A$="FRANCOS SUIZOS", para que la ejecución continúe en la línea siguiente, la 64, que a su vez nos envía a la 7Ø para la normal terminación del programa.

La instrucción IF-THEN también se puede escribir así: IF-THEN GO TO. Por ejemplo:

```
17Ø IF X↑2<Y+5 THEN GO TO 32Ø
```

Pero detrás de THEN, en vez de GO TO se pueden poner también otras instrucciones como:

```
17Ø IF X↑2<Y+5 THEN PRINT "X=";X
17Ø IF X↑2<Y+5 THEN LET X=Y-3
```

4.4 ORGANIGRAMAS O DIAGRAMAS DE FLUJO

Veamos un sencillo programa que nos da los cuadrados de los N primeros números naturales:

```
1Ø REM CUADRADOS DE LOS N PRIMEROS NUMEROS
2Ø INPUT N
3Ø LET I=I+1
4Ø LET C=I*I
5Ø PRINT "EL CUADRADO DE";I;" ES";C
6Ø IF I<N THEN 3Ø
7Ø END
```

Explicaremos brevemente este programa:

a) Con la línea 2Ø el ordenador nos pide el dato N

b) En la línea 3Ø hemos establecido un *contador*. En efecto: cada vez que el ordenador ejecuta esta línea, la variable I va tomando sucesivamente los valores 1, 2, 3, ... Observa que antes de la ejecución de esta línea por primera vez, la variable I vale Ø (ya que no se le ha asignado ningún valor previamente). La primera vez que se ejecute quedará: I=Ø+1=1 es decir, I tomará el valor 1. La segunda vez: I=1+1=2, etc.

c) En la línea 4Ø se calcula el cuadrado del número y se guarda en C.

d) Con la línea 5Ø se imprime el número y su cuadrado.

e) En la línea 6Ø se controla el valor de I, y si todavía no se ha llegado hasta el número N, se recomienza el proceso a partir de 3Ø. Por el contrario, si I es igual a N (ya no es menor) se pasa a la línea siguiente donde termina la ejecución del programa.

Todo este proceso se acostumbra a expresar gráficamente utilizando unos símbolos convenidos que explicaremos más adelante:

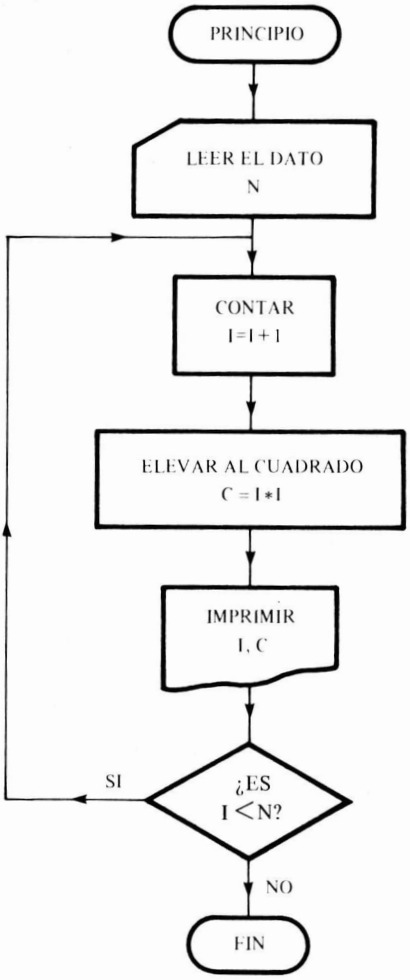


Fig. 4.1

Este esquema se conoce con el nombre de *organigrama o diagrama de flujo.* En él se recoge, a grandes rasgos, la sucesión de operaciones que hay que llevar a cabo para conseguir un determinado objetivo.

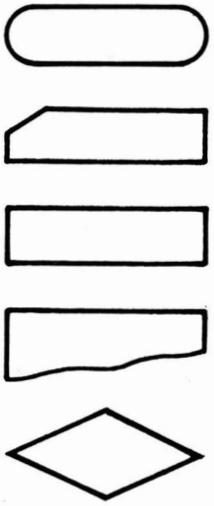
Hasta ahora hemos trabajado con programas sencillos que no necesitaban diseñar previamente el organigrama. A medida que en los programas proliferan las instrucciones GO TO, IF-THEN y otras que veremos, son más útiles los organigramas, especialmente durante la etapa de aprendizaje.

El diagrama permite seguir de una manera clara los pasos que dará el ordenador al ejecutar el programa. Es conveniente hacerlo y, antes de programar. recorrer mentalmente el organigrama como si el ordenador fueras tú, para com-

probar que no hay errores. Tienes que esforzarte para que los programas funcionen a la primera, y el organigrama es una ayuda muy interesante.

Los símbolos convenidos que nosotros vamos a utilizar para la representación de los organigramas son los siguientes:

- El principio y fin se representa por elipses o figuras similares.
- La entrada de datos se representa por
- La asignación de valores a variables o la realización de cálculos se enmarcan en rectángulos.
- La impresión de resultados, expresiones, etc., por
- Las decisiones lógicas se representan por un rombo.



4.5 DOS EJEMPLOS INTERESANTES

Para iniciarnos en el diseño de organigramas, y su posterior traducción a un programa en BASIC, vamos a ver dos ejemplos:

Ejemplo 1:

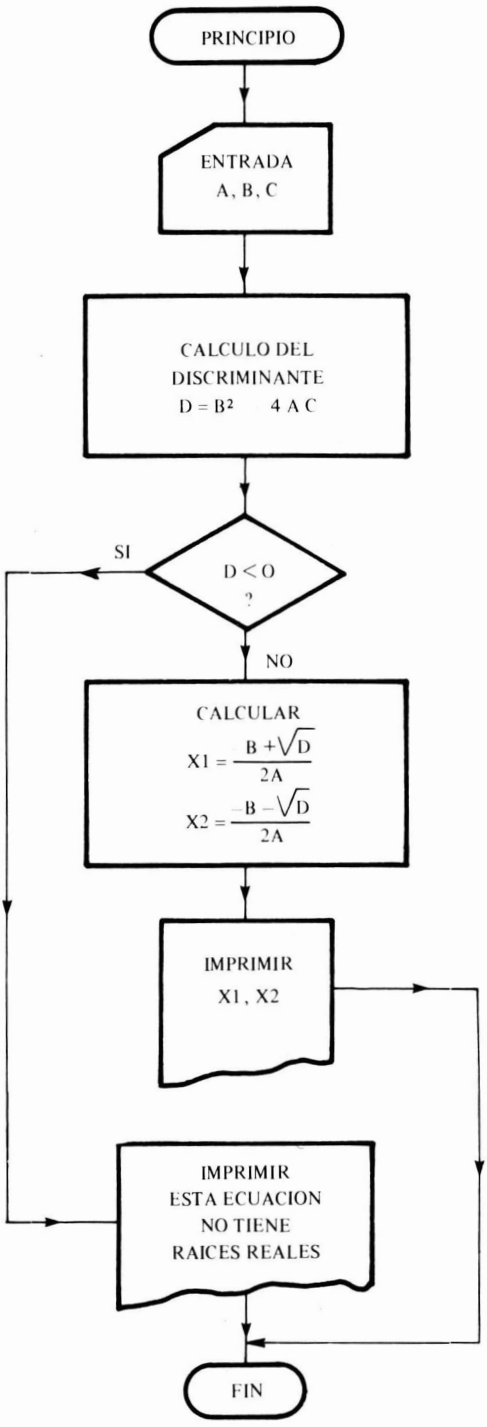
Hallar las *raíces reales* de la ecuación de segundo grado $AX^2 + BX + C=O$, donde A, B y C son números reales cualesquiera y $A \neq O$. Sabido es que la fórmula que nos da las raíces de la ecuación es:

$$X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

que en realidad se desdobla en estas dos:

$$X1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \qquad X2 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

Para que se puedan hallar estas raíces reales es necesario que el discrimi-



PROGRAMA

```
1Ø REM ECUACION DE SEGUNDO GRADO
2Ø INPUT "DA VALORES PARA A, B Y C"; A, B, C
3Ø LET D=B↑2-4*A*C
4Ø IF D<Ø THEN 11Ø
5Ø LET X1=(-B+D↑.5)/(2*A)
6Ø LET X2=(-B-D↑.5)/(2*A)
7Ø PRINT "LAS RAICES SON:"
8Ø PRINT
9Ø PRINT "X1=";X1,"X2=";X2
1ØØ GO TO 12Ø
11Ø PRINT "LA ECUACION NO TIENE RAICES REALES"
12Ø END
```

Fig. 4.2

nante de la ecuación, $D=B^2 - 4AC$, no sea negativo. (Si D=0 las dos raíces resultarán iguales).

En la página anterior damos un organigrama y en paralelo ponemos el correspondiente programa en BASIC para que se pueda cotejar.

Ejemplo 2:

Imaginemos que tenemos una serie de datos:

$$174, 163, 168, 174, \ldots$$

que, por fijar ideas, supondremos que son las tallas, en centímetros, de una población. Se trata de diseñar un organigrama y el correspondiente programa que nos dé la media aritmética de los datos introducidos uno a uno en el ordenador.

Recordemos que para calcular la media se utiliza la fórmula:

$$M = \frac{\text{Suma de los datos}}{\text{Número de datos}}$$

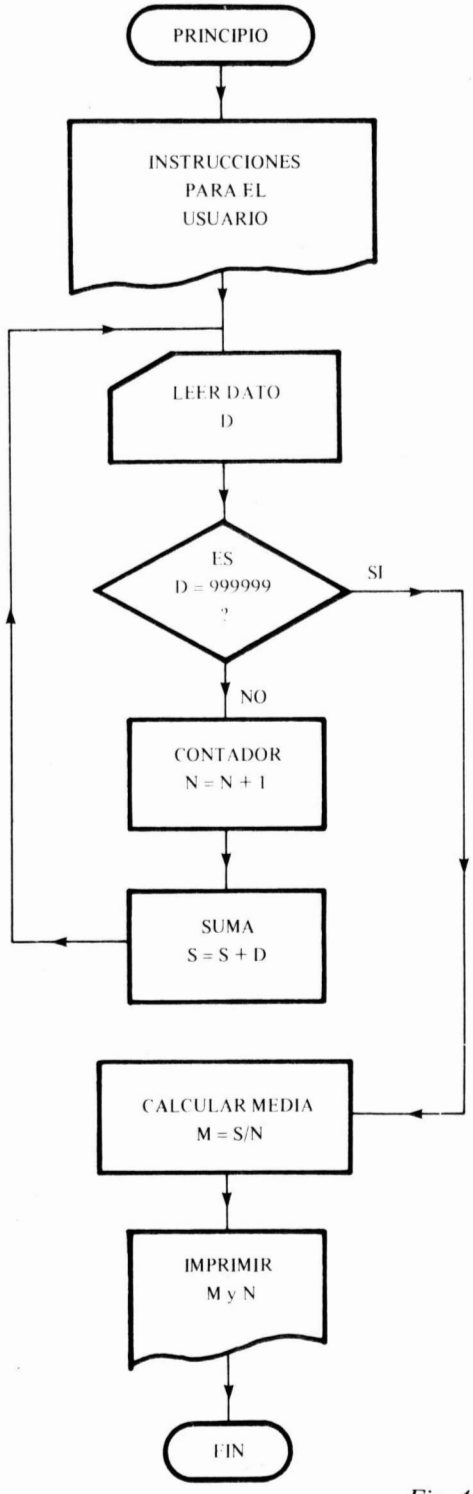
Para poder ejecutar el programa no es necesario conocer de antemano el número de datos: podemos conseguir que sea el propio ordenador el que vaya contándolos a medida que los vamos introduciendo.

Estableceremos, pues, una variable N que representará el número de datos introducidos hasta el momento (contador) y otra variable S que recogerá la suma parcial de los datos (al final en S estará la suma total, en N el número total y la media será M = S/N).

El único problema que se presenta es el de cómo avisar al ordenador de que ya hemos acabado de introducir el último dato y que tiene que proceder a calcular la media e imprimir el resultado. Esto se puede resolver introduciendo en la lista un último *dato ficticio (el testigo)* que en este ejemplo puede ser el dato falso 999999, ya que no va a haber ninguna persona con esta talla. Es claro que este dato ni se suma ni se cuenta; únicamente dice que el dato anterior a él era el último de la lista.

Para que el programa pueda ser usado por una persona distinta del programador, es conveniente que imprima unas instrucciones previas para el usuario.

Recogiendo todas estas ideas tenemos el organigrama y programa siguientes:




```
1Ø REM MEDIA DE UNA SERIE
2Ø PRINT "CADA VEZ QUE
   APAREZCA UNA INTERRO-
   GACION";
3Ø PRINT "EN LA PANTALLA
   ESCRIBE UN NUMERO DE
   LA";
4Ø PRINT "SERIE Y PULSA
   RETURN"
5Ø PRINT "DESPUES DEL UL-
   TIMO DATO INTRODUCE";
6Ø PRINT "EL DATO FICTICIO
   999999 Y PULSA RETURN"

7Ø INPUT "DATO";D


8Ø IF D=999999 THEN 12Ø


9Ø LET N=N+1


1ØØ LET S=S+D
11Ø GO TO 7Ø


12Ø LET M=S/N


13Ø PRINT "LA MEDIA ES M=";M

14Ø PRINT

15Ø PRINT "EL NUMERO DE
   DATOS ES N=";N


16Ø END
```

Fig. 4.3

RECUERDA	
Bifurcación incondicional	GO TO
Bifurcación condicional	IF - THEN
Contador	IF - GO TO
Organigrama	
Operador lógico	
=	
>	
<	
<>	
<=	
>=	
OR	
AND	

EJERCICIOS

4.1) Haz un programa que escriba en la pantalla los 20 primeros números naturales.

4.2) Castiga al ordenador con un programa que le haga escribir 100 veces "VACA SE ESCRIBE CON V".

4.3) Haz un programa que escriba los 15 primeros números naturales y sus cuadrados y cubos (utiliza comas para la impresión).

4.4) Haz un programa similar al anterior que escriba los números, su raíz cuadrada y su raíz cúbica. La presentación podría ir precedida de dos líneas de encabezamiento como las siguientes:

NUMERO RAIZ CUADRADA RAIZ CUBICA

4.5) Mejora el ejercicio 3.36 ESCRITURA DE UN SOBRE, poniendo justo antes del END un GO TO 2Ø, y sustituyendo las instrucciones LET de las líneas 2Ø, 3Ø y 4Ø por INPUT. Así te preguntará los datos para escribir muchos sobres, igual que hemos hecho en la primera pregunta de este capítulo con el programa 1.4.

4.6) Mejora el programa de las divisas en su primera versión (contestación con un número) pero añadiendo las libras esterlinas, marcos alemanes y yens japoneses. Te quedará mejor si justo antes del END añades un GO TO a la línea en que se pregunta "CUANTAS PESETAS QUIERES CAMBIAR".

4.7) Imaginando que el ordenador no sabe multiplicar, haz un programa de multiplicación de números enteros por adiciones sucesivas. (Puedes utilizar variables enteras, A% por ejemplo, pero entonces no podrás exceder su límite: 32 767).

4.8) Suponiendo que el ordenador no sabe dividir, haz un programa de división de dos números por sustracciones sucesivas. División entera que al final dé el cociente sin decimales y el resto.

4.9) Una bañera tiene un grifo y un sumidero. La bañera tiene capacidad de L litros, el grifo echa G litros por minuto y el sumidero desagua S litros por minuto. Si la operación empezó cuando la bañera tenía LØ litros, calcula cuánto tarda en vaciarse o llenarse, según que G sea mayor o menor que S. (Si son iguales el programa no acaba nunca por lo que debes sacar un mensaje). Haz el programa no mediante ecuaciones, sino viendo de minuto en minuto el estado de la bañera. Al final podrías pasar el resultado a horas y minutos.

4.10) Dados tres números cualesquiera haz un programa que los ordene de menor a mayor. (Puedes utilizar IF-THEN con OR y AND). Te conviene hacer el organigrama.

4.11) ¿Cuántas campanadas da un reloj desde la hora H hasta las 24 horas? Haz un programa.

4.12) Mejora el programa del ejercicio 3.39) sobre el adulador cibernético, de forma que las alabanzas que diga el ordenador varíen con el sexo y edad del usuario (Puedes utilizar IF-THEN con AND).

4.13) Haz un programa que sepa encontrar en una serie de diez números el mayor de ellos.

4.14) Haz un programa que pida dinero al usuario hasta sobrepasar una cantidad que figure en la primera línea del programa (cantidad desconocida para el usuario). Al sobrepasar la cantidad el ordenador ha de decir: "YA TENGO BASTANTE. NECESITABA... PESETAS. LUEGO ME HAS DADO... PESETAS DE MAS".

4.15) Haz un programa que calcule el factorial de un número. Al ejecutarlo no funcionará para números superiores a 33, pues se rebasa el valor máximo de las variables numéricas del BASIC. Trata de añadirle después la condición de que el número sea entero, y que saque mensaje de error en caso contrario.

4.16) A alguno de los programas que hayas hecho añádele al principio unas líneas para conseguir lo siguiente: el programa va a tener una clave y sólo podrá utilizarlo quien la conozca. Nada más empezar preguntas al usuario la clave. Si la sabe puede seguir el problema, si no sacas un mensaje como: "TOP SECRET. COMO NO SABES LA CLAVE NO TIENES ACCESO AL PROGRAMA".

4.17) En una tienda automatizada de ultramarinos hay J Kg de judías, L de lentejas y G de garbanzos, siendo sus precios respectivos P1, P2 y P3. Nuestro ordenador, que es el vendedor, pregunta qué producto quiere comprar el usuario (es más cómodo, como en el caso de la compra de divisas, que seleccione el producto que le interesa por un número: 1, 2 ó 3). Después pregunta qué cantidad y responde una de dos: o que no tiene tanto, o que sí tiene, y el precio total de la compra. Por fin pregunta cuánto le pagamos, y termina contestando lo que nos devuelve (prever que algún "listo" podría tratar de pagar de menos).

4.18) Escribe un programa que empiece pidiéndote un número N. Luego calculará la suma S de $1^2 + 2^2 + 3^2 + \ldots$ hasta que esa suma sea igual o mayor que N. Entonces debe escribir N, S y el último número que ha elevado al cuadrado y sumado a S.

4.19) Haz un programa que escriba los 50 primeros números, otro que escriba los 80 primeros y otro los primeros 200 números. Como nuestra pantalla sólo tiene 25 líneas, para que quepan todos en la pantalla necesitarás usar comas o puntos y comas.

4.20) Utiliza el algoritmo del ejercicio 4.8 (división entera) para decir si un número es divisible por 3 ó no (resto 0 ó no).

4.21) Haz un programa que escriba todos los números menores que 50 y no divisibles por 3.

4.22) Interesa hacer un programa de conversión de longitudes para los ingleses. El ordenador pedirá una longitud en metros, y la convertirá a yardas, pies y pulgadas. Una yarda tiene 0,9144 m , y está dividida en 3 pies, y cada pie en 12 pulgadas. (Utiliza el algoritmo de división entera).

4.23) Programar la entrada a la cueva de Alí Babá. La cueva sólo se abre si se pronuncia la frase mágica.

4.24) Taquilla automática: un ordenador controla el número de plazas libres en un autobús de 40 plazas. A medida que la gente le va pidiendo uno o varios billetes quedan menos plazas libres. Si, por ejemplo, alguien pide cinco plazas y sólo le quedan tres contestas con el siguiente mensaje: "NO TENGO TANTAS PLAZAS. SOLO ME QUEDAN 3. LE INTERESAN?". Cuando el autobús esté completo saca el mensaje de COMPLETO y se termina el programa.

4.25) Una línea de helicópteros tiene paradas en Santander, Palencia y Valladolid (los trayectos son 1 y 2). El helicóptero tiene 16 plazas. Haz un programa semejante al anterior, pero el mensaje de COMPLETO sólo aparecerá cuando lo estén los dos trayectos. (Utiliza OR y AND).

4.26) Quieres que el ordenador te pregunte mediante un INPUT las notas de los 35 alumnos de la clase, y que vaya contando los muy deficientes (MD $<$ 2), insuficientes (2 $\leqslant$ I $<$ 5), suficientes (5 $\leqslant$ S $<$ 6), bienes (6 $\leqslant$ B $<$ 7), notables (7 $\leqslant$ N $<$ 8,5) y sobresalientes (8,5 $\leqslant$ SB). Al final deberá aparecer en pantalla el número de alumnos con cada nota. Haz un programa para esto.

4.27) En una tienda venden pañuelos por cajas de 10, y también sueltos. Los hay de diversas calidades: de 1200, 1500 y 1800 ptas./caja. Llega un comprador. Si pide cajas enteras se le venden sin más, pero si pide pañuelos sueltos se recarga el precio: un 10% si cada pañuelo vale menos de 140 ptas. y un 5% si vale más. Haz un programa que dé el importe total según el número de pañuelos pedidos (todos de la misma calidad).

4.28) Haz un programa para enseñar a un niño la tabla de multiplicar. Primero le preguntarás (INPUT) qué número quiere repasar, y él dirá, por ejemplo, el 7. Entonces sacas en la pantalla 7*1=?. Si el alumno contesta bien ha de salir el mensaje "RESPUESTA CORRECTA", si no el mensaje sería "TE HAS EQUIVOCADO. LA RESPUESTA ES 7*1=7". Luego lo mis-

mo con 7*2 y así hasta 7*9. Al final sacarás en la pantalla el número de errores cometidos, que si es menor o igual que 2 le dirán al niño que ha aprobado, y si no que ha suspendido.

4.29) Retoca el programa anterior para aplicarlo a alumnos más retrasados. Si el alumno se equivoca ante la pregunta 7*3=?, sacarás el mensaje: "LA RESPUESTA NO ES CORRECTA. PRUEBA OTRA VEZ". Y si se equivoca de nuevo, le dices como antes "TE HAS EQUIVOCADO. LA RESPUESTA ES 7*3=21". (Te interesa hacer el organigrama, y puedes utilizar OR o AND).

4.30) Haz un programa para calcular el recibo de la luz, que se obtiene de la siguiente forma: en concepto de "POTENCIA CONTRATADA", que suelen ser 5 ó 6 Kw en una casa, se pagan unas 150 ptas./Kw; esto es un gasto fijo. La energía consumida medida en Kw.h se paga a unas 10 ptas. el Kw.h si no se ha llegado a rebasar el primer bloque, cuyo límite es POTENCIA CONTRATADA*200. Los Kw.h que exceden del primer bloque se pagan a 9 ptas./Kw.h. Un ejemplo de recibo puede ser el siguiente:

POTENCIA			ENERGIA CONSUMIDA					
			Bloque 1			Bloque 2		
Kw	Pts./Kw	Pts.	Kw.h	Pts./Kw.h	Pts.	Kw.h	Pts./Kw.h	Pts.
6	150	900	1200	10	12000	129	9	1161
							TOTAL	14061

Puedes poner las tarifas (150, 10 y 9) como asignaciones LET, y la potencia contratada (6) y la energía consumida (1329) como entradas INPUT. Si se hubieran consumido menos de 6·200=1200 Kw.h, todos se facturarían en el primer bloque.

4.31) La raíz cuadrada de un número real positivo se puede aproximar mediante el siguiente algoritmo: se calculan los términos de la sucesión

X1=1, X2=(Q/X1+X1)/2, X3=(Q/X2+X2)/2, X4=(Q/X3+X3)/2, etc.

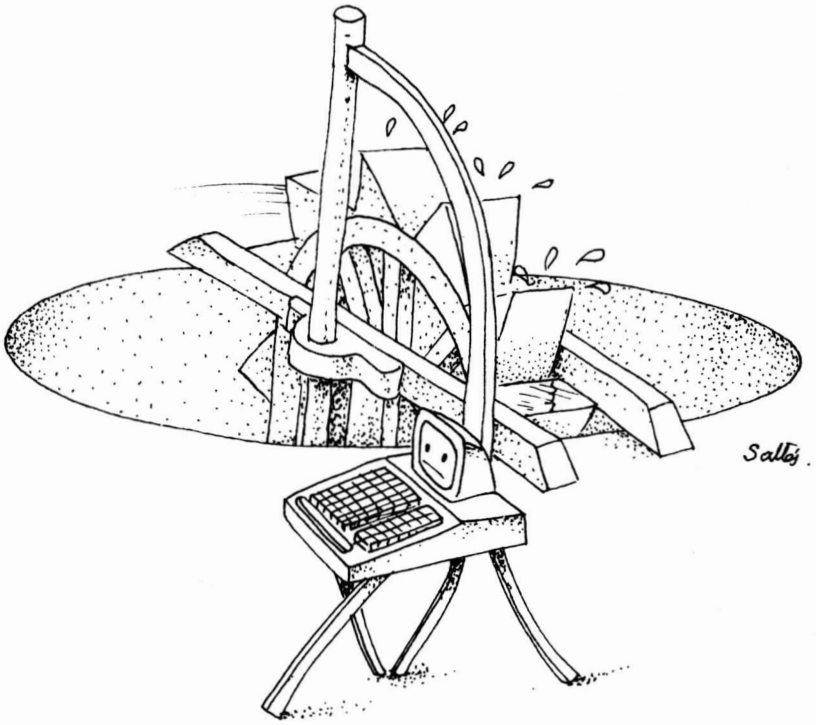
Haz un programa que, tras preguntar el valor de Q, vaya calculando e imprimiendo X1, X2, ... hasta que dos términos consecutivos se diferencien menos que 0,0001. Darás ese resultado aproximado por bueno, y como comprobación escribe al final Q↑.5

4.32) Haz el organigrama del siguiente programa:

```
10 REM ALGORITMO DE BISECCION PARA HALLAR EL LOGA-
   RITMO DECIMAL DE UN NUMERO X > 1
20 INPUT "ESCRIBE UN NUMERO MAYOR QUE 1";X
30 INPUT "CUANTAS CIFRAS ENTERAS TIENE X";K
40 LET A=1*10↑(K-1)
50 LET B=1*10↑K
60 INPUT "SE DESEA UN ERROR MENOR QUE";E
70 LET M=K-1
80 LET N=K
90 IF E<N-M THEN 190
100 LET Q=(M+N)/2
110 LET C=(A*B)↑.5
120 IF X>C THEN 160
130 LET B=C
140 LET N=Q
150 GO TO 90
160 LET A=C
170 LET M=Q
180 GO TO 90
190 PRINT M;"< LOG (";X;")<";N
200 END
```

5

PROCESOS ITERATIVOS: BUCLES



5.1 CICLOS FOR–NEXT

Con frecuencia queremos que el ordenador repita una serie de instrucciones muchas veces, para distintos valores de una variable, que se va incrementando cada vez según una cantidad prefijada. Un sencillo ejemplo va a aclararnos esto:

Supongamos que deseamos imprimir los cuadrados de todos los números naturales comprendidos entre 5 y 12, ambos incluidos. Aquí, la acción que hay que repetir ocho veces es "elevar un número I al cuadrado e imprimir el resulta-

do". Y esto hay que hacerlo primero con I=5, luego con I=6, y así sucesivamente, hasta llegar a hacerlo con I=12. En este ejemplo la variable I se incrementa de uno en uno.

Pues bien, el ordenador es un trabajador infatigable y puede ejecutar la acción cuantas veces nosotros queramos, con tal que lo programemos bien. Para ello contamos con unas instrucciones que en nuestro ejemplo son:

```
    FOR  I=5 TO 12      (desde I=5 hasta 12)
y   NEXT  I             (siguiente I)
```

He aquí un programa que resuelve el problema anterior:

```
1Ø  REM  CUADRADOS DE NUMEROS DEL 5 AL 12
2Ø  FOR  I=5  TO  12
3Ø       LET  C=I*I     ; (Estas líneas se repiten 8 veces, con I=5,
4Ø       PRINT  C       )  6, ..., 12).
5Ø  NEXT  I
6Ø  END
```

El resultado de la ejecución del programa es la impresión:

```
 25
 36
 49
 64
 81
1ØØ
121
144
```

El programa presenta la típica estructura de los *ciclos* FOR-NEXT: la línea 2Ø FOR I=5 TO 12 indica que todo lo que va entre ella y la instrucción NEXT I, en nuestro caso las líneas 3Ø y 4Ø, se va a repetir para los valores I=5, I=6, ..., I=12. Fíjate que la variable que aparece tras FOR (I en este caso) ha de ser la misma que figure detrás de NEXT.

Para diseñar los organigramas en los que aparezcan los ciclos FOR-NEXT, o *bucles,* utilizaremos los siguientes convenios gráficos:

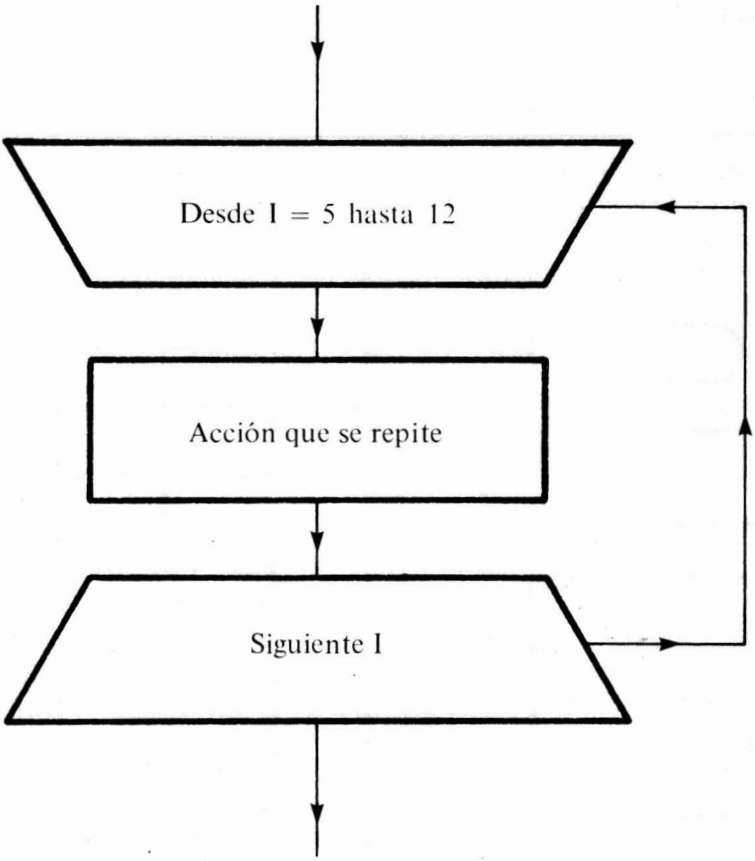


Fig. 5.1

Este problema se podría haber resuelto de otra forma con los recursos de los capítulos anteriores: partiendo de I=5, se establece un contador I=I+1, y una instrucción IF para que cuando el contador sobrepase el valor 12 no se ejecute la bifurcación y se termine el programa. Un programa que responde a estas ideas es:

```
1Ø REM CUADRADOS DE NUMEROS DEL 5 AL 12
2Ø LET I=5
3Ø LET C=I*I
4Ø PRINT C
5Ø LET I=I+1
6Ø IF I<=12 THEN 3Ø
7Ø END
```

Resulta interesante comparar los organigramas correspondientes a los dos programas anteriores:

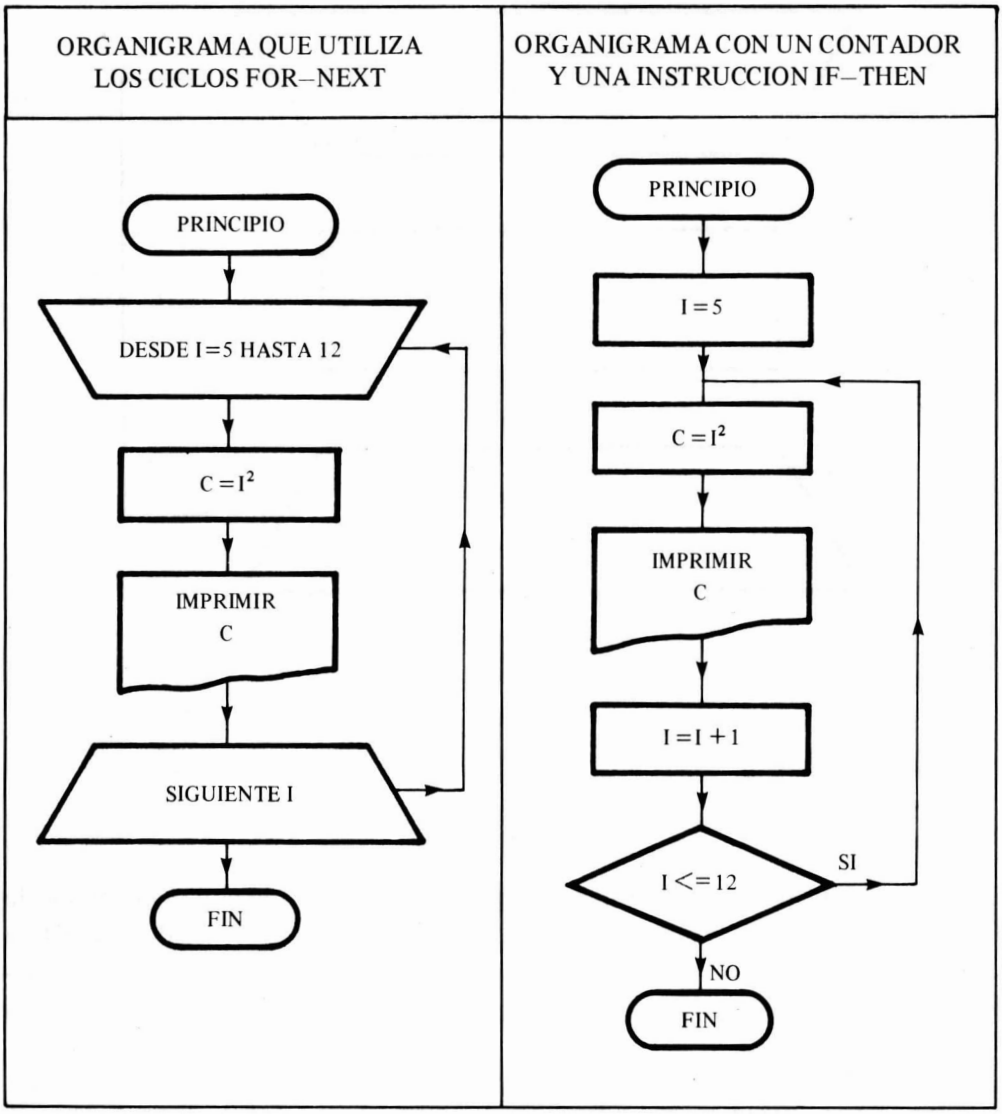


Fig. 5.2

En realidad el organigrama de la derecha "explica" el de la izquierda. Entre otras cosas puedes observar que:

1) La acción siempre se ejecuta por lo menos una vez, puesto que siempre se realiza para el valor inicial de I, ya que el control sobre I se realiza una vez que la acción del bucle ha sido ejecutada.

2) El *valor final* de la variable I, fuera ya de los ciclos, es siempre más grande que el *límite superior* de I. En nuestro ejemplo el límite superior de I es 12 y el valor final de I es 13; pero observa que no se ejecuta la acción para I=13.

Vamos a utilizar los ciclos FOR-NEXT entre unos límites que podríamos calificar de "atípicos", pero que sirven para clarificar los dos puntos anteriores:

```
1Ø FOR I=5 TO 5
2Ø       LET C=I*I
3Ø       PRINT C
4Ø NEXT I
5Ø PRINT "VALOR FINAL DE I FUERA DEL CICLO" ; I
6Ø END
```

Este programa imprime:

```
25
VALOR FINAL DE I FUERA DEL CICLO 6
```

En el programa anterior vamos a modificar la línea 1Ø, sustituyéndola por la siguiente:

```
1Ø FOR I=5 TO 2
```

El programa modificado imprime el mismo resultado que el primitivo. Que la I vaya "aumentando" desde 5 hasta 2 es absurdo. Lo que sucede es que primero se realiza el bucle para I=5 y después se comprueba que I ha rebasado ya el límite 2.

5.2 STEP: PASO

La instrucción FOR-NEXT es la sentencia reina del BASIC. Vamos a sacarle todo su jugo, toda su potencia, en los apartados que siguen.

¿Cómo escribirías un programa para obtener los cuadrados de los números pares hasta 20? Si estuviéramos aún en la lección anterior utilizarías una bifurcación y un contador, pero esta vez el contador iría de dos en dos: I=I + 2. Un programa podría ser:

```
1Ø REM CUADRADOS DE LOS NUMEROS PARES DE 2 A 2Ø
2Ø LET I=2
3Ø LET C=I*I
4Ø PRINT C
5Ø LET I=I+2
6Ø IF I<2Ø THEN 3Ø
7Ø END
```

Y el resultado de su ejecución sería en la pantalla:

```
 4
 16
 36
 64
 1ØØ
 144
 196
 256
 324
```

La manera de escribir este programa utilizando la instrucción FOR-NEXT es la siguiente:

```
1Ø REM CUADRADOS DE LOS NUMEROS PARES DE 2 A 2Ø
2Ø FOR I=2 TO 2Ø STEP 2←Observa el cambio
3Ø       LET C=I*I
4Ø       PRINT C
5Ø NEXT I
6Ø END
```

Hemos incluido en la instrucción FOR de la línea 2Ø el STEP (paso o salto), que indica cuánto se va a incrementar la variable I en cada nueva ejecución de las instrucciones del bucle o iteracción. En el organigrama habrá que incluir el STEP:

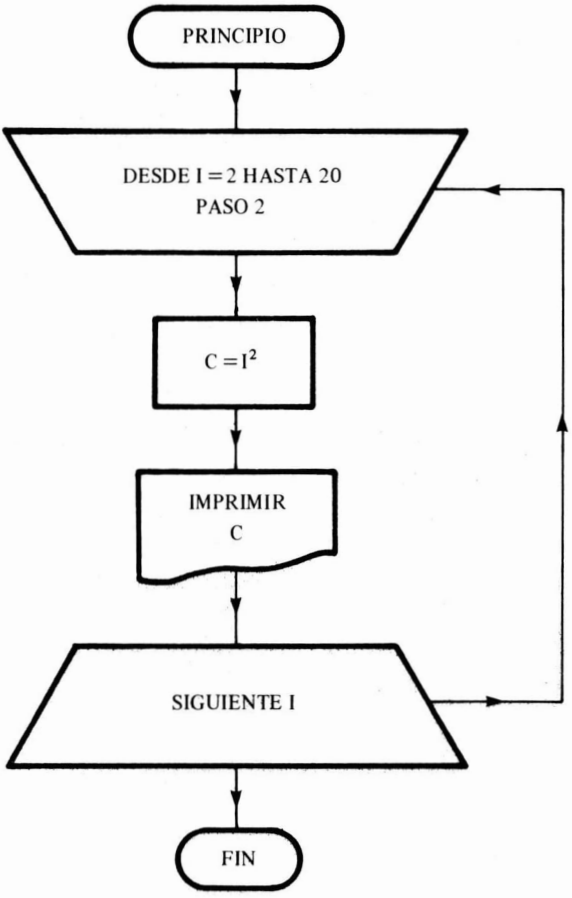


Fig. 5.3

Cuando el paso es 1 no es necesario ponerlo, pero si es cualquier otro valor hay que escribir STEP... El STEP no tiene por qué ser un número entero, puede tener decimales. Por ejemplo:

```
1Ø  FOR  I=1Ø  TO  2Ø  STEP  .7
•
•
•
5Ø  NEXT  I
```

El bucle se sigue ejecutando mientras sea I $\leqslant$ 20, y la última iteración sería para I=19.8. En la siguiente, I valdrá 20.5 y por ser 20.5 > 20 se pasa a la línea siguiente a NEXT I, que es la 6Ø. El valor final de I fuera del ciclo será I=20.5.

El STEP puede ser también negativo. Mira por ejemplo una solución del ejercicio 4.15 (Programa del cálculo del factorial de un número) utilizando la instrucción FOR-NEXT con STEP −1.

```
1Ø REM CALCULO DEL FACTORIAL DE UN NUMERO
2Ø INPUT N
3Ø LET F=1
4Ø FOR I=N TO 1 STEP -1
5Ø LET F=F*I
6Ø NEXT I
7Ø PRINT "EL FACTORIAL DE";N;" ES";F
8Ø END
```

Supongamos que al INPUT N de la línea 2Ø contestamos con el valor 7. Veamos qué valores va tomando F.

En la primera iteración el valor de F era 1, y N vale 7, luego en la línea 5Ø tenemos F=1*7 (=7). En la segunda ejecución de las instrucciones del bucle I vale 6, y como F valía 7, ahora resulta F=7*6. En la tercera vuelta I vale 5, y se tiene F=7*6*5. Y así sucesivamente hasta que I vale 1. En ese momento F vale 7*6*5*4*3*2*1, es decir, F=7! Ahí se termina el bucle y se pasa, en la línea 7Ø, a imprimir el resultado:

```
EL FACTORIAL DE 7 ES 5Ø4Ø
```

Hay que resaltar que cuando el paso es negativo, el bucle se continúa ejecutando mientras I $\geqslant$ 1 (en nuestro ejemplo), al contrario que cuando el paso era positivo.

5.3 VARIABLES Y FORMULAS EN LA INSTRUCCION FOR-NEXT

En el ejemplo anterior, sobre el factorial de un número, puedes ver que el valor inicial de la variable del bucle, no es una constante, sino la variable N, cuyo valor se da previamente mediante el INPUT de la línea 2Ø. Lo mismo se puede hacer para el valor final de la variable del bucle y para el paso: pueden ser constantes, pero pueden ser también variables, e incluso pueden ser expresiones. Así, serían perfectamente válidas las siguientes líneas:

```
5Ø FOR A=7 TO B↑2 STEP (B-1)/2
5Ø FOR V2=A+B TO 3*A
5Ø FOR LØ=(IN+NI)/2 TO 3.1416*R STEP .1
```

Siempre detrás del FOR hay una variable (A, V2, LØ, en los ejemplos anteriores) que es la variable de ejecución del bucle, y que no puede ser ni una constante ni una expresión.

Una advertencia: cuando en un bucle FOR-NEXT utilices variables o expresiones, como en los tres ejemplos superiores, su valor se calcula sólo la primera vez, cuando se inicia el bucle. Por ejemplo, si en un momento B vale 7, la línea:

```
5Ø FOR A=7 TO B↑2 STEP (B-1)/2          (1)
```

queda sustituida por la siguiente:

```
5Ø FOR A=7 TO 49 STEP 3                 (2)
```

Aunque B cambie de valor, por ejemplo dentro del bucle, la línea (1) ha quedado calculada para B=7 y sustituida por la línea (2). Sólo si volviéramos a empezar el ciclo, volvería a buscar el posible nuevo valor de B, por ejemplo 5, para sustituir la línea (1) por la siguiente:

```
5Ø FOR A=7 TO 25 STEP 2
```

5.4 BUCLES SIN SALIDA: INTERRUPCION DE UN PROGRAMA

El siguiente ejemplo es un bucle que no termina nunca:

```
•
•
•
5Ø FOR I=1Ø TO 2Ø
6Ø I=I-1
•
•
•
9Ø NEXT I
```

La variable I empieza valiendo 10. En la línea 6Ø pasa a valer 9, y en la 9Ø vuelve a valer 10, y retorna a la línea 5Ø. Así se reitera una y otra vez el proceso, sin parar nunca.

El mismo problema te daría la línea:

```
5Ø FOR A=3 TO 7 STEP Ø
```

En este bucle A va a valer siempre 3, sin llegar nunca a 7.

En estos casos el ordenador está trabajando sin parar, y hasta ahora no hemos visto remedios para este desenfreno, por lo que lo único que podrías hacer sería apagar el ordenador. Lo malo es que así se ha perdido el programa, que quizá no tenías grabado, y el trastorno puede ser grave.

Para estos casos los ordenadores tienen la tecla [STOP]. Pulsando la tecla [STOP] el programa se para, y aparece en la pantalla el mensaje:

```
BREAK AT LINE 7Ø
```

si era la línea 7Ø la que se ejecutaba en ese instante. Se dice que el ordenador pasa así al modo "COMANDO", ya que puedes utilizar los diversos comandos que conoces, RUN, NEW, y en particular LIST para estudiar el bucle y ver dónde está el fallo.

La tecla [STOP] tiene más aplicaciones. Por ejemplo, cuando un programa da resultados que tú sabes que son incorrectos, lo puedes chequear interrumpiéndolo con [STOP], y preguntándole los valores que en ese momento tienen las variables. En el modo "COMANDO" tecleas:

```
PRINT A [RETURN]  (Sin número de línea)
```

y la máquina da el valor de A, o de la variable que quieras pedirle.

STOP es también una instrucción, que puede incluirse como tal, con número de línea en un programa. Al llegar a ella el programa se para y saca el mensaje:

```
BREAK AT LINE 9Ø
```

si era en la línea 9Ø donde estaba el STOP. Esto es útil por ejemplo para fraccionar las salidas: a veces la salida de un programa son muchas líneas, y como

salen deprisa puedes perderlas por arriba de la pantalla. Con STOP puedes hacer que cada veinte líneas, por ejemplo, el programa se pare, y te dé tiempo a observar los resultados.

Para continuar un programa detenido por la tecla STOP o por la instrucción STOP, se escribe el comando CONT (de continuar), y la ejecución sigue.

Sin embargo, si has rectificado el programa, porque tenía errores, o modificas alguna variable, no se puede continuar; hay que volver a empezar desde el principio. Si escribes CONT en este caso, la máquina contesta algo así como:

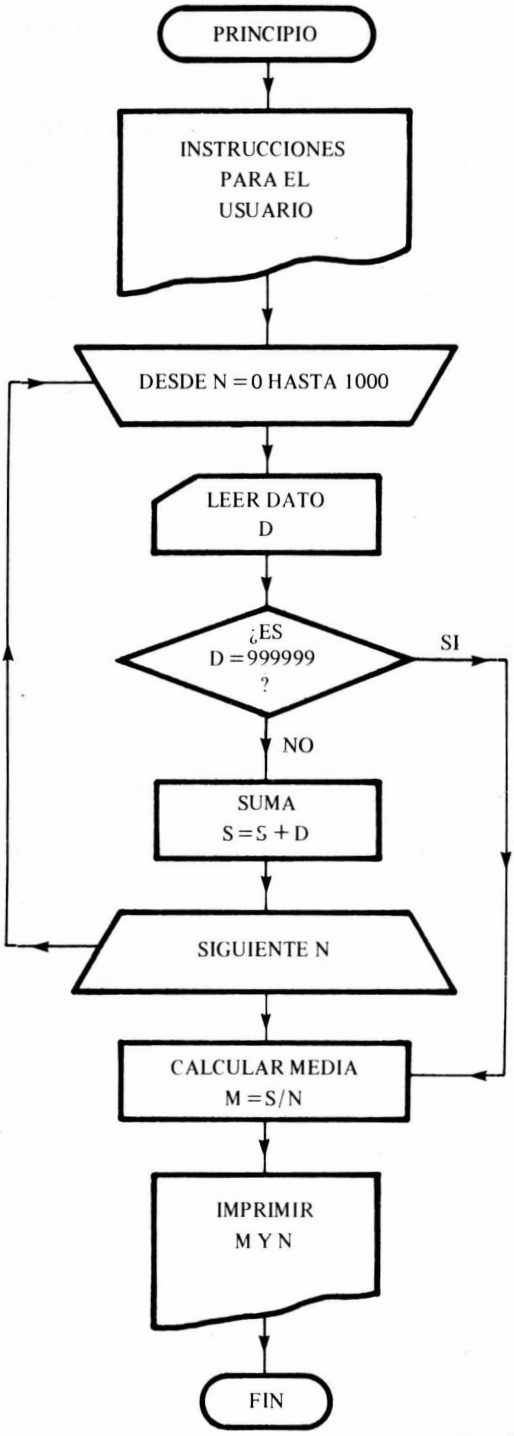
CAN'T CONTINUE ERROR

Has de teclear RUN para empezar desde el principio.

5.5 ENTRADAS Y SALIDAS EN BUCLES FOR-NEXT

Para salir de un bucle no es necesario terminar su ejecución hasta el final. Se pueden poner salidas intermedias, por ejemplo mediante bifurcaciones condicionales (IF-THEN).

Hagamos mediante un bucle FOR-NEXT el último ejemplo de la lección anterior:




```
10 REM MEDIA DE UNA SERIE QUE NO PASE DE 1000 DATOS
20 PRINT "CADA VEZ QUE APAREZCA UNA INTERROGACION"
30 PRINT "EN LA PANTALLA ESCRIBE UN NUMERO DE LA SERIE"
40 PRINT "Y PULSA RETURN"
50 PRINT "DESPUES DEL ULTIMO DATO INTRODUCE"
60 PRINT "EL DATO FICTICIO 999999"
70 FOR N=0 TO 1000
80 INPUT "DATO";D
90 IF D=999999 THEN 120
100 LET S=S+D
110 NEXT N
120 LET M=S/N
130 PRINT "LA MEDIA ES M=";M
140 PRINT
150 PRINT "EL NUMERO DE DATOS ES N=";N
160 END
```

Fig. 5.4

Ya conoces de la lección anterior el funcionamiento de este programa. Interesa únicamente añadir, que si quieres meter más de 1.000 datos no puedes hacerlo con este programa, pues el NEXT N de la línea 11Ø, al llegar a N=1.000 te saca del bucle pasándote a la línea 12Ø, para calcular la media e imprimir los resultados, sin opción a introducir más datos. Pero lo arreglarás fácilmente poniendo por ejemplo:

```
7Ø FOR N=Ø TO 1ØØØØØ
```

o incluso límites superiores, de forma que estés seguro de que nunca llegarás a ellos.

Hemos visto que son lícitas las salidas de un bucle sin pasar por la instrucción NEXT. Sin embargo, nunca son lícitas las entradas en un bucle sin pasar por la instrucción FOR. Un diagrama como el siguiente supondrá un mensaje de error cuando se ejecute el programa correspondiente.

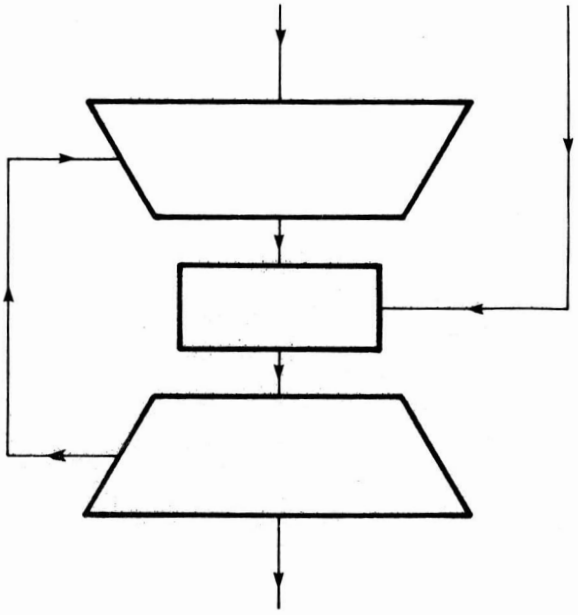


Fig. 5.5

La máquina contestará:

```
NEXT WITHOUT FOR ERROR
```

o un mensaje similar.

5.6 BUCLES ANIDADOS

Entre las instrucciones que se pueden poner en un bucle FOR-NEXT, para que se repita su ejecución tantas veces como se quiera, está la propia instrucción FOR-NEXT: se puede poner un bucle dentro de otro. Veámoslo con un ejemplo: escribir aquéllos números de tres cifras, tales que son iguales a la suma de los cubos de las tres cifras.

Es decir: si A, B y C son tres enteros del 0 al 9, queremos que el número ABC=100A + 10B + C (donde A ≠ 0 pues si no el número tendría en realidad dos cifras) sea igual a $A^3 + B^3 + C^3$.

```
10 REM  EJERCICIO NUMERICO
20 FOR  A=1  TO  9
30      FOR  B=0  TO  9
40           FOR  C=0  TO  9
50                LET  N=100*A+10*B+C
60                LET  X=A*A*A+B*B*B+C*C*C
70                IF  N<>X  THEN  90
80                PRINT  N
90           NEXT  C
100     NEXT  B
110 NEXT  A
120 END
```

En la línea 20 comienza la ejecución del bucle de variable A, dando a A el valor 1. La primera instrucción que incluye este ciclo es comenzar otro en la línea 30 con la variable B, dándole el valor 0; y a su vez, la primera instrucción que realiza éste, es otro bucle FOR-NEXT en el que se asigna a C el valor 0. Se tiene A=1, B=0 y C=0; luego en la línea 50, N vale 100, el primer número de tres cifras.

En la línea 60 se calcula el valor $X=A^3 + B^3 + C^3$. En la línea 70 se comparan los valores de N y X, y sólo si se cumple la condición, se salta la línea 80.

El NEXT C de la línea 90 nos remite a la 40, pero ahora C vale 1. Se trata del número 101, para el que se calcula X, y como $1^3 + 0^3 + 1^3 = 2 \neq 101$ se salta otra vez a la línea 90 que nos remite de nuevo a la 40 pero con C=2.

Este proceso termina cuando C vale 9. Entonces, con independencia de que haya aparecido ya en la pantalla algún resultado, como C ya ha alcanzado el valor 9, se pasa a la línea siguiente, que es NEXT B. Ahora volvemos a la lí-

nea 3Ø pero B vale 1, y en la línea 4Ø se reitera el bucle de C empezando por el valor C=0. Es decir, se van a estudiar los números comprendidos entre 110 y 119. Después B valdrá 2, e irá creciendo hasta llegar a 9, hasta estudiar los números 190 a 199.

En este momento el bucle de B se ha terminado y se pasa a la línea 11Ø, al NEXT A, que supone una vuelta a empezar con A=2, B=0 y C=0. Sucesivamente se estudian todos los números entre 100 y 999. El organigrama sería:

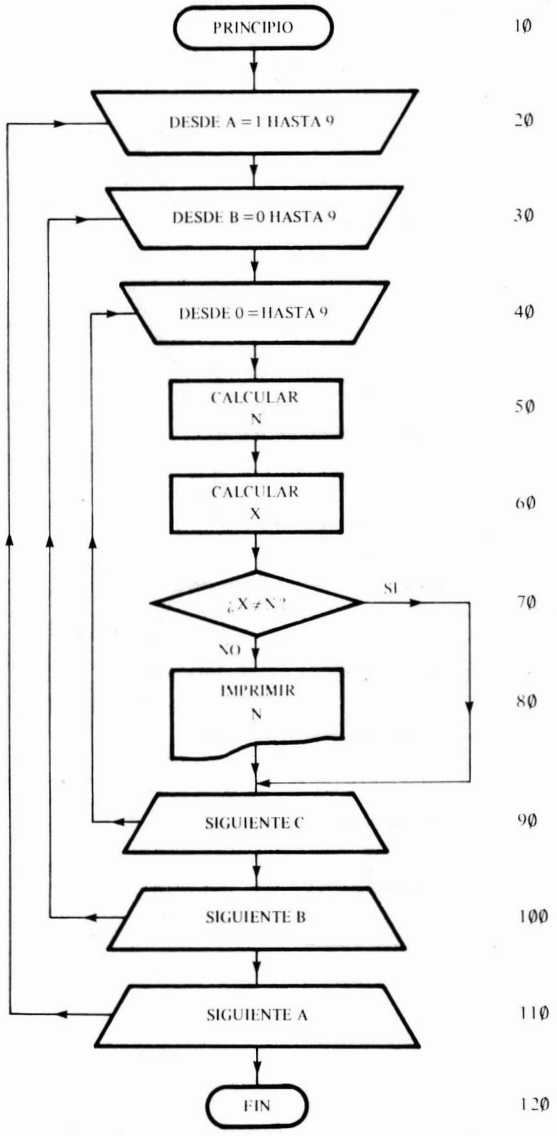


Fig. 5.6

Haz el programa como ejercicio y verás que tiene cuatro soluciones.

Para terminar, otra regla que debes conocer es que no eś correcto "solapar" los bucles FOR-NEXT. Así, mientras es lícito anidarlos, incluso varias veces, no lo es solaparlos. El siguiente organigrama da lugar a un programa que no funciona, habría mensaje de error.

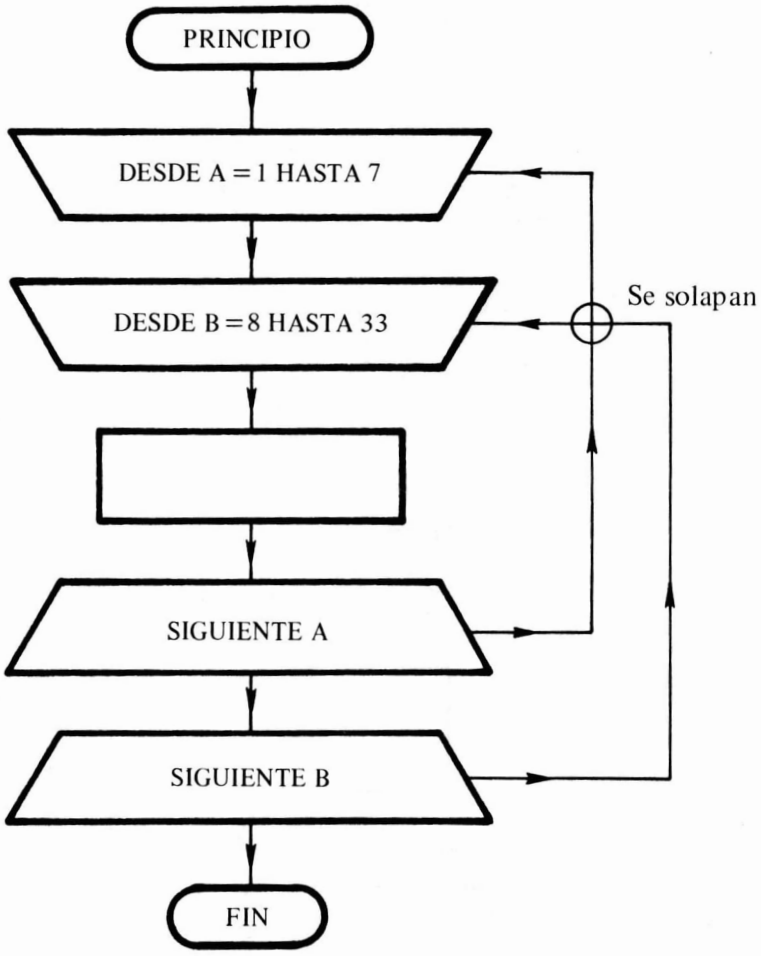


Fig. 5.7

Dicho de otra forma: *en los bucles anidados el primer bucle que se abre es el último que se cierra.*

RECUERDA	
Bucles	FOR-TO
Se ejecutan al menos una vez	STEP
Iteraciones	NEXT
Bucles sin salida	
Salidas de un bucle: lícitas	
Entradas a un bucle: error	
Bucles anidados	
Interrupción de un programa	STOP tecla STOP instrucción
Reanudación de un programa	CONT comando

EJERCICIOS

Muchos ejercicios del capítulo 4 se resuelven más fácilmente y de forma más clara utilizando la instrucción FOR-NEXT. Trata de mejorar así los siguientes ejercicios:

4.1) 4.2) 4.3) 4.4) 4.7)
4.8) 4.11) 4.19) 4.21) 4.26)
4.28) 4.29)

Puedes resolver otros como en el ejercicio de la pregunta 5.5: un bucle de límites amplios y una salida mediante un dato testigo o alguna bifurcación condicional:

4.5) 4.9) 4.18) 4.31)

5.1) A continuación se muestran algunos esqueletos de bucles FOR-NEXT. Indicar cuáles, si es que hay alguno, están escritos incorrectamente:

a)
```
10 FOR K=K1 TO K2
•
•
•
50 K=K+1
•
•
•
90 NEXT K
```

b)
```
10 FOR C1=0 TO 50 STEP 5
•
•
•
35 FOR C2=0 TO C1
•
•
•
65 NEXT C2
•
•
•
100 NEXT C1
```

c)
```
10 FOR J=1 TO N
•
•
•
40 FOR K=1 TO M
•
•
•
70 NEXT J
•
•
•
120 NEXT K
```

d)
```
10 FOF X=A TO B STEP C
•
•
•
40 NEXT C
```

5.2) Hacer el ejercicio 4.18) con una pequeña modificación: que en la pantalla no sólo se obtengan los resultados finales, sino que también se vayan imprimiendo los resultados parciales de la sucesión: 1^2, $1^2 + 2^2$, $1^2 + 2^2 + 3^2$, ... hasta llegar al resultado final, sacando entonces un mensaje adecuado.

5.3) Hacer el problema 4.24). Sólo si hay plazas libres el ordenador saca el INPUT: CUANTOS BILLETES DESEA? Si el viaje ya está completo en la pantalla sale COMPLETO y se termina el programa.

5.4) Programar la tabla de multiplicar del 7 de modo que aparezca de la forma:

$$7* \ldots = \ldots$$

Con el segundo factor variando de 0 a 9. Primero STEP 1, luego STEP 0,5.

5.5) Construir una tabla de valores de la función $y=x^2 + 3x - 7$, desde $X=1$ hasta $X=2$, aumentando X de décima en décima.

5.6) Calcular los siguientes números:

$$a = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{100}$$

$$b = \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{8} + \ldots + \frac{1}{100}$$

$$c = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots + \frac{1}{512}$$

$$d = 15\,!$$

$$e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{10!}$$

El último puedes hacerlo mediante dos bucles anidados .

5.7) Calcular los siguientes números combinatorios:

$$a = \binom{30}{13} \qquad b = \binom{36}{9} \qquad c = \binom{49}{21}$$

(Atención: los valores 36! y 49! pueden exceder el valor máximo de una variable en tu ordenador).

5.8) Construir un programa para enseñar las tablas de sumar a un niño (parecido al programa 4.28).

5.9) Sabiendo que la ecuación $X^3 - X^2 - 8X - 5 = 0$ tiene las tres soluciones en el intervalo [−2,4], hallarlas con un error menor que una centésima. (Escribir una tabla de valores variando la X de centésima en centésima. Los cambios de signo corresponden a intervalos con soluciones).

5.10) Calcular la suma de los cien primeros enteros impares.

5.11) El valor de sen (x), que estudiaremos en el próximo capítulo, puede aproximarse mediante la siguiente función:

$$\text{sen}\,(x) \simeq x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \quad \text{(x en radianes)}$$

Haz el programa de una tabla de la función seno de 0° a 45° con valores de grado en grado.

(Atención: has de pasar los grados a radianes).

5.12) Al nacer un niño, su madre le abre una cartilla de ahorros y le ingresa todos los años 1.000 ptas. el día 1 de enero. Al dinero que ingresa la madre se le añaden unos intereses anuales del 8% sobre la cantidad que en ese momento tenga ahorrada. El abono de los intereses se hace el 31 de diciembre de cada año. Después de 25 años, el chico retira sus ahorros. ¿Cuánto dinero le dan?

5.13) He aquí un algoritmo (procedimiento) para calcular el valor de π:

En una circunferencia de radio unidad, el lado del cuadrado inscrito vale $L_1 = \sqrt{2}$ y el perímetro del cuadrado es PERIM = $4\sqrt{2}$.

El octógono inscrito tiene de lado:

$$L_2 = \sqrt{(L_1/2)^2 + (1 - L_1/2)^2} = \sqrt{L_1^2/2 + 1 - L_1}$$

y su perímetro será PERIM = 8*L_1

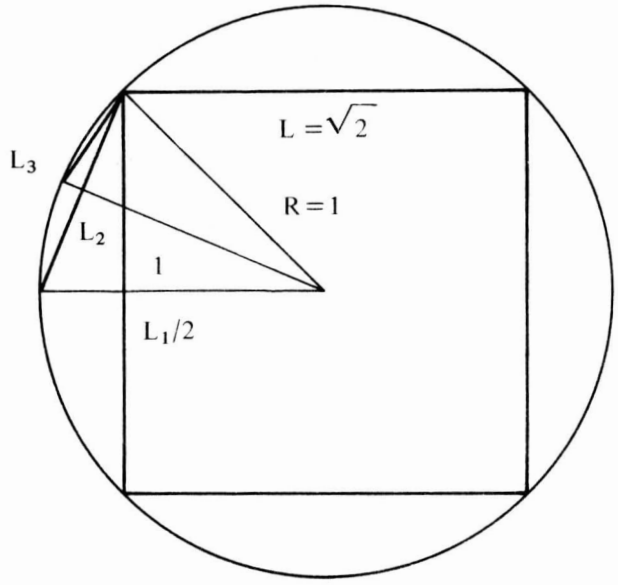


Fig. 5.8

Si aumentamos en progresión geométrica de razón 2 el número de lados del polígono inscrito, la longitud del lado que resulta es:

$$L_n = \sqrt{2 - \sqrt{4 - L_{n-1}^2}}$$

y su perímetro será: $PERIM = 2^{n+1} * L_n$ (1)

Cuando n tiende a infinito, el lado tiende a cero, el polígono inscrito "tiende" a la circunferencia, y su perímetro a $2\pi R$ (en este caso hacemos R = 1). Por lo que dividiendo el perímetro calculado en (1) entre 2 tenemos aproximaciones de π cada vez más exactas.

Un programa podría ser:

```
1Ø  REM  ALGORITMO PARA EL CALCULO DE π
2Ø  L=2↑.5
3Ø  PRINT  "N. LADOS", "APROXIMACION DE π"
4Ø  PRINT  "========", "================"
5Ø  FOR  I=2  TO  2Ø
6Ø  L=(2-(4-L↑2)↑.5)↑.5
7Ø  PERIM=L*2↑ (I+1)
8Ø  PRINT  2↑(I+1), PERIM/2
9Ø  NEXT  I
1ØØ  END
```

Te encontrarás con un problema de falta de precisión del ordenador: cuando L es muy pequeño, la máquina tomará $L^2 = 0$ y en la línea 6Ø se tiene $L = \sqrt{2 - \sqrt{4}} = 0$

Y si el lado es 0, el perímetro es 0 y π valdría 0. Para resolver bien este problema el ordenador tendría que utilizar muchas cifras significativas, o tomar valores más próximos al cero que $1Ø^{-38}$

5.14) INTEGRACION NUMERICA: METODO DE LOS RECTANGULOS

Dada una función Y = F (X), continua en el intervalo cerrado [A, B], un valor aproximado de la integral

$$\int_A^B F(X)dX$$

puede calcularse por el método de los rectángulos, que vamos a justificar en el caso de que la gráfica de la función F, esté siempre por encima del eje de abscisas, aunque el método es válido cualquiera que sea el signo de los valores que tome la función.

En el caso de que F tome valores positivos, la integral coincide con el área de la región plana limitada por la gráfica de F, el eje de abscisas y las rectas de ecuaciones X = A y X = B (ver Figura 5.9).

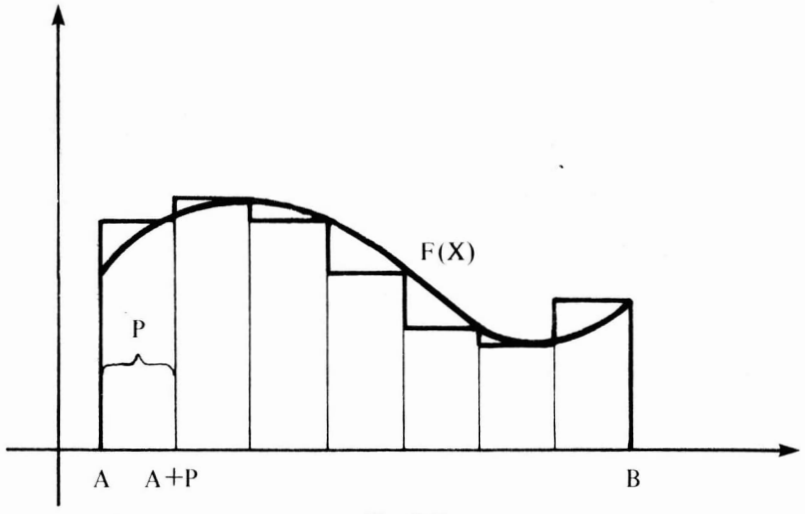


Fig. 5.9

Para hallar un valor aproximado de la integral dividimos el segmento [A, B] en N partes iguales (en la figura se ha dividido en 7 partes iguales).

Cada una de estas partes tiene por longitud el número P=(B−A)/N, que será la base de cada uno de los rectángulos que aparecen en la figura.

Cada rectángulo tiene una altura igual al valor que toma la función en el extremo de la derecha de cada intervalo parcial; así la altura del primer rectángulo es F(A + P), la del segundo es F(A + 2P), y así sucesivamente, la del último será F(A + NP) = F(B).

La suma de las áreas de los N rectángulos será un valor aproximado de la integral:

$$S = P \cdot F(A+P) + P \cdot F(A+2P) + \ldots + P \cdot F(A+NP)$$

Si queremos una aproximación mejor tendremos que dividir el segmento [A, B] en un número mayor de partes.

A continuación damos un organigrama para hallar el valor de S para una función F. Los valores A y B de los extremos del intervalo y el número N de divisiones serán metidos en el programa mediante INPUT.

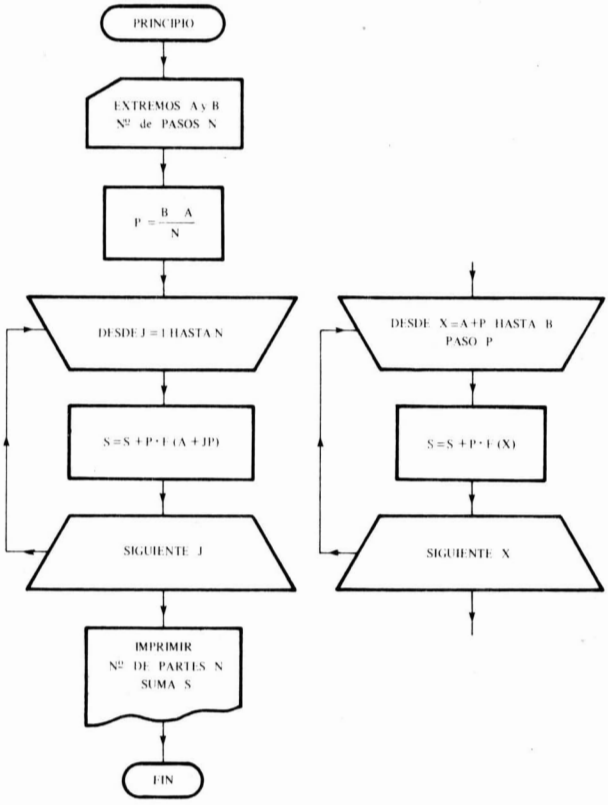


Fig. 5.10

Otra posibilidad, que puede acortar el tiempo de ejecución del programa, es sacar factor común P:

$$S = P\cdot(F(A+P) + F(A+2P) + \ldots + F(A+NP))$$

Mediante el ciclo FOR-NEXT se puede calcular la suma del paréntesis, asignándole la variable SU por ejemplo. Al final

$$S = P\cdot SU$$

Hacer un programa para calcular un valor aproximado de las integrales:

a) $\int_1^2 \frac{1}{X}\,dx$ b) $\int_{-3}^{3} e^{-x^2}\,dx$

(En BASIC e^X es EXP(X))

6

FUNCIONES DE LIBRERIA

6.1 INTRODUCCION

En el capítulo 2 vimos las operaciones aritméticas: +, *, etc. Igual que las calculadoras de bolsillo, los microordenadores con lenguaje BASIC pueden hacer otras operaciones como: hallar logaritmos, calcular valores de funciones trigonométricas, exponenciales, etc. Es lo que vamos a ver en este capítulo.

Estas "operaciones" se llaman *funciones de librería.* No se designan por una tecla, como en las calculadoras, sino por las iniciales de la función o una abreviatura (del inglés). En las operaciones aritméticas se trabaja con dos canti-

dades (suma, producto, etc., de dos cantidades). Las funciones de librería suelen trabajar sobre una sóla cantidad que se llama argumento y se pone entre paréntesis.

6.2 PARTE ENTERA DE UN NUMERO: INT

La parte entera de 3,25 es 3; la parte entera de 0,277 es 0 y la parte entera de −3,27 es −4 (¡Cuidado!). Por definición la parte entera de un número real X es el mayor de los enteros menores o iguales que X. La parte entera se toma siempre por defecto.

El nombre en BASIC de la función parte entera es INT. Así, por ejemplo, en el programa siguiente se tiene:

```
1Ø LET A=INT(5.225)        A = 5
2Ø LET B=INT(−7.25)        B = −8
3Ø LET C=INT(6)            C = 6
4Ø LET D=INT(−6)           D = −6
•
•
•
```

El argumento de las funciones de librería puede ser una constante, como en el ejemplo anterior, pero puede ser también una variable e incluso una fórmula. Esto es válido para todas las funciones de librería. Mira el siguiente ejemplo:

```
1Ø INPUT N
2Ø PRINT INT(N),INT(N↑2),INT(N↑3)
3Ø END
```

Si al correr el programa contestamos a la interrogación del INPUT con el valor −2,3, se tiene N=−2,3, N^2 = 5,29 y N^3 = −12,167, y el programa escribirá:

```
−3          5          −13
```

A primera vista puede parecernos que la función INT es de poca utilidad.

Sin embargo esta función es una de las más importantes del BASIC. Veamos una primera aplicación: *la división entera de números naturales.*

La división

37	8
5	4

es una *división entera.* Dados dos números, dividendo (37) y divisor (8); se trata de hallar otros dos, cociente (4) y resto (5) que cumplan las condiciones:

1ª Dividendo = divisor . cociente + resto

2ª $0 \leqslant$ resto $<$ divisor

En nuestro ejemplo:

1ª $37 = 8.4 + 5$

2ª $0 \leqslant 5 < 8$

Efectuar una división entera es hallar el cociente entero y el resto. Mediante la función INT podemos hallar ambos. La mayor parte de los ordenadores no efectúan directamente la división entera. Si nosotros dividimos 37/8, la respuesta es 4,625, pero a partir de aquí podemos hallar el cociente y el resto enteros.

El cociente entero será INT(37/8). El ordenador calcula primero el argumento y resulta INT (4.625) que da 4. El resto tiene la siguiente expresión:

```
37 - INT(37/8)*8
```

aplicando la primera condición de la división entera. Es la diferencia entre el dividendo (37) y el cociente por el divisor (4*8).

Todo esto se aplica frecuentemente, por ejemplo, para ver si un número A es divisible por otro B. La división será exacta cuando el resto sea 0. De ordinario puedes emplear otra expresión más sencilla: A es divisible por B si, y sólo si,

```
INT(A/B)=A/B
```

ya que la expresión anterior equivale a decir que A/B no tiene decimales, es una división exacta.

Otra aplicación de la función INT es fijar el número de decimales para

la presentación en pantalla. Habrás hecho ya programas que te dan unas salidas que inundan la pantalla de números decimales, perdiendo claridad. Si quieres escribir un valor aproximado de una cantidad A, con sólo dos decimales, puedes poner:

B = INT(A*1ØØ)/1ØØ

Primero se multiplica el número por 100 y se halla la parte entera, perdiendo el resto de los decimales. Después se divide por 100 y queda la cantidad inicial pero sólo con dos decimales. Por ejemplo:

PI = INT(3.14159*1ØØ)/1ØØ = INT(314.159)/1ØØ = 314/1ØØ = 3.14

6.3 NUMEROS ALEATORIOS: RND

Un número aleatorio es aquel que se escoge al azar entre los de un conjunto dado. Por ejemplo: al lanzar un dado obtenemos un número natural aleatorio comprendido entre 1 y 6. Si a la cara de una moneda le asignamos el número 0 y a la cruz el 1, al lanzarla obtenemos el número aleatorio 0 o el 1.

En cada uno de estos ejemplos los números 1 al 6, o los números 0 ó 1, se presentan con igual probabilidad. Pero también cabe imaginarse otros ejemplos en los que unos resultados tienen más probabilidad que otros usando, por ejemplo, dados o monedas trucados.

En programas de juegos o de simulaciones interesa con frecuencia que el ordenador "piense" un número al azar entre dos números dados. Para eso dispone de una función, RND, abreviatura de random = azar, que da un número aleatorio comprendido entre 0 y 1.

Así, por ejemplo, RND(5) puede ser 0.74372578. En la siguiente ejecución RND(5) será otro número aleatorio. Ejecuta el siguiente programa:

```
1Ø REM EXPERIMENTO ALEATORIO
2Ø FOR I=1 TO 5
3Ø LET A=RND(7)                    (1)
4Ø PRINT A
5Ø NEXT I
6Ø END
```

Cada vez obtendrás un resultado distinto. Nosotros lo hemos ejecutado dos veces y hemos obtenido los siguientes valores:

```
RUN
 .416Ø32766
 .661745879
 .98Ø122777
 .74Ø416917
 .917968972
READY

RUN
 .Ø959731361
 .757597443
 .Ø12Ø646175
 .127581985
3.4Ø876882E-Ø3
```

Como ves son valores más o menos uniformemente repartidos. En un lugar hemos puesto RND(5) y en el programa RND(7), y es que el efecto de "azar" que realiza la función RND es independiente del argumento, que puede ser también una variable o una fórmula. Con todo, este es un punto en el que puedes profundizar consultando el manual de instrucciones de tu microordenador. Por ejemplo, algunos no utilizan argumento, y ponerlo daría un mensaje de error, y utilizan además la instrucción RANDOMIZE.

Los números aleatorios que obtenemos con la función RND están comprendidos entre 0 y 1, esto es:

$$Ø < \text{RND}(1ØØ) < 1$$

¿Cómo llegar, entonces, mediante la función RND a simular el lanzamiento de un dado? Tendríamos que obtener aleatoriamente valores enteros del 1 al 6, y lo vamos a ver en tres pasos sucesivos. Inicialmente tenemos:

$$Ø < \text{RND}(2) < 1$$

Si se multiplica por 6 el valor de RND, resulta:

$$Ø < 6*\text{RND}(2) < 6$$

Puedes sustituir en el programa (1) la línea 3Ø por la siguiente:

```
3Ø LET A=6*RND(2)
```

Al ejecutarlo hemos obtenido los valores

```
3.27417616
5.1729Ø744
Ø.55671922
1.43246516
2.622Ø5
```

comprendidos entre 0 y 6.

Si tomamos su parte entera y ponemos un punto y coma (;) al final de la línea 4Ø, nos queda:

```
3  5  Ø  1  2
```

Observa que el menor número que se puede obtener es el 0 y el mayor el 5.

Si, por fin, les sumamos la unidad, resultan los números:

```
4  6  1  2  3
```

que son los puntos que se pueden obtener al lanzar un dado. De estas ideas obtenemos la fórmula que hay que emplear en la simulación del dado:

$$D = INT(6*RND(2))+1$$

Un programa podría ser:

```
1Ø REM SIMULACION LANZAMIENTO DADO
2Ø FOR I=1 TO 1Ø
3Ø LET D=INT(6*RND(3))+1
4Ø PRINT D;
5Ø NEXT I
6Ø END
```

que nos ha dado como resultados:

```
RUN
3  4  5  6  1  3  1  4  5  3
READY

RUN
5  1  2  1  4  2  1  4  6  4
READY
```

6.4 FUNCIONES TRIGONOMETRICAS

Los microordenadores tienen las siguientes funciones trigonométricas:

SIN(X)	Calcula el seno de X	sen x
COS(X)	Calcula el coseno de X	cos x
TAN(X)	Calcula la tangente de X	tg x
ATN(X)	Calcula el arco tangente de X	arctg x

Algunos ordenadores tienen definidas también arco seno y arco coseno.

En estas funciones los ángulos van siempre en radianes. Así, si pones SIN(30) la máquina "entiende" seno de 30 radianes, es decir, seno de $(30 \cdot 180)/\pi$ grados: sen 1718,87° = −0,988. Si lo que querías era el seno de 30 grados has de pasarlo a radianes:

```
SIN(3Ø*3.14159/18Ø)=.5
```

De igual forma ATN(1)=.7853981, que pasado a grados es:
.7853981*18Ø/3.14159=45°, arco cuya tangente es 1.

6.5 LA FUNCION PRINT TAB

En cada línea de impresión de la unidad de salida (pantalla o impresora) cabe un número determinado de caracteres. Supongamos que en nuestro ordenador el número de caracteres por línea es 40. Esto quiere decir que en cada

línea hay 40 posiciones donde imprimir, numeradas del 0 al 39 (otras máquinas pueden numerar del 1 al 40). Veamos qué significado tiene la instrucción siguiente:

```
6Ø PRINT TAB(25);A
```

Cuando se ejecute esta sentencia el cursor correrá hacia la derecha hasta colocarse en la posición 25, e imprimirá el valor de A a partir de ahí. Es un papel idéntico al del TABULADOR de las máquinas de escribir corrientes.

Emplearemos una sentencia TAB cuando nos interese escribir caracteres en posiciones determinadas. Vimos en el tercer capítulo la utilización de la coma (,), y del punto y coma (;) en instrucciones PRINT para tener salidas claras y elegantes. La función TAB es todavía más potente en este sentido. TAB se suele terminar en punto y coma (;).

Como en las demás funciones, el argumento puede ser un número, una variable, o una expresión, eso sí, con valores positivos. Podríamos escribir:

```
TAB(17)
TAB(I)
TAB(INT(3*C) )
```

Mira el siguiente ejemplo:

```
9Ø PRINT TAB(3);"NUMERO DE HIJOS";TAB(K);N
```

Si K es suficientemente grande, 25 por ejemplo, se escribirá NUMERO DE HIJOS a partir del espacio 3 y a partir del 25 el valor de N. Sin embargo si K es pequeño, 10 por ejemplo, al escribir NUMERO DE HIJOS se ha rebasado la posición 10. Cuando el cursor se encuentra a la derecha de la posición señalada por TAB, la función TAB se ignora y la impresión se efectúa a continuación. Se imprimiría el valor de N a continuación de NUMERO DE HIJOS, como si estuvieran separados sólo por un punto y coma (;) prescindiendo de TAB(K).

Si K no fuera entero, la función TAB trabajaría sobre su parte entera, y así TAB(27.38) equivale a TAB(27), y TAB(27.97) equivale también a TAB(27).

Veamos el siguiente ejercicio: programar la impresión de la tabla:

posición 1	posición 11	posición 21
X	X/2	X/3
1	.5	.333333333
2	1	.666666666
3	1.5	1
4	2	1.33333333
5	2.5	1.66666666
6	3	2
7	3.5	2.33333333
8	4	2.66666666
9	4.5	3
1Ø	5	3.33333333

Un programa podría ser:

```
1Ø REM TABLA DE MITADES Y TERCIOS
2Ø PRINT TAB(1);"X";TAB(11);"X/2";TAB(21);"X/3"
3Ø FOR X=1 TO 1Ø
4Ø PRINT TAB(Ø);X;TAB(1Ø);X/2;TAB(2Ø);X/3
5Ø NEXT X
6Ø END
```

Observa que en la línea 4Ø los argumentos de TAB son 0, 10 y 20, inferiores en una unidad a los de la línea 2Ø. Esto se debe a que al imprimir X, X/2 o X/3 en la línea 4Ø el primer espacio lo ocupa el signo, y como aquí es positivo se omite, y queda el primer espacio en blanco; por lo que el número queda desplazado un lugar hacia la derecha. Por eso hemos desplazado un lugar hacia la derecha la cabecera X, X/2 y X/3.

El siguiente programa nos asoma a las posibilidades gráficas que tiene el ordenador con la función TAB.

```
1Ø REM REPRESENTACION GRAFICA
2Ø FOR I=1 TO 15
3Ø PRINT TAB(4Ø-I);"*"
4Ø NEXT I
5Ø END
```

cuyo resultado en pantalla es:

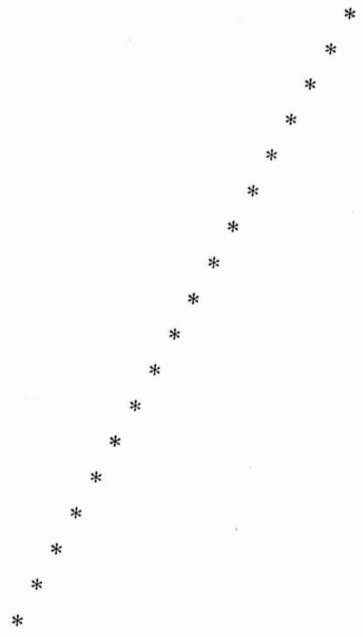


Fig. 6.1

La función TAB también es útil para la edición de textos. Supongamos que queremos disponer de un programa que, tomando los datos de nombre y dirección, escriba sobres como éste:

SR. D. JOSE ALVAREZ FERNANDEZ
CALLE BELLAVISTA 37
MADRID – 43

He aquí un programa:

```
1Ø  REM  ESCRITURA DE UN SOBRE
2Ø  INPUT  "NOMBRE Y APELLIDOS";N$
3Ø  INPUT  "CALLE";C$
4Ø  INPUT  "POBLACION";P$
5Ø  PRINT  TAB(6);"SR. D. "+N$
6Ø  PRINT
7Ø  PRINT  TAB(6);"CALLE";TAB(12);C$
8Ø  PRINT
9Ø  PRINT  TAB(12);P$
1ØØ  END
```

6.6 OTRAS FUNCIONES PARA FORMULAS Y OPERACIONES

SQR( ) Es una contracción de SQUARE ROOT = RAIZ CUADRADA. Halla la raíz cuadrada del argumento encerrado entre paréntesis, y da mensaje de error si el argumento es negativo. Función análoga a ↑.5

ABS( ) Calcula el valor absoluto del argumento.

LOG( ) Halla el logaritmo neperiano, o en base e, del argumento. Para obtener un logaritmo decimal hay que aplicar la fórmula del cambio de base:

$$\log_{10} N = LOG(N)/LOG(10)$$

EXP( ) Calcula el valor del número e elevado al argumento. Es la función exponencial, con la ventaja de que tiene en memoria el número e con muchos decimales y no hay que teclearlo.

SGN( ) Signo. Toma el valor 1 si el argumento es positivo, −1 si es negativo y 0 si es nulo.

6.7 DEFINICION DE UNA FUNCION CUALQUIERA: DEF FN

Además de todas las funciones de librería, el BASIC ofrece la posibilidad de que utilicemos cualquier otra función, definiéndola previamente. Un ejemplo de definición de una función es el siguiente:

```
20 DEF FNY(X)=3*X↑2+7*X+6
```

La instrucción es DEF FN (definición de función) seguida del nombre de la función, Y en nuestro caso, y de la variable entre paréntesis (X). La función arriba definida es:

$$y(x) = 3x^2 + 7x + 6$$

Otros ejemplos de definición de funciones son:

```
2Ø DEF FNZ1(T)=SIN(T)*3-T↑2
2Ø DEF FNPA(A)=7*A/LOG(A)
2Ø DEF FNM(X)=A*X↑2-A↑2*X
```

Los nombres de las funciones siguen las mismas reglas que los nombres de las variables, aunque algunos "micros" admiten una letra únicamente como nombre.

La forma de utilizar las funciones definidas, para distintos valores de la variable, es (siguiendo con el primer ejemplo):

```
5Ø LET Y1=FNY(-2)
```

Esta instrucción, en la que se incluye FN (función), su nombre (Y), y el valor del argumento (−2), calcula el valor de Y para ese argumento, que resulta ser 4, y asigna a Y1 ese valor: Y1 = 4. Cada vez que la máquina encuentre en la ejecución de un programa una función FN seguida del nombre, y del argumento entre paréntesis, busca la función definida con ese nombre (en cualquier sitio del programa en que se encuentre) y la calcula para el argumento señalado, utilizándose luego en una instrucción LET, como en el caso anterior, o PRINT o IF-THEN, etc.

Veamos un ejemplo completo en el que hacemos uso de una función definida por nosotros. Supongamos que necesitamos una tabla de valores de la función $y=3x^2 + 7x + 6$, para todos los valores enteros de x comprendidos entre −4 y 4. He aquí un programa:

```
1Ø REM TABLA DE VALORES DE UNA FUNCION
2Ø DEF FNY(X)=3*X↑2+7*X+6
3Ø PRINT "X", "Y"
4Ø FOR X=-4 TO 4
5Ø PRINT X, FNY(X)
6Ø NEXT X
7Ø END
```

Corriendo el programa obtenemos:

X	Y
−4	26
−3	12
−2	4
−1	2
Ø	6
1	16
2	32
3	54
4	82

La instrucción DEF FN puede estar en cualquier lugar del programa, pero por cuestión de claridad suele ponerse al principio o al final, y si hay varias, se ponen juntas. Sin embargo hay aparatos que exigen que la definición de la función (DEF FN...) sea anterior a su utilización (FN...) y entonces es mejor definir las funciones al principio del programa.

Aunque el lenguaje BASIC permite definir funciones de varias variables, algunos microordenadores no las admiten. No serían lícitas instrucciones como:

```
7Ø  DEF  FNT(X, Y, Z)=X↑2+Y↑2+Z↑2
```

Sin embargo, es posible conseguir el mismo efecto de otra forma. Considera la definición de función:

```
2Ø  DEF  FNM(X)=A*X↑2+A↑2*X
```

Es una función de una variable, X, que si la utilizamos para X=7

```
24Ø  LET  K=FNM(7)
```

proporcionará el valor $49A + 7A^2$. En esta expresión se toma para A el valor que tiene en el curso del programa en ese momento. Si al llegar a 24Ø A vale 2, se tiene K=49*2+7*4=126. En realidad hemos tratado a A como a una segunda variable.

6.8 FUNCIONES PARA TRABAJAR CON PALABRAS

Hasta ahora hemos tratado muy poco de las variables alfanuméricas. La única "operación" que se ha definido para ellas es la suma (+) entendida como yuxtaposición, y los únicos operadores los representados por los signos = y <>. ¿Cómo hacer para ordenar alfabéticamente una lista de apellidos? Nos gustaría también saber entresacar las iniciales de los nombres completos de las personas. Y, aumentando la dificultad, sería interesante conjugar verbos o corregir faltas de ortografía. El BASIC tiene bastantes funciones para trabajar con variables alfanuméricas o de cadena, y vamos a ver las más importantes.

La función LEN(A$) cuenta el número de caracteres de una cadena (length=longitud). Por ejemplo el programa:

```
10 LET A$="BASIC"
20 LET N=LEN(A$)
30 PRINT N
40 END
```

cuenta en la línea 20 el número de letras de la palabra "BASIC" y lo asigna a N, que valdrá 5. Por tanto la línea 30 imprime el número 5. No hace falta que la palabra "BASIC" se asigne a la variable A$; se pueden contar sus letras directamente, poniéndola entre comillas, y se obtiene el mismo resultado:

```
10 LET N=LEN ("BASIC")
20 PRINT N
30 END
```

Tres funciones muy parecidas entre sí son las siguientes:

LEFT$ (A$,N): extrae de la cadena A$, N caracteres empezando por la izquierda (LEFT=izquierda).

RIGHT$ (A$,N): extrae de la cadena A$ los últimos N caracteres, los de la derecha (RIGHT=derecha).

MID$(A$,P,N): extrae de la cadena A$, N caracteres empezando por el caracter intermedio que ocupe la posición P (MIDDLE =medio).

Como ejemplo vamos a hacer un juego de palabras: dada una palabra, o variable alfanumérica cualquiera, el ordenador la escribirá en sentido inverso:

```
1Ø REM JUEGO DE PALABRAS
2Ø INPUT "QUE PALABRA QUIERES INVERTIR";A$
3Ø LET L=LEN(A$)
4Ø FOR I=1 TO L
5Ø PRINT MID$(A$,L-I+1,1);
6Ø NEXT I
7Ø END
```

Si al INPUT de la línea 2Ø contestas con la palabra "ROMA", en la línea 3Ø, L vale 4, y mira lo que hace el bucle: de la palabra ROMA escribe el caracter de la posición L–I+1=4 –1 + 1 = 4, es decir, escribe la A. Tras el NEXT I, escribe otra vez un caracter, pero esta vez de la posición 4 – 2 + 1 = 3, es decir, la M. En el paso siguiente tendremos 4 – 3 + 1 = 2, que corresponde a la O, para terminar con la R. Ha escrito la palabra AMOR.

Otro ejemplo sencillo es entresacar las iniciales del nombre y apellidos de una persona:

```
1Ø REM INICIALES
2Ø PRINT "CUALES SON TU NOMBRE Y APELLIDOS"
3Ø INPUT N$
4Ø PRINT LEFT$(N$,1);".";    (escribe la 1ª inicial)
5Ø LET A$=" "
6Ø FOR I=2 TO LEN(N$)
7Ø IF MID$(N$,I,1)<>A$ THEN 9Ø
8Ø PRINT MID$(N$,I+1,1);".";    (escribe las demás iniciales)
9Ø NEXT I
1ØØ END
```

El funcionamiento de este programa es sencillo de comprender. Aumentando la dificultad hagamos un programa para entresacar las palabras de una frase. Las palabras se caracterizan por estar comprendidas entre dos espacios en blanco, excepto la primera y última palabras de la frase. Un programa podría ser:

```
1Ø REM PALABRAS DE UNA FRASE
2Ø PRINT "QUE FRASE QUIERES DESGUAZAR"
3Ø INPUT F$
4Ø LET L=LEN(F$)
5Ø LET A$=" "
6Ø LET R=1 (primera posición de la palabra a escribir)
7Ø FOR I=1 TO L
8Ø IF MID$(F$,I,1)<>A$ THEN 11Ø
9Ø PRINT MID$(F$,R,I-R)
1ØØ LET R=I+1 (primera posición de la siguiente palabra a escribir)
11Ø NEXT I
12Ø PRINT RIGHT$ (F$,L-R+1)
13Ø END
```

donde la línea 120 es necesaria pues la última palabra no termina en un espacio en blanco, y el bucle no da orden de imprimirla: ha de escribirse con independencia. Te recordamos que una cadena puede tener hasta 255 caracteres. Si en el INPUT de la línea 30 tienes una limitación más restrictiva (por ejemplo, dos líneas, con un máximo de 80 caracteres) puedes alargar la frase mediante varios INPUT y suma de cadenas.

Hay otras dos funciones que son inversas entre sí: VAL(A$) extrae de la variable A$ los primeros caracteres numéricos hasta llegar al primero no numérico. Por ejemplo, la línea:

```
3Ø LET P=VAL("127SEAT")
```

asigna a P el valor numérico 127. VAL, abreviatura de value = valor, extrae, hasta donde puede, un valor numérico de una cadena de caracteres.

En cierto modo recíproca, la función STR$(N) convierte un número en una cadena, permitiendo aplicarle las funciones anteriores. Por ejemplo, las líneas:

```
2Ø LET N=85324
3Ø LET A$=STR$(N)
```

asignan a la variable N el número 85324, y a la variable A$ el valor "85324", concebido como cadena, no como número. Por ejemplo a la variable A$ le podemos aplicar el primer programa de este apartado, y la escribirá al revés: "42358".

RECUERDA	
ABS valor absoluto ATN arco tangente COS coseno DEF FN definición de función EXP exponencial base e FN uso de la función INT parte entera	LOG logaritmo neperiano PRINT TAB tabulador RND número aleatorio SGN signo SIN seno SQR raíz cuadrada TAN tangente
LEN(A$) longitud: número de caracteres VAL(A$) valor numérico STR$(N) convierte en alfanumérica	LEFT$(A$,N) subcadena izquierda RIGHT$(A$,N) subcadena derecha MID$(A$,P,N) subcadena intermedia

EJERCICIOS

6.1) Hacer un programa que pida un número entero, y que al dárselo escriba todos los números que lo dividan.

6.2) Hacer un programa que pida un número y conteste si es primo o no (es parecido al anterior).

6.3) Obtener todos los números primos menores que 342.

6.4) Descomponer un número en sus factores primos. El procedimiento a seguir es el mismo que cuando lo resuelves con papel y bolígrafo: se toma el número y se prueba a ver si 2 lo divide, incluso varias veces; luego el 3, etcétera. Y en cada caso afirmativo se anota el factor, y se divide el número por él antes de seguir.

6.5) El algoritmo de Euclides para hallar el M.C.D. de dos números (supongamos 16 y 6) consiste en lo siguiente: se hace la división entera del mayor entre el menor:

$$\begin{array}{r|l} 16 & 6 \\ \hline 4 & 2 \end{array}$$

Si el resto es 0, el M.C.D. es el divisor. Si no, el M.C.D. es el mismo que el del divisor y el resto, por lo que se hace la división entera:

$$\begin{array}{r|l} 6 & 4 \\ \hline 2 & 1 \end{array}$$

Como el resto no es cero se vuelve a hacer la división entera entre el divisor y el resto:

$$\begin{array}{r|l} 4 & 2 \\ \hline 0 & 2 \end{array}$$

Al ser la división exacta el M.C.D. es el divisor (2). Viendo el organigrama de la Fig. 6.2, haz el programa correspondiente.

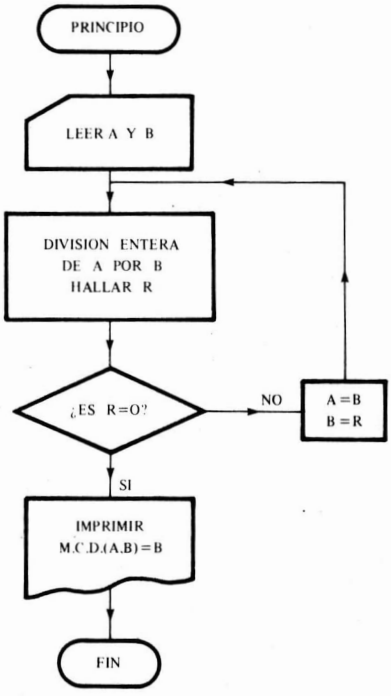


Fig. 6.2

6.6) Completa el programa anterior de forma que calcule también el m.c.m., sabiendo que M.C.D. x m.c.m. = producto de los dos números dados.

6.7) En el apartado 6.2 se ha visto con la función INT un ejemplo para truncar decimales de forma que sólo salgan dos decimales, por ejemplo. De ordinario será más conveniente *redondear* que *truncar*. Si redondeamos el número A = 3,73875431 a dos cifras decimales, se obtiene B = 3,74, cuya aproximación es mayor que la del truncamiento 3,73.

Para redondear a dos cifras decimales puedes usar la sentencia:

LET B = INT(A*1ØØ+.5)/1ØØ

Escribe un programa para redondear a dos cifras decimales cualquier número A. Luego mejora el programa para redondear un número A a n decimales.

6.8) Haz un programa en el que el ordenador, después de pedirte un número natural menor que 100, te conteste la forma de pagar ese dinero con el menor número posible de monedas de 50, 25, 5 y 1 ptas. (al llegar a estudiar el "duro" y las pesetas sueltas, utiliza la función INT).

6.9) Sabiendo que la respuesta es única, hallar un número de cuatro cifras, de la forma aabb, que sea cuadrado perfecto.

6.10) Un número, cuadrado perfecto, más 2 da un cubo perfecto. Hallar el número.

6.11) Sabiendo que la respuesta es única, hallar un número de seis cifras capicúa y cuadrado perfecto.

6.12) Hallar el menor múltiplo de 13 que dividido por 2, 3, ..., 12 da de resto 1.

6.13) Hallar tres números menores que 10 que, elevados al cuadrado y sumados, dan un cuadrado perfecto.

6.14) Un tejado lleva 40 piezas de pizarra por m^2 o fracción. El rendimiento de un obrero es de 10 m^2 por jornada de 8 horas. Cada pieza se coloca con dos clavos, que se venden en cajas de 400 clavos a 1.500 ptas. la caja. Sabiendo que cada pieza de pizarra cuesta 35 ptas. y que la mano de obra cuesta 3.000 ptas./día o fracción, hacer un programa que, tras pedir los

metros cuadrados del tejado del cliente, conteste con el coste por los tres conceptos y el coste total.

6.15) La población de una ciudad ha evolucionado de la siguiente forma:

1935	1.238.400	1960	2.362.057
1940	1.401.107	1965	2.716.366
1945	1.580.880	1970	3.123.821
1950	1.786.055	1975	3.592.394
1955	2.053.963	1980	4.237.525

Queremos representar esta evolución mediante un diagrama de barras horizontales. Para ello deberás utilizar el signo ■. Tras escribir el año y un espacio en blanco, pondrás las barras de la longitud proporcional a la población. Has de cuidar el problema de escalas: dispones de 34 casillas que has de aprovechar al máximo. Por ejemplo a 4.237.525, que es la cifra más alta, le puedes hacer corresponder las 34 casillas, o un número de espacios próximo a 34.

6.16) Simular el lanzamiento de una moneda N veces. Al final interesa saber el número de lanzamientos y el número de caras y cruces. Durante los lanzamientos puedes hacer que cada vez que salga "cara" aparezca en la pantalla un carácter (0 por ejemplo) y por cada "cruz" un signo + (con punto y coma (;) en la instrucción PRINT correspondiente).

6.17) Se lanza una moneda tres veces seguidas. El resultado será una terna formada por caras y cruces (0 y +). Realiza un programa que nos dé diez de estas ternas.

6.18) Queremos simular el lanzamiento de una moneda trucada, de modo que la probabilidad de cara sea 0,75 y la de cruz 0,25. Haz un programa para N lanzamientos, como el ejercicio 16.

6.19) Vamos a simular de nuevo el lanzamiento de una moneda, pero esta vez el programa se parará cuando se estabilicen las frecuencias relativas. Después de cada lanzamiento se halla la frecuencia relativa de las caras. Cuando esa frecuencia difiera de la del lanzamiento anterior en menos de 0,01 el programa parará, indicando el número de lanzamientos, el de caras y cruces, y las dos últimas frecuencias relativas de caras redondeadas a 4 decimales. Piensa las dificultades que pueden presentarse en las primeras tiradas.

6.20) Hacer un programa en el que el ordenador lance monedas, y se pare cuando haya lanzado cuatro caras consecutivas. Dar entonces el número de lanzamientos, el de caras y su frecuencia relativa.

6.21) Haz un programa que simule el lanzamiento de un dado N veces, y que al final saque en la pantalla las frecuencias relativas de 1, 2, ..., 6 y la puntuación media.

El programa puede servir para verificar la función RND: las seis frecuencias relativas deben tener valores próximos, más próximos cuanto mayor sea N.

6.22) Repetir el lanzamiento de un dado, escribiendo cada 10 lanzamientos las frecuencias relativas de cada punto (1, 2, ..., 6).

Haz que el programa se pare cuando en dos lanzamientos consecutivos todas las frecuencias relativas hayan variado menos de 0,02.

6.23) Mediante la función RND la pantalla del ordenador va a proponer a un niño sumas y multiplicaciones. Cuando el niño contesta,el ordenador responde si la solución es correcta o si ha fallado. El programa va contando el número de fallos y aciertos.

6.24) Simula la extracción de los tres primeros premios de la lotería nacional (Números desde el 00000 al 59999). A un mismo número no le pueden tocar dos premios.

6.25) Haz un programa de simulación de una ruleta: tiene 36 números a los que se puede apostar, y el 0. Si sale el 0, todas las apuestas son para la casa. Si aciertas el número, el premio es tu apuesta multiplicada por 36. Además los 36 números se clasifican en pares o impares, y pasa (19 ó mayor) o falta (18 ó menor). (También en rojo y negro, pero esto complicaría el programa). Puedes jugar a pares o impares y a pasa o falta; si ganas en estos casos el premio es tu jugada multiplicada por dos. Empieza el juego con P ptas. y ve haciendo apuestas. El ordenador debe decir cuánto dinero te queda, y no permitirte apostar más de lo que tienes. El juego se acaba si te quedas sin dinero.

6.26) En vez del adulador cibernético vamos a hacer el Criticón Cibernético de Luxe. Mediante la función RND puede elegir entre una lista de críti-

cas. Otra función RND elegirá el número de críticas que se harán (entre dos y cuatro por ejemplo). No importa que una crítica se repita.

6.27) Flechazo al ordenador: la máquina te pregunta el nombre. A continuación lo imprime 15 veces, una en cada línea, situándolo aleatoriamente a la izquierda, derecha o centro de la pantalla, y añadiéndole signos de admiración (!) también en número aleatorio, comprendido entre 1 y 5. (Es una combinación de RND y TAB, y si quieres que el nombre no se salga de una línea para escribirse en la siguinte, has de tener en cuenta su longitud: LEN).

6.28) Cada vez que un submarino avista un barco enemigo sólo tiene tiempo de lanzar cuatro torpedos antes de ser localizado y perseguido. Cada torpedo tiene 1/3 de probabilidades de hundir el barco. Si un torpedo hunde un barco en la pantalla aparecerá H, y no se lanzan más torpedos contra ese barco. Si no le da, aparecerá A y seguirá la serie de cuatro torpedos. Simular unos días de guerra en los que el submarino se encuentra diez barcos enemigos en total. Se sugiere que los resultados con cada barco aparezcan en una sóla línea de pantalla, en total 10 líneas.

6.29) UNA APLICACION DE LA FUNCION RND: EL CALCULO ALEATORIO DE π

Consideremos el cuadrado de lado 1 y el cuadrante de círculo de la Fig. 6.3. Supongamos que lanzamos un dardo al azar sabiendo con seguridad que va a dar en el cuadrado, pero no necesariamente en el cuadrante de círculo.

La probabilidad de que el dardo caiga en el cuadrante es la proporción que hay entre las áreas de las dos figuras. Más concretamente:

$$P = \frac{\text{Area del cuadrante}}{\text{Area del cuadrado}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

Supongamos ahora que lanzamos N dardos sobre el cuadrado y que M de ellos dan en el cuadrante. La frecuencia relativa del suceso "dar en el cuadrante" es:

$$f = \frac{M}{N}$$

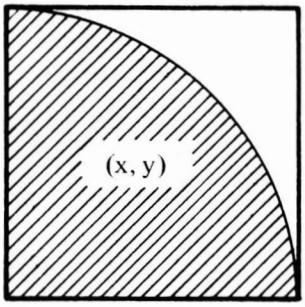


Fig.6.3

Si el número de lanzamientos es muy grande, la frecuencia relativa se aproximará a la probabilidad P, por tanto:

$$f = \frac{M}{N} \simeq \frac{\pi}{4} = P \qquad \text{(para N "grande")}$$

De aquí podemos despejar π :

$$\pi \simeq \frac{4M}{N} \qquad \text{(para N "grande")}$$

Esto es: un valor aproximado de π es igual al cuádruplo del número de dardos que han caído en el cuadrante de círculo, dividido por el número de lanzamientos efectuados.

Para hallar una aproximación no es necesario lanzar los dardos, basta simular los lanzamientos. Para ello obtenemos dos números aleatorios x e y comprendidos entre 0 y 1 (para que el dardo caiga en el cuadrado con seguridad), y suponemos que el dardo ha ido a parar al punto (x, y). El punto está dentro del cuadrante si:

$$x^2 + y^2 < 1$$

Con estas ideas, construye un organigrama y el correspondiente programa, para hallar un valor aproximado de π lanzando mil dardos.

6.30) Representa en una tabla los valores de sen x, cos x, y tg x. Vamos a hacer una presentación elegante. A la izquierda aparecerá el valor de x, desde 1° hasta 45°, variando de 1 en 1. Redondearemos el valor de sen x, cos x y tg x a cinco decimales. Para no "perder" los resultados por la parte superior de la pantalla, debido a la rapidez con que se obtienen, la

ejecución se parará mediante una instrucción STOP, cada 15°. Pon también la cabecera correspondiente.

6.31) Un paisano tiene un prado cuadrado de lado L, y tiene una vaca que pasta en él. Ata a la vaca con una cuerda al centro de uno de los lados. Quiere saber la longitud que ha de dar a la cuerda, para que la vaca alcance exactamente la mitad de la superficie del prado, y así no se indigeste. Introduce L mediante un INPUT. Haz que R vaya creciendo desde R = L/2 hasta que la superficie de pasto sea $L^2/2$, con incrementos de R iguales a L/1000. La fórmula de la superficie de pasto es:

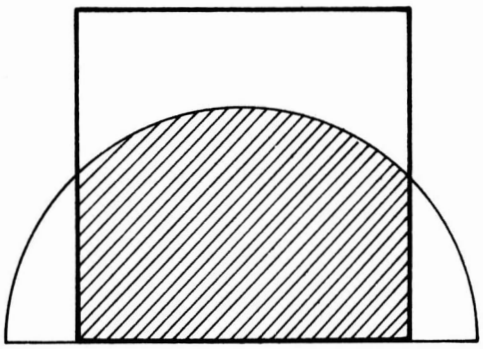


Fig. 6.4

```
S = 3.14159*R↑2/2+L/2*(R↑2-L↑2/4)↑.5-ATN( (R↑2-L↑2/4)↑.5*
                                *2/L)*R↑2
```

6.32) Problema semejante al anterior. Dadas dos circunferencias de igual radio R, hallar a qué distancia D deben estar sus centros para que las tres superficies de la figura 6.5 sean iguales. La fórmula de S_2 es:

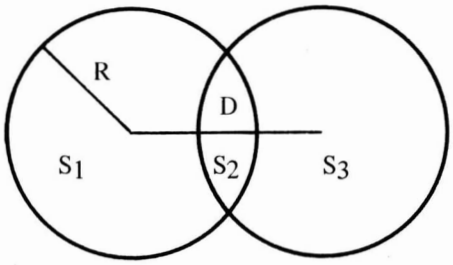


Fig. 6.5

```
S2 = 2*(ATN( (R↑2-D↑2/4)↑.5*2/D)*R↑2-D/2*(R↑2-D↑2/4)↑.5)
```

6.33) Hacer el siguiente programa que será útil para la navegación marítima y

aérea: dados los datos de longitud y latitud de dos puntos de la superficie terrestre, hallar la distancia entre ellos, suponiendo que la Tierra es esférica. Primero habrás de distinguir si los dos puntos están en el mismo hemisferio o en distinto, para restar sus latitudes o sumarlas respectivamente, y lo mismo para la longitud. Después habrás de aplicar la siguiente fórmula:

$$X = \cos(A1)*\cos(A2)$$

$$D = -6367.5 *(\text{arctg}\,(X/\sqrt{1-X^2}\,) + \pi/2)$$

Donde A1 y A2 son, en radianes, las diferencias de longitud y latitud de los dos puntos.

6.34) Ejecuta en tu microordenador el siguiente programa:

```
1Ø REM  REPRESENTACION DE LA FUNCION SENO
2Ø FOR I=Ø TO 2Ø STEP .3
3Ø PRINT TAB(19+19*SIN(I));"*"
4Ø NEXT I
5Ø END
```

6.35) Representa en la pantalla simultáneamente, utilizando la función TAB, las tablas de multiplicar de los números 2, 3, 5, 7 y 9 hasta el factor 15.

6.36) Hacer un ejercicio semejante al 6.3) que represente en la pantalla simultáneamente los 100 primeros números primos.

6.37) Utilizando la función TAB, escribe una tabla que represente:

a) En su extremo izquierdo los nombres abreviados de los equipos de primera división, con no más de 10 caracteres para cada equipo (puedes hacerlo, por sencillez, con un número más reducido de equipos: diez por ejemplo).

b) Para cada equipo una barra horizontal que represente el número de partidos ganados en el Campeonato Nacional de Liga. (Parecido al ejercicio 6.15), pero con la función TAB).

6.38) Una empresa de transporte de viajeros por carretera sirve la línea Madrid-Toledo, con paradas intermedias en Getafe, Parla, Torrejón, Illescas, Cabañas, Yuncos y Olías. Desde las 8 h hasta las 21,30 h salen autobuses en cada sentido, desde los extremos de la línea, cada hora y media. El precio por km es de 3,50 ptas., y además los viajeros que utilicen las es-

taciones de Madrid o Toledo deberán de pagar un extra de 5 ptas. Supondremos que no hay limitación de plazas. La empresa necesita un programa parecido a los de la RENFE, que tras preguntar día, hora, origen y final del trayecto y distancia en kms , rellene el siguiente billete:

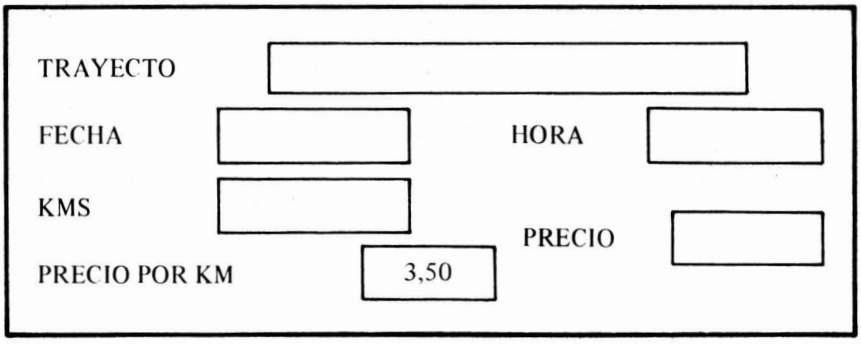


Fig. 6.6

6.39) Hacer el programa anterior pero sin necesidad de que se introduzcan los kilómetros desde el teclado: que, según el trayecto deseado, el programa incluya la distancia entre esas dos poblaciones.

6.40) Representa la función $y = x^2$ en el intervalo $(-7, 7)$ en un programa semejante al 6.34).

6.41) Escribir en la pantalla, como si fueran los vértices de un cuadrado las siguientes letras:

A B

D C

Hacer un programa que, mediante un INPUT, "gire" las letras en el sentido de las agujas del reloj si pulsamos R, y en sentido contrario si pulsamos C.

6.42) Hacer un programa que, dados dos puntos en la pantalla por sus "coordenadas" (fila y columna), trace una línea discontinua de asteriscos entre ambos.

6.43) Hacer un programa que dé una tabla de valores de cualquier función entre dos números y para un incremento dado. Te sugerimos que las

instrucciones para el usuario vayan en las primeras líneas de la siguiente forma:

```
1Ø PRINT "TABLA DE VALORES DE UNA FUNCION"
2Ø PRINT "=============================="
3Ø PRINT
4Ø PRINT "      SI NO HAS DEFINIDO AUN LA FUNCION"
5Ø PRINT "HAZLO AHORA ESCRIBIENDO PREVIAMENTE"
6Ø PRINT "1ØØ DEF FNY(X)=..."
7Ø PRINT "      SI YA LA HAS DEFINIDO TECLEA RUN 1ØØ"
8Ø STOP
```

6.44) Haz un programa para calcular la integral definida de una función en un intervalo, semejante al ejercicio 5.14) pero utilizando la definición de una función. Puedes utilizar un comienzo semejante al del ejercicio anterior.

6.45) RAICES REALES DE UNA ECUACION $f(x)=0$ POR BIPARTICION

Si una función continua en un intervalo toma valores de distinto signo en sus extremos, es que la función se anula en algún punto intermedio. (Teorema de Bolzano). Vamos a hallar por bipartición ese punto (al menos uno) en que se anula.

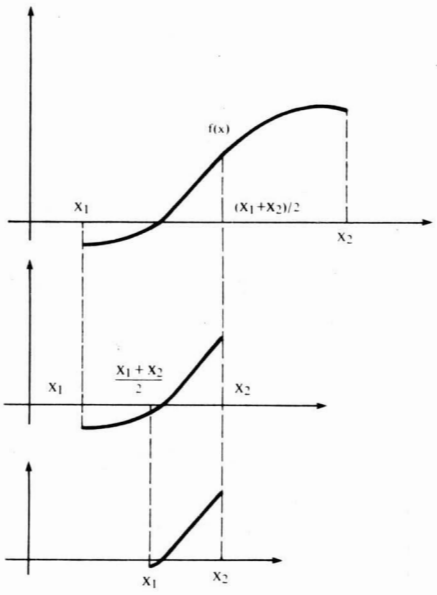


Fig. 6.7

En la figura 6.7 se representa el caso de que f(X1) es negativo y f(X2) es positivo. (El caso contrario es análogo). Tomamos el *punto medio* del intervalo [X1, X2], y vemos el valor de la función. Supongamos que es positivo (como en la figura). Entonces la raíz ha de estar entre X1 y (X1 + X2)/2; llamamos a este punto medio X2 y prescindimos del estudio de la función a su derecha.

Se repite el mismo estudio en el nuevo intervalo más pequeño, se vuelve a tomar el punto medio, y vemos el valor de la función. Supongamos que es negativo (como en la figura). Entonces la raíz ha de estar entre (X1 + + X2)/2 y X2; llamamos a este punto medio X1 y prescindimos del estudio de la función a su izquierda.

Tal vez en alguna de estas biparticiones podemos dar exactamente con la raíz: f [(X1 + X2)/2] = 0; lo que sí es seguro es que cada vez nos acercamos más a ella. Al ordenador no le cuesta nada operar y repetir todas las veces que haga falta este proceso, hasta tener acotada la raíz en un intervalo [X1, X2] muy pequeño. Si tomamos entonces su punto medio como raíz, cometemos un error menor que (X1−X2)/2.

Apoyándote en estas ideas, y en el siguiente organigrama, haz un programa para hallar raíces. La función puede introducirse mediante la instrucción DEF FN..., y el intervalo y error mediante INPUT.

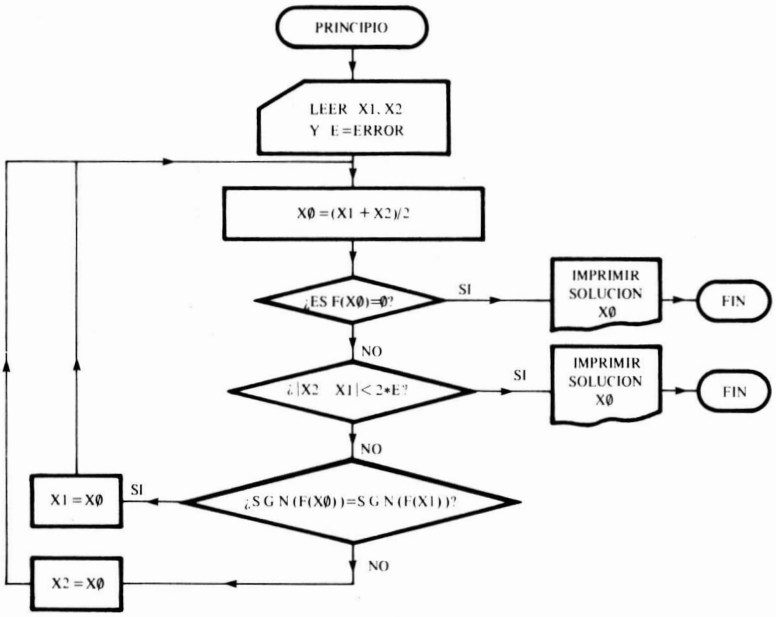


Fig. 6.8

6.46) VALOR MAXIMO DE UNA FUNCION

Diseña el organigrama y el programa correspondiente para hallar el valor máximo de una función cualquiera, f, en un intervalo [a,b]. Hay que utilizar un algoritmo semejante al del ejercicio anterior: divide cada vez el intervalo, no en dos partes, sino en diez, cuyas abscisas serán X_0, X_1, X_2, ..., X_{10}; toma el valor X_i que haga mayor la función; como nuevo intervalo a estudiar tomarás $[X_{i-1}, X_{i+1}]$ que volverás a dividir en diez, y así sucesivamente, hasta obtener una aproximación satisfactoria. Piensa que en algunos casos el método puede fallar.

6.47) Haz un programa que, mediante un INPUT, pida una frase y vuelva a escribirla, pero intercalando un guión entre cada dos caracteres. (Observación: el espacio en blanco también es un carácter a separar entre guiones de las letras contiguas).

6.48) Haz un programa que, tras pedir una "frase sin signos de puntuación", cuente el número de vocales, espacios en blanco y consonantes de la frase. (El número de consonantes, si no hay signos de puntuación, es la longitud de la cadena menos el número de vocales y el de espacios en blanco).

6.49) Necesitamos un programa que pida, mediante un INPUT, una frase. Luego ha de preguntar qué carácter deseas buscar; examinará la cadena y cada vez que encuentre el carácter señalará en qué posición está. Al final sacará el número total de veces que lo ha encontrado.

6.50) Queremos contar cuántas veces aparece una palabra en una frase. Si, por ejemplo, buscamos la palabra "sol", es obvio que las palabras "sola", o "solar", no nos sirven: la palabra "sol" debe tener a derecha e izquierda un espacio en blanco, o un signo de puntuación. Como ejemplo concreto cuenta las veces que aparece "vuesa merced" en un capítulo del Quijote.

6.51) Haz un programa que suprima todos los espacios en blanco de una cadena.

6.52) Haz un programa en el que el ordenador te pida un número de más de cinco cifras y posteriormente te dé las dos cifras de la izquierda, las cuatro de más a la derecha, y la del centro. (Si el número de cifras es par, en el centro no hay una cifra, sino dos. Toma la que prefieras de ellas).

6.53) Queremos un programa que pida al teclado un número positivo de seis o

más cifras. Vamos a ir haciendo una serie de operaciones y queremos que en la pantalla aparezcan los sucesivos resultados parciales, con los textos explicativos correspondientes: el número se multiplicará por dos; nos quedaremos con las cuatro cifras de la izquierda; nos las escribirá de nuevo pero separadas por guiones; y por fin el resultado de sumar esas cuatro cifras.

6.54) Haz un programa que te pida el infinitivo de un verbo de la primera conjugación, y a continuación escriba su pretérito imperfecto y su presente de subjuntivo.

6.55) OTROS EJERCICIOS CON CADENAS DE CARACTERES

Vamos a utilizar otras dos funciones ya más complejas que son: ASC(A$) y CHR$(N). Cada carácter del BASIC tiene un código numérico. El código ASCII de los caracteres que más te interesan es:

CARACTER	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
CODIGO	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81
CARACTER	R	S	T	U	V	W	X	Y	Z	Ø	1	2	3	4	5	6	7
CODIGO	82	83	84	85	86	87	88	89	90	48	49	50	51	52	53	54	55
CARACTER	8	9	"	$	(	)	*	+	,	–	.	/	;	<	=	>	?
CODIGO	56	57	34	36	40	41	42	43	44	45	46	47	59	60	61	62	63
CARACTER	↑		ESPACIO				RETURN				DELETE						
CODIGO	94		32				13				20						

La función ASC(A$) asigna a un carácter su código, y la función CHR$(N) asigna a un código N el carácter correspondiente. Mira el siguiente programa y el resultado de su ejecución:

```
1Ø  A=6
2Ø  B=ASC("6")              (B vale 54)
3Ø  C$=CHR$(12*A)           (C$ vale "H")
4Ø  C=ASC(C$)               (C vale 72)
5Ø  D$=CHR$(B+2Ø)           (D$ vale "J")
6Ø  PRINT B;C$;C;D$
7Ø  END
RUN
54 H 72 J
```

Un ejemplo de programa puede ser la escritura en clave. Imagínate que un espía dispone de un ordenador para su trabajo. Si tiene que transmitir un mensaje, puede traducir las letras a sus códigos; después opera con los códigos (por ejemplo les suma a todos el número 4) y los descodifica de nuevo, pasando los nuevos códigos a sus correspondientes caracteres. Resulta algo completamente desfigurado, que sólo sabrá entender quien conozca la clave.

```
1Ø REM ESCRITURA EN CLAVE
2Ø PRINT "MENSAJE"
3Ø INPUT A$
4Ø FOR I=I TO LEN(A$)
5Ø B$=MID$(A$,I,1)          (se toma un carácter)
6Ø B=ASC(B$)+4              (se pasa a código y se suma 4)
7Ø C$=CHR$(B)               (se pasa al nuevo carácter)
8Ø D$=D$+C$                 (se acumula en D$ el mensaje)
9Ø NEXT I
1ØØ PRINT D$                (se escribe el mensaje cifrado)
11Ø END
```

Si al INPUT de la línea 3Ø, en la ejecución del programa, contestas "OPERACION RINOCERONTE", este programa escribe:

"STIVEGMSR$VMRSGIVSRXI"

que sólo sabrá descifrar quien conozca la "clave": restar 4 a los códigos de las letras.

Estas funciones se utilizan para ordenar alfabéticamente listas de palabras, pero este tipo de ejercicios requiere utilizar variables de índice, que es el objeto del próximo capítulo. Los ejercicios los haremos después.

6.56) Dos trenes T1 y T2 tienen parte del recorrido común (L3) y parte distinto (L1 y L2), y circulan en un circuito cerrado en la misma dirección. Van con velocidad constante V1 y V2, y tienen una estación en la mitad de su trayecto común y otra en la mitad de su trayecto particular (E1 y E2). Salen a las 8 de la mañana y se pasan el día circulando con paradas de 1 minuto en las estaciones. Como es natural no se pueden adelantar en el trayecto común, aunque sí pueden coincidir en E3. Haz un programa en el que salga el horario en el que van pasando por las agujas A y B. En cada vuelta el primero que pasa por A ha de ser el mismo que el

primero que pasa por B; si no, hay descarrilamiento. Cuando haya descarrilamiento saca un mensaje en la pantalla señalando la hora del choque, y termina el programa. (Es difícil. Haz el organigrama).

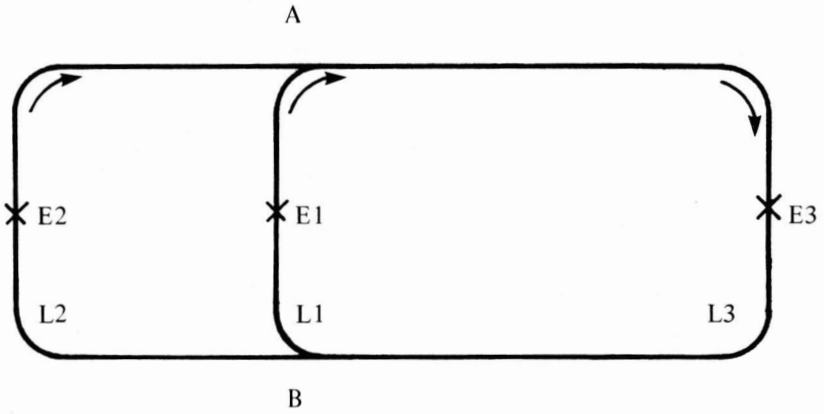


Fig. 6.9

7
LISTAS Y TABLAS

7.1 VARIABLES CON INDICE

Hasta aquí hemos visto un tipo de variables, tales como A, A1, A2, etc. El lenguaje BASIC permite definir otro tipo de variables numéricas:

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

Se llaman *variables con índice* y están formadas por un nombre de variable, que en el ejemplo es A, seguido de un número natural entre paréntesis.

Conviene imaginarse las variables con índice como cajas contiguas que guardan números. Así por ejemplo:

A(1)	1
A(2)	4
A(3)	9
A(4)	16
A(5)	25

El conjunto ordenado de estas variables se llama *lista.* En el ejemplo anterior tenemos una lista de cinco números.

Existen también *variables de cadena con índice* donde se pueden guardar, por ejemplo, los nombres de una lista:

A$(1)	JOSE PEREZ FERNANDEZ
A$(2)	LUIS GARCIA GARCIA
A$(3)	ANA GOMEZ RAMIREZ
A$(4)	ISABEL RODRIGUEZ RUIZ
A$(5)	LORENZO RUIZ GARCIA

Las variables con índice hacen aumentar notablemente el número de variables que podemos manejar en BASIC. Sin embargo, ésta no es ni la única, ni la más importante ventaja de las variables con índice. Lo más interesante es que *el índice puede ser variable.* Así por ejemplo, la variable A(N) será A(1) si N es 1, será A(2) si N es 2, etc. La posibilidad de que el índice varíe hace realmente útiles a estas variables. Veámoslo en algunos ejemplos:

Supón que quieres definir la lista numérica anterior. El siguiente programa podría servir para ello:

```
1Ø REM  DEFINICION DE UNA LISTA
2Ø FOR  N=1  TO  5
3Ø LET  A(N)=N*N
4Ø NEXT  N
5Ø END
```

Si el valor de la variable L(N) no guardara relación con el índice N, como ocurre con la lista:

3.25	Ø.3	4.6	7.8	Ø.76	4.3	2Ø
L(1)	L(2)	L(3)	L(4)	L(5)	L(6)	L(7)

podríamos definir la lista L(N) mediante INPUT dentro de un ciclo FOR-NEXT:

```
1Ø REM  DEFINICION DE UNA LISTA ARBITRARIA
2Ø FOR  N=1  TO  7
3Ø INPUT  L(N)
4Ø NEXT  N
5Ø END
```

Corriendo el programa introduciremos sucesivamente los valores de la lista:

```
RUN
? 3.25
? Ø.3
  •
  •
  •
? 2Ø
```

El índice de una variable puede ser una fórmula. Por ejemplo, A(2*I+3). Si el valor de la fórmula no es un entero positivo, el ordenador toma como índice la parte entera de la fórmula.

Algunos ordenadores admiten como índice el Ø; es decir, tienen variables tales como A(Ø), L(Ø), etc.

Cuando definimos una variable de índice A(N), el ordenador reserva automáticamente posiciones de memoria para las variables A(1), A(2),... hasta A(1Ø). Si nuestra lista tiene más de 10 elementos, por ejemplo 100, entonces hay que advertir al ordenador que reserve sitio para 100 variables. Esta reserva de plaza se hace con la instrucción DIM A(1ØØ), precedida de un número de línea. DIM es una abreviatura de DIMENSION. El número encerrado entre paréntesis es el índice más alto que pensamos utilizar en el programa.

Supongamos que queremos simular cien lanzamientos de un dado, guardar los resultados en una lista, D(1), D(2), ..., (D1ØØ) y por último imprimirlos:

```
1Ø REM  CIEN TIRADAS DE UN DADO
2Ø DIM  D(1ØØ)
3Ø FOR  J = 1  TO  1ØØ
4Ø LET  D(J) = INT(6*RND(J) ) + 1
5Ø NEXT  J
6Ø FOR  N = 1  TO  1ØØ
7Ø PRINT  D(N);
8Ø NEXT  N
9Ø END
```

Dado que las variables con índice ocupan mucha memoria, únicamente las usaremos cuando realmente necesitemos guardar los resultados. Si en el ejemplo anterior lo único que se pretendía era la impresión en pantalla de los resultados, no necesitábamos guardar cada resultado en una variable con índice, sino, sencillamente, imprimirlos a medida que los íbamos obteniendo.

7.2 FABRICACION DE BOTELLAS

Una primera aplicación de las variables con índice puede ser la resolución de un problema clásico: en la pasta de vidrio para la fabricación de cien botellas hay cincuenta partículas de impureza que se distribuyen al azar durante el proceso de fabricación. Si una botella resulta con una o más partículas de impureza, es considerada defectuosa. ¿Como cuántas botellas resultaron aceptables en esta fabricación?

Vamos a resolver el problema simulando el proceso de fabricación. Para ello consideramos una cuadrícula de 10 x 10, como la de la Figura 7.1, y numeramos las casillas del 1 al 100. Cada casilla representa una botella.

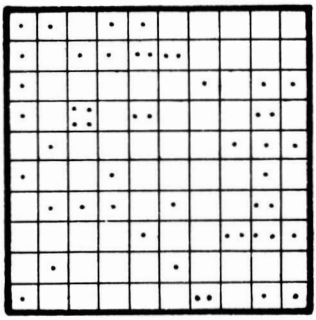


Fig. 7.1

Ahora obtenemos un número aleatorio entre 1 y 100. Supongamos que sale el 33. Esto quiere decir que a la botella número 33 le ha tocado una partícula de impureza. En la casilla correspondiente nosotros marcaremos un punto. Repitiendo la operación otras 49 veces, tendremos 50 puntitos repartidos entre las casillas. El número de casillas libres será el número de botellas buenas.

Todo esto se puede realizar en el ordenador, estableciendo las variables de índice B(1), B(2), ..., B(100) y sumando una unidad a B(N) cada vez que aparece el número aleatorio N. Después de las 50 extracciones de números aleatorios, contamos cuántas variables B(I) tienen valor cero; así obtenemos el número de botellas en buen estado.

```
10 REM  FABRICACION DE BOTELLAS
20 DIM  B(100)
30 FOR  I=1  TO  50
40 LET  N=INT(100*RND(I) ) + 1
50 LET  B(N)=B(N)+1
60 NEXT  I
70 REM  RECUENTO DE BOTELLAS BUENAS
80 FOR  J=1  TO  100
90 IF  B(J)=0  THEN  LET  C=C+1
100 NEXT  J
110 PRINT  "NUMERO DE BOTELLAS BUENAS=";C
120 END
```

Corriendo el programa hemos obtenido 61 botellas buenas.

7.3 EL MAYOR DE LOS NUMEROS DE UNA LISTA

Un problema que se te presentará con alguna frecuencia será el de encontrar el mayor de los números de una lista y el lugar que ocupa en ella. Por ejemplo, en la lista:

16	13	9	2	−4	0	23	4
L(1)	L(2)	L(3)	L(4)	L(5)	L(6)	L(7)	L(8)

el número mayor es el 23 y ocupa el lugar 7.

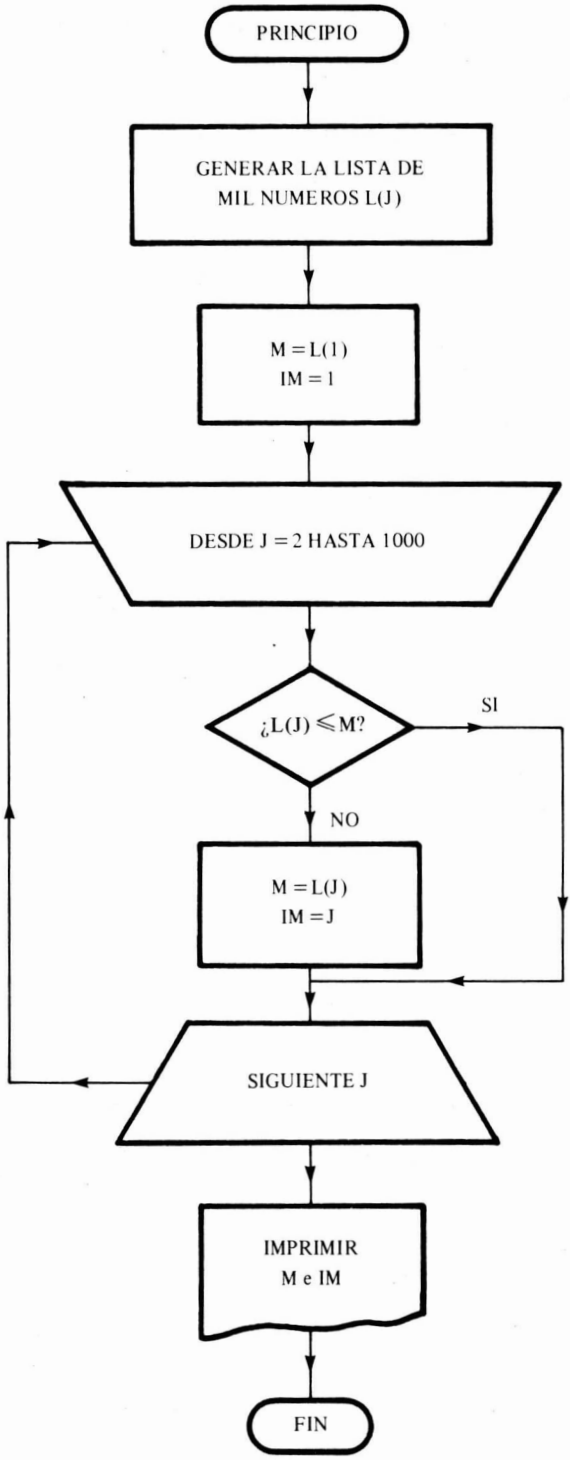

PRINCIPIO
GENERAR LA LISTA DE MIL NUMEROS L(J)
M = L(1)
IM = 1
DESDE J = 2 HASTA 1000
¿L(J) ≤ M?
SI
NO
M = L(J)
IM = J
SIGUIENTE J
IMPRIMIR M e IM
FIN

Fig. 7.2

Nosotros resolvemos inmediatamente el problema: un vistazo a la lista y descubrimos que el 23 es el mayor de los números y que su lugar es el 7. Pero debemos ser conscientes de todas las pequeñas operaciones que hemos realizado para llegar a esta conclusión, de lo contrario nunca se lo podremos comunicar al ordenador. El pequeño tamaño de la lista hace que no reparemos en el proceso mental de búsqueda del valor máximo. Será bueno, entonces, que imaginemos una lista de mil números, por ejemplo.

Ahora tendremos que recorrer la lista desde el principio hasta el fin. Necesitaremos una variable M, donde vamos a guardar el mayor "provisional", y otra variable IM, en la que guardaremos el lugar que ocupa dicho valor máximo "provisional". Inicialmente hacemos M=L(1) e IM=1, esto quiere decir que en un principio tomamos como "número mayor" el primero de la lista. El valor de M lo comparamos con L(2); si L(2) es mayor que M, hacemos M=L(2) e IM=2, en caso contrario dejamos las variables como están. Luego hacemos la comparación de M con L(3), y así sucesivamente, hasta llegar a la comparación con L(1000). Al final, en M está el mayor de los números y en IM el lugar que ocupa dicho valor máximo. Podría ocurrir que el valor máximo se encontrara en varios lugares de la lista; en este caso en IM se recoge sólo el primero de los lugares con valor máximo. En el organigrama de la izquierda (Fig. 7.2), al que incorporamos como paso previo la generación de la lista, se expresan estas ideas.

He aquí un programa en el que previamente se genera una lista de mil números aleatorios comprendidos entre 1 y 10000.

```
10  REM  VALOR MAXIMO DE UNA LISTA DE MIL NUMEROS
20  REM  GENERACION DE LA LISTA
100  DIM  L(1000)
110  FOR  J = 1  TO  1000
120  LET  L(J) = INT(10000*RND(J) ) + 1
130  NEXT  J
200  REM  BUSQUEDA DEL MAXIMO
210  LET  M = L(1)
220  LET  IM = 1
230  FOR  J = 2  TO  1000
240  IF  L(J) <= M  THEN  270
250  LET  M = L(J)
260  LET  IM = J
270  NEXT  J
280  PRINT  "EL MAYOR NUMERO ES";M
290  PRINT  "QUE OCUPA EL LUGAR";IM
300  END
```

7.4 ORDENACION DE LISTAS

Un problema clásico es ordenar de menor a mayor los números de una lista de N elementos: L(1), L(2), L(3), ..., L(N).

Con objeto de economizar memoria, la ordenación se hace *in situ*, es decir, sobre la misma lista, cambiando los números de unos lugares a otros. Los métodos de ordenación pueden clasificarse en tres categorías:

1. Ordenación por intercambio
2. Ordenación por inserción
3. Ordenación por selección

A continuación vamos a exponer estos métodos. Te aconsejamos vivamente que cortes una hoja de papel en ocho trozos y escribas un número en cada uno. Colocando los papeles en un orden arbitrario, trata de ponerlos en orden creciente y estudia las comparaciones y movimientos que hay que hacer hasta llegar a la lista ordenada.

7.4.1 ORDENACION POR INTERCAMBIO

Se trata de comparar, recorriendo la lista L(1), L(2), ..., L(N) desde el principio, elementos contiguos, intercambiándolos si no están en orden. La primera vez se compara L(1) con L(2), y se pone en L(2) el mayor de ambos. A continuación se compara L(2) con L(3), haciendo que L(3) valga el mayor de los dos, y así hasta L(N). Al terminar de recorrer la lista, tendremos el mayor elemento colocado en último lugar, es decir, bien colocado. Por tanto, en la segunda vuelta la comparación ha de hacerse hasta el penúltimo elemento. Hay que recorrer la lista N-1 veces, pues cuando tengamos colocados el último, el penúltimo, ..., el segundo; el que queda es el primero.

```
100 REM  ORDENACION POR INTERCAMBIO
110 FOR  I=1  TO  N-1
120 FOR  J=1  TO  N-I
130 IF  L(J)<=L(J+1)  THEN  170
140 LET  X=L(J)
150 LET  L(J)=L(J+1)
160 LET  L(J+1)=X
170 NEXT  J
180 NEXT  I
190 END
```

Observación: Si deseas ensayar este programa en el ordenador, deberás previamente, definir la lista L(J) y asignar el valor que corresponda a la variable N. Después deberás dar instrucciones para que se imprima la lista en el nuevo orden. Así podrás comprobar que los números quedan ordenados de menor a mayor.

7.4.2 ORDENACION POR INSERCION

Este método lo usan generalmente los jugadores de cartas para colocarlas y los profesores para ordenar los ejercicios de los alumnos alfabéticamente.

Se recorre la lista L(1), L(2), ..., L(N) de izquierda a derecha y cada elemento se coloca en su sitio en la parte izquierda de la lista previamente ordenada. Se toman los dos primeros elementos y se ordenan si no lo estaban. Después se toma L(3) y se compara con los anteriores, poniéndolo en primero, segundo o tercer lugar, según le corresponda. Y así se sigue hasta L(N).

```
200 REM ORDENACION POR INSERCION
210 FOR I=2 TO N
220 FOR J=1 TO I-1
230 IF L(J)<=L(I) THEN 295
240 LET X=L(I)
250 FOR K=I TO J+1 STEP -1
260 LET L(K)=L(K-1)
270 NEXT K
280 LET L(J)=X
290 GO TO 297
295 NEXT J
297 NEXT I
299 END
```

Hacemos la misma observación que para el método anterior.

7.4.3 ORDENACION POR SELECCION DEL MINIMO

Es el método que utilizamos cuando queremos reescribir una lista de números en orden creciente: se selecciona el más pequeño de la lista y se coloca en primer lugar, de la lista restante se selecciona el más pequeño y se coloca en segundo lugar, y así sucesivamente. Se realizan estas operaciones N−1 veces, puesto que cuando tengamos ordenados todos excepto uno, éste es el último.

```
300 REM  ORDENACION POR SELECCION
310 FOR  I=1  TO  N-1
320 LET  MIN=L(I)
330 FOR  J=I  TO  N
340 IF  MIN<=L(J)  THEN  370
350 LET  MIN=L(J)
360 LET  K=J
370 NEXT  J
380 LET  L(K)=L(I)
390 LET  L(I)=MIN
395 LET  K=0
397 NEXT  I
399 END
```

Hay que hacer la misma observación que en los dos métodos anteriores.

7.4.4 ORDENACION POR SELECCION DEL MINIMO Y DEL MAXIMO

Es una transformación del método anterior (selección del mínimo). Se recorre la lista para seleccionar el mínimo y el máximo, que se colocan en primer y último lugar respectivamente. En la parte restante de la lista se vuelven a seleccionar el mínimo y el máximo, para colocarlos en segundo y penúltimo lugar, y así sucesivamente.

```
400 REM  ORDENACION  POR  SELECCION  DEL  MINIMO  Y  DEL
     MAXIMO
405 FOR  I=1  TO  INT(N/2)
410 LET  MI=L(I)
415 LET  MA=L(I)
420 FOR  J=I  TO  N-I+1
425 IF  MI<L(J)  THEN  440
430 LET  MI=L(J)
```

```
435 LET K1 = J
44Ø IF MA > L(J) THEN 455
445 LET MA = L(J)
45Ø LET K2 = J
455 NEXT J
46Ø LET L(K1) = L(I)
465 LET L(I) = MI
47Ø IF K2 = I THEN K2 = K1
475 LET L(K2) = L(N-I+1)
48Ø LET L(N-I+1) = MA
485 NEXT I
49Ø END
```

En este programa hay que hacer la misma observación que en los anteriores.

7.5 VARIABLES CON INDICE DOBLE

Hasta aquí hemos utilizado variables con un sólo índice. Usaremos ahora variables con dos índices, de la forma:

T(2,4)

Las variables con índice doble se utilizan para representar tablas de números:

T(1,1)	T(1,2)	T(1,3)	T(1,4)
T(2,1)	T(2,2)	T(2,3)	T(2,4)
T(3,1)	T(3,2)	T(3,3)	T(3,4)

Esta es una tabla de tres filas y cuatro columnas. Los índices de estas variables también pueden variar. Así, T(I,J) es la variable T(2,4) cuando I es 2 y J es 4.

Cuando en un programa se establece una variable T(I,J) con índice doble, el ordenador automáticamente reserva lugar de memoria para todas las variables que se obtienen variando los índices I y J hasta un valor máximo de 10. Si alguno de los dos índices va a superar el valor de 10, entonces debes hacer una

reserva con la instrucción DIM. Así, por ejemplo, DIM T(12,14) reserva las variables T(I,J) para valores I $\leqslant$ 12 y J $\leqslant$ 14. Podremos disponer, entonces, de un una tabla de 12 filas y 14 columnas. Si tu ordenador admite el índice Ø, entonces dispones de una fila y una columna más.

También puede interesarte utilizar la instrucción DIM para ahorrar memoria. Si vas a manejar una tabla de 3 x 4 y escribes DIM T(3,4), reservas espacio sólo para doce variables y no para cien.

Existen también variables de cadena con índice doble, como, por ejemplo T$(I,J).

Con objeto de mostrar el manejo de las variables con dos índices vamos a dar un programa muy simple: se trata de asignar, mediante INPUT, valores a la variable A(I,J) de 12 filas y 3 columnas; y, posteriormente, imprimir la tabla en la pantalla del ordenador.

```
1Ø   REM  IMPRESION DE UNA TABLA DE NUMEROS
2Ø   DIM  A(12,3)
3Ø   FOR  I=1  TO  12
4Ø        FOR  J=1  TO 3
5Ø             INPUT  A(I,J)
6Ø        NEXT  J
7Ø   NEXT  I
8Ø   FOR  I=1  TO  12
9Ø        FOR  J=1  TO  3
1ØØ            PRINT  A(I,J),
11Ø       NEXT  J
12Ø       PRINT
13Ø  NEXT  I
14Ø  END
```

Observa la coma (,) final de la línea 1ØØ: es necesaria para que las impresiones de los tres elementos de cada línea de la tabla se efectúen en la misma línea de pantalla y separados.

La instrucción PRINT de la línea 12Ø es necesaria para que la impresión del primer elemento de cada línea de la tabla se haga al comienzo de una línea de pantalla. Si no tuviéramos esta instrucción PRINT, el primer elemento de la segunda fila de la tabla se imprimiría en la cuarta zona de la primera línea de pantalla, y así sucesivamente.

RECUERDA	
Variables con un índice: lista	A(3)
El índice puede ser variable e incluso una expresión	A(I) A(3*I–5)
Dimensionar la variable si tiene más de 10 elementos	DIM A(5Ø)
Ordenación de listas	
Variables con dos índices: tabla	T(3,4), T(I,J), T(I+3, J*2), DIM T(12,14)
Variables de cadena con índice	A$(3), A$(I,J), DIM A$(35)

EJERCICIOS

7.1) Haz un programa que sume los elementos de una lista de números. Previamente deberás generar la lista.

7.2) Simula el lanzamiento de un dado mil veces; cuenta las frecuencias de cada número utilizando una variable de índice e imprime las frecuencias obtenidas. Es parecido al ejercicio 6.21), y comprobarás la ventaja de programarlo con variables de índice.

7.3) Haz un programa para hallar el menor de los números de una lista y el lugar que ocupa.

7.4) Haz un programa para hallar el menor y el mayor de los números de una lista.

7.5) Dada una tabla de 13 x 13 números, tomar cada fila y sumar todos sus elementos; y lo mismo con las columnas. Hacer al final la suma de los

totales de las filas y la de los totales de las columnas y comprobar que ambas coinciden.

7.6) Haz un programa que calcule la media aritmética y la desviación típica de una lista de números.

7.7) Construye un organigrama y un programa para que, introduciendo el nombre y la fecha de nacimiento de cien personas, el ordenador imprima el nombre y la fecha de nacimiento de la persona más joven. (Supondremos que sólo hay una persona "más joven". Te conviene escribir las fechas de nacimiento como un sólo número: 16 de marzo de 1965 = = 19650316).

7.8) Dada una tabla de cinco filas y tres columnas escribir dicha tabla y su traspuesta.

7.9) Consideremos en la parte superior izquierda de la pantalla, diez líneas y veinticinco espacios. Imprimir en ella cincuenta puntos aleatoriamente.

7.10) A unas elecciones se presentan N candidatos; introduciremos sus nombres en el ordenador mediante variables alfanuméricas con índice. Después de la votación se recuentan los votos: mediante la función RND iremos introduciendo los votos en variables numéricas con índice, correlativas (con igual índice) a las variables de los nombres de los candidatos, y esto hasta llegar a V votos. La pantalla mostrará al final la lista ordenada de los candidatos según los votos obtenidos.

7.11) Simular unas carreras de caballos con apuestas. Primero darás nombre a los caballos participantes mediante variables alfanuméricas con índice. Después se admitirán apuestas, que irás introduciendo en variables numéricas con el mismo índice que el caballo al que se juega. Cada boleto de apuestas vale 100 ptas. Por último, mediante la función RND, el ordenador dará el nombre del caballo ganador, dividirá el dinero total recaudado, entre el dinero apostado al caballo ganador, cifra que multiplicada por 100 señala el dinero que se entregará por cada boleto acertante.

7.12) Simular una carrera de once caballos numerados del 2 al 12. El avance de cada caballo se decide lanzando dos dados: la suma obtenida es el

número del caballo que avanza un espacio. Se acaba la carrera cuando el primero llega a la meta. (Conviene representar el avance de los caballos por una especie de diagrama de barras).

7.13) De los cuatro métodos de ordenación de listas estudiados en este capítulo ¿cuál crees tú que es el más rápido? Diseña un programa que te permita medir estos tiempos. Puedes cronometrar con tu reloj o, mejor aún, con el reloj interno del ordenador (¡si lo tiene!).

7.14) Representa en la pantalla todas las permutaciones de la palabra BASIC, obligando a que el segundo y cuarto lugar los ocupe una vocal.

7.15) De una baraja española, el ordenador debe repartir cinco cartas a cada uno de los cuatro jugadores de una partida. Una carta ya dada no puede volver a salir. Puedes utilizar variables con dos índices: para el palo y para el número. La variable valdrá 0 si esa carta no se reparte, y 1, 2, 3 ó 4, si se reparte al jugador correspondiente.

7.16) Haz un programa para jugar a las siete y media.

7.17) Siguiendo las instrucciones del ejercicio 6.55) y los métodos de ordenación de listas de este capítulo, haz un programa para ordenar alfabéticamente una lista de nombres. Puedes hacerlo por varios métodos.

7.18) Haz un programa por el que el ordenador pregunte los nombres de N personas, y nos permita archivar su edad, estado, peso e ingresos anuales (0 en su caso). Después el programa ha de preguntar qué dato queremos repasar, por ejemplo: edad 35, o peso 72, etc. A cada respuesta nuestra contestará la máquina con la lista de personas que se encuentran en esa situación (o NINGUNO en su caso).

7.19) Un almacén tiene departamentos de ferretería, droguería y bebidas. Asigna diez artículos a cada departamento mediante variables de cadena con índice. Por ejemplo F$(1)="TORNILLOS", D$(1)="DETERGENTE", B$(1)="VINO", F$(2)=... y así hasta diez. El almacén sólo atiende a las personas o entidades que tienen contrato con él, para el correspondiente departamento: Puedes asignar a S$(I,J) los nombres de los socios (I) abonados al departamento J (J vale 1, 2 ó 3). Deseamos que cuando llegue alguien a comprar, se verifique primero si tiene contrato con el departamento que busca; en caso afirmativo que aparezca la lista de artículos de

ese departamento y se le pregunte cuál desea. Has de prever el mensaje correspondiente a error en la petición.

7.20) LA COLECCION DE CROMOS

¿Cuántos cromos hay que comprar para completar un álbum de 100? Se supone que compramos los cromos de uno en uno y que todos tienen la misma probabilidad de salir. El álbum consistirá en las variables de índice:

C(1), C(2), C(3), ..., C(100)

Comprar un cromo es equivalente a extraer un número aleatorio del 1 al 100, (I), y sumar la unidad a la variable C(I), como hicimos en la fabricación de botellas.

Desde el principio escrutamos el álbum para ver si faltan cromos (aunque sabemos que antes de las cien primeras extracciones es imposible que el álbum esté completo). Desde el momento en que encontramos una variable con valor cero (no hay cromos) compramos otro cromo, pero guardando en una variable auxiliar este índice, que representa el número del primer cromo que falta. El siguiente escrutinio lo empezaremos, precisamente, por dicho número.

Cuando hayamos comprobado que el valor de todas las variables es distinto de cero hacemos imprimir el número de extracciones de cromos, que previamente hemos tenido que ir contando. Si, además, hacemos imprimir los valores de las variables C(I) en forma de una tabla de 10 x 10, podemos saber cuántas veces ha salido cada cromo.

A continuación proponemos un organigrama para este problema, a partir del cual tienes que diseñar tu programa que, en definitiva, es lo que se te pide.

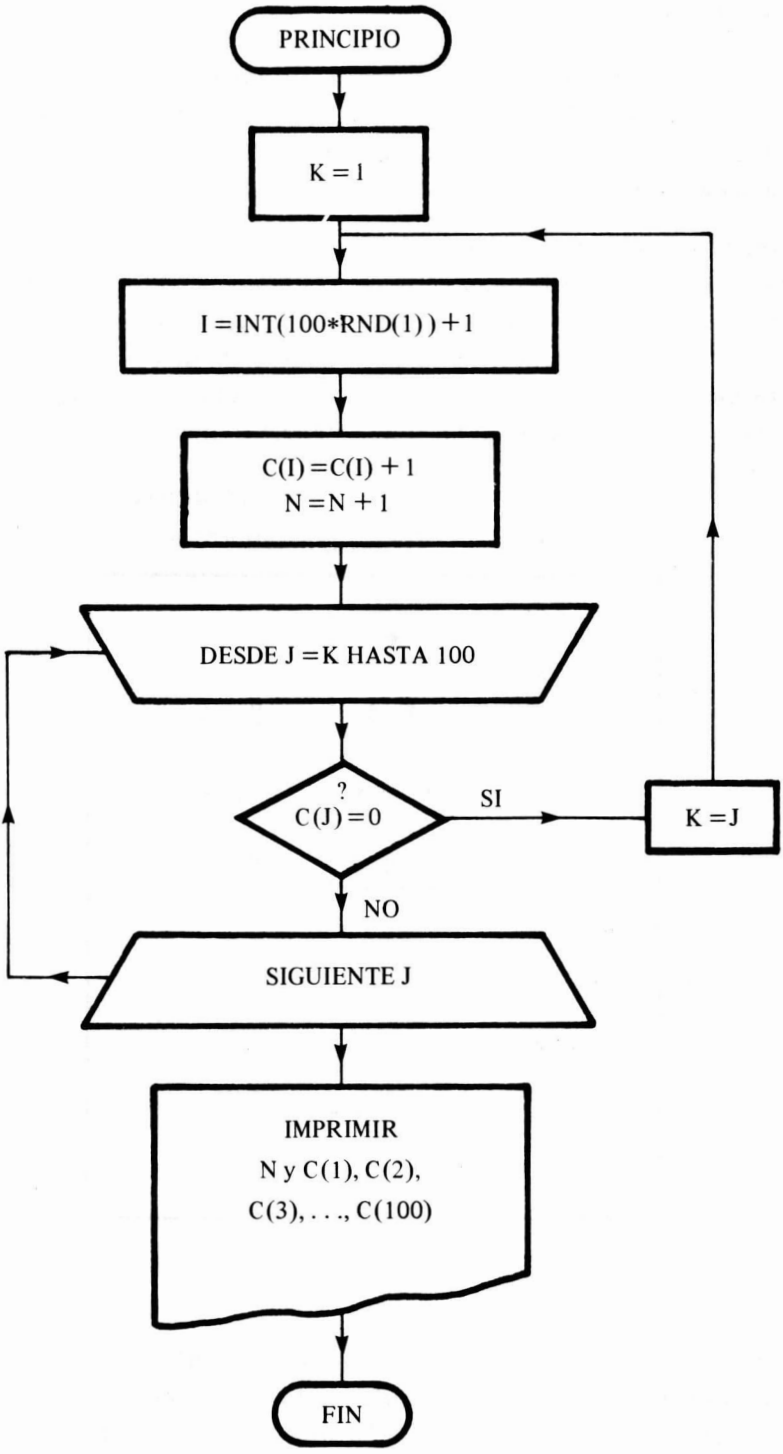

PRINCIPIO
K = 1
I = INT(100*RND(1)) + 1
C(I) = C(I) + 1
N = N + 1
DESDE J = K HASTA 100
?
C(J) = 0
SI
K = J
NO
SIGUIENTE J
IMPRIMIR
N y C(1), C(2),
C(3), . . ., C(100)
FIN

Fig. 7.3

7.21) EL DESTINO DE UNA URNA

Se parte de una urna cuya composición inicial es de una bola blanca y otra negra. Se extrae una bola al azar y se la devuelve a la urna acompañada de otra del mismo color. Se repite el proceso hasta que en la urna haya cien bolas.

Podemos considerar que la urna es un sistema que cambia de estado a lo largo de las distintas etapas. Desde la etapa inicial (etapa 2, porque en la urna hay dos bolas) hasta la etapa final (etapa 100) la urna atraviesa distintos estados, entendiendo por estados cada una de las composiciones posibles de la urna; así, por ejemplo, en la etapa 3 hay dos estados posibles: una blanca y dos negras o dos blancas y una negra.

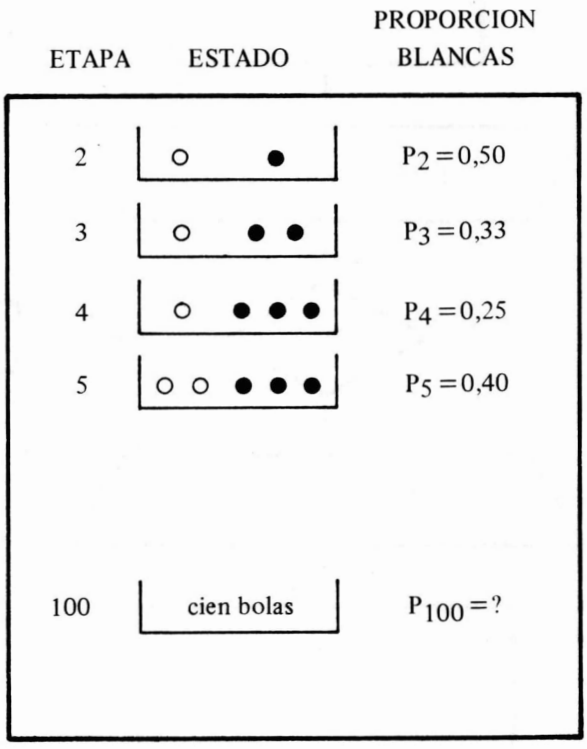


Fig. 7.4

En la etapa 100 hay 99 estados posibles. ¿A cuál de estos 99 estados llegará la urna? Imposible predecirlo: todos ellos son igualmente probables; es decir, la probabilidad de que la urna llegue a un estado final concreto es 1/99.

Aunque al principio no podamos predecir qué estado final alcanzará la urna, sí que lo podemos hacer (con cierta seguridad) cuando hayan transcurrido algunas etapas, porque se puede comprobar, mediante simulación, que *la proporción de bolas blancas se estabiliza* a lo largo del proceso.

Diseña un programa de modo que en cada etapa se imprima en pantalla la proporción de bolas blancas. Comprobarás el hecho notable de que dicha proporción se estabiliza a partir de una cierta etapa.

7.22) JUEGO DEL SUBMARINO 1

En un casillero de 6 x 6 el ordenador sitúa al azar un submarino, fuera del alcance de nuestra vista y del sónar. Vamos lanzando cargas de profundidad en diversas casillas. Si una carga da al submarino, el ordenador contesta "HUNDIDO". Si da en una casilla contigua a la del submarino, el ordenador contestará "SUBMARINO CERCA". Si la carga da en otro lugar la contestación es "AGUA".

Al hundir el submarino el ordenador ha de sacar en pantalla el número de cargas de profundidad utilizadas; así puede haber competición entre varios jugadores, ganando quien consiga el hundimiento con el menor número de cargas. (No es necesaria la representación gráfica del juego).

7.23) JUEGO DEL SUBMARINO 2

Variante del juego anterior en un casillero de 10 x 10. En cada carga lanzada la máquina contesta la distancia (en casillas) al submarino, y si la distancia es 0 contesta "HUNDIDO". Pero en esta variante el submarino se mueve: cada vez que lanzamos una carga se desplaza aleatoriamente una casilla. (Si va hacia los bordes rebotará en ellos). Igual que antes se trata de hundir el submarino con el menor número posible de cargas.

7.24) RESOLVER SISTEMAS DE ECUACIONES LINEALES POR EL METODO DE GAUSS (o de triangulación)

El programa debe resolver sistemas de N ecuaciones con N incógnitas. Para simplificar la explicación veamos cómo se opera si se trata de un sistema de 3 ecuaciones con 3 incógnitas:

$$A(1,1)\cdot X1 + A(1,2)\cdot X2 + A(1,3)\cdot X3 = A(1,4)$$
$$A(2,1)\cdot X1 + A(2,2)\cdot X2 + A(2,3)\cdot X3 = A(2,4)$$
$$A(3,1)\cdot X1 + A(3,2)\cdot X2 + A(3,3)\cdot X3 = A(3,4)$$

Si A(1,1) es distinto de cero, a la segunda ecuación se le resta la primera multiplicada por A(2,1) y dividida por A(1,1). De igual forma a la tercera ecuación se le resta la primera multiplicada por A(3,1) y dividida por A(1,1). De esta manera queda un sistema de la forma:

$$\begin{aligned} A(1,1)\cdot XI + A(1,2)\cdot X2 + A(1,3)\cdot X3 &= A(1,4) \\ A(2,2)\cdot X2 + A(2,3)\cdot X3 &= A(2,4) \\ A(3,2)\cdot X2 + A(3,3)\cdot X3 &= A(3,4) \end{aligned}$$

donde los coeficientes tienen valores no necesariamente iguales a los iniciales.

Si A(1,1) fuera igual a cero, al no poder dividir por cero tomaríamos en vez de la primera ecuación la segunda; y si A(2,1) también fuera cero tendríamos que tomar en vez de la primera ecuación la tercera. Eso sí, uno de los tres ha de ser distinto de cero pues de lo contrario se trataría en realidad de un sistema de tres ecuaciones con dos incógnitas.

El paso siguiente consiste en restar a la tercera ecuación la segunda multiplicada por A(3,2) y dividida por A(2,2), con lo que queda un sistema de la forma:

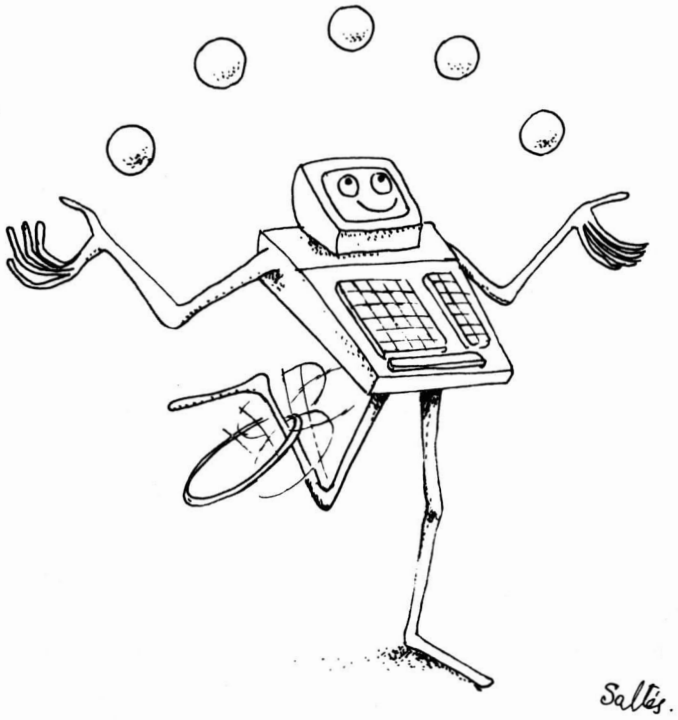
$$\begin{aligned} A(1,1)\cdot X1 + A(1,2)\cdot X2 + A(1,3)\cdot X3 &= A(1,4) \\ A(2,2)\cdot X2 + A(2,3)\cdot X3 &= A(2,4) \\ A(3,3)\cdot X3 &= A(3,4) \end{aligned}$$

en el que el cálculo de X3 es inmediato. Conociendo X3, se obtiene enseguida X2 de la segunda ecuación, y de ambos X1 en la primera.

Para el segundo paso queda salvar la posibilidad de que A(2,2) sea cero. Puedes tratarlo como en el caso de que A(1,1) fuera cero. La diferencia es que aquí pueden ser nulos a la vez A(2,2) y A(3,2). Entonces es seguro que el sistema no es compatible determinado, por lo que podrías sacar en pantalla "ESTE SISTEMA NO TIENE SOLUCION UNICA". Si te ves con fuerzas distingue el caso de que el sistema sea indeterminado o de que sea incompatible.

8

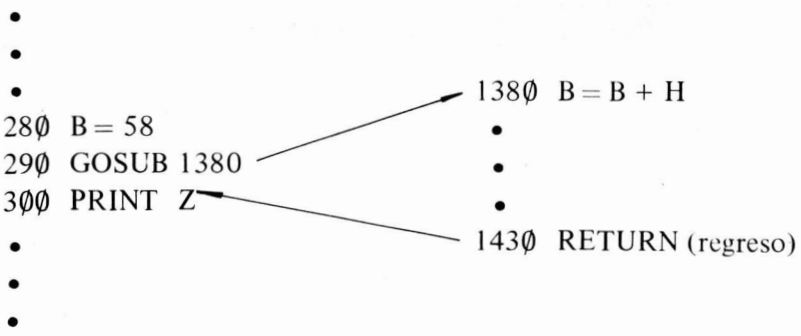
SUBRUTINAS



8.1 INTRODUCCION

A veces ocurre en un programa que las líneas que van de 36Ø a 5ØØ efectúan el mismo trabajo que las que van de 61Ø a 75Ø y que las comprendidas entre 124Ø y 138Ø. Ahorraríamos mucho trabajo si pudiésemos programar una sola vez todos aquellos trabajos idénticos que se realizan varias veces. A estas partes "portátiles" de un programa, que figuran en él una sola vez, pero que se pueden ejecutar tantas veces como haga falta, y en los lugares del programa que las necesitemos, las llamamos *subrutinas*. Si tenemos una subrutina escrita en

las líneas que van de la 138Ø a la 143Ø de un cierto programa, y necesitamos usarla en la 29Ø, la situación será la siguiente:



Al llegar a la línea 28Ø, la variable B toma el valor 58; en 29Ø, la orden GOSUB 138Ø es un direccionamiento incondicional a la línea 138Ø, que además recuerda al ordenador que cuando tropiece con una orden RETURN (regreso), habrá de continuar la ejecución del programa en la línea siguiente a la 29Ø, la 3ØØ en este ejemplo.

Vamos a ver un ejemplo sencillo que ilustra el funcionamiento de un programa dotado de dos subrutinas, una de suma y otra de producto; la primera nos proporciona los resultados en un texto "vertical" y la segunda en uno "horizontal".

```
1ØØ   LET A=4
11Ø   LET B=5
12Ø   GOSUB 1ØØØ
2ØØ   GOSUB 2ØØØ
3ØØ   LET A=3
31Ø   LET B=1Ø
32Ø   GOSUB 2ØØØ
4ØØ   LET A=2
41Ø   LET B=1
42Ø   GOSUB 1ØØØ
43Ø   GOSUB 2ØØØ
5ØØ   PRINT "ESTO ES TODO"
51Ø   END
```

```
1ØØØ  PRINT "SUBRUTINA PRIMERA"
1Ø1Ø  LET S1=S1 + 1
1Ø2Ø  PRINT "UTILIZACION";S1
1Ø3Ø  PRINT "A=";A
1Ø4Ø  PRINT "B=";B
1Ø5Ø  PRINT "A + B =";A + B
1Ø9Ø  PRINT "FIN DE SUBR. PRIMERA"
11ØØ  RETURN
2ØØØ  PRINT "SUBRUTINA SEGUNDA";
2Ø1Ø  LET S2=S2+1
2Ø2Ø  PRINT "UTILIZACION";S2
2Ø3Ø  PRINT "A=";A;"   Y B=";B,
2Ø4Ø  PRINT "LUEGO A*B =" ;A*B
2Ø8Ø  PRINT "FIN DE  SUB.SEGUNDA"
2Ø9Ø  PRINT
2Ø95  PRINT
21ØØ  RETURN
```

Como diagrama de flujo asociado a este programa tenemos el siguiente:

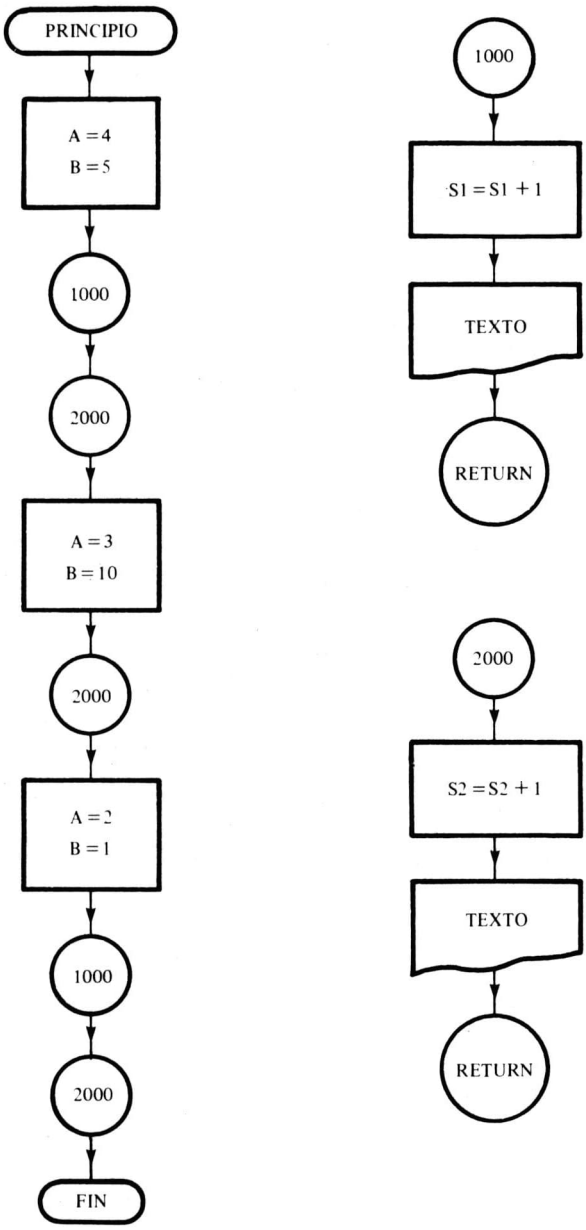


Fig. 8.1

Es posible que una subrutina tenga, a su vez, subrutinas, de forma semejante a los bucles, que podían estar unos dentro de otros, anidados.

8.2 PROGRAMACION DESCENDENTE

Si necesitamos conocer con exactitud y precisión cómo es una zona del mundo, enviar una partida de exploradores no es precisamente la mejor forma de lograrlo: conseguiremos tener descripciones pintorescas y detalladas de algunos lugares, pero no conoceremos con exactitud las distancias y posiciones relativas de unos respecto a otros; sabremos cómo se recortan contra el cielo los picos de algunas montañas, pero careceremos de un conocimiento preciso de la cordillera que forman, y no tendremos la certeza de que no haya algún pequeño valle escondido donde paste el unicornio. La forma más apropiada de proceder hoy sería la siguiente:

1º Delimitar en una fotografía hecha desde un satélite artificial la zona que se pretende explorar.

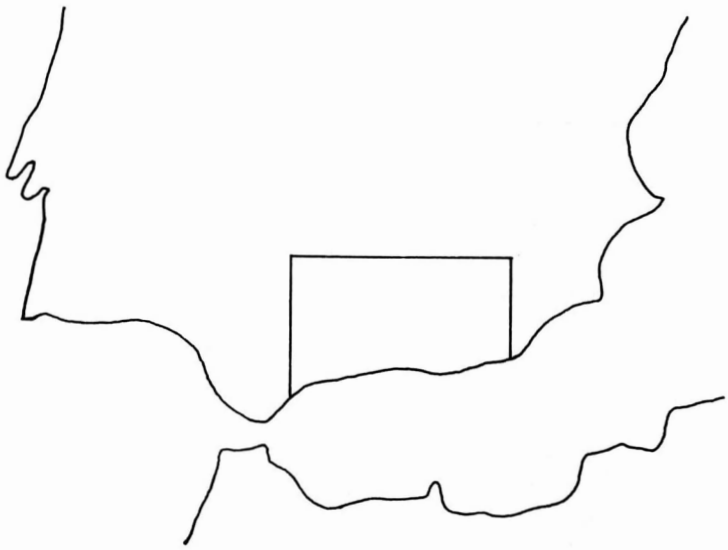


Fig. 8.2

2º Con aviones volando a gran altura (25.000 m) podemos hacer pasadas sobre la zona acotada en 1º, de tal forma que, si el ángulo que cubre la cámara fotográfica que transporta el avión es de 90°, la fotografía que toma es de un cuadrado de 50 km de lado;

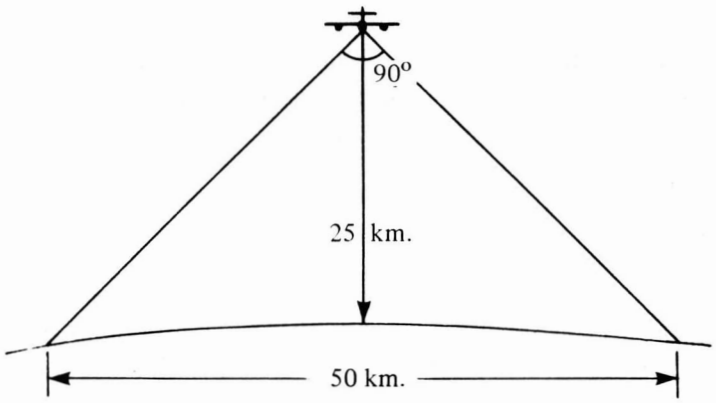


Fig. 8.3

así podemos dividir la zona inicial en un cierto número de zonas menores –en nuestro ejemplo serán 24– como aparece en la siguiente figura.

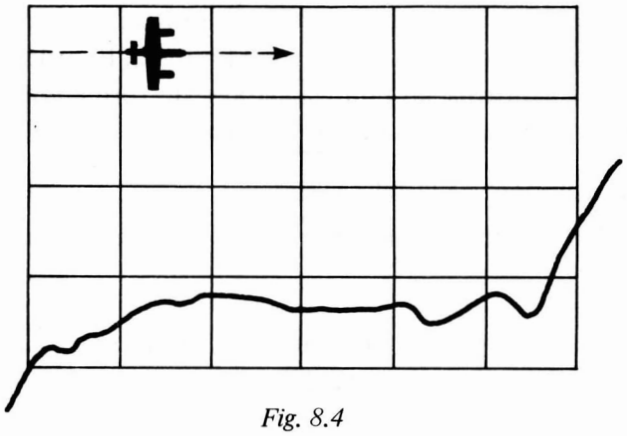


Fig. 8.4

3º Repetimos el proceso anterior utilizando, ahora, aviones que vuelen a media altura (5.000 m) y lleven el mismo tipo de cámara; así obtendremos fotos de cuadrados de 10 kms de lado.

4º Si luego efectuamos vuelos a "sólo" mil metros de altura, obtenemos, de cada uno de los cuadrados anteriores, otros veinticinco de dos km de lado. Estas fotos ampliadas hasta 1 m de lado –cosa nada exagerada para lo que es la cartografía aérea– nos permiten contar a la gente que toma el sol en las playas (una persona de 2 m de altura aparecerá allí como una rayita de 1 mm de longitud, una carretera será una cinta oscura de no menos de 2 mm de ancho, etcétera). Este es el momento en que se puede comenzar a inspeccionar el terreno a pie; pero no empezaremos enviando un equipo de químicos para analizar

la tierra del primer tiesto que encuentren, y cristalógrafos para que estudien la forma de los granos de esa tierra; el análisis sobre el terreno se puede llevar a esos extremos, pero siguiendo un proceso análogo al que nos ha guiado hasta este momento: *de arriba a abajo;* así, si está el unicornio, le veremos.

Si aplicamos este método de exploración a la elaboración de algoritmos para la resolución automática de problemas, estaremos haciendo lo que se llama *programación descendente.*

Las subrutinas, que nacieron de la necesidad de no repetir innecesariamente un trabajo ya hecho, resultan muy apropiadas para facilitar la programación descendente.

8.3 HABLANDO CON EL ORDENADOR

Vamos a aplicar estos principios enunciados a un caso concreto. Deseamos programar en el ordenador una conversación limitada. El ordenador nos hace algunas preguntas y controla las contestaciones, de forma que no admitirá ninguna que considere absurda. Las preguntas que nos va a hacer son:

1ª ¿En qué número de la calle vives?

No aceptará valores inferiores a 1 ni superiores a 400.

2ª ¿En qué piso?

El bajo será 0 y el sótano −1; no aceptará valores menores, ni tampoco mayores que 20.

3ª ¿Tienes hermanos?

Sólo aceptará sí o no.
Si la contestación a la pregunta anterior ha sido afirmativa, preguntará

4ª ¿Cuántos?

La contestación habrá de estar comprendida entre 1 y 22.
No se admitirán contestaciones numérica no enteras.

El planteamiento que hacemos de este problema es el siguiente:

Principio
1. Hacer la primera pregunta

2. Hacer la segunda pregunta
3. Hacer la tercera pregunta
4. Si se contesta afirmativamente la tercera, entonces
   Hacer la cuarta pregunta
   Fin de Si (aquí termina el módulo 4)
   Fin

El diagrama de flujo asociado es:

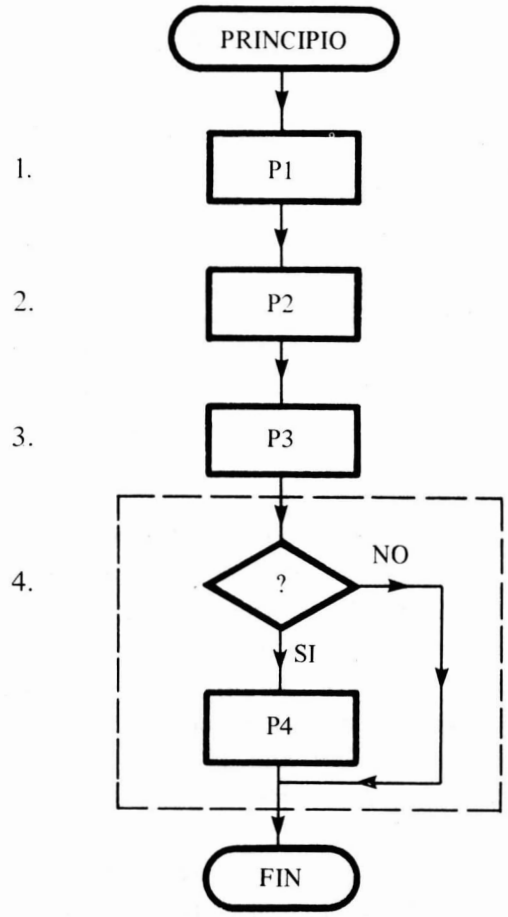


Fig. 8.5

Si refinamos un poco el planteamiento anterior, tenemos:

Inicio
1. Hacer la primera pregunta
   hasta que no haya error en la contestación
2. Hacer la segunda pregunta
   hasta que no haya error en la contestación
3. Hacer la tercera pregunta
   hasta que no haya error en la contestación
4. Si la contestación a la tercera es afirmativa
   entonces
        Hacer la cuarta pregunta
        hasta que no haya error en la contestación
   Fin de Si
   Fin

Podemos traducir esto, de manera inmediata, al lenguaje de los esquemas gráficos, según figura en el organigrama de la derecha, Fig. 8.6.

A la vista de este diagrama, resulta evidente que el programa consiste en preguntar lo mismo un cierto número de veces; bastará tener cuidado para saber qué admitir como contestación correcta o incorrecta cada vez. De todas formas, se ha de excluir de este proceso de homogeneización la pregunta tercera, que tiene contestación no numérica.

La variable E es un indicador de error; sólo tomará dos valores: 0, cuando no está activada, y 1, cuando se ha cometido un error (a este tipo de indicadores se les suele llamar banderas). La utilizaremos para saber si se ha de repetir, o no, la pregunta; sirve para guiar al ordenador durante la ejecución del programa. Tras saber que se ha de repetir una pregunta –porque E vale 1, hay respuesta errónea– es necesario desactivar el indicador, pues en caso contrario, aunque la siguiente respuesta sea correcta, E seguiría valiendo 1; así pues, antes de repetir una pregunta, hacemos E = 0.

En cada una de las preguntas de contestación numérica escribiremos, en primer lugar el texto de la pregunta en la pantalla y, luego, al introducirse un valor como contestación, veremos si es entero, mayor que el valor mínimo admitido, A, y menor que el máximo admitido B. Un diagrama de flujo que simboliza estas actuaciones se puede ver en la Fig. 8.7.

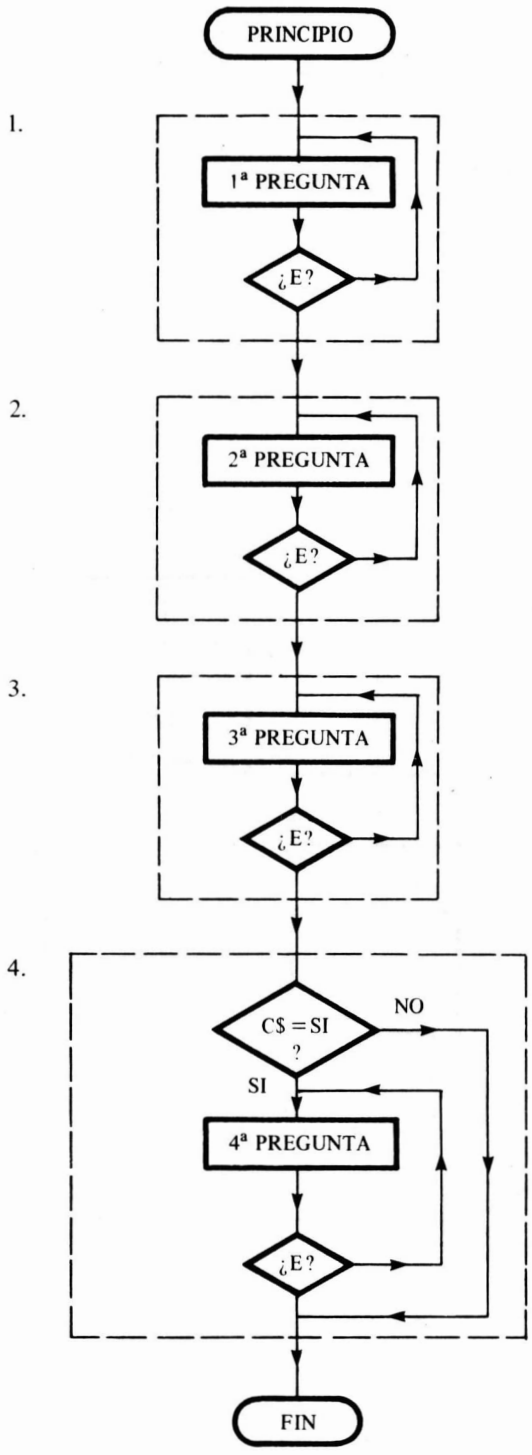

PRINCIPIO
1.
1ª PREGUNTA
¿E?
2.
2ª PREGUNTA
¿E?
3.
3ª PREGUNTA
¿E?
4.
C$ = SI
?
NO
SI
4ª PREGUNTA
¿E?
FIN

Fig. 8.6

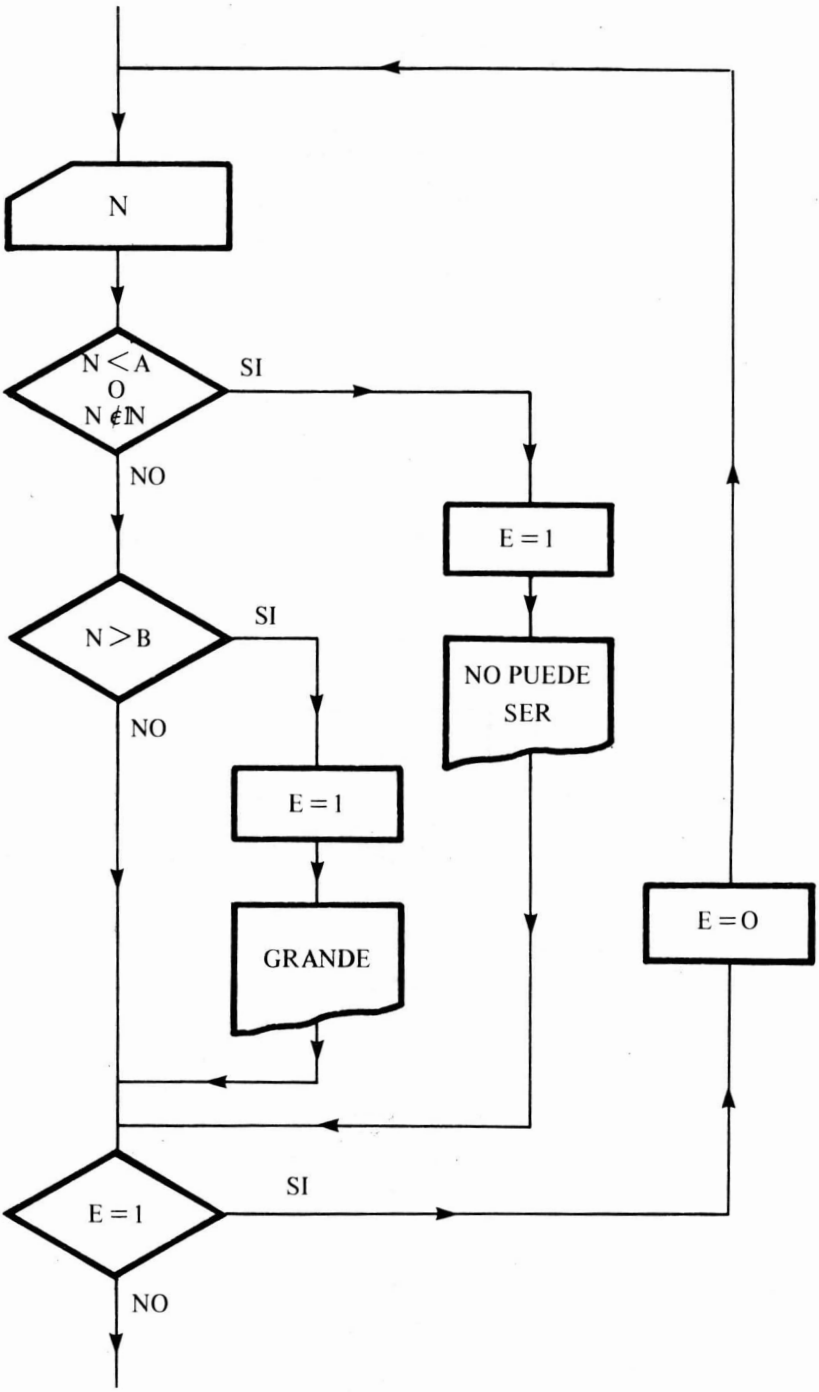


Fig. 8.7

El proceso total se simbolizaría, entonces, así:

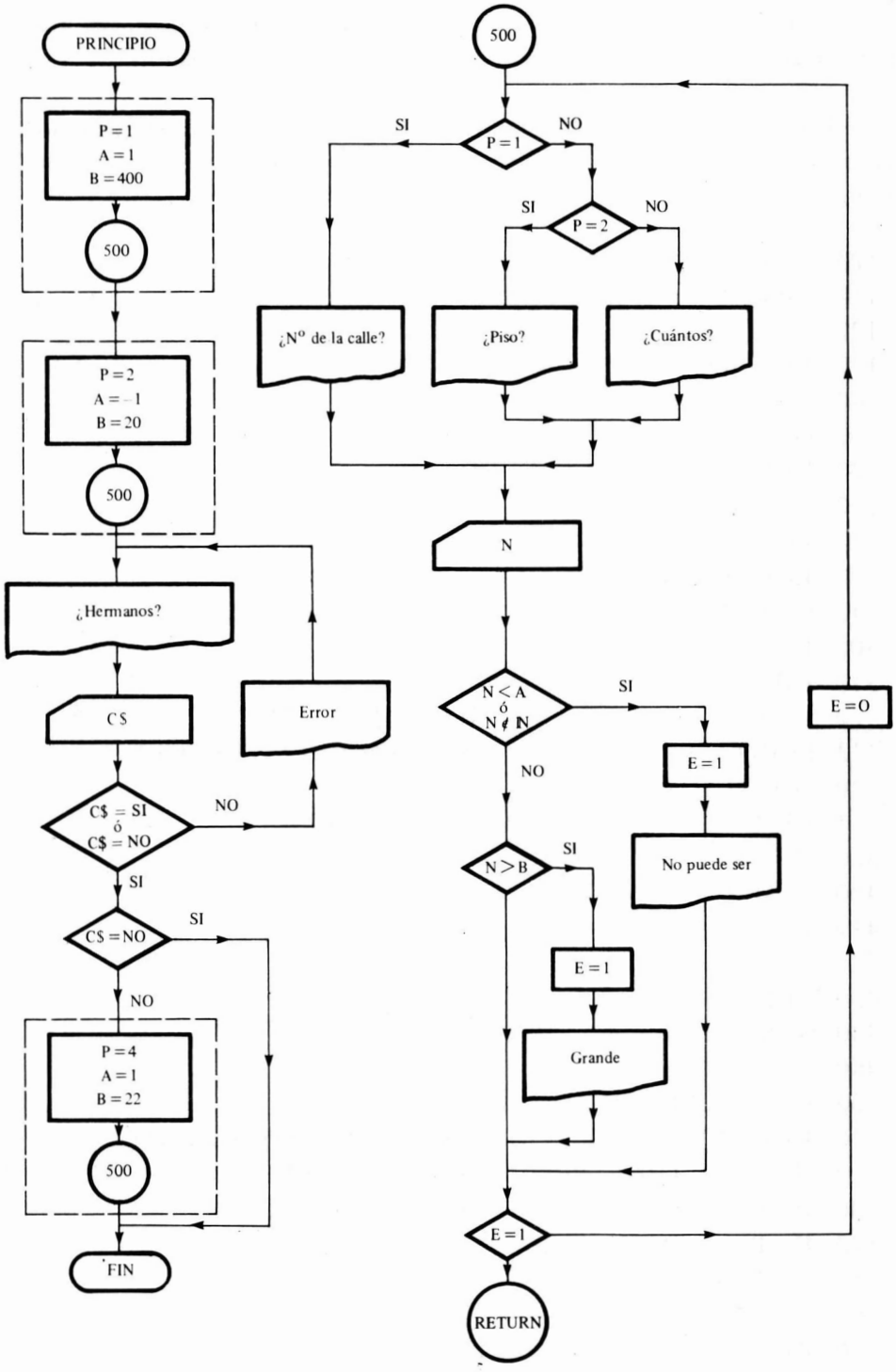

PRINCIPIO
1ª
P = 1
A = 1
B = 400
500
2ª
P = 2
A = 1
B = 20
500
¿Hermanos?
C$
Error
C$ = SI ó C$ = NO
NO
SI
C$ = NO
SI
NO
4ª
P = 4
A = 1
B = 22
500
FIN
500
SI
P = 1
NO
SI
P = 2
NO
¿Nº de la calle?
¿Piso?
¿Cuántos?
N
N < A ó N ≠ N
SI
NO
E = 1
N > B
SI
No puede ser
E = 1
Grande
E = 1
RETURN
E = 0

Fig. 8.8

Como vemos, la subrutina es un poco más complicada que cada una de las preguntas, para poder, así, poner en pantalla el texto apropiado.

Este programa nos repetiría un cierto número de veces la misma tarea, con la única excepción de la tercera pregunta. Un programa que se obtiene de forma inmediata a partir de este diagrama de flujo es el siguiente:

```
1ØØ REM CONVERSACION
11Ø REM PREGUNTA PRIMERA
12Ø LET P=1
13Ø LET A=1
14Ø LET B=4ØØ
15Ø GOSUB 5ØØ
2ØØ REM PREGUNTA SEGUNDA
21Ø LET P=2
22Ø LET A=1
23Ø LET B=2Ø
24Ø GOSUB 5ØØ
3ØØ REM PREGUNTA TERCERA
31Ø INPUT "TIENES HERMANOS";C$
32Ø IF C$="SI" OR C$="NO" THEN 35Ø
33Ø PRINT "ERROR. ESA CONTESTACION NO ES CORRECTA"
34Ø GO TO 31Ø
35Ø REM SI NO TIENE HERMANOS SE ACABA
36Ø IF C$="NO" THEN 48Ø
4ØØ REM CUARTA PREGUNTA
41Ø LET P=4
42Ø LET A=1
43Ø LET B=22
44Ø GOSUB 5ØØ
48Ø GO TO 76Ø
5ØØ REM SUBRUTINA
51Ø REM TEXTO DE LA PREGUNTA CORRESPONDIENTE
52Ø IF P=1 THEN PRINT "EN QUE NUMERO DE LA CALLE VIVES?"
53Ø IF P=2 THEN PRINT "EN QUE PISO VIVES?"
54Ø IF P=4 THEN PRINT "CUANTOS?"
55Ø REM CONTESTACION
56Ø INPUT N
6ØØ REM RESPUESTA CORRECTA?
61Ø IF N<A OR N<>INT(N) THEN 65Ø
```

```
62Ø  IF  N >B THEN  68Ø
63Ø  REM  NO HAY ERROR
64Ø  GO TO  7ØØ
65Ø  PRINT  "NO PUEDE SER ESE NUMERO"
66Ø  LET  E = 1
67Ø  GO TO  7ØØ
68Ø  PRINT  "ESE NUMERO ES DEMASIADO GRANDE"
69Ø  LET  E=1
7ØØ  REM  SI NO HAY ERROR SE ACABA
71Ø  IF  E=Ø  THEN  75Ø
72Ø  LET  E=Ø
73Ø  GO TO  51Ø
75Ø  RETURN
76Ø  END
```

8.4 LA OPCION MULTIPLE ON–GOTO; ON–GOSUB

Cuando la contestación a una pregunta presenta tan solo dos alternativas, por ejemplo Sí o No, una contestación o todas las demás, un número o todos los demás, la programación es cómoda, pues tenemos una instrucción apropiada (IF–THEN); pero si la opción es múltiple, por ej. P=1 ó P=2 ó P=3 ó P=4, no tenemos aún una instrucción adecuada para decidirnos por uno cualquiera de estos valores y hemos de ir optando entre uno cualquiera y los restantes, hasta que conseguimos aislar el valor elegido.

Para remediar esta situación contamos con la instrucción ON–GOTO que se comporta así:

```
•
•
•
13Ø  ON  I  GOTO  4ØØ, 5ØØ, 6ØØ, 132Ø, 15ØØ
14Ø  LET  R = R + 3
•
•
•
```

Si I vale 1, entonces se transfiere el control a 4ØØ
Si I vale 2, entonces se transfiere el control a 5ØØ

Si I vale 3, entonces se transfiere el control a 6ØØ
Si I vale 4, entonces se transfiere el control a 132Ø
Si I vale 5, entonces se transfiere el control a 15ØØ
Si I es 6, más que 6, o si I es menor que 1, entonces el control se transfiere a 14Ø.

Ilustraremos esto con un ejemplo:

Al final de una carrera los 8 corredores han de pulsar una tecla numérica, indicadora del lugar en que llegaron a la meta. El ordenador tiene introducido un programa que felicita a los tres primeros, en forma diferente según el puesto ocupado, y a los demás les recuerda que no han alcanzado el premio; además controla las trampas.

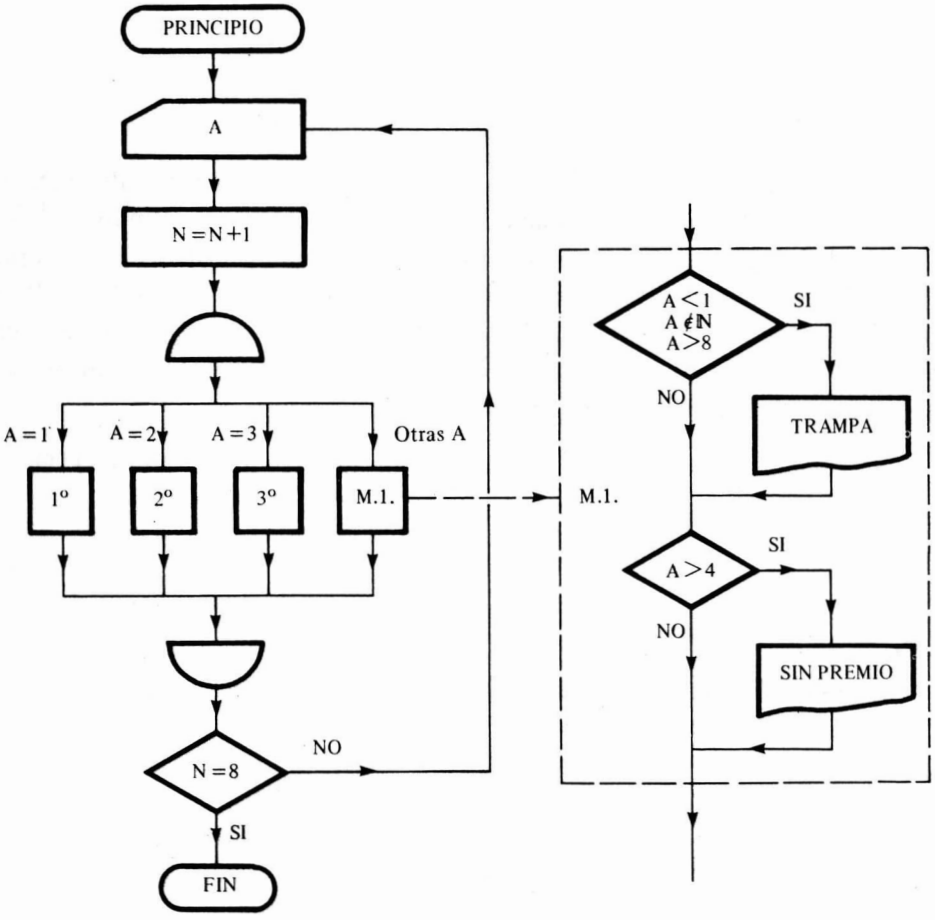


Fig. 8.9

Para simplificar, en este programa no se comprueba si varios dicen ocupar la misma posición.

```
100 REM CARRERA
110 PRINT Borrado de pantalla
120 INPUT "EN QUE LUGAR HA LLEGADO"; A
130 LET N=N+1
140 ON A GO TO 1000, 2000, 3000
150 REM SI A NO ES UNO, DOS NI TRES, LA EJECUCION SIGUE
    POR AQUI
160 REM PUEDE HABER LLEGADO A ESTA POSICION?
170 IF A<1 OR INT(A)<>A OR A>8 THEN PRINT "ERES UN
    TRAMPOSO. ESA POSICION NO EXISTE"
180 GO TO 4000
190 IF A>=4 THEN PRINT "LO SIENTO. NO HAY PREMIO"
200 GO TO 4000
1000 REM SI A ES UNO SE VIENE DE 140 A ESTE PASO
1010 PRINT "ENHORABUENA CAMPEON"
1020 GO TO 4000
2000 REM SI A ES DOS SE VIENE DE 140 A ESTE PASO
2010 PRINT "MUY BIEN, HA FALTADO MUY POCO"
2020 GO TO 4000
3000 REM SI A ES TRES SE VIENE DE 140 A ESTE PASO
3010 PRINT "HA CORRIDO MUY BIEN"
4000 PRINT
4010 PRINT
4020 PRINT
4030 IF N=8 THEN 5000
4040 REM A UN FALTAN
4050 PRINT "EL SIGUIENTE";
4060 GO TO 120
5000 END
```

Veamos otro ejemplo. Utilizaremos de nuevo esta instrucción para resolver la ecuación de segundo grado. Seguiremos el diagrama:

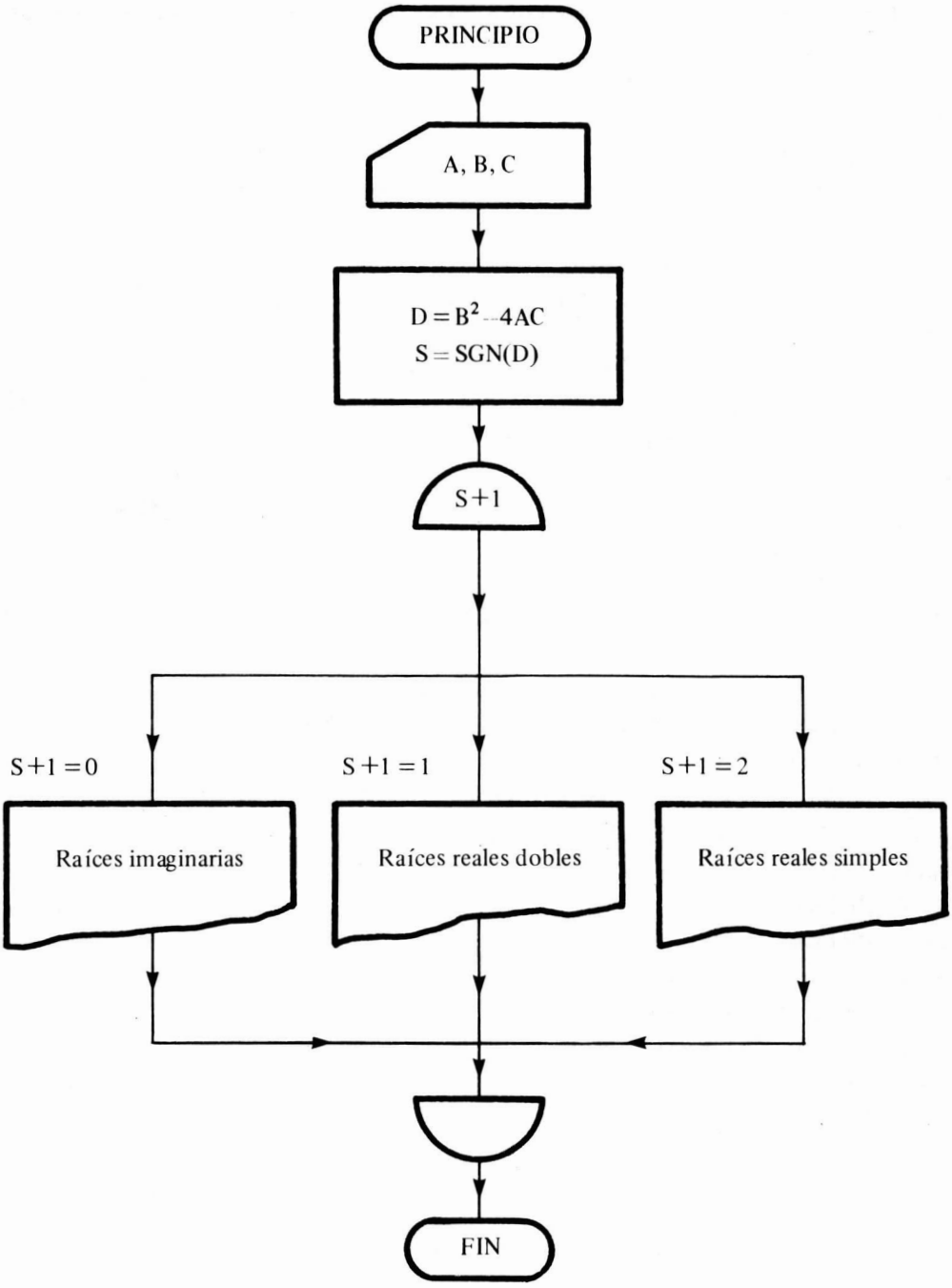

PRINCIPIO
A, B, C
D = B² - 4AC
S = SGN(D)
S+1
S+1 = 0
S+1 = 1
S+1 = 2
Raíces imaginarias
Raíces reales dobles
Raíces reales simples
FIN

Fig. 8.10

El programa sería:

```
1ØØ  REM  RESOL. EC. SEGUNDO GRADO
11Ø  REM  ENTRADA DE COEFICIENTES
14Ø  PRINT  "INTRODUZCA LOS COEFICIENTES
     DE AX↑2 + BX + C = Ø"
15Ø  INPUT  "A = ";A
16Ø  INPUT  "B = ";B
17Ø  INPUT  "C = ";C
2ØØ  REM  DISCRIMINANTE
23Ø  LET  D = B↑2 - 4*A*C
25Ø  REM  PREPARACION PARA BIFURC. MULTIPLE
28Ø  LET  S = SGN(D) + 1
3ØØ  ON  S  GO TO  5ØØ, 7ØØ
32Ø  REM  SI D < Ø ENTONCES S + 1 = Ø Y LA EJECUCION SIGUE
     AQUI
35Ø  PRINT  "LAS RAICES SON IMAGINARIAS CONJUGADAS"
36Ø  PRINT
37Ø  PRINT  "XI = ";-B/(2*A);"+";ABS(SQR(ABS(D)  )/(2*A)  );"I"
38Ø  PRINT  "X2= ";-B/(2*A);"-";ABS(SQR(ABS(D)  )/(2*A)  ):"I"
39Ø  GO TO  8ØØ
5ØØ  REM  SI D = Ø  ENTONCES S+1 = 1 Y LA EJECUCION
     VIENE AQUI
54Ø  PRINT  "RAIZ REAL DOBLE"
55Ø  PRINT
56Ø  PRINT  "X = ";-B/(2*A)
58Ø  GO TO  8ØØ
7ØØ  REM  SI D > Ø ENTONCES S+1 = 2 Y LA EJECUCION
     VIENE AQUI
73Ø  PRINT "DOS RAICES REALES DISTINTAS"
74Ø  PRINT
75Ø  PRINT "X1 = ";(-B+SQR(D)  )/2*A)
76Ø  PRINT "X2 = ";(-B-SQR(D)  )/(2*A)
8ØØ  END
```

En lugar de utilizar el direccionamiento múltiple, se podría hacer con subrutinas, en una forma análoga:

```
9Ø   .....
1ØØ  ON  K  GOSUB  1ØØØ, 12ØØ, 31ØØ
12Ø  .....
```

si K cumple $1 \leqslant K < 2$, se ejecuta la subrutina 1000 y se regresa a la línea 120; si $2 \leqslant K < 3$, se ejecuta la 12000 y se regresa a 120; si $3 \leqslant K < 4$, es la 31000; si $1 > K$ ó $4 \leqslant K$, entonces se pasa directamente a la 120.

RECUERDA	
Subrutina	GOSUB
Programación descendente	RETURN
Opción múltiple en el salto incondicional	ON–GOTO
Opción múltiple en las subrutinas	ON–GOSUB

EJERCICIOS

8.1) Llamamos por teléfono varias veces a un número equivocado. Preparar un programa con contestaciones cada vez más secas.

8.2) El abuelito "gagá". Simular con el ordenador un abuelito que cuando le preguntan por sus nietos siempre quiere saber por cuál. Si es por la niña, dice que "es un angel"; si le dicen que Carlos, contesta que "no estudia nada" y de Pepe que "es un bestia, no sabe más que darme balonazos". (Utiliza la instrucción ON–GOTO).

8.3) Haz un programa semejante al del ejercicio 7.2 utilizando la instrucción ON–GOTO.

8.4) Haz lo mismo con ON–GOSUB.

8.5) Examina el signo del número contenido en una variable X. Si es negativo, que el programa escriba NEGATIVO; si es positivo deberá salir en pantalla POSITIVO, y si es cero, CERO (Utiliza la función SGN y la instrucción ON–GOTO).

8.6) Haz un programa análogo al del cambio de divisas del apartado 4.2, en su primera versión, pero ahora ofreciendo la venta de más divisas, 6 u 8, por ejemplo. (Emplea la instrucción ON–GOTO).

8.7) Haz lo mismo con ON–GOSUB.

8.8) Utiliza una subrutina que analice si un número es primo o no para resolver el problema OCTOBER. Se tiene la siguiente multiplicación:

```
      X X X X X X
            X X
  ---------------
      X X X X X X
    X X X X X X
  ---------------
    O C T O B E R
```

donde tanto las X como las letras son números (ninguna letra vale cero). Los datos son los siguientes:

1º) son primos cada uno de los siguientes números

```
O C T O          T O B E R
  C T O            O B E R
  C T O B            B E R
    T O              B E
    T O B E            E R
                         R
```

2º) el número OCTOBER tiene un divisor primo menor que 10.

8.9) ¿Puede una subrutina contener más de una instrucción RETURN?

8.10) Explicar el error del siguiente programa:

```
3Ø  GOSUB  1ØØ
•
•
•
1ØØ  REM  SUBRUTINA A
•
•
•
12Ø  GOSUB  2ØØ
•
•
•
16Ø  RETURN
•
•
•
2ØØ  REM  SUBRUTINA  B
•
•
•
225  GOSUB  1ØØ
•
•
•
245  RETURN
25Ø  END
```

8.11) ¿Por qué es incorrecta la siguiente instrucción?

```
5Ø  ON  N$  GO TO  6Ø, 7Ø, 9Ø
```

8.12) Prepara un programa que realice la suma, resta, multiplicación o división de dos polinomios, a nuestra elección (Utiliza el principio de la programación descendente). Los grados de los polinomios no pueden ser mayores de 10.

8.13) Programar un controlador de vuelo automático que atienda las llamadas de hasta cinco aviones que pretendan aterrizar en el aeropuerto (nosotros seremos los pilotos de los aviones). Los criterios de preferencia serán los siguientes:

1) Avería grave a bordo.

2) Combustible escaso.

3) Avería leve a bordo.

Si el combustible es muy escaso puede ser prioritario frente a la avería grave (tú has de pormenorizar estos criterios). El tiempo necesario para el aterrizaje de un avión, desde que recibe la orden será de 4 minutos. Entre dos aterrizajes ha de mediar, cuando menos, un minuto.

8.14) Utilizando programación descendente, repite el ejercicio 7.19.

8.15) Empleando subrutinas, programa el ejercicio 7.20, considerando cuatro coleccionistas que intercambian los cromos repetidos. Los cromos se compran en sobres de tres cromos cada uno.

8.16) Prepara un programa que nos permita jugar a la brisca contra el ordenador. No puede quedar ninguna regla sobreentendida. Cuida la presentación. La máquina reparte las cartas y es uno de los dos jugadores.

8.17) Análogo al anterior, pero jugando al tute (para dos jugadores).

8.18) Haz los programas anteriores para cuatro jugadores.

8.19) Los sistemas situados a menos de un parsec (3.26 años luz) del nuestro, que tienen mayor probabilidad de poseer algún planeta habitable son:

Estrella	Distancia (a.l.)	Probabilidad
Alfa Centauro	4,3	0,107
Epsilón Eridano	10,8	0,033
Tau Ballena	12,2	0,036
70 Ofiuco	17,3	0,057
Eta Casiopea	18,0	0,057
Sigma Dragón	18,2	0,036
36 Ofiuco	18,2	0,042
HR 7703	18,6	0,020
Delta Pavo	19,2	0,057
82 Eridano	20,9	0,057
Beta Hidra Macho	21,3	0,037
HR 8832	21,4	0,011

Programa el ordenador de tu astronave de forma que permita al piloto tener en pantalla, a elección:

1º) La lista ordenada por distancia a nuestro sistema, con esta distancia expresada en:
   a) años luz
   b) parsecs

2º) La lista ordenada por probabilidad de poseer planetas habitables.

8.20) La velocidad que ha de alcanzar una astronave para poder escapar de la atracción de nuestro sistema planetario (velocidad de escape) es, aproximadamente, 43,6 km /seg. La velocidad que alcanza una nave al cabo de T segundos de estar sujeta a una aceleración A es VT = VI+A*T,siendo VI la velocidad inicial. Consideremos que una astronave que ha de transportar una tripulación humana y ha de estar acelerando un período considerable de tiempo, debe tomar como A el valor 10 m /seg ^{2}.

Según la teoría de la relatividad, el tiempo a bordo de la nave, TN, depende de la velocidad de ésta según la fórmula

$$TN = TT\sqrt{1 - V^2/c^2}$$

(TT es el tiempo-Tierra; c es la velocidad de la luz, $3 \cdot 10^5$ km /seg).

Completar el programa del ejercicio anterior para que el piloto pueda, tras elegir una estrella y una velocidad de crucero, conocer:

1º) El tiempo que se va a tardar en alcanzar la velocidad de crucero, tras lograr la velocidad de escape (medida en años, meses, días y horas), en
   a) tiempo de la nave
   b) tiempo de la tierra

2º) Idem. para completar el viaje.

8.21) Prepara un programa que traduzca un mensaje cualquiera empleando el código Morse:

Ø – – – – –	d – . .	q – – . –
1 . – – – –	e .	r . – .
2 . . – – –	f . . – .	s . . .

3 . . . – –	g – – .	t – .
4 . . . . –	h	u . . –
5	i . .	v . . . –
6 –	j . – – –	w . – –
7 – – . . .	l . – . .	x – . . –
8 – – – . .	m – –	y – . – –
9 – – – – .	n – .	z – – . .
a . –	ñ – – . – –	punto . – . – . –
b – . . .	o – – –	? . . – – . .
c – . – .	p . – – .	comienzo – . – . –
		fin . . . – . –

(Utiliza la instrucción ON–GOSUB y la función CHR$( )).

9

MANEJO DE DATOS: FICHEROS

9.1 INTRODUCCION

En los primeros capítulos hemos visto dos instrucciones para asignar a unas variables determinados valores, que ahora llamaremos datos; eran las instrucciones LET e INPUT. Hemos aprendido a manejar esos datos y a presentar unos resultados; los datos, una vez utilizados, se traducían en unos resultados en la pantalla, o en el papel de la impresora.

Pero en algunos programas se manejan inmensos Bancos de Datos. El movimiento de las cuentas corrientes en una sucursal bancaria, o la cantidad de in-

formación que supone el censo de una población cara a las elecciones, presentan un enfoque de la Informática bastante distinto del que hemos visto hasta aquí.

Estos programas gigantes no suelen hacerse en BASIC sino en otros lenguajes; muchas veces en Ensamblador o en código de máquina, e incluso se tiene un ordenador con un sólo programa: para llevar la contabilidad por ejemplo.

Sin embargo, si reduces tus pretensiones, ciñéndolas a los datos de tus compañeros de clase, por ejemplo, el BASIC puede resolver tus problemas. ¿Has pensado en las complicaciones que surgen?

La tarea de introducir los datos mediante instrucciones LET o INPUT es lenta, incómoda y a veces da lugar a equivocaciones. Esta información ha de grabarse de tal forma que no sea preciso volverla a teclear nunca más.

Sin embargo, pueden ser necesarias modificaciones en los datos. Nos interesa, entonces, que estas modificaciones se puedan efectuar con facilidad.

En otras ocasiones los resultados de un programa quedan en la pantalla o en la impresora, y para trabajar con ellos, para utilizarlos en un nuevo programa, habrá que teclearlos otra vez. Nos gustaría arbitrar un procedimiento mediante el cual los resultados queden grabados, guardados de alguna manera fácilmente accesible.

Queda suficientemente claro que con las instrucciones LET e INPUT no podemos ir muy lejos en este terreno. El BASIC ofrece dos posibilidades: archivo de los datos en líneas de programa y archivo de datos y de resultados en ficheros de memoria exterior: cassettes o diskettes.

9.2 LOS DATOS EN LINEAS DE PROGRAMA

Cuando en un programa vamos a trabajar con un volumen de datos, numéricos, alfanuméricos o de ambos tipos, de cierta entidad, las instrucciones LET e INPUT resultan lentas e incómodas, y se suele utilizar el par de sentencias DATA y READ, que trabajan siempre juntas, son correlativas.

Las instrucciones DATA son simples almacenes de datos separados por comas (,). Por ejemplo:

```
3Ø DATA 4,16.2,-11,1Ø.123
4Ø DATA OSCAR,CARLOS
5Ø DATA 4,16.2,OSCAR,-11,1Ø.123,CARLOS
```

Las instrucciones DATA se comportan de forma muy parecida a las definiciones de funciones: **DEF FN**:

- no son ejecutables. El programa pasa por ellas sin hacer nada.
- un programa puede llevar tantas instrucciones DATA como queramos y, por una cuestión de orden y claridad, es conveniente situarlas todas juntas, al principio o al final del programa.
- cuando el ordenador encuentre una instrucción **READ**, buscará el DATA correspondiente, en cualquier lugar del programa en que se encuentre, de modo parecido a lo que sucedía con FN y DEF FN.
- no importa que algunos datos se queden sin leer; pero si se intentan leer más datos de los disponibles se comete un error, que será denunciado por el mensaje correspondiente.

La instrucción READ, por ejemplo:

```
7Ø READ A,B
8Ø READ C,D,A$
9Ø READ B$,E,F,C$
```

manda al ordenador que lea (READ = Leer) *a partir del primer valor no leído todavía* que figure en una sentencia DATA. Si las líneas escritas como ejemplo formaran parte de un único programa real, se tendría:

```
A  = 4
B  = 16.2
C  = -11
D  = 1Ø.123
A$ = OSCAR
B$ = CARLOS
E  = 4
F  = 16.2
C$ = OSCAR
```

EJEMPLO 1. Calcularemos la longitud de tandas de circunferencias.

```
100 REM  CALCULO DE LA LONGITUD DE TANDAS DE CIRCUN-
     FERENCIAS
110 READ  R
120 LET  L=2*π*R
130 PRINT  "PARA RADIO";R;"LONGITUD";L
140 GO  TO  110
150 END
```

Antes de ejecutar el programa hemos de introducir las líneas de datos, por ejemplo al principio del programa:

```
10 DATA  2,5.1,6,0.1
20 DATA  11.2,2.3
```

Al correr el programa no sucede nada hasta la línea 110. Por orden del READ, el ordenador busca el primer valor de DATA no leído aún, que es el 2, el primero de todos, y R=2. Como consecuencia de las líneas siguientes L=2*π*2=12.5663706, y en pantalla aparece:

```
PARA RADIO  2 LONGITUD  12.5663706
```

De nuevo en la línea 110, se busca otro valor para R. Ahora el primer dato no leído es 5.1 y en la pantalla sale:

```
PARA RADIO  5.1 LONGITUD  32.044245
```

La variable R tomará sucesivamente los valores 6 y 0.1; después de eso, agotados los datos de la línea 10, los tomará de la 20, y R valdrá 11.2 y 2.3 por lo que en la pantalla irá saliendo:

```
PARA RADIO  6 LONGITUD  37.6991118
PARA RADIO .1 LONGITUD .628318531
PARA RADIO  11.2 LONGITUD  70.371675
PARA RADIO  2.3 LONGITUD  14.45153262
OUT OF DATA ERROR
```

El mensaje final nos indica que, después de haber leído todos los DATA, en 110 se ha intentado leer un nuevo dato, y como están todos leídos sale un

mensaje de error. Ha sido una forma, poco elegante todavía, de terminar el programa.

Los valores de las líneas DATA forman una especie de *banco de datos* que se leen ordenadamente, empezando por los de la primera línea, cuando éstos se agotan se pasa a los de la segunda línea, y así sucesivamente.

EJEMPLO 2. Hagamos un ejercicio parecido al ejercicio 2 del capítulo 4, para calcular la media de edad de una población. Supongamos que contamos con la siguiente información:

Edad	1	2	3	4	...	107	108	109	110
Núm. Pers.	835	902	871	790	...	13	7	3	2

Un programa sería:

```
1Ø REM EDAD MEDIA
2Ø REM LOS DATA EN LAS LINEAS SIGUIENTES A 2ØØ
3Ø REM EL PRIMER VALOR SERA LA EDAD Y EL SEGUNDO LA
   FRECUENCIA
4Ø REM AL FINAL PONER COMO DATO FICTICIO UNA FRECUEN-
   CIA NEGATIVA
5Ø FOR I=1 TO 1ØØØ
6Ø READ E,F
7Ø IF F<Ø THEN 11Ø
8Ø LET S=S+E*F
9Ø LET N=N+F
1ØØ NEXT I
11Ø LET MEDIA=S/N
12Ø PRINT "LA MEDIA ES=";MEDIA
5ØØ END
```

Antes de ejecutar este programa será necesario añadir los DATA:

```
2ØØ DATA 1,835, 2,9Ø2,3,871,4,79Ø,...
•
•
•
34Ø DATA 1Ø7,13,1Ø8,7,1Ø9,3,11Ø,2,111,-1 (Testigo)
```

El programa es fácil de entender. Se ha establecido un bucle FOR–NEXT de límites amplios, para que de hecho la salida del bucle sólo se produzca por

el IF de la línea 7Ø, es decir, cuando se llegue al dato testigo, que es el último y que no debe contarse en N ni en S.

EJEMPLO 3. Veamos ahora una aplicación con datos alfanuméricos. Necesitamos un programa que imprima etiquetas para pegar en sobres de correspondencia comercial, que hemos de enviar a los representantes de una serie de firmas. Poseemos una lista en la que figuran los nombres de las empresas, y los de sus representantes, así como las direcciones de éstos. Escribimos el programa:

```
1ØØ REM DIRECCIONES PARA SOBRES
11Ø READ EM$          (Empresa)
12Ø READ NO$          (Nombre)
13Ø READ DI$          (Dirección)
14Ø READ PO$          (Población)
15Ø PRINT NO$
16Ø PRINT
17Ø PRINT EM$
18Ø PRINT
19Ø PRINT DI$
2ØØ PRINT
21Ø PRINT
22Ø PRINT PO$
23Ø PRINT
24Ø PRINT
25Ø PRINT
26Ø GO TO 11Ø
27Ø  DATA  PERAMATIC,SR. BAJO,C/SUCIA 24, VILLANARDOS
28Ø  DATA  INTERCASA,SR. CALVO,C/LARGA 376, VILLA TOBAS
29Ø  DATA  SUBAVIA,SR. GORDO,C/ANCHA 13, VILLALBA
•
•
•
5ØØ  END
```

Al ejecutar el programa obtenemos:

SR. BAJO
PERAMATIC
C/ SUCIA 24

VILLANARDOS

SR. CALVO
INTERCASA
C/ LARGA 376

VILLATOBAS

SR. GORDO
SUBAVIA
C/ ANCHA 13

VILLALBA

Se puede observar que los DATA de cadenas admiten espacios en blanco y puntos, y que no necesitan comillas. Sólo si queremos que una cadena contenga una coma (,) será necesario encerrarla entre comillas.

9.3 RESTORE

Si los datos que se utilizan van a ser necesarios en más de una ocasión dentro del mismo programa, es preciso evitar que, tras haberlos leído una vez, se "agoten", como ocurría hasta ahora. La instrucción RESTORE (Realmacenar) regenera los valores de DATA, hace que el próximo valor a leer sea el primero del primer DATA, como si comenzáramos de nuevo la lectura.

EJEMPLO 4. Completaremos el Ejemplo 2 con el cálculo de la desviación típica. La desviación típica es:

$$S = \sqrt{\frac{\Sigma\,(x_i - \bar{x})^2 \cdot f_i}{n}}$$

Para calcular $(x_i - \bar{x})^2 \cdot f_i$ es necesario conocer previamente la media $(\bar{x})$; siguiendo con el ejemplo 2 elaboramos este programa:

EJEMPLO 2
Calcula la media $\bar{x}$, y el número de personas, n

```
1Ø  REM MEDIA Y DESVIACION TIPICA DE LA
    EDAD
2Ø  REM LOS DATA EN LAS LINEAS SIGUIENTES
    A 2ØØ
3Ø  REM EL PRIMER VALOR SERA LA EDAD Y EL
    SEGUNDO LA FRECUENCIA
4Ø  REM AL FINAL PONER COMO DATO FICTICIO
    UNA FRECUENCIA NEGATIVA
5Ø  FOR I=1 TO 1ØØØ
6Ø  READ E,F
7Ø  IF F<Ø THEN 11Ø
8Ø  LET S=S+E*F
9Ø  LET N=N+F
1ØØ NEXT I
11Ø LET MEDIA=S/N
12Ø PRINT "LA MEDIA ES=" ;MEDIA
13Ø RESTORE
14Ø FOR I=I TO 1ØØØ
15Ø READ E,F (La lectura recomienza debido a
    RESTORE)
16Ø IF F<Ø THEN 19Ø
17Ø LET S=S+(E-MEDIA)↑2*F
18Ø NEXT I
19Ø LET DESV=(S/N)↑.5
195 PRINT "LA DESVIACION TIPICA ES=";DESV
2ØØ DATA 1,835,2,9Ø2,3,871,4,79Ø,...
•
•
•
34Ø DATA 1Ø7,13,1Ø8,7,1Ø9,3,11Ø,2,111,-1
5ØØ END
```

9.4 LOS DATOS EN FICHEROS DE MEMORIA EXTERNA

Las instrucciones READ, DATA y RESTORE son sólo una mejora de las instrucciones LET e INPUT. Un inconveniente grave es que sirven para almacenar y leer datos previos, pero no para grabar los resultados de un programa. Además, la modificación de los datos exige modificar líneas de programa, cosa

siempre pesada y lenta. Estas instrucciones son insuficientes para un tratamiento serio de la información.

¿Cómo se trabajaría con un fichero o archivo de oficina? La primera operación, después de comprarlo, es abrirlo e ir metiendo fichas, puesto que estaba vacío. Para ordenar las fichas caben dos posibilidades: o bien por orden de entrada, a medida que llegan (digamos por antigüedad), o bien con un criterio como el orden alfabético o de un determinado código.

También los ficheros informáticos se pueden construir de estas dos formas. Los primeros se llaman *ficheros de acceso secuencial,* y llevan los datos uno detrás de otro. Se podrían imaginar como una gran variable de índice, que admitiera índices altos, para muchos valores; pero en la que los datos no siguen otro orden o regla de clasificación más que el de llegada, están yuxtapuestos.

Los segundos, los ficheros bien clasificados, se llaman *ficheros de acceso directo.* El orden que siguen no es el alfabético, sino un código que indica *su posición en el disco* en que se han grabado: les señalamos una posición y se graban en (o se leen de) ella.

9.5 FICHEROS SECUENCIALES

Los ficheros secuenciales son más sencillos de manejar, y tienen la pequeña ventaja de ocupar un poco menos que los de acceso directo. Las cassettes sólo admiten ficheros de acceso secuencial, no admiten ficheros de acceso directo.

Las instrucciones para manejar ficheros secuenciales varían notablemente con la marca y modelo del microordenador. Se utilizan básicamente cuatro instrucciones, que nosotros llamaremos OPEN, CLOSE, INPUT#, y PRINT#.

Antes de trabajar con un fichero hay que abrirlo, como si se tratara de un archivo de oficina con llave. Cuando se ha terminado el trabajo hay que cerrarlo: si un programa llega al END con ficheros abiertos, suele salir un mensaje de error. La apertura de un fichero se hace con la instrucción OPEN. En ficheros secuenciales es la instrucción más delicada, pues hay que indicar tres cosas:

- nombre del fichero (si se trata de un fichero nuevo, que comenzamos a cargar, éste es el momento de "bautizarlo").
- número del fichero. Tu microordenador tendrá capacidad para traba-

jar simultáneamente con 7 u 8 ficheros. Al abrir un fichero has de asignarle un número que lo distinguirá de los demás ficheros en acción.

– si el fichero se abre para meter datos en él, o para leer los datos que tiene. Puede sorprenderte que al abrir un fichero haya que señalar esto, pero es así.

Veamos algunos ejemplos:

```
1ØØ OPEN 5,1,"NOTAS"
```

abre el fichero "NOTAS" asignándole el número 5, con el código 1, que supondremos que es el de escribir, es decir, meter datos en el fichero. Como es un fichero nuevo, el nombre de "NOTAS" se lo hemos puesto ahora, en la línea 1ØØ.

```
13Ø OPEN 3,Ø,"NOMBRES"
```

Esta instrucción abre el fichero llamado "NOMBRES" asignándole el número 3, con el código Ø, que supondremos es el de leer datos del fichero.

Para cerrar un fichero basta señalar su número. Para los ejemplos anteriores sería:

```
22Ø CLOSE 5
27Ø CLOSE 3
```

Para escribir datos en un fichero se utiliza la instrucción PRINT#, que trabaja igual que PRINT sólo que añadiendo el símbolo # y el número del fichero:

```
12Ø PRINT# 5:N,7,A$,"BYE"
18Ø PRINT# 5:"HOLA",K,.ØØ1
```

Estas dos instrucciones graban en el fichero 5 los datos: valor de N, 7, valor de A$, "BYE", "HOLA", valor de K y .001; los mismos que habrían salido en la pantalla quitando en las dos instrucciones los caracteres #5. El primer signo de puntuación, que en nuestro ejemplo es dos puntos (:) suele ser distinto de los restantes (en nuestro ejemplo comas (,)). Para utilizar estas instrucciones sin error es necesario que el fichero 5 se haya abierto previamente para grabar.

Para leer datos de un fichero se utiliza la instrucción INPUT#, cuyo funcionamiento es análogo al de PRINT#.

```
12Ø  INPUT# 3:A,B,C$,D$
18Ø  INPUT# 3:I,J,K,L$
```

Estas dos últimas líneas habrían leído cuatro datos cada una, asignándolos respectivamente a las variables:

A, B, C$ y D$
I, J, K y L$

EJEMPLO 5. Un profesor quiere llevar las notas de sus alumnos en un fichero de ordenador. Lo primero que necesita es introducir en el fichero los nombres y las notas que ya posee de los alumnos, que supondremos serán tres.

```
1Ø  REM  FICHERO DE NOTAS DE 35 ALUMNOS
2Ø  OPEN  1,1 "NOTAS"
3Ø  FOR  I=1  TO  35
4Ø  INPUT  "NOMBRE":N$
5Ø  INPUT  "NOTAS";A,B,C
6Ø  PRINT#  1:N$,A,B,C
7Ø  NEXT  I
8Ø  CLOSE  1
9Ø  END
```

Tras abrir el fichero se inicia un bucle en el que, por 35 veces, el programa preguntará el nombre y notas de un alumno, y a continuación se van grabando en el fichero. Al acabar con las 35 personas se termina el bucle y se cierra el fichero. El profesor ya tiene sus datos en una cassette.

Pero hace un nuevo examen, y desea anotar la cuarta nota en cada alumno de su fichero. El profesor podría hacer un programa como éste:

```
5    REM  ACTUALIZACION DE NOTAS
1Ø   DIM  A$(35),A(35),B(35),C(35),D(35)
2Ø   OPEN  3,Ø,"NOTAS"
3Ø   FOR  I=1  TO  35
4Ø   INPUT#  3:A$(I),A(I),B(I),C(I)
5Ø   NEXT  I
6Ø   CLOSE  3  (Queda leído el fichero)
7Ø   OPEN  5,1, "NOTAS"
8Ø   FOR  I=1  TO  35
```

```
9Ø  PRINT "NUEVA NOTA DE";A$(1);
1ØØ INPUT D(I)
11Ø PRINT# 5:A$(I),A(I),B(I),C(I),D(I)
12Ø NEXT I
13Ø CLOSE 5
14Ø END
```

Es necesario, como se hace en el primer bucle, volcar el archivo en la memoria del ordenador. En el segundo bucle se van añadiendo notas, y escribiendo los valores de antes y el nuevo, en el archivo.

Si a fin de curso el profesor quiere dar la nota media, de 7 notas que supondremos que tiene de cada alumno, le puede servir el siguiente programa:

```
1Ø  REM NOTA MEDIA
2Ø  OPEN 2,Ø, "NOTAS"
3Ø  FOR I=1 TO 35
4Ø  INPUT# 2:S$,A,B,C,D,E,F,G
5Ø  LET M=(A+B+C+D+E+F+G)/7
6Ø  PRINT A$,M
7Ø  NEXT I
8Ø  CLOSE 2
9Ø  END
```

9.6 FICHEROS DE ACCESO DIRECTO

Los ficheros de acceso secuencial tienen limitaciones importantes si nuestro enfoque de la Informática es ambicioso. En ellos para modificar un dato es necesario leer por completo el archivo y volverlo a escribir corregido. Si para corregir una sola nota, de un alumno que ha reclamado por ejemplo, hay que leer y reescribir todo el fichero, esto ya no es tan cómodo. En una cassette, incluido el tiempo de maniobra, rebobinado, etc., puede llevar unos cuantos minutos.

Es igualmente grave que, de ordinario, cuando se trabaja con cassette, para modificar o añadir datos a un fichero es necesario volcar su contenido en la memoria del ordenador. Un fichero importante excede esa capacidad, y no puede ser tratado mediante acceso secuencial.

Estos problemas se solventan con los ficheros de acceso directo, que utili-

zan unidades de discos (en microordenadores generalmente discos pequeños flexibles, floppy disks).

De un fichero de acceso directo se pide que haga con rapidez las siguientes operaciones:

- altas (añadir nuevas fichas al fichero)
- modificaciones
- consultas de una ficha
- bajas
- proporcionar un listado

Observa por ejemplo la tercera operación. Para consultar un dato en un fichero secuencial hay que leerlos todos, al menos hasta llegar a él. Con el fichero de acceso directo esto no es necesario.

El único inconveniente es el precio de las unidades de discos, que hará que no las adquieras si no te son realmente necesarias. Lo malo es que en cuestiones de discos, manejando ficheros de acceso directo, las instrucciones varían muchísimo entre unos equipos y otros, y se sale de los objetivos de este libro el hacer aquí una especie de denominador común. El día que te decidas a afrontar este problema tendrás que resolverlo para tu caso concreto, leyendo el manual del equipo o consultando a tus proveedores.

RECUERDA	
DATA READ RESTORE	OPEN CLOSE INPUT# PRINT#

EJERCICIOS

9.1) La máquina lee unos nombres de personas almacenados en unas líneas DATA, y va escribiendo los nombres en la pantalla, precedidos de los números 1, 2, 3, etc.

9.2) Usando las instrucciones READ–DATA asignar los valores −1.6E−6, −5ØØ,.4Ø77, MAYO, OCTUBRE, 1ØØ, 11Ø, 12Ø, 13Ø, 14Ø, 15Ø a las variables C1, C2, C3, X$, Y$, Z(1), Z(2), Z(3), Z(4), Z(5) y Z(6). Que al final aparezca en la pantalla:

C1 = −1.6E−6
C2 = −5ØØ
C3 = 4Ø77
X$ = MAYO
• • •
• • •
• • •

9.3) Resuelve el ejercicio 7.10) de simulación de unas elecciones introduciendo los nombres de los candidatos en instrucciones DATA.

9.4) Introducir en instrucciones DATA los equipos de primera división de fútbol y el número de partidos ganados por cada equipo. Que el programa saque en pantalla los nombres de todos los equipos, y a continuación, a la derecha, a partir de un cierto lugar (función TAB), un diagrama de barras con tantos espacios como victorias haya logrado ese equipo.

9.5) Resuelve el ejercicio 6.15) de diagrama de barras de la evolución de una población, introduciendo los valores mediante instrucciones DATA.

9.6) Prepara un programa que permita formar una tabla de 6 x 6 elementos (6 filas y 6 columnas) tomando números que figuren en sentencias DATA.

9.7) Preparar un programa que cree simultáneamente dos tablas de distinto número de elementos, una numérica y otra alfanumérica (por ejemplo DIM A$(3,4),B(5,2)), a partir de sentencias DATA.

9.8) Tenemos una serie de sentencias DATA en las que figuran los nombres y edades de quince personas. Formar dos tablas de cinco filas y tres co-

lumnas cada una, una numérica y otra alfanumérica. El programa ha de permitir la exhibición de la tabla que solicitemos, hasta que ordenemos finalizar.

9.9) Preparar un programa que, utilizando para almacenamiento de un texto sentencias DATA, cuente el número de vocales, letras y palabras.

9.10) En un programa semejante al anterior calcular la frecuencia absoluta de cada letra.

9.11) Hacer otro programa parecido a los anteriores, pero calculando la frecuencia relativa de las palabras de longitud 1, 2, 3, . . ., 20.

9.12) Construir un programa que, tomando la información necesaria de líneas DATA, forme la tabla periódica de los elementos químicos. Para cada elemento ha de figurar, además de su símbolo, su número atómico, peso atómico y configuración electrónica. Te recomendamos utilizar una sola cadena para cada elemento conteniendo toda esa información. En la cadena los dos primeros lugares se pueden reservar para el símbolo, los tres siguientes para el número atómico, cinco más para el peso atómico, etc.; los lugares que un elemento no necesite se dejan a la izquierda y se llenan con guiones, por ejemplo:

```
-H --1 1.ØØ8 1...
HE --2 4.ØØ3 2...
LI --3 7.Ø16 21...
```

Este programa, además de formar la tabla, ha de permitir:

a) Comprobar su correcto almacenamiento tanto por filas como por columnas.

b) Proporcionarnos en la pantalla toda la tabla periódica en su disposición tradicional, apareciendo sólo los símbolos de los elementos.

c) Conocer, para un elemento cualquiera, su número atómico, peso atómico y configuración electrónica.

9.13) Completar el programa anterior con una subrutina que permita conocer, para un compuesto dado cuya fórmula introduciremos desde el teclado, su peso molecular.

9.14) Un problema de topografía: se desea hacer el levantamiento topográfico de un terreno (un plano con curvas de nivel y situación de algunos lugares destacados). Para eso se establece un recorrido dentro del terreno.

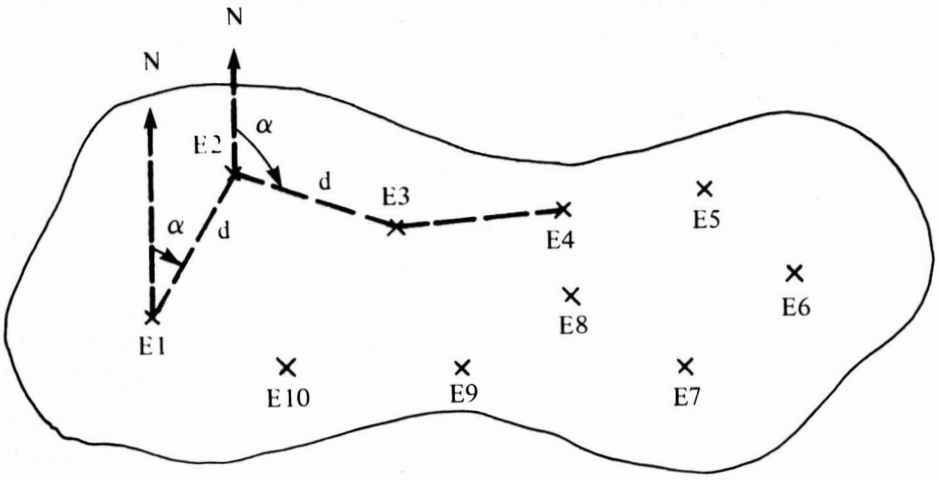


Fig. 9.1

Se conocen las coordenadas (X1,Y1) de E1 y su cota (Z1). Mediante el taquímetro (aparato de topografía) se mira desde cada punto (Estación) al siguiente, y se obtienen tres datos: desnivel (Δz), distancia d y ángulo α con el Norte. Así las coordenadas de E2 serán:

$$X2 = X1 + d.\text{sen}\,\alpha$$
$$Y2 = Y1 + d.\cos\alpha$$

y la cota de E2 será

$$Z2 = Z1 + \Delta Z$$

Hacer un programa que, introduciendo en líneas DATA los valores d_i, α_i e Δz_i que se observan en cada estación, vaya calculando y sacando en la pantalla las coordenadas y cota de las estaciones.

9.15) INTERPOLACION POR EL METODO DE LAGRANGE

Se define como polinomio de Lagrange el polinomio de grado n:

$$P(X) = y_0 \cdot P_0(X) + y_1 \cdot P_1(X) + \ldots + y_n \cdot P_n(X)$$

Donde $$P_0(X) = \frac{(X - X_1) \cdot (X - X_2) \ldots (X - X_n)}{(X_0 - X_1) \cdot (X_0 - X_2) \ldots (X_0 - X_n)}$$

$$P_1(X) = \frac{(X - X_0) \cdot (X - X_2) \ldots (X - X_n)}{(X_1 - X_0) \cdot (X_1 - X_2) \ldots (X_1 - X_n)}$$

$$P_n(X) = \frac{(X - X_0) \cdot (X - X_1) \ldots (X - X_{n-1})}{(X_n - X_0) \cdot (X_n - X_1) \ldots (X_n - X_{n-1})}$$

El polinomio de Lagrange pasa por los puntos (X_0,Y_0), (X_1,Y_1), . . ., (X_n,Y_n), y se utiliza para interpolar, para aproximar el valor de Y, correspondiente a valores intermedios de X.

Haz un programa capaz de interpolar para cualquier valor intermedio de X el valor de Y, conociendo una serie de puntos (X_i,Y_i) por los que pasa la función, y que introducirás en líneas DATA.

9.16) En un fichero constan los apellidos y el nombre de todos los alumnos de un curso. Preparar un programa que forme, a partir de este fichero, otro en el que estos mismos alumnos estén ordenados alfabéticamente.

9.17) Prepara un programa que recupere un dato de un lugar, L, de un fichero y lo inserte en otro, J, de otro fichero.

9.18) Haz un programa que sume los valores que figuran en dos ficheros que contienen datos numéricos, y que son de igual longitud, y que introduzca los resultados en un nuevo fichero, llamado "suma", creado a este fin.

9.19) En un almacén al por mayor se prepara semanalmente un fichero en que figuran, para cada artículo, su número de registro, nombre, precio, existencias al principio de la semana y al final de ésta. Prepara, según los prin principios de la programación descendente, un programa que:

a) almacene esta información en un fichero (considera solamente diez artículos).

b) avise si de algún artículo quedan en el almacén menos existencias de las que han sido vendidas durante la semana.

c) nos presente un balance semanal en que para cada artículo figure su número de registro, nombre, número de unidades vendidas en la semana, precio unitario y montante de las ventas por ese artículo. Finalmente han de aparecer las ventas totales.

9.20) Los cuatro miembros de un grupo rock han llegado a un acuerdo sobre el porcentaje que corresponde a cada uno por los ingresos de las diferentes canciones; este porcentaje varía de una a otra. Realiza un programa que contenga estos datos en sentencias DATA, y que les permita conocer los ingresos de cada uno en función de los ingresos que proporciona al grupo una cierta canción.

9.21) Prepara un programa que forme un fichero de listín telefónico personal. Ha de permitir la introducción de cualquier nuevo número en el lugar adecuado, de acuerdo con el nombre del abonado (orden alfabético). Para obtener un teléfono hemos de teclear el apellido y nombre del abonado, y el programa contestará el teléfono. Por si de algún amigo no recordamos el apellido, si tecleamos sólo el nombre, el programa contestará con los apellidos y teléfono de todos los que tengan ese nombre, por ejemplo:

TFNO. DESEADO? LOPEZ, CARLOS
LOPEZ,CARLOS 6 78 9Ø 12

o bien:

TFNO. DESEADO? CARLOS
LOPEZ, CARLOS 6 78 9Ø 12
PEREZ, CARLOS 4 56 78 9Ø

Sugerencia: utiliza una sola variable alfanumérica para introducir apellido, nombre y teléfono.

9.22) Haz un programa análogo al anterior, pero que además proporcione nombre y apellido del abonado si le damos el número.

9.23) Forma un fichero con la tabla siguiente: (datos de 1972):

PAIS	COCHES	POBLACION	SUPERFICIE
GRAN BRETAÑA	$13.6 \cdot 10^6$	$55.9 \cdot 10^6$	$244 \cdot 10^3$ Km²
BELGICA	2.3 "	9.8 "	30.5 " "
DINAMARCA	1.2. "	5 "	43.1 " "
FRANCIA	13.4 "	52.1 "	547 " "
ALEMANIA FEDERAL ..	15.6 "	62 "	248.6 " "
IRLANDA	0.4 "	3 "	70.3 " "
ITALIA	12.5 "	54.9 "	301.2 " "
LUXEMBURGO	0.1 "	0.4 "	2.6 " "
HOLANDA	2.8 "	13.4 "	40.8 " "

Haz un programa que nos dé el número de habitantes por coche. Haz otro programa que nos dé la población y el número de coches de aquellos países que tengan una extensión menor de 100.000 km².

9.24) Con los datos del ejemplo anterior prepara un programa que nos proporcione la lista de todos los países de la tabla, ordenados según elijamos, de acuerdo con:

a) Habitantes por coche

b) Habitantes por km²

c) Coches por km²

9.25) En la siguiente tabla se presentan las veinte primeras industrias en ventas, en USA en 1976. Construye con la tabla un fichero.

LUGAR	EMPRESA	VENTAS EN $.10^6	ACTIVIDAD
1	Exxon	48.630	Petróleo
2	General Motors	47.181	Automoción
3	Ford	28.839	Automoción
4	Texaco	26.451	Petróleo
5	Mobil Oil	26.062	Petróleo
6	Standard Oil of California	19.434	Petróleo
7	Gulf Oil	16.451	Petróleo
8	I.B.M.	16.304	Electrónica

LUGAR	EMPRESA	VENTAS EN $. 10^6	ACTIVIDAD
9	General Electric	15.697	Electrónica
10	Chrysler	15.537	Automoción
11	I.T.T.	11.764	Teléfonos
12	Standard Oil	11.532	Petróleo
13	Shell Oil	9.229	Petróleo
14	U.S. Steel	8.604	Acero
15	Atlantic Richfield	8.462	Petróleo
16	Du Pont	8.361	Químicas
17	Continental Oil	7.957	Petróleo
18	Western Electric	6.930	Electrónica
19	Procter & Gamble	6.512	Prod. de Consumo
20	Tenneco	6.389	Petróleo

Haz un programa que, tecleando la actividad, conteste con todas las empresas que se dedican a ella, y los porcentajes de cada una, por ejemplo:

ACTIVIDAD? AUTOMOCION
GENERAL MOTORS 51.53 %
FORD 31.5Ø %
CHRYSLER 16.97 %

9.26) Nuestra nave espacial posee un ordenador cocinero que contiene un fichero en el que constan las calorías por 100 gramos de los siguientes alimentos:

Pan	263	Pasta
Pastas	377	"
Arroz	362	"
Vaca	194	Carne
Pollo	200	"
Conejo	180	"
Jamón Serrano	500	"
Jamón York	420	"
Salchichas	343	"
Bacalao	107	Pescado

Merluza	74	"
Salmón	143	"
Sardina	115	"
Lenguado	84	"
Salmonete	113	"
Judías	335	Legumbre
Lentejas	340	"
Garbanzos	320	"
Acelgas	27	Verdura
Alcachofas	27	"
Coliflor	25	"
Judías Verdes	35	"
Patatas	83	"
Espinacas	20	"
Leche	65	Lácteos
Queso	370	"
Naranjas	45	Fruta
Manzanas	58	"
Melón	20	"
Nueces	646	"
Uvas	66	"
Peras	63	"
Huevos (uno)	81	Huevo
Mantequilla	716	Grasa
Aceite Oliva	891	"

En instrucciones DATA tiene la relación existente entre la edad de una persona y las calorías diarias a consumir:

EDAD	CALORIAS
2	1000
6	1400
10	2000
15	3200
20	3500
Hombre	3000
Mujer	2200
65-70	2000-2500
más de 70	1800-2200

así como la edad de los diferentes viajeros:

DATA PACO, 3Ø , CARLOS, 4 , . . .

Haz un programa que permita a cada viajero, tras identificarse, elegir algunos de los alimentos que va a tomar en un día; tras ello el ordenador completará el menú determinando:

a) Los demás alimentos, teniendo en cuenta que diariamente se ha de tomar carne o pescado, verdura y fruta, huevos y grasa, y se pueden alternar las pastas y las legumbres (dejar alguna capacidad de elección o variación a la máquina).

b) Las cantidades de cada alimento según criterios que tú mismo has de prefijar.

9.27) Ampliar el programa para que los primeros días de la semana (calendario galáctico) se elija el menú semanal.

9.28) A continuación reproducimos un cuadro de datos de las provincias españolas, para que con ellos puedas hacer más ejercicios de ficheros:

PROVINCIAS	Población de derecho (31-12-76)	Número municipios (31-12-75)	Superficie Km^2	Matrimonios Año 1976	Nacimientos Año 1976	Defunciones Año 1976	Fall. menor un año Año 1976	Viviendas familiares Año 1976	Turismos matriculados Año 1976
Alava	245.669	59	3.047	1.488	4.472	1.614	51	57.066	5.986
Albacete	331.734	86	14.858	2.404	6.213	3.038	69	112.809	4.885
Alicante	1.079.244	138	5.863	7.705	21.224	8.930	180	367.601	23.890
Almería	390.449	103	8.774	3.077	8.170	3.216	117	127.004	5.821
Avila	188.738	262	8.048	1.256	2.067	1.775	34	76.463	2.554
Badajoz	638.672	162	21.657	4.482	10.382	6.153	89	211.281	7.045
Baleares	611.194	65	5.014	4.337	10.913	5.662	113	213.512	14.493
Barcelona	4.485.086	310	7.733	30.447	74.858	31.631	883	1.183.718	122.716
Burgos	348.675	439	14.269	2.236	5.910	2.992	61	155.918	6.559
Cáceres	425.044	218	19.945	3.304	5.986	3.875	73	126.153	4.675
Cádiz	946.999	42	7.385	7.752	22.117	6.858	276	232.544	11.824
Castellón	414.744	140	6.679	2.854	7.006	4.014	17	157.391	9.269
Ciudad Real	479.147	98	19.749	3.161	7.478	4.249	87	160.095	6.155
Córdoba	714.632	75	13.718	5.534	12.874	5.878	148	211.783	9.867
Coruña (La)	1.064.976	93	7.876	7.898	17.905	9.306	288	281.126	18.791
Cuenca	220.499	232	17.061	1.492	2.716	2.255	16	93.675	2.590
Gerona	447.657	223	5.886	3.510	7.538	4.406	63	161.832	13.762
Granada	742.764	169	12.531	5.311	13.864	6.079	225	221.206	7.241
Guadalajara	146.179	295	12.190	683	1.785	1.322	16	62.239	1.973
Guipúzcoa	690.594	81	1.997	4.866	12.773	4.450	134	164.750	16.258
Huelva	402.629	79	10.085	3.050	7.316	3.920	101	117.043	5.694
Huesca	212.360	209	15.671	1.203	2.539	2.115	18	71.780	4.974
Jaén	647.532	96	13.498	4.496	10.789	5.426	137	204.279	5.628
León	529.524	222	15.468	4.146	7.486	4.856	79	171.560	8.637
Lérida	348.388	230	12.028	2.359	5.664	3.393	39	108.489	8.601
Logroño	243.140	176	5.034	1.762	4.049	2.194	39	80.562	5.074
Lugo	409.061	66	9.803	2.773	4.884	4.875	76	122.580	5.475
Madrid	4.576.253	179	7.995	31.545	93.704	27.434	839	1.132.793	124.691
Málaga	928.069	99	7.276	6.466	19.036	7.754	172	274.466	13.094
Murcia	890.974	43	11.317	6.530	18.591	7.232	208	274.223	14.227

PROVINCIAS	Población de derecho (31-12-76)	Número municipios (31-12-76)	Superficie Km^2	Matrimonios Año 1976	Nacimientos Año 1976	Defunciones Año 1976	Fall. menor un año Año 1976	Viviendas familiares Año 1976	Turismos matriculados Año 1976
Navarra	490.531	264	10.421	3.595	8.633	4.271	81	131.688	12.118
Orense	431.774	92	7.278	2.630	4.773	4.309	49	136.356	4.986
Oviedo	1.111.917	78	10.565	8.016	17.746	9.462	208	327.620	22.303
Palencia	183.824	203	8.029	1.401	2.505	1.873	38	62.859	3.122
Palmas (Las)	661.629	34	4.065	4.966	14.377	4.195	175	154.498	15.350
Pontevedra	857.326	61	4.477	6.420	16.627	6.348	230	208.095	15.160
Salamanca	351.785	359	12.336	2.645	5.235	3.310	75	121.568	6.743
Sta. Cruz de Tenerife	680.498	53	3.208	4.535	12.955	4.387	150	153.901	11.469
Santander	495.185	102	5.289	3.994	9.028	4.145	131	150.783	9.830
Segovia	149.013	215	6.949	1.064	2.070	1.419	10	5.343	2.666
Sevilla	1.386.187	102	14.001	11.101	32.058	11.399	315	362.118	21.946
Soria	102.967	185	10.287	569	1.114	1.039	11	42.681	1.967
Tarragona	490.056	177	6.283	3.571	9.044	4.854	56	174.665	11.562
Teruel	153.247	234	14.804	824	1.597	1.732	8	73.092	2.342
Toledo	465.916	204	15.368	3.362	7.208	4.249	76	156.421	6.455
Valencia	1.969.062	263	10.763	14.589	38.168	17.288	368	618.225	43.520
Valladolid	458.020	229	8.202	3.098	8.938	3.136	95	128.326	10.236
Vizcaya	1.178.055	96	2.217	7.917	23.230	7.995	163	301.019	26.585
Zamora	229.834	252	10.559	1.667	2.818	2.357	37	81.556	3.430
Zaragoza	801.029	292	17.194	5.549	13.642	6.903	135	249.393	15.144
TOTAL	36.448.481	8.194	504.750	259.640	662.084	291.573	7.089	10.630.131	739.493

FUENTE: Instituto Nacional de Estadística.

9.29) LA INSTRUCCION GET

Otra instrucción, más que de manejo de datos, de entrada de datos, es la instrucción GET.

Es una instrucción semejante a INPUT muy utilizada en representaciones gráficas y en juegos. Al igual que INPUT, GET acepta un dato desde el teclado, pero se diferencia de INPUT en lo siguiente:

1º GET no saca interrogación en la pantalla, no "estropea" una representación gráfica.

2º GET no detiene la ejecución del programa: GET repasa el teclado buscando si has pulsado algo. Si has tecleado algún número, GET N asigna a N ese valor; si has tecleado una letra, GET A$ da a A$ ese valor, igual que INPUT N e INPUT A$. Si no has tecleado nada, GET P da a P el valor Ø y GET Z$ da a Z$ la cadena " " (cadena nula).

3º GET no saca en la pantalla tu respuesta y no necesita que se pulse RETURN.

De ordinario se suele utilizar de la siguiente manera:

```
4Ø GET A$                    4Ø GET N
5Ø IF A$="" THEN 4Ø          5Ø IF N=Ø THEN 4Ø
```

organizando un bucle del que sólo saldrás al pulsar una tecla, y teniendo cuidado de si la tecla ha de ser un número o una letra.

9.30) Ejecuta el siguiente programa y razona su funcionamiento:

```
1Ø GET A$
2Ø IF A$="" THEN 1Ø
3Ø PRINT A$
4Ø GO TO 1Ø
5Ø END
```

9.31) Ejecuta y razona el siguiente programa:

```
1Ø GET A$
2Ø IF A$="" THEN 1Ø
3Ø IF A$=CHR$(13) THEN 6Ø
4Ø PRINT A$;
```

```
5Ø  GO  TO  1Ø
6Ø  PRINT "“ OTRA LINEA"
7Ø  GO  TO  1Ø
8Ø  END
```

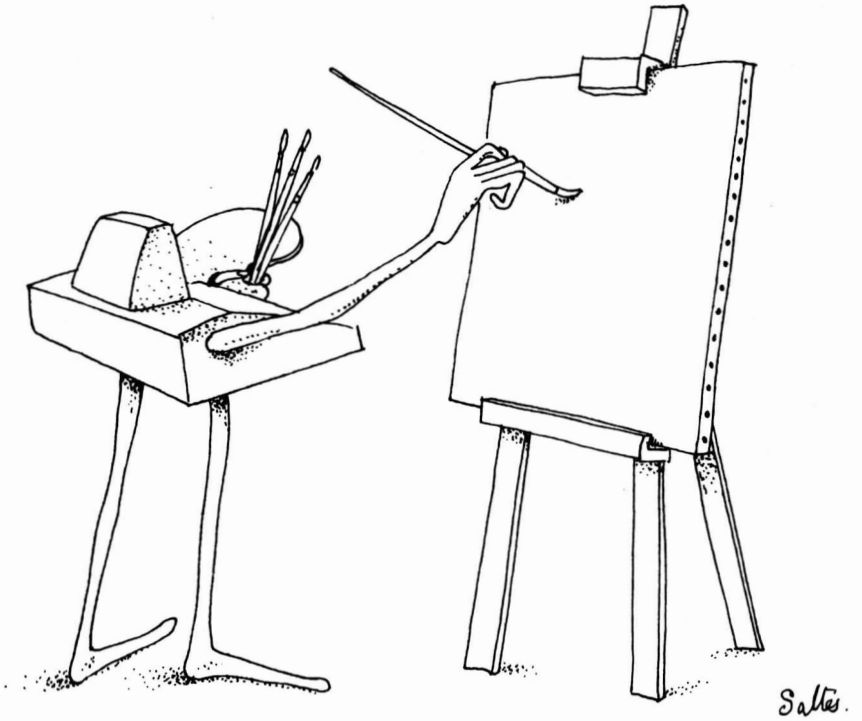
Observación: 13 es el código de la tecla [RETURN].

9.32) Diviértete con el siguiente programa:

```
1Ø  GET  A$
2Ø  IF  A$="  "  THEN  1Ø
3Ø  FOR  I=1  TO  12
4Ø  PRINT
5Ø  NEXT  I
6Ø  PRINT  TAB(17);"HUY"
7Ø  FOR  I=1  TO  12
8Ø  PRINT
9Ø  NEXT  I
1ØØ  GO  TO  1Ø
11Ø  END
```

APENDICE

ESTUDIO DE LAS POSIBILIDADES GRAFICAS DE LOS MICROORDENADORES



A.1 INTRODUCCION

Hay ordenadores cuyo sistema operativo permite situar un carácter (numérico, alfabético o gráfico) en un lugar cualquiera de la pantalla, indicando dicho lugar por un par de coordenadas. Otros están dotados de alta resolución, pudiendo entonces dibujar puntos, por sus coordenadas, en una pantalla dividida en 30.000, 50.000, 100.000 puntos, según ordenadores. Pero ni lo uno ni lo otro es imprescindible para sacar partido gráfico a nuestro ordenador. En este capítulo estudiamos la explotación de las posibilidades gráficas que ofrecen los

microordenadores dotados de monitor de T.V. *sin* alta resolución. El microordenador que hemos utilizado nosotros descompone la pantalla en 25 filas (desde la 0 a la 24) y 40 columnas (desde la 0 a la 39), lo que da lugar a 1000 cuadrados, en cada uno de los cuales podemos representar un símbolo numérico, literal o gráfico, estos últimos dentro de un pequeño alfabeto o juego de caracteres disponible.

Un primer problema con el que nos encontramos es la no estandardización, no ya de las instrucciones, sino de las coordenadas en la pantalla. Hemos optado por situar el origen en el vértice superior izquierdo, que es la opción más generalmente adoptada.

Además de las instrucciones standard del BASIC, hemos utilizado las siguientes, que por no ser propias del lenguaje, sino depender del constructor, cada uno las llama de una manera:

- borrado: borra la pantalla y sitúa el cursor en el primer lugar de ésta (se encuentra en todos los microordenadores);
- ir al origen: no borra la pantalla pero lleva el cursor al primer lugar.

A.2 TABLAS

Parece obligado comenzar cualquier estudio sobre graficación con un ejemplo de utilización de la función TAB para disponer apropiadamente una tabla en la pantalla. La orden TAB(X) lleva el cursor a la columna X, dentro de la fila en que se encuentra. Veamos un programa que nos proporciona una tabla numérica:

```
1Ø  (borrado de pantalla)
2Ø  REM ENCABEZAMIENTO
3Ø  PRINT TAB(5);"NUMERO";TAB(16);"DOBLE";TAB(25):"CUA-
    DRADO"
4Ø  FOR I=3 TO 34
5Ø  PRINT TAB(I);"*";
6Ø  NEXT I
65  PRINT
7Ø  REM TABLA
8Ø  FOR N=1 TO 2Ø
9Ø  PRINT TAB(6);N;TAB(16);2*N;TAB(26);N*N
```

```
1ØØ  NEXT  N
11Ø  GO  TO  11Ø
12Ø  END
```

El programa es muy sencillo: primero borra la pantalla (instrucción 1Ø); escribe el encabezamiento y lo subraya (hasta la fila 6Ø); posteriormente, escribe los valores requeridos para los números que van del 1 al 20; por último, la ejecución cae en un ciclo sin fin (instrucción 11Ø). Tanto la instrucción 1Ø como la 11Ø tienen una función principalmente estética; la primera evita la permanencia en pantalla de símbolos anteriores a la ejecución; la segunda, la llegada de símbolos posteriores a ésta; unos y otros podrían estropear la claridad de los textos o dibujos, o llevarnos a confusión.

A.3 REPRESENTACION DE FUNCIONES CON TAB

Otra posibilidad que ofrece la función TAB es la de representar gráficamente funciones. Es inmediato pensar que si el origen de coordenadas puede situarse en el vértice superior izquierdo y TAB(X) sitúa el cursor en la columna X, entonces para funciones FNF(X) positivas, TAB(FNF(X)) situará el cursor en la columna FNF(X). Así, considerando x creciente hacia abajo e y creciendo de izquierda a derecha, podemos representar f(x) = 70/(x+1) con este sencillo programa:

```
1Ø   (Borrado)
2Ø   FOR  X=1  TO  23
3Ø   PRINT  TAB (7Ø/(X + 1) ); "*"
4Ø   NEXT  X
5Ø   GO  TO  5Ø
6Ø   END
```

cuyo segmento principal, las líneas 2Ø a 4Ø, cumple la función de imprimir en el lugar adecuado un asterisco y esto lo hace 23 veces.

Si queremos utilizar TAB para representar funciones cualesquiera, hemos de introducir un condicional que sólo permita actuar a esta instrucción cuando su argumento esté entre 0 y 39. El diagrama de flujo asociado será:

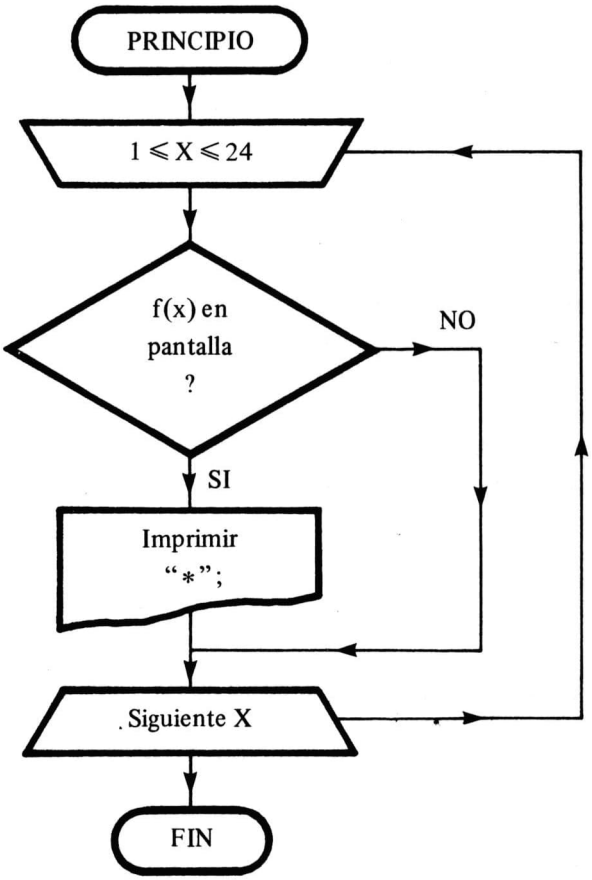


Fig. A. 1

Para la función $y = x^2 + 3$, el programa quedaría así:

```
5    DEF FNF (X)=X↑2+3
1Ø   (Borrado)
2Ø   FOR X=1 TO 24
3Ø   IF FNF(X)>=Ø AND FNF(X)<=38 THEN PRINT TAB
     (FNF (X) );"*";
35   PRINT
4Ø   NEXT X
5Ø   GO TO 5Ø
6Ø   END
```

En general, para representar cualquier función hemos de estudiar dos cuestiones previas: cómo centrarla en la pantalla mediante traslaciones y cómo elegir la escala más conveniente. Si quisiéramos representar la función seno habría que dar dos pasos:

a) Para centrarla, tendremos que representar la función y = sen (x) + 19.

b) Pero la función seno oscila entre −1 y +1, y va a salir una línea vertical casi recta en el centro de la pantalla. Para qué la amplitud de la representación abarque toda la pantalla, tendremos que representar la función y = 15*sen (x) + 19, y así los puntos se moverán entre los espacios: 15*(−1) + 19 = 4 y 15*(1) + 19 = 34.

En la práctica sustituimos la línea 5 por:

```
5  DEF  FNF (X) = 19 + 15*SIN(X/5)
```

donde la expresión X/5 como argumento del seno, surge de la necesidad de adaptar la escala de las X, teniendo en cuenta que las funciones trigonométricas en BASIC trabajan en radianes.

A.4 REPRESENTACION DE MAS DE UNA FUNCION

Lo que no puede hacer la función TAB es tomar, en una fila, un valor menor que otro que ya haya tomado en ella. Para poder representar simultáneamente dos funciones, es necesario comenzar en cada fila por la de menor ordenada. Veamos un ejemplo:

```
5    REM  FUNCIONES
1Ø   DEF  FNF(X) = 18*COS(X/5)
2Ø   DEF  FNG(X) = 18*SIN(X/5)
3Ø   REM  INTERVALO DE REPRESENTACION
4Ø   INPUT  "ORIGEN DEL INTERVALO";P
5Ø   INPUT  "FIN DEL INTERVALO";F
6Ø   REM  REPRESENTACION
7Ø   (Borrado)
8Ø   FOR  X=P  TO  F  STEP  (F−P)/23
9Ø   IF  INT(FNF(X) ) = INT(FNG(X) )  THEN  C = 1
1ØØ  IF  INT(FNF(X) ) < INT(FNG(X) )  THEN  C = 2
```

```
11Ø  IF  INT(FNF(X) ) > INT(FNG(X) )  THEN  C = 3
12Ø  ON  C  GOSUB  2ØØ, 25Ø, 3ØØ
13Ø  NEXT  X
14Ø  GO  TO  14Ø
2ØØ  REM  SE CRUZAN LAS DOS CURVAS
21Ø  PRINT  TAB(FNF(X) + 18);"1"
22Ø  RETURN
25Ø  REM  EL VALOR DE F(X) ES MENOR QUE EL DE G(X)
26Ø  PRINT  TAB(FNF(X) + 18);"1";TAB(FNG(X) + 18);"2"
27Ø  RETURN
3ØØ  REM  EL VALOR DE F(X) ES MAYOR QUE EL DE G(X)
31Ø  PRINT  TAB(FNG(X) + 18);"2";TAB(FNF(X) + 18);"1"
32Ø  RETURN
4ØØ  END
```

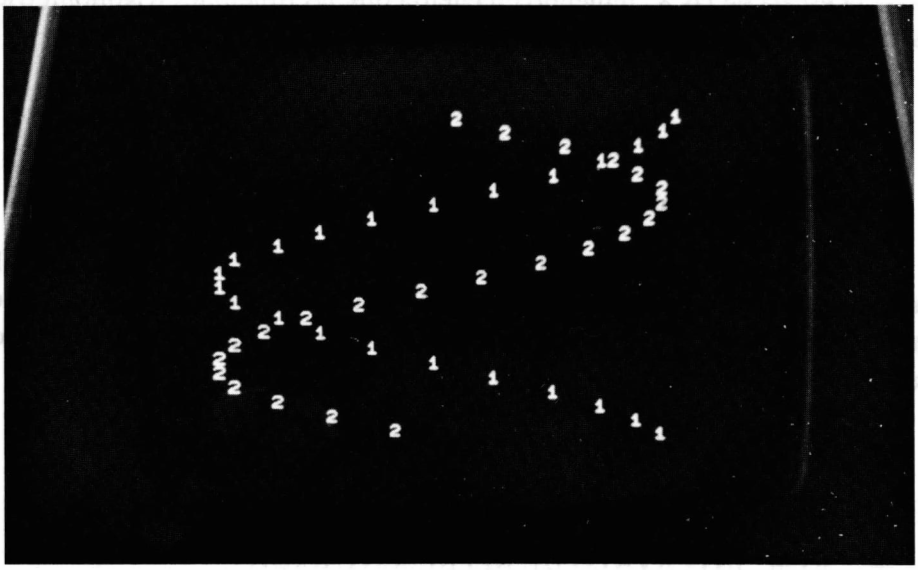


Fig A. 2

A.5 SEGMENTOS

También podemos emplear TAB para dibujar segmentos horizontales en una fila cualquiera. Para ello, tras borrar la pantalla, bajaremos el cursor a la fila requerida y, usando la función TAB, dibujaremos en los lugares precisos.

Como ya dijimos en el apartado 8.2, el empleo de las subrutinas resulta extremadamente útil para favorecer la programación descendente. Podemos descomponer un proceso en programas parciales que desarrollamos aparte, empleando subrutinas, de manera que la posible complejidad o la longitud de un proceso parcial, no enturbie la simplicidad del general. Este procedimiento se puede emplear repetidamente en un mismo problema.

Aunque alguno de los programas que siguen pueden ser realizados de manera más simple sin emplear subrutinas, con fines didácticos y metodológicos hemos preferido utilizarlas para unificar el tratamiento de los problemas.

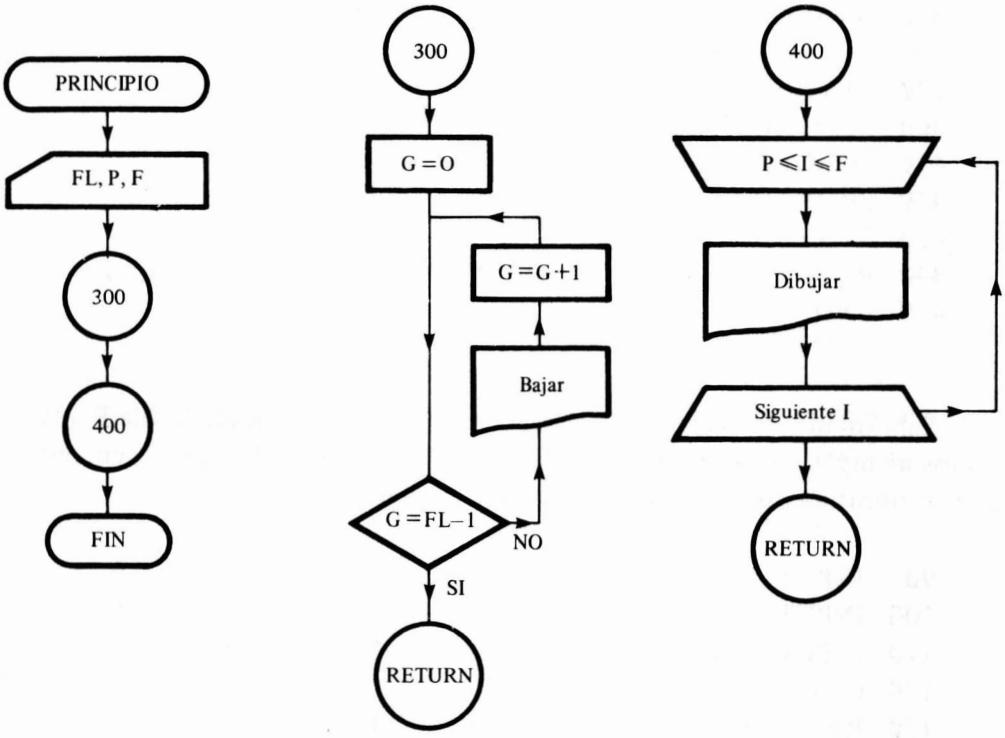


Fig. A. 3

El programa asociado a estos diagramas será:

```
1Ø   REM  SEGMENTO HORIZONTAL EN LA FILA FL
9Ø   INPUT  "FILA";FL
1ØØ  INPUT  "PRINCIPIO";P
11Ø  INPUT  "FIN";F
13Ø  (Borrado)
```

```
15Ø  REM  BAJAR A LA FILA
16Ø  GOSUB  3ØØ
18Ø  REM  DIBUJO DEL SEGMENTO HORIZONTAL
19Ø  GOSUB  4ØØ
2ØØ  GO  TO  2ØØ
3ØØ  REM  SUBRUTINA DE BAJADA
31Ø  G=Ø
32Ø  REM  SI G=FL–1 HEMOS BAJADO BASTANTE
325  IF  FL=Ø  THEN  37Ø
33Ø  IF  G=FL–1  THEN  37Ø
34Ø  PRINT
35Ø  G=G+1
36Ø  GO  TO  33Ø
37Ø  RETURN
4ØØ  REM  SUBRUTINA DEL DIBUJO
41Ø  FOR  I=P  TO  F
42Ø  PRINT  TAB(I);"*";
43Ø  NEXT  I
44Ø  RETURN
45Ø  END
```

Para dibujar un segmento vertical en la columna C, desde la fila P a la F, vamos a emplear otro programa análogo, en el que nos volveremos a encontrar con la subrutina 3ØØ, que rara vez vamos a abandonar ya.

```
9Ø   INPUT  "COLUMNA";C
1ØØ  INPUT  "PRINCIPIO";P
11Ø  INPUT  "FIN";F
12Ø  (Borrado)
13Ø  REM  BAJADA A LA FILA PRINCIPIO
14Ø  FL=P
15Ø  GOSUB  3ØØ
2ØØ  REM  DIBUJO DEL SEGMENTO VERTICAL
21Ø  GOSUB  5ØØ
22Ø  GO  TO  22Ø
3ØØ  REM  SUBRUTINA DE BAJADA
31Ø  G=Ø
32Ø  REM  SI G=FL–1 HEMOS BAJADO BASTANTE
33Ø  IF  G=FL–1  THEN  37Ø
```

```
34Ø  PRINT
35Ø  G=G + 1
36Ø  GO TO 33Ø
37Ø  RETURN
5ØØ  REM  SUBRUTINA DEL SEGMENTO VERTICAL
51Ø  FOR  I=P  TO  F
52Ø  PRINT  TAB(C);"*"
53Ø  NEXT  I
54Ø  RETURN
6ØØ  END
```

Entretenerse en dibujar segmentos puede parecer pueril, pero su dominio nos abre el camino a otros tipos de representaciones, el más simple de los cuales tal vez sea el diagrama de barras.

A.6 DIAGRAMA DE BARRAS HORIZONTALES

Representar la información, contenida en una tabla, mediante un diagrama de barras horizontales es trivial si los valores a representar están comprendidos entre 0 y 39. Nosotros vamos a pedirle al programa que, antes de representar, calcule la escala apropiada. Necesitamos un algoritmo que haga lo siguiente:

1. Escribir el encabezamiento general
2. Hacer lectura de datos
   hasta que se encuentre con "ZZZ" (testigo)
   hacer
      Hacer escritura del nombre de la fila hasta un máximo de diez caracteres
      Hacer cuenta de la cantidad de filas (N)
   Fin de Hacer
3. Hacer cálculo de la escala (subrutina 6ØØ)
4. Hacer N veces (índice I)
      Hacer representación del segmento horizontal (subrutina 7ØØ)
   Siguiente vez
   FIN

Elaboramos un diagrama de flujo apropiado:

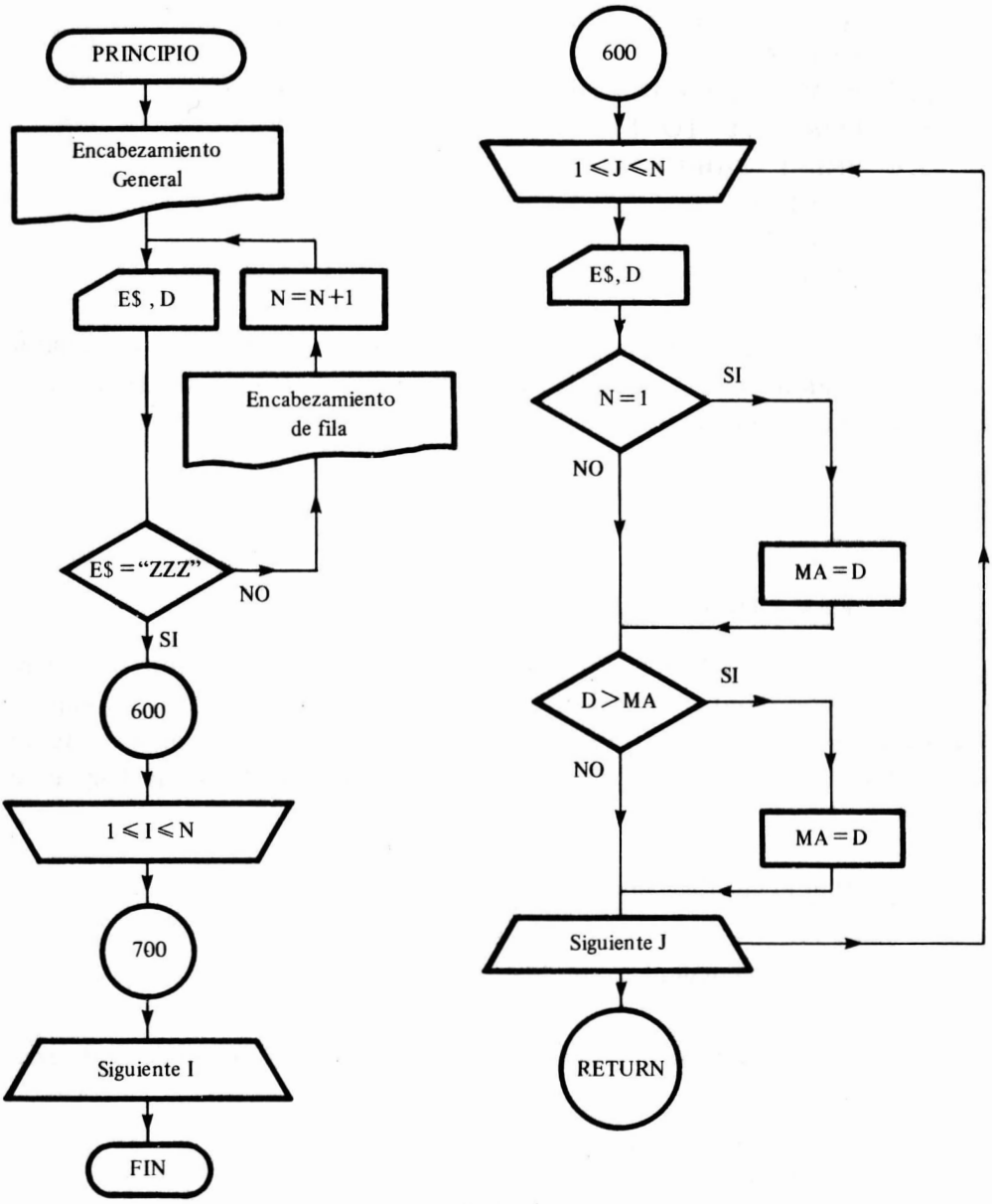


Fig. A. 4

No representamos el diagrama de flujo de la subrutina 7ØØ, que dibuja el segmento horizontal, porque su única particularidad es que no se puede utilizar la instrucción FOR–NEXT, ya que si, con el cambio de escala, algún segmento fuera nulo, no habría de ser dibujado.

El programa resultante es el siguiente:

```
2Ø   REM  DIAGRAMA DE BARRAS HORIZONTALES
3Ø   (Borrado)
4Ø   REM  ENCABEZAMIENTO
5Ø   PRINT  TAB(15);"AÑO 1975"
6Ø   PRINT  TAB(15);"- - - - -"
7Ø   PRINT
8Ø   PRINT
9Ø   REM  ENCABEZAMIENTO DE LAS BARRAS
1ØØ  READ  E$,D
11Ø  IF  E$ = "ZZZ"  THEN  16Ø
12Ø  PRINT  LEFT$(E$,1Ø)
13Ø  PRINT
14Ø  N = N + 1
15Ø  GO  TO  1ØØ
16Ø  REM  SUBRUTINA PARA CALCULO DE ESCALA
17Ø  RESTORE
18Ø  GOSUB  6ØØ
19Ø  REM  REPRESENTACION
2ØØ  (Cursor al origen)
2Ø5  PRINT
21Ø  PRINT
215  PRINT
22Ø  PRINT
23Ø  FOR  I = 1  TO  N
24Ø  GOSUB  7ØØ
25Ø  NEXT  I
26Ø  GO  TO  26Ø
6ØØ  REM  CALCULO DE LA ESCALA
6Ø5  REM  OBTENCION DEL MAXIMO
61Ø  FOR  J = 1  TO  N
62Ø  READ  E$, D
64Ø  IF  N = 1  THEN  MA = D
66Ø  IF  D > MA  THEN  MA = D
67Ø  NEXT  J
68Ø  RESTORE
69Ø  RETURN
7ØØ  REM  BARRA CORRESPONDIENTE
71Ø  REM  LEER DATO NUMERICO
```

```
72Ø   READ  E$,D
73Ø   REM  CAMBIO DE ESCALA
74Ø   D=(D*29)/MA
745   L=Ø
75Ø   REM   ¿HAY ALGO QUE REPRESENTAR?
76Ø   L=L+1
765   IF  INT(D)=Ø  THEN  81Ø
77Ø   REM  SI HAY
78Ø   PRINT  TAB(9+L);"*";
78Ø   D=D-1
8ØØ   GO  TO  75Ø
81Ø   REM  NO HAY QUE REPRESENTAR
82Ø   PRINT
83Ø   PRINT
84Ø   RETURN
1ØØØ  DATA  CINES, 8669, TEATROS, 68, TEATR. CINES, 1Ø05
1Ø1Ø  DATA  CAMP. FUTBOL, 323Ø, PL. TOROS, 35Ø, ZZZ, -1
2ØØØ  END
```

En el caso de valores muy similares en las diferentes barras, puede que estemos más interesados por las diferencias entre ellos que por su similitud. Lo que podemos hacer es utilizar una escala relativa. Igual que antes al valor máximo le asignábamos la longitud total disponible, ahora, además, al mínimo le asignaremos el cero. Se puede utilizar el programa anterior, sin más que intercalar lo necesario para localizar el valor mínimo:

```
63Ø   IF  N=1  THEN  MI=D
65Ø   IF  D<MI  THEN  MI=D
```

además hemos de rectificar la línea 74Ø para el cálculo de escala:

```
74Ø   D=( (D-MI)*29)/(MA-MI)
```

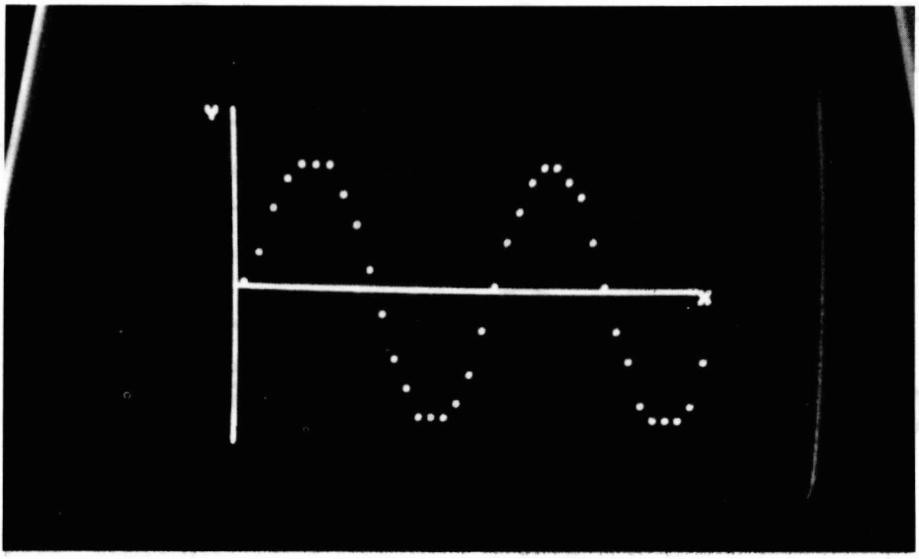


Fig. A. 5

A.7 CURVAS CON EL EJE OX HORIZONTAL

Utilizamos dos subrutinas principales, la primera para dibujar los ejes y la segunda para dibujar la curva. Como hemos optado por representar la curva en los cuadrantes de abscisa positiva, el eje vertical queda pegado al margen izquierdo de la imagen; tras dibujar este eje, el cursor sube al origen y baja hasta la fila necesaria (Subrutina 13ØØ) para dibujar el eje OX. La segunda subrutina comienza por estudiar el valor de la imagen de cada X y, si queda en los valores de pantalla, baja el cursor a la fila correspondiente y dibuja en la columna que corresponde al valor x (Ver Figs. A.5 y A.6).

El programa correspondiente será:

```
1Ø    REM  CURVA
2Ø    DEF  FNF(X)=9*SIN(X/3)
3Ø    REM  DIBUJO DE LOS EJES
4Ø    GOSUB  1ØØ
5Ø    REM  DIBUJO DE LA CURVA
6Ø    GOSUB  25Ø
9Ø    GO  TO  9Ø
1ØØ   REM  DIBUJO DE LOS EJES
11Ø   REM  EJE VERTICAL
115   (Borrado)
12Ø   REM  PARAMETROS APROPIADOS
```

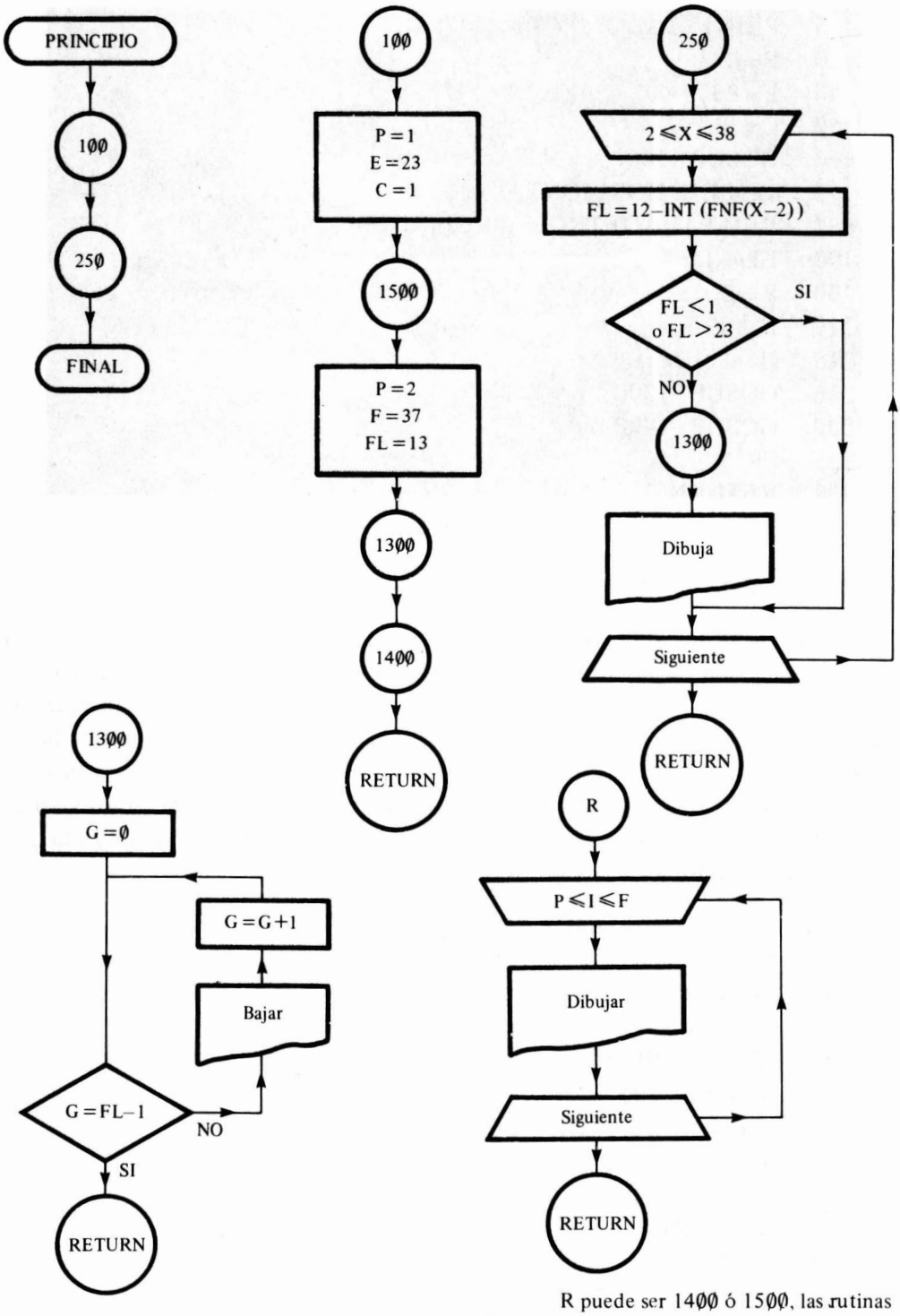


Fig. A. 6

R puede ser 1400 ó 1500, las rutinas respectivas son casi iguales.

```
125  PRINT TAB(Ø);"Y"
13Ø  P=1
14Ø  F=23
15Ø  C=1
16Ø  GOSUB 15ØØ
17Ø  REM EJE HORIZONTAL
18Ø  REM PARAMETROS APROPIADOS
19Ø  FL=13
2ØØ  P=2
21Ø  F=37
215  (Ir al origen)
216  GOSUB 13ØØ
22Ø  GOSUB 14ØØ
225  PRINT "X";
23Ø  RETURN
25Ø  REM REPRESENTACION DE LA CURVA
26Ø  FOR X=2 TO 38
265  (Ir al origen)
27Ø  FL=12-INT(FNF(X-2))
28Ø  REM ? ES EXCESIVA LA ORDENADA?
29Ø  IF (FL<1) OR (FL>23) THEN 33Ø
3ØØ  REM BAJADA
31Ø  GOSUB 13ØØ
32Ø  PRINT TAB(X);"·"
33Ø  NEXT X
34Ø  RETURN
13ØØ REM BAJADA
131Ø G=Ø
133Ø IF G=FL-1 THEN 137Ø
134Ø PRINT
135Ø G=G+1
136Ø GO TO 133Ø
137Ø RETURN
14ØØ REM
141Ø FOR I=P TO F
142Ø PRINT TAB(I);"-";
143Ø NEXT I
144Ø RETURN
15ØØ REM
15Ø5 (Ir al origen)
```

```
151Ø  FOR  I=P  TO  F
152Ø  PRINT  TAB(C);" | "
153Ø  NEXT  I
154Ø  RETURN
READY
```

A.8 NUBES DE PUNTOS CORRESPONDIENTES A DISTRIBUCIONES ESTADISTICAS BIDIMENSIONALES

Estas representaciones, como resulta fácil comprender, son muy semejantes a las anteriores; un programa que permite realizarlas es el siguiente:

```
1      REM  CAMPO DE LAS VARIABLES
2Ø     GOSUB  1ØØ
3Ø     REM  DIBUJAR EJES
4Ø     GOSUB  2ØØØ
5Ø     REM  REPRESENTAR LOS PUNTOS
6Ø     GOSUB  25ØØ
7Ø     GO  TO  7Ø
1ØØ    REM  MAXIMOS Y MINIMOS
11Ø    READ  XA, YA
12Ø    XI=XA
13Ø    YI=YA
14Ø    READ  X, Y
15Ø    IF  X<Ø  THEN  21Ø
16Ø    IF  X<XI  THEN  XI=X
17Ø    IF  X>XA  THEN  XA=X
18Ø    IF  Y<YI  THEN  YI=Y
19Ø    IF  Y>YA  THEN  YA=Y
2ØØ    GO  TO  14Ø
21Ø    RESTORE
22Ø    RETURN
13ØØ   REM RUTINA DE BAJADA
131Ø   G=Ø
1315   IF  FL=Ø  THEN  137Ø
132Ø   REM  SI G=FL–1 ESTAMOS EN LA FILA
133Ø   IF  G=FL–1  THEN  137Ø
134Ø   PRINT
135Ø   G=G+1
136Ø   GO  TO  133Ø
```

```
137Ø  RETURN
2ØØØ  REM
2Ø2Ø  (Borrado)
2Ø3Ø  PRINT TAB(Ø);MID$(STR$(YA),2,4);
2Ø4Ø  FOR I=1 TO 22
2Ø5Ø  IF I=22 THEN PRINT TAB(Ø);MID$(STR$(YI),2,4);
2Ø6Ø  PRINT TAB(4);" | "
2Ø7Ø  NEXT I
2Ø8Ø  FOR I=5 TO 38
2Ø9Ø  PRINT TAB(I);"-";
21ØØ  NEXT I
211Ø  PRINT
212Ø  PRINT TAB(5);MID$(STR$(XI),2,4);TAB(34);MID$(STR$(XA),
      2,4)
213Ø  RETURN
25ØØ  REM NUBE
25Ø5  (Ir al origen)
251Ø  READ X,Y
252Ø  IF X<Ø THEN 26ØØ
253Ø  X=( (X-XI)*34/(XA-XI) )+5
254Ø  FL=INT(21-(Y-YI)*21/(YA-YI) )
255Ø  REM BAJADA A FILA CORRECTA
256Ø  GOSUB 13ØØ
257Ø  PRINT TAB(X);"."
258Ø  GO TO 25Ø5
26ØØ  RETURN
3ØØØ  DATA 1Ø1,21Ø,113,25Ø,125,3ØØ,14Ø,31Ø,173,34Ø,18Ø,4ØØ,-1,-1
31ØØ  END
```

Animamos al lector a elaborar los diagramas de flujo correspondientes a este programa y a los restantes.

A.9 ¿COMO COMPORTARSE EN SOCIEDAD CUANDO NO SE POSEE DOMINIO DEL CURSOR?

Cuando el ordenador en que estamos trabajando no dispone de la posibilidad de mover libremente el cursor, la mejor estrategia nos parece barrer la pantalla por filas, de arriba a abajo, y en cada fila de izquierda a derecha; así podemos producir una imagen en forma análoga a como lo hace la televisión.

Esto es lo que hacemos en el programa que sigue. En él situamos el origen en el centro de la pantalla y en cada punto, (x, y), estudiamos si se cumple y = f(x); en caso afirmativo dibujamos un punto de la curva; en los lugares en que x = 0 dibujamos un segmento vertical. Lo anterior lo hacemos para los puntos de los cuadrantes superiores; luego dibujamos el eje OX y repetimos, lo ya hecho, para los cuadrantes inferiores.

Es imprescindible que los valores de f(x) sean enteros; en caso contrario no habrá representación, pues nunca será y = f(x), ya que y es entero.

```
1Ø    REM  PROGRAMA PRINCIPAL
2Ø    REM  CUADRANTES SUPERIORES
3Ø    DEF  FNF(X) = INT (0·25*(X12+3*X-40) )
5Ø    P = 11
6Ø    F = 1
7Ø    GOSUB  1ØØØ
8Ø    REM  EJE  HORIZONTAL
9Ø    GOSUB  2ØØØ
95    PRINT
1ØØ   REM  CUADRANTES INFERIORES
11Ø   P = -1
12Ø   F = -11
13Ø   GOSUB  1ØØØ
14Ø   GOTO  14Ø
1ØØØ  REM  CUADRANTES  SUPERIORES
1Ø1Ø  REM  PARA CADA FILA
1Ø2Ø  REM  Y=P  TO  F  STEP  -1
1Ø3Ø  REM  PARA CADA ELEMENTO DE ELLA
1Ø4Ø  A=  -19
1Ø5Ø  B=  -1
1Ø6Ø  REM  ¿PASA POR EL LA CURVA?
1Ø7Ø  GOSUB  5ØØØ
1Ø8Ø  REM  ¿CORTA AL EJE?
1Ø9Ø  REM  SI
11ØØ  IF  FNF(Ø)=Y  THEN PRINT  TAB(19); "*";
111Ø  REM  NO
112Ø  IF  FNF(Ø)<>Y  THEN PRINT  TAB(19); "!";
113Ø  REM  PARA LAS X POSITIVAS
114Ø  A=1
115Ø  B=19
116Ø  GOSUB  5ØØØ
```

```
117Ø  REM  SIGUIENTE FILA
118Ø  PRINT
119Ø  NEXT  Y
12ØØ  RETURN
2ØØØ  REM  EJE DE ABSCISAS
2Ø1Ø  REM  SEMIEJE NEGATIVO
2Ø2Ø  A= -19
2Ø3Ø  B= -1
2Ø4Ø  GOSUB  3ØØØ
2Ø5Ø  REM  ORIGEN
2Ø6Ø  GOSUB  4ØØØ
2Ø7Ø  REM  SEMIEJE POSITIVO
2Ø8Ø  A=1
2Ø9Ø  B=19
21ØØ  GOSUB  3ØØØ
211Ø  RETURN
3ØØØ  REM  ¿DONDE HAY CURVA?
3Ø1Ø  FOR  X=A  TO  B
3Ø2Ø  IF  FNF(X)=Ø THEN PRINT  TAB(X+19); "*";
3Ø3Ø  IF  FNF(X)<>Ø  THEN PRINT  TAB(X+19); "-";
3Ø4Ø  NEXT  X
3Ø5Ø  RETURN
4ØØØ  REM  ?CORTA AL ORIGEN?
4Ø1Ø  IF  FNF(Ø)=Ø  THEN PRINT  TAB(19); "*";
4Ø2Ø  IF  FNF(Ø)<>Ø  THEN PRINT  TAB(19); "+";
4Ø3Ø  RETURN
5ØØØ  REM  ¿PASA POR UN PUNTO?
5Ø1Ø  FOR  X=A  TO  B
5Ø2Ø  REM  EN CASO AFIRMATIVO, SEÑALARLO
5Ø3Ø  IF  FNF(X)=Y  THEN PRINT  TAB(X+19); "*";
5Ø4Ø  NEXT  X
5Ø5Ø  RETURN
51ØØ  END
```

A.10 REPRESENTACION DE SUPERFICIES

Una manera sencilla de representar superficies es por sus curvas de nivel, pero esto sólo es posible, por lo general, cuando se dispone de alta resolución. El método más cercano, en baja resolución, es la representación de la cota de cada punto. Eso hemos hecho en el programa que sigue, que en lo demás conti-

núa el sistema de barrido que ya hemos presentado.

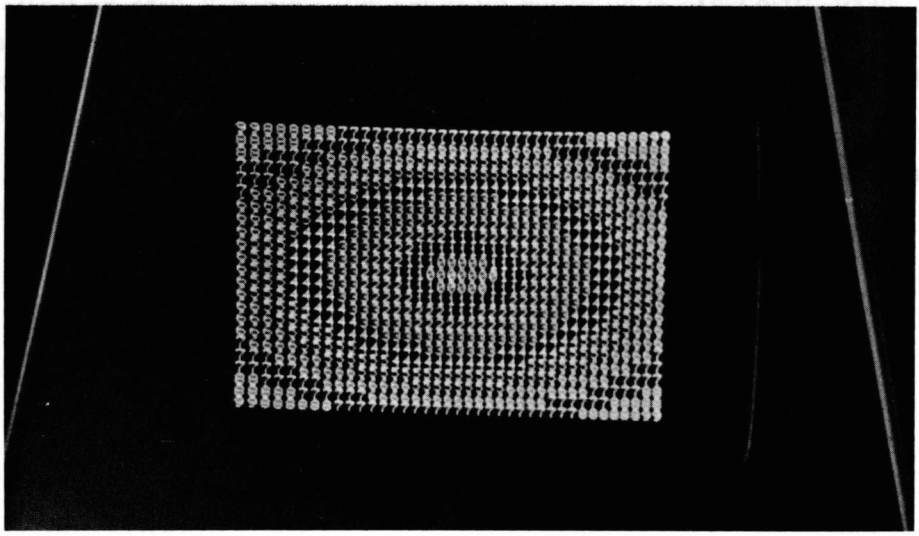


Fig. A. 7

```
1Ø   (Borrado)
2Ø   REM  PARA TODAS LAS FILAS
3Ø   FOR  Y=11  TO  -11  STEP -1
9Ø   REM  PARA CADA LUGAR DE ESA FILA
1ØØ  FOR  X=-19  TO  19
11Ø  REM  VALOR DE LA FUNCION EN EL PUNTO
12Ø  Z=SQR(Ø·8*X↑2+2*Y↑2)/3
13Ø  REM  REPRESENTARLO
14Ø  PRINT TAB(X+19);MID$(STR$(INT(Z) ), 2, 1);
15Ø  NEXT  X
16Ø  NEXT  Y
17Ø  GOTO  17Ø
18Ø  END
```

Si no deseamos una representación numérica, sino gráfica, podemos sustituir la línea 14Ø por

```
135   IF  Z > 5  THEN  Z = 5
14Ø   ON  Z  GOSUB  3ØØ, 35Ø, 4ØØ, 45Ø, 5ØØ
145   PRINT  TAB(X + 19 );A$;
```

añadiendo, además, las subrutinas:

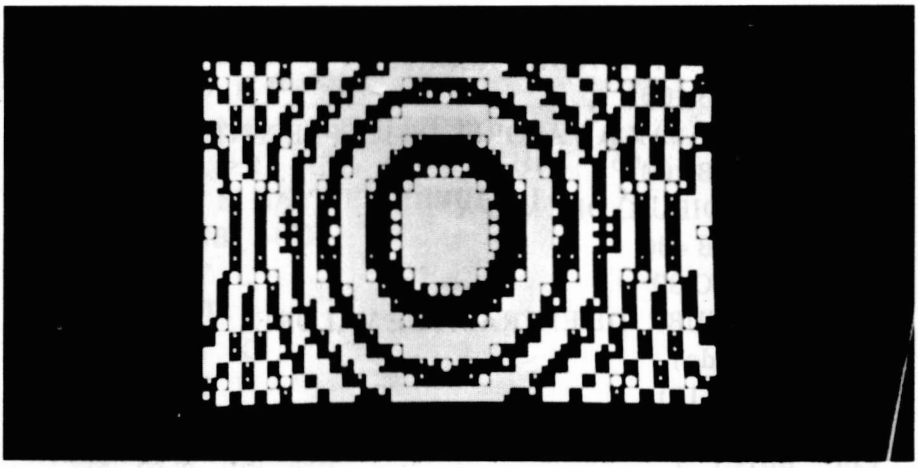


Fig. A. 8

```
3ØØ   A$ = "  "
31Ø   RETURN
35Ø   A$ = " ."
36Ø   RETURN
4ØØ   A$ = "•"
41Ø   RETURN
45Ø   A$ = "•"
46Ø   RETURN
5ØØ   A$ = "•"
51Ø   RETURN
```

A.11 MAPAS

El mapa de España consta de lo que, geométricamente hablando, serían 17 regiones disjuntas. Hemos superpuesto a este mapa un acetato cuadriculado con 25 filas y 40 columnas y ha resultado que, por ejemplo, Andalucía está situada en varias filas (ver líneas 3ØØØ a 3Ø18) la primera de ellas la 15; además, estas filas son 8, y en la primera presenta dos segmentos disjuntos, luego necesitamos cuatro valores, dos por segmento (su origen y extremo). Atendiendo a estas características, hemos construido una tabla para cada región utilizando sentencias DATA. Con estas tablas podemos representar la distribución de los valores de una variable. Estos valores los asignamos en las líneas 13Ø a 16Ø; en las líneas 2ØØ a 295 hacemos un cambio de escala para que los valores extremos sean 0 y 9.

```
1Ø    REM MAPA
2Ø    REM ENTRADA DE VALORES
3Ø    GOSUB 1ØØ
4Ø    REM PASAR A VALORES DE REPRESENTACION
5Ø    GOSUB 2ØØ
6Ø    REM DIBUJO DEL MAPA
7Ø    GOSUB 4ØØ
8Ø    GO  TO 8Ø
1ØØ   REM SUBR. DE ENTRADA DE VALORES
11Ø   (Borrado)
12Ø   DIM L(17)
13Ø   FOR  I=1  TO  17
14Ø   PRINT  I
15Ø   INPUT  L(I)
16Ø   NEXT  I
17Ø   RETURN
2ØØ   REM SUBR. DE CAMBIAO DE ESCALA
2Ø5   REM VALOR PROVISIONAL DEL MAXIMO
21Ø   MA=L(1)
22Ø   REM  BUSQUEDA DEL MAXIMO DEFINITIVO
23Ø   FOR  I=2  TO  17
24Ø   IF  L(I)>MA  THEN  MA=L(I)
25Ø   NEXT  I
26Ø   REM  CAMBIO DE ESCALA
27Ø   FOR  I=1  TO  17
28Ø   L(I)=9.9*L(I)/MA
29Ø   NEXT  I
295   RETURN
4ØØ   REM  SUBR. DE DIBUJO DE MAPA
41Ø   REM  PARA CADA ZONA
415   (Borrado)
42Ø   FOR  J=1  TO  17
425   (Cursor al origen)
43Ø   REM LEER FILA DE COMIENZO, Y NO. DE FILAS Y COLUM-
      NAS
44Ø   READ  FL, NF, NC
45Ø   REM  BAJAR A ESA FILA [FL]
46Ø   GOSUB 13ØØ
47Ø   REM  PARA CADA FILA
48Ø   FOR  K=1  TO  NF
```

```
49Ø   REM  LEER PRINC. Y FIN, DE 2 EN 2
5ØØ   FOR  L=1  TO  NC-1 STEP 2
51Ø   READ  P, F
52Ø   REM  ?HAY SEGMENTO?
53Ø   IF  (L>1)  AND  (P=Ø)  THEN  56Ø
54Ø   REM  SI HAY. IR A DIBUJARLO
55Ø   GOSUB  14ØØ
56Ø   NEXT  L
57Ø   PRINT
58Ø   NEXT  K
6ØØ   NEXT  J
61Ø   RETURN
13ØØ  REM  BAJADA
13Ø5  IF  FL=Ø  THEN  137Ø
131Ø  G=Ø
132Ø  REM  SI G=FL-1  HEMOS BAJADO BASTANTE
133Ø  IF  G=FL-1  THEN  137Ø
134Ø  PRINT
135Ø  G=G+1
136Ø  GO  TO  133Ø
137Ø  RETURN
14ØØ  REM  SUBR. DE SEGMENTO HORIZONTAL
141Ø  FOR  I=P  TO  F
142Ø  REM IMPRIMIMOS EL NO. APROPIADO EN EL LUGAR CO-
      RRECTO
143Ø  PRINT  TAB(I);MID$(STR$(L(J) ),2,1);
144Ø  NEXT  I
145Ø  RETURN
3ØØØ  REM ANDALUCIA
3ØØ2  DATA  15, 8, 4
3ØØ4  DATA  9, 1Ø, 13, 16
3ØØ6  DATA  8, 17, Ø, Ø
3ØØ8  DATA  4, 18, Ø, Ø
3Ø1Ø  DATA  3, 18, Ø, Ø
3Ø12  DATA  5, 18, Ø, Ø
3Ø14  DATA  6, 11, Ø, Ø
3Ø16  DATA  7, 9, Ø, Ø
3Ø18  DATA  8, 9, Ø, Ø
3Ø4Ø  REM ARAGON
3Ø42  DATA  2, 9, 2
```

```
3Ø44 DATA 19, 23
3Ø46 DATA 19, 23
3Ø48 DATA 19, 23
3Ø5Ø DATA 17, 22
3Ø52 DATA 17, 23
3Ø54 DATA 18, 22
3Ø56 DATA 19, 21
3Ø58 DATA 18, 21
3Ø6Ø DATA 19, 19
3Ø7Ø REM ASTURIAS
3Ø72 DATA Ø, 3, 2
3Ø74 DATA 4, 9
3Ø76 DATA 5,8
3Ø78 DATA 5, 5
3Ø9Ø REM BALEARES
3Ø92 DATA 8, 5, 2
3Ø94 DATA 35, 35
3Ø96 DATA 32, 32
3Ø98 DATA 31, 33
31ØØ DATA 32, 32
31Ø2 DATA 28, 28
311Ø REM CANARIAS
3112 DATA 17, 6, 6
3114 DATA 37, 37 Ø, Ø, Ø, Ø
3116 DATA 36, 36, Ø, Ø, Ø, Ø
3118 DATA 24, 24, 36, 36, Ø, Ø
312Ø DATA 28, 29, Ø, Ø, Ø, Ø
3122 DATA 26, 26, 28, 29, 32, 33
3124 DATA 24, 24, 32, 33, Ø, Ø
314Ø REM CANTABRIA
3142 DATA Ø, 3, 2
3144 DATA 1Ø, 13
3146 DATA 1Ø, 11
3148 DATA 11, 11
316Ø REM CASTILLA–LEON
3162 DATA 2, 1Ø, 4
3164 DATA 9, 9, 12, 13
3166 DATA 6, 1Ø, 12, 13
3168 DATA 5, 13, Ø, Ø
317Ø DATA 5, 16, Ø, Ø
```

```
3172 DATA 5, 16, Ø, Ø
3174 DATA 6, 16, Ø, Ø
3176 DATA 5, 12, Ø, Ø
3178 DATA 5, 11, Ø, Ø
3180 DATA 5, 10, Ø, Ø
3182 DATA 8, 9, Ø, Ø
3190 REM CATALUÑA
3192 DATA 2, 6, 4
3194 DATA 24, 25, 29, 31
3196 DATA 24, 30, Ø, Ø
3198 DATA 24, 29, Ø, Ø
3200 DATA 23, 28, Ø, Ø
3202 DATA 24, 26, Ø, Ø
3204 DATA 23, 24, Ø, Ø
3220 REM EUSKADI
3222 DATA Ø, 3, 2
3224 DATA 14, 17
3226 DATA 14, 16
3228 DATA 14, 15
3240 REM EXTREMADURA
3242 DATA 10, 7, 2
3244 DATA 5, 7
3246 DATA 5, 8
3248 DATA 4, 8
3250 DATA 4, 10
3252 DATA 5, 9
3254 DATA 4, 8
3256 DATA 5, 7
3270 REM GALICIA
3272 DATA Ø, 6, 4
3274 DATA 2, 3, Ø, Ø
3276 DATA Ø, 4, Ø, Ø
3278 DATA Ø, 4, Ø, Ø
3280 DATA Ø, 4, Ø, Ø
3282 DATA Ø, 4, Ø, Ø
3284 DATA Ø, Ø, 2, 3
3300 REM MADRID
3302 DATA 7, 3, 2
3304 DATA 12, 13
3306 DATA 12, 14
```

```
3308  DATA  11, 14
3320  REM MANCHA [CASTILLA-LA]
3322  DATA  7, 9, 4
3324  DATA  14, 17, 0, 0
3326  DATA  15, 18, 0, 0
3328  DATA  15, 17, 0, 0
3330  DATA  10, 18, 0, 0
3332  DATA  9, 18, 0, 0
3334  DATA  9, 19, 0, 0
3336  DATA  11, 20, 0, 0
3338  DATA  10, 19, 0, 0
3340  DATA  11, 12, 17, 17
3350  REM MURCIA
3352  DATA  14, 4, 2
3354  DATA  20, 20
3356  DATA  18, 20
3358  DATA  18, 21
3360  DATA  19, 20
3370  REM NAVARRA
3372  DATA  1, 4, 2
3374  DATA  17, 18
3376  DATA  16, 18
3378  DATA  17, 18
3380  DATA  18, 18
3390  REM RIOJA
3392  DATA  3, 2, 2
3394  DATA  14, 16
3396  DATA  14, 17
3410  REM VALENCIA
3412  DATA  8, 8, 2
3414  DATA  22, 23
3416  DATA  22, 23
3418  DATA  20, 22
3420  DATA  19, 22
3422  DATA  20, 22
3424  DATA  21, 23
3436  DATA  21, 23
3448  DATA  21, 22
4000  END
```

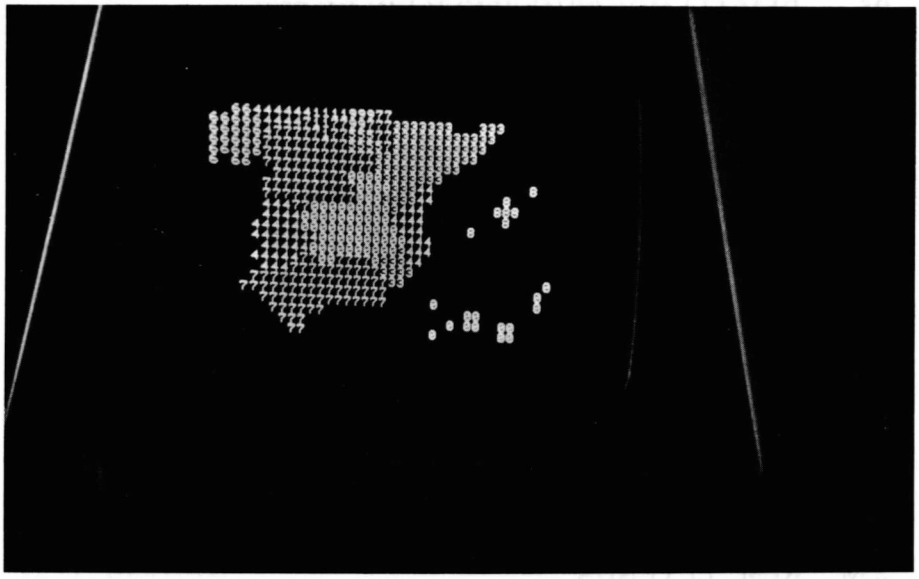


Fig. A. 9

A.12 GRAFICOS ANIMADOS

En ocasiones puede ser conveniente utilizar un gráfico con movimiento. Hay dos posibilidades: que el operador mueva la imagen o que lo haga el ordenador.

Hemos preparado dos ejemplos, ambos relacionados con el laberinto de Creta. En el primero es el operador quien mueve la imagen; el ordenador, tras dibujar el laberinto, proporciona instrucciones para guiar a Teseo. La mayor parte del programa se dedica al laberinto, que se dibuja utilizando segmentos horizontales y verticales: son las líneas 5 a 24Ø, 13ØØ a 137Ø (rutina de bajada ya conocida); 14ØØ a 154Ø, que dibujan el laberinto con los datos almacenados en las líneas 2ØØØ a 2126. Las líneas 25Ø a 31Ø son de instrucciones, y la línea clave es la 43Ø, que decide la dirección de Teseo en cada paso.

```
5     REM
1Ø    (Borrado)
2Ø    FOR J=1 TO 2
3Ø    REM  LEEMOS NO. DE FILAS Y DE COLUMNAS
4Ø    READ  NF, NC
5Ø    REM  PARA CADA FILA
6Ø    FOR  K=1  TO  NF
7Ø    REM  LEEMOS EL TERMINO DESTACADO
8Ø    READ  TD
```

```
9Ø    FOR L=1 TO NC-1 STEP 2
95    REM LEEMOS PRINCIPIO Y FIN DE SEG.
1ØØ   READ P, F
11Ø   REM SI P=Ø NO HAY SEGMENTO
12Ø   IF P=Ø THEN 2ØØ
13Ø   REM BAJADA
14Ø   IF J=1 THEN FL=TD
15Ø   IF J=2 THEN FL=P
16Ø   GOSUB 13ØØ
17Ø   REM DIBUJO DE SEGMENTO APROPIADO
18Ø   IF J=2 THEN C=TD
19Ø   ON J GOSUB 14ØØ, 15ØØ
195   (Curso al origen)
2ØØ   NEXT L
22Ø   NEXT K
24Ø   NEXT J
25Ø   REM LEYENDA
26Ø   FL=18
27Ø   GOSUB B 13ØØ
28Ø   PRINT "TESEO VA HACIA EL NORTE CON... 1"
29Ø   PRINT "                EL ESTE CON ..... 2"
3ØØ   PRINT "                EL SUR CON..... 3"
31Ø   PRINT "                EL OESTE CON.... 4"
33Ø   REM SITUAMOS A TESEO
34Ø   (Cursor al origen)
35Ø   FL=16
36Ø   GOSUB 13ØØ
37Ø   PRINT TAB(27);".";
4ØØ   REM MOVIMIENTOS DE TESEO
41Ø   GET T$
42Ø   IF T$=" " THEN 41Ø
43Ø   ON VAL(T$) GOSUB 16ØØ, 165Ø, 17ØØ, 175Ø
44Ø   GO TO 41Ø
13ØØ  REM BAJADA
131Ø  G=Ø
132Ø  REM SI G=FL-1 HEMOS BAJADO BASTANTE
133Ø  IF G=FL-1 THEN 137Ø
134Ø  PRINT
135Ø  G=G+1
136Ø  GO TO 133Ø
137Ø  RETURN
```

```
1400  REM  LINEAS HORIZONTALES
1410  FOR  I=P  TO  F
1420  PRINT  TAB(I);"■";
1430  NEXT  I
1440  RETURN
1500  REM  LINEAS VERTICALES
1510  FOR  I=P  TO  F
1520  PRINT  TAB (C);"■"
1530  NEXT  I
1540  RETURN
1600  REM  NORTE
1610  (Cursor a la izquierda, cursos arriba);".";
1620  RETURN
1650  REM  ESTE
1660  PRINT  ".";
1670  RETURN
1700  REM  SUR
1710  (Cursor a la izquierda, cursor abajo);".";
1720  RETURN
1750  REM  OESTE
1760  (Cursor a la izquierda, cursor a la izquierda);".";
1770  RETURN
2000  REM  DATOS DE LAS HORIZONTALES
2002  DATA  9, 6
2004  DATA  1, 11, 29, 0, 0, 0, 0
2006  DATA  3, 17, 20, 22, 25, 0, 0
2008  DATA  5, 17, 21, 23, 27, 0, 0
2010  DATA  7, 13, 15, 17, 19, 21, 23
2012  DATA  9, 19, 21, 23, 27, 0, 0
2014  DATA  11, 15, 17, 19, 20, 22, 24
2016  DATA  12, 24, 24, 26, 28, 0, 0
2018  DATA  13, 17, 20, 0, 0, 0, 0
2020  DATA  14, 11, 26, 28, 29, 0, 0
2100  REM  DATOS DE LAS VERTICALES
2102  DATA  12, 4
2104  DATA  11, 1, 14, 0, 0
2106  DATA  13, 7, 12, 0, 0
2108  DATA  15, 8, 9, 11, 13
2110  DATA  17, 4, 4, 8, 10
2112  DATA  19, 8, 8, 0, 0
```

```
2114 DATA 2Ø, 1Ø, 1Ø, Ø, Ø
2116 DATA 21, 6, 8, Ø, Ø
2118 DATA 22, 12, 13, Ø, Ø
212Ø DATA 25, 4, 8, Ø, Ø
2122 DATA 26, 1Ø, 11, Ø, Ø
2124 DATA 27, 3, 4, 7, 1Ø
2126 DATA 29, 1, 14, Ø, Ø
3ØØØ END
```

Para el movimiento de la imagen por el ordenador, hemos optado por representar, en el mismo laberinto, al Minotauro. Sustituimos las líneas 25Ø a 45Ø y 16ØØ a 177Ø por las siguientes:

```
3ØØ  REM MINOTAURO
31Ø  (Cursor al origen)
32Ø  REM SITUACION INICIAL
33Ø  FL=4
34Ø  GOSUB 13ØØ
345  C=14
35Ø  F=4
36Ø  PRINT TAB(14);"*"
37Ø  (Cursor al origen)
38Ø  PRINT
389  FOR I=1 TO 2ØØ
39Ø  S=1+RND(7)*4
4ØØ  ON S GOSUB 8ØØ, 82Ø, 84Ø, 86Ø
41Ø  FOR I=1 TO 5
42Ø  IF F<>I THEN 48Ø
43Ø  FOR J=12 TO 16
44Ø  IF J=C THEN PRINT TAB(J);"*";
45Ø  IF J<>C THEN PRINT TAB(J); "      ";
46Ø  NEXT J
465  PRINT
47Ø  GO TO 49Ø
48Ø  PRINT TAB(12);"      "
49Ø  NEXT I
5ØØ  GO TO 37Ø
8ØØ  IF F<>1 THEN F=F-1
81Ø  RETURN
```

```
82Ø   IF  C<>16  THEN  C=C+1
83Ø   RETURN
84Ø   IF  F<>5  THEN  F=F+1
85Ø   RETURN
86Ø   IF  C<>12  THEN  C=C-1
87Ø   RETURN
```

En las líneas 39Ø y 4ØØ el ordenador decide hacia dónde mover la imagen, pero sólo lo hace si las paredes lo permiten, subrutinas 8ØØ a 87Ø.

Indice alfabético