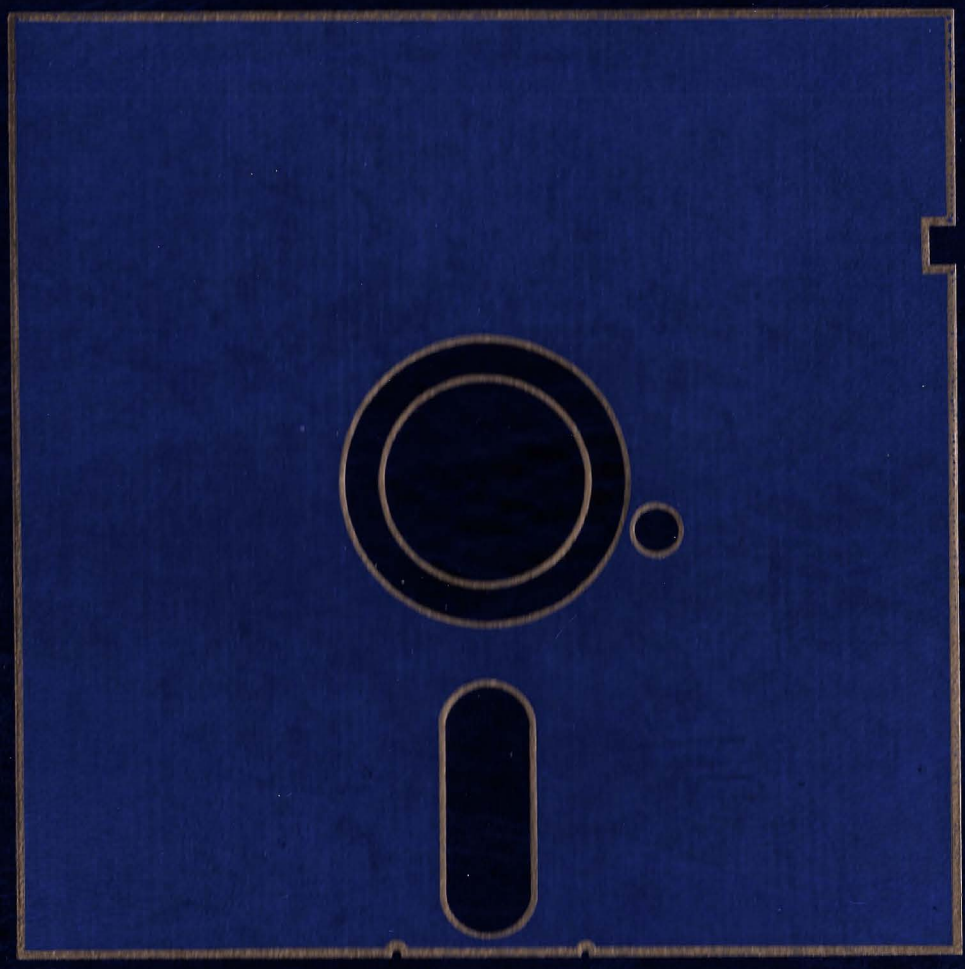
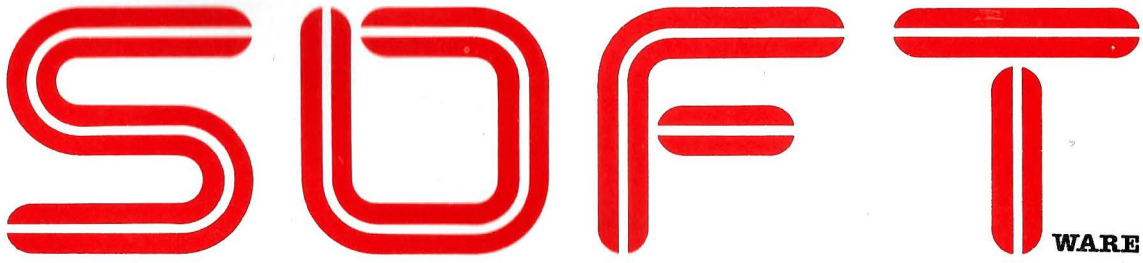
SOFTWARE





Nueva Lente/Ingelek

una publicación

EDICIONES NUEVA LENTE, S. A.,
y EDICIONES INGELEK, S. A.

Director-Editor:
Por NUEVA LENTE, S. A.:
MIGUEL J. GOÑI
Por INGELEK, S. A.:
ANTONIO M. FERRER
Director de producción:
RICARDO ESPAÑOL
Jefe de producción:
SANTOS ROBLES

Director de la obra:
FRANCISCO LARA (PL/3)
Colaboradores:
MANUEL MUÑOZ
DAVID SANTAOLALLA
SANTIAGO RUIZ
LUIS COCA
MIGUEL ANGEL VILA
MIGUEL ANGEL SANCHEZ
VICENTE ROBLES

Diseño e ilustración:
JOSE OCHOA (PL/3)
Maquetación:
JUAN JOSE DIAZ SANCHEZ
Fotografía:
(Equipo Gálata)
EDUARDO AGUDELO y
ALBINO LOPEZ
Redaccion:
PL/3 calc

Dirección, Redacción y
Administración:
Benito Castro, 12. 28028 Madrid
Tels. 245 45 98 y 246 73 67

ISBN, tomo primero: 84-7534-103-9
ISBN, tomo segundo: 84-7534-104-7
ISBN, tomo tercero: 84-7534-105-5
ISBN, tomo cuarto: 84-7534-106-3
ISBN de la obra: 84-7534-101-2

Fotomecánica:
OCHOA, S. A.
Ricardo Ortiz, 74. 28017 Madrid
Impresión:
GRAFICAS REUNIDAS, S. A.
Avda. de Aragón, 56. 28027 Madrid
Depósito legal: M. 41.955-1984
PRINTED IN SPAIN

Macrolenguajes gráficos

Otras formas de manejar los gráficos

En la exposición hecha hasta ahora sobre los gráficos se ha hablado de los comandos más habituales en BASIC. Sin embargo, en las tablas de compatibilización de los capítulos precedentes se observan importantes huecos. Ello no significa que los correspondientes aparatos carezcan de instrucciones para el manejo de gráficos. Lo que ocurre es que utilizan otros métodos para ese tratamiento.

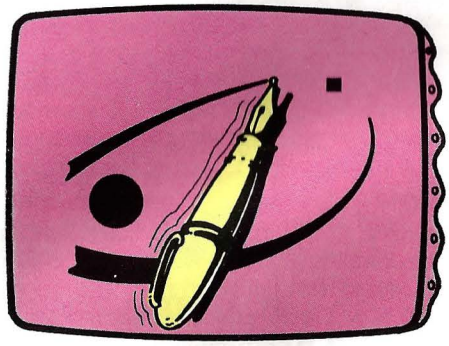
OTROS METODOS

En los microordenadores más "grandes", los denominados profesionales, se acostumbra a utilizar un monitor monócromo como pantalla de representación, y, con ello, el empleo de gráficos queda muy restringido. En muchos casos, la pantalla se contempla como un periférico más. Esto lleva a la necesidad de entablar un diálogo CPU-periférico complejo. El manejo de la pantalla como un "terminal externo" obliga al uso de códigos de control, enviados a través de un canal que se emplea como si se tratara de un archivo secuencial. De esta forma, cada efecto se ha de codificar en un complejo lenguaje de códigos.

En otros aparatos se emplea un lenguaje parecido, pero al que se accede directamente desde el BASIC. Estos últimos son los denominados "macrolenguajes gráficos".

Un macrolenguaje viene a ser un sublenguaje de comandos asociado a una palabra clave del BASIC. Los macrolenguajes gráficos suelen emplear comandos para el desplazamiento del cursor de gráficos. Con esos desplazamientos se consigue definir el dibujo, por el método de recorrerlo. Algo muy similar a la técnica empleada en el denominado Turtle Graphicas del Logo (véanse los capítulos dedicados a este lenguaje).

De los diferentes macrolenguajes gráficos, se comenta en estas páginas el propio del dialecto BASIC de Microsoft, por ser el más ampliamente utilizado.

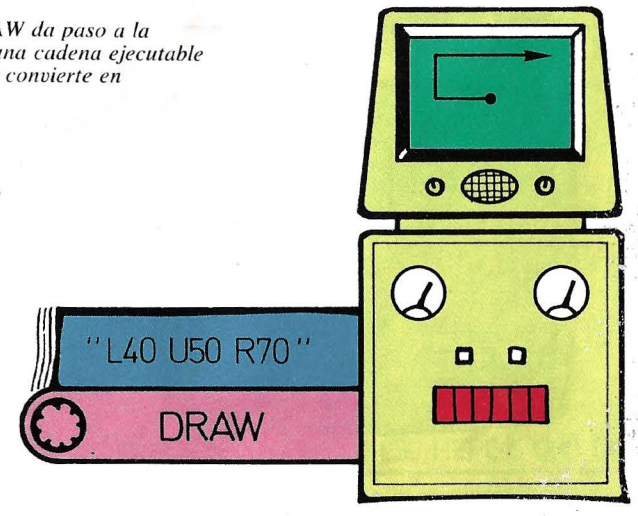


El macrolenguaje gráfico: una forma de hablar con el ordenador de temas pictóricos.

LA FILOSOFIA DEL MACROLENGUAJE

En el BASIC de Microsoft se hace uso de un macrolenguaje gráfico asociado a la palabra reservada DRAW. Este comando es el que nos introduce en el nuevo lenguaje de comandos. En cierto modo guarda una ligera relación con el otro comando DRAW, el comentado en el primer capítulo de gráficos. Aquél trazaba una recta desde el último punto referenciado hasta el indicado mediante sus coordenadas relativas. Es decir, una vez situadas en un punto se saltaba a otro indicando el incremento que debían sufrir ambas coordenadas. Para ello se empleaban dos enteros que especificaban dicho incremento. El nuevo comando DRAW también toma como base el último punto trazado. En su argumento se indica la lista de instruccio-

El comando DRAW da paso a la introducción de una cadena ejecutable que el ordenador convierte en gráficos.



nes del macrolenguaje que se desean ejecutar.
Así pues, el empleo del macrolenguaje gráfico se facilita por medio de DRAW seguido de una lista de instrucciones. La lista toma el aspecto de una cadena alfanumérica. De esta forma, la ejecución de DRAW se limita a la evaluación de la cadena que le sigue. Como es natural, la cadena ha de seguir unas reglas, definidas por el macrolenguaje.
El empleo de una cadena ejecutable proporciona una gran flexibilidad al macrolenguaje. Esa cadena no sólo puede ser almacenada en una variable dispuesta al efecto, sino que puede ser tratada como cualquier otra cadena caracteres (mediante LEFT$, RIGHT$, MID$, etc.). Dicha característica permite crear programas que construyan sus propios gráficos.

EL PRIMER PASO

Ya se ha comentado que el comando DRAW utiliza como punto de partida el último punto trazado. Para iniciar un dibujo se puede marcar ese punto inicial por medio del comando PSET.
Una vez posicionado el cursor de gráficosse trazarán líneas con DRAW. Para ello se emplean cuatro letras del macrolenguaje: U(up) arriba, D(down) abajo, L(left) izquierda y R(right) derecha. Estas son las cuatro direcciones principales de desplazamiento del macrolenguaje. A continuación de cada comando-letra se ha de especificar la magnitud del movimiento. Dicha magnitud se mide en pixels y será, por lo tanto, un número entero. He aquí el primer programa que utiliza este macrolenguaje:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "R50D50L50U50"
40 GOTO 40
```

En este ejemplo, tras fijar el punto 100,100, se ha desplazado el cursor 50 pixels a la derecha, 50 abajo, 50 a la izquierda y 50 arriba. Todo ello completa un cuadrado de lado 50 cuyo vértice superior derecho se encuentra en el punto 100,100. Como puede apreciarse en el listado, las cuatro órdenes de la cadena ejecutable no se separan entre sí, van contiguas.
Este programa podría complicarse un poco más empleando los comandos de tratamiento de cadenas. El siguiente ejemplo permite trazar un cuadrado de lado variable:

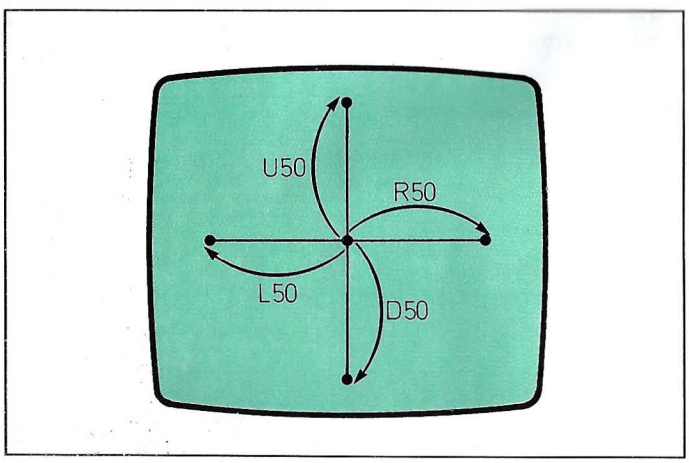
```
 5 INPUT A
10 SCREEN 2
20 PSET (100,100)
30 A$=STR$(A)
40 C$="R"+A$+"D"+A$+"L"+A$+"U"+A$
50 DRAW C$
60 GOTO 60
```

En la línea número cinco se recoge la longitud del lado, introducida por el operador. La línea 30 se encarga de pasar ese dato numérico a cadenas de caracteres. La cadena ejecutable se forma por la concatenación de sus partes en la línea 40. Por fin, la línea 50 efectúa la ejecución de la cadena recién creada.
Podrían haberse recogido diferentes valores para la longitud de cada lado. Con ello no se hubiese trazado un cuadrado sino cualquier otra figura. Para ello hubieran hecho falta cuatro variables, una para cada lado.

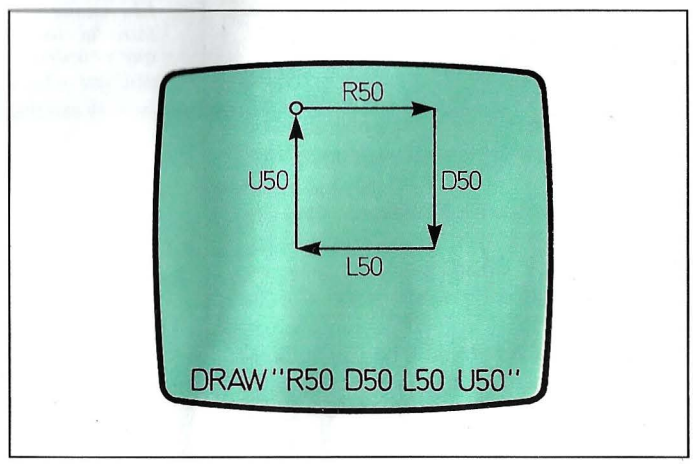
TRAZANDO DIAGONALES

Además de las direcciones horizontal y vertical se pueden emplear las que forman un ángulo de 45 grados con éstas. Las nuevas direcciones funcionan de la misma forma que las anteriores. Sus letras identificativas son: E arriba a la derecha, F abajo a la derecha, G abajo a la izquierda y H arriba a la izquierda. Si las anteriores se identificaban por coincidir con las iniciales de la palabra ingesa, éstas se pueden recordar por marcar su orden el movimiento de las agujas del reloj. Empleando estas nuevas instrucciones se puede trazar un cuadrado apoyado sobre un vértice:

```
10 SCREEN 2
20 PSET (100,100)
```



Los cuatro comandos principales del macrolenguaje gráfico permiten el movimiento vertical y horizontal.



Trazado de un cuadrado mediante una sencilla cadena de instrucciones.

```
30 DRAW "E50F50G50H50"
40 GOTO 40
```

También se puede hacer uso del manejo de cadenas para determinar la longitud del lado.

```
 5 INPUT A
10 SCREEN 2
20 PSET (100,100)
30 A$=STR$(A)
40 C$="E"+A$+"F"+A$+"G+A$+"H"+A$
50 DRAW C$
60 GOTO 60
```

Teniendo acceso a esas ocho direcciones se pueden cfear figuras relativamente complejas. Como ejemplo se muestra un pequeño programa que traza el contorno de una pajarita de papel:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "R30D30F30L30H15G15U30E15H15"
40 GOTO 40
```

La manipulación de cadenas permite reproducir efectos variados sobre los gráficos definidos. El siguiente programa representa el gráfico producido por una cadena dada y la imagen espectacular de éste:

```
 5 INPUT C$
10 SCREEN 2
20 PSET (100,100)
30 DRAW C$
40 A$=""
50 FOR I=1 TO LEN(C$)
60 B$=MID$(C$,I,1)
70 F$=B$
80 IF B$="R"THEN F$="L"
90 IF B$="L"THEN F$="R"
100 IF B$="E"THEN F$="H"
110 IF B$="F"THEN F$="G"
120 IF B$="G"THEN F$="F"
130 IF B$="H"THEN F$="E"
140 A$=A$+F$
150 NEXT I
160 DRAW A$
170 GOTO 170
```

En este ejemplo, el bucle FOR de las líneas 50 a 150 recorre todos los caracteres de la cadena C$. Las instrucciones IF de las líneas 80-130 cambian los trazos que van hacia la derecha para que vayan hacia la izquierda y viceversa. Para ello se emplea la variable intermedia F$. Esta variable contiene el valor del carácter, y su contenido se varía si dicho carácter proporciona un movimiento en la dirección horizontal. La línea 140 reconstruye la cadena A$ a partir de C$ y las modificaciones efectuadas. Por último, la línea 160 ejecuta la cadena calculada.

MOVIMIENTO HACIA UN PUNTO ABSOLUTO

El movimiento visto hasta ahora permite ocho direcciones de desplazamiento. En la especificación de dicho movimiento se incluye el salto en pixels. Ello implica un movimiento relativo al último punto trazado. También puede indicarse el desplazamiento por las coordenadas absolutas del punto final del trazo. Ello se consigue con la letra M (de mover), a la que habrán de adjuntarse dos datos: los correspondientes a las coordenadas absolutas del punto en cuestión. El siguiente ejemplo traza el mismo cuadrado del principio utilizando esta nueva posibilidad:

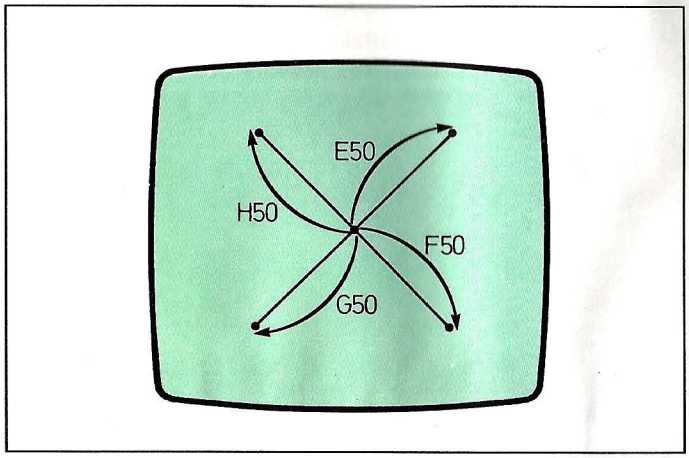
```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "M150,100M150,
   150M100,150M100,100"
40 GOTO 40
```

En este caso se ha supuesto que el origen de coordenadas se encuentra, como es habitual, en la esquina superior izquierda de la pantalla. Si no es así, se trazará el cuadrado hacia arriba, en lugar de hacia abajo.

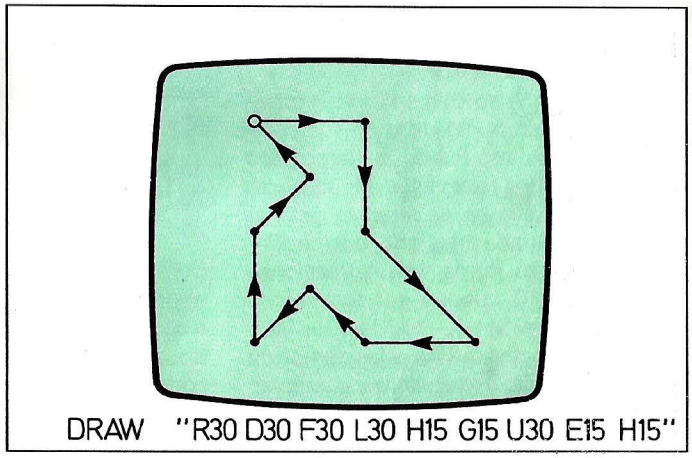
En el ejemplo se pone de manifiesto la formulación de esta nueva posibilidad. Tras la letra M se añaden ambas coordenadas (horizontal y vertical) separadas por una coma.

Este nuevo método de trazado parece más complicado que el anterior. Sin embargo, proporciona mayor potencia de dibujo, al no estar limitado el trazo a las ocho direcciones indicadas. Además de ello, la opción M permite conocer en cada momento la situación exacta del cursor gráfico, cosa que no se logra con el método anterior.

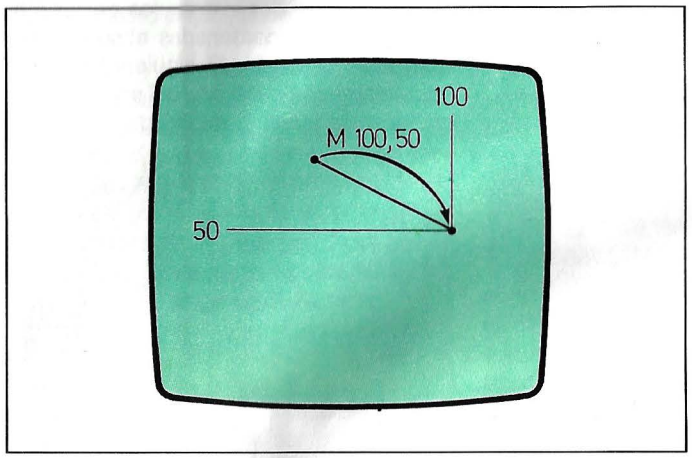
Para trazar una figura compleja con la opción M debe emplearse una filosofía dis-



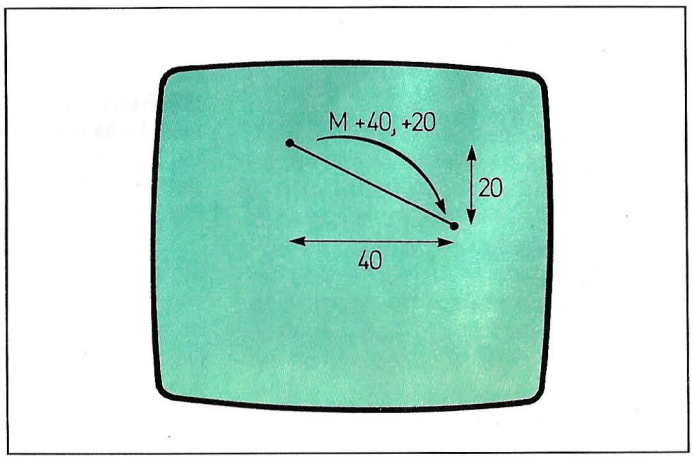
El trazado de líneas inclinadas se encomienda a los subcomandos E, F, G y H.



Realización de un dibujo utilizando los subcomandos de desplazamiento en las ocho direcciones.



Los desplazamientos a un punto dado por sus coordenadas absolutas se realizan haciendo uso del comando M.



El comando M también permite el desplazamiento a un punto dado por sus coordenadas relativas al último punto trazado.

tinta. El método anterior obligaba a conocer la situación relativa del punto siguiente. Con M, por el contrario, es preciso identificar las coordenadas absolutas de cada extremo de segmento. Una vez conocidas éstas, se tratará de recorrer los correspondientes puntos en el orden adecuado; esto es, siguiendo el trazo continuo. Se vuelve a mostrar el trazado del contorno de la pajarita, esta vez empleando las coordenadas absolutas de cada punto.

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "M130,100M130,130M160,160M130,
    160M115,145M100,160M100,130M115,115
    M100,100"
40 GOTO 40
```

Con el empleo de las coordenadas absolutas se observa otro efecto. Las figuras así trazadas aparecerán siempre en el mismo sitio. Por el contrario, al utilizar los desplazamientos relativos, la posición de la figura dependerá de la situación del punto inicial (el marcado con PSET).

La instrucción M no sólo permite el movimiento hacia un punto dado en sus coordenadas absolutas. Puede ser empleado con coordenadas relativas al último punto trazado. Para ello ha de situarse delante de la coordenada el signo correspondiente. De esta forma, las instrucciones U10,E10 y R10 pueden ser formuladas también como M+0,−10,M+10,−10 y M+10,0 respectivamente, y las demás de una forma similar. No obstante, el empleo de coordenadas relativas con M permite trazar rectas con todo tipo de inclinación, ya que no se limita a las ocho direcciones fijas en los otros comandos.

CAMINANDO SIN DIBUJAR

Tal como se ha visto hasta ahora, se pueden realizar figuras conexas; esto es, de un trazo continuo. La forma de cambiar de sitio el cursor sin por ello trazar una línea parece relegada al empleo de PSET. No obstante, puede prescindirse de ese comando. Ello resultará útil a la hora de encadenar varias figuras en una sola cadena ejecutable.

La letra B colocada delante de un comando-letra de movimiento producirá un desplazamiento sin trazo. Esta letra es la inicial de "blank", que en inglés significa "en blanco". Así pues, no es preciso anteponer un comando PSET a la ejecución de una cadena gráfica. El ejemplo inicial del cuadrado puede resumirse en una sola cadena.

```
10 SCREEN 2
20 DRAW "BM100,100R50D50L50U50"
30 GOTO 30
```

Como se aprecia en el ejemplo propuesto, el prefijo B sólo afecta a la siguiente instrucción, y no necesita de más parámetros. Este prefijo puede ser empleado con el resto de comandos-letra, permitiendo el trazado de varias figuras separadas. En el siguiente programa se dibujan dos cuadrados sin punto de conexión entre ambos.

```
10 SCREEN 2
20 DRAW "BM100,100R50D50L50U50
    BM100,155R50D50L50U50"
30 GOTO 30
```

En este caso se ha vuelto a emplear el prefijo B con el comando M. El conjunto marca la posición de partida del segundo cuadrado en el punto 100,155. El mismo resultado se obtiene empleando el comando de desplazamiento relativo R. He aquí el nuevo programa.

```
10 SCREEN 2
20 DRAW "BM100,100R50D50L50U50
    BR55R50D50L50U50"
30 GOTO 30
```

Dada la posición de fin de trazado del cuadrado, la instrucción BM100,155 es equivalente a BR55. Otra posibilidad es la ofrecida por el prefijo N. Este es sólo utilizable con los comandos de desplazamiento relativo y hace que el cursor vuelva al punto de partida tras marcar el trazo. El siguiente programa hace uso de esta característica para dibujar un asterisco.

```
10 SCREEN 2
20 DRAW "BM100,100NU50NE50NR50NF50
    ND50NG50NL50NH50"
30 GOTO 30
```

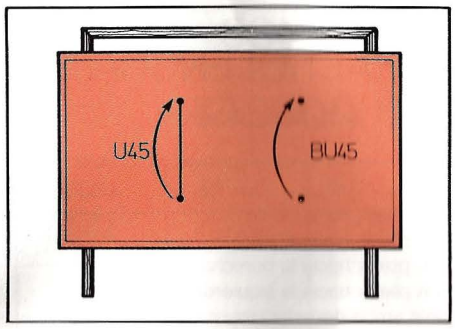
En este último ejemplo se aprecia que todas las instrucciones de desplazamiento relativo van precedidas por la letra N. Con ello se consigue que, tras dibujar cada brazo, se vuelva a la posición original; es decir, el punto central 100,100. De esta forma se evita una orden de vuelta atrás. La cadena "U50D50" queda así abreviada en "NU50".

ROTACION

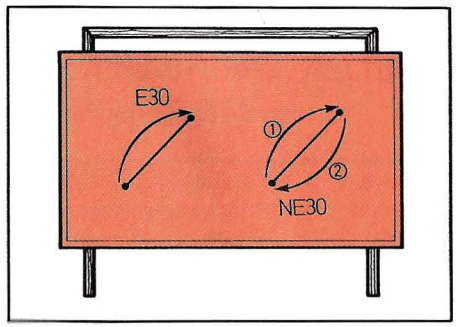
La mayor parte de las veces se desea utilizar una figura repetidamente. Para ello se puede almacenar la cadena generadora de dicha figura en una variable. Si la cadena utiliza comandos relativos, ello permitirá situar la figura en distintas posiciones de la pantalla. Véase un ejemplo en el que se dibuja el mismo cuadrado en diferentes lugares:

```
 10 SCREEN 2
 20 A$="R40D40L40U40"
 30 DRAW "BM20,20"
 40 DRAW A$
 50 DRAW "BM20,100"
 60 DRAW A$
 70 DRAW "BM100,20"
 80 DRAW A$
 90 DRAW "BM100,100"
100 DRAW A$
200 GOTO 200
```

Pero, una vez creada la cadena, ésta sólo puede ser reubicada en otro lugar. Para alterar su forma se precisa un tratamiento de la propia cadena. Esto no es siempre cierto. Por ejemplo, se puede especificar una rotación de la figura creada. Para ello se hace uso de una nueva letra-comando. Se trata de la letra A (de Angulo), que seguida de un número varia la orientación de las instrucciones que le sigan.



El prefijo B permite el desplazamiento sin trazar la trayectoria.

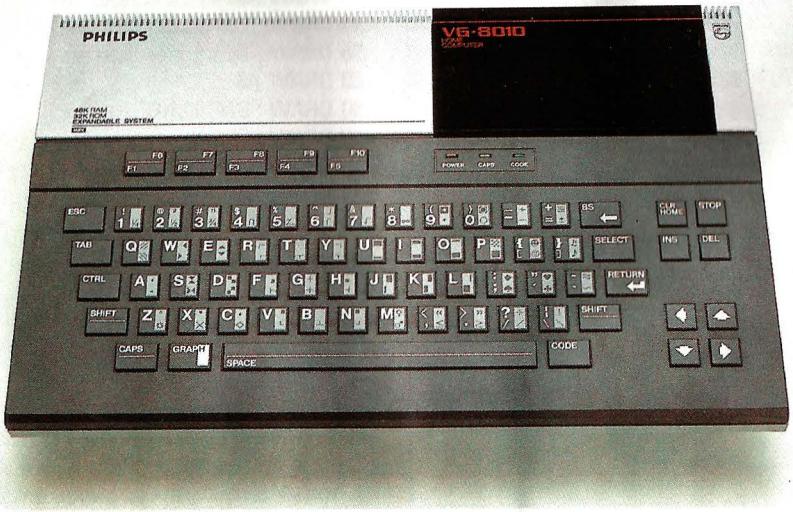


Otro prefijo importante es N. Este devuelve el cursor gráfico al punto en donde se inició la ejecución del comando.

El comando A admite como argumento un entero del 0 al 3. El cero indica la orientación inicial: el comando U moverá hacia arriba. El resto de números marca un ángulo de desfase de 90, 180 y 270 grados respectivamente. Ello quiere decir que tras la ejecución de A1, el comando U efectuará un movimiento hacia la derecha de la pantalla, con A2U moverá hacia abajo, etc. El resto de comandos relativos variará su dirección de ataque solidariamente con el comando U.
Esta nueva función gráfica permite repetir una misma figura con cuatro orientaciones distintas. Esto mismo es lo que se realiza en el siguiente programa:

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM100,100"
40 DRAW A$
50 DRAW "A1"
60 DRAW A$
70 DRAW "A2"
80 DRAW A$
90 DRAW "A3"
100 DRAW A$
200 GOTO 200
```

En este ejemplo se aprecia que la orientación estipulada por el comando A se mantiene hasta que se ejecute un nuevo comando A. El programa utiliza la misma cadena para trazar cuatro cuadrados diferentes, uno en cada una de las cuatro direcciones posibles. Para volver a seleccionar la orientación inicial es preciso ejecutar un comando A0, de lo contrario seguirá vigente la última adoptada.
Cabe destacar que el comando A, si bien altera el ángulo tomado como referencia en los comandos relativos, no varía en absoluto el efecto del comando M. Esto es debido a que M se apoya en las coordenadas absolutas y éstas se mantienen siempre fijas.



Los ordenadores adscritos a la norma MSX poseen el macrolenguaje gráfico asociado al comando DRAW dentro de su intérprete BASIC.

ESCALA

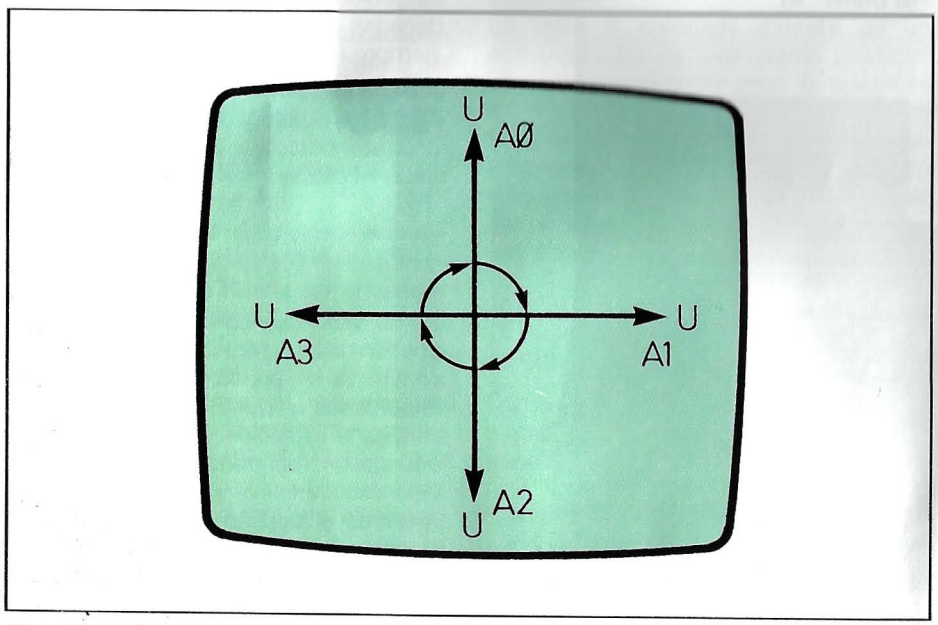
Se ha visto que con una cadena fija se pueden trazar distintos tipo de figuras. Por una parte se puede alterar la situación, imponiendo un punto de partida diferente para cada ejecución. Por otra, variando la orientación, por medio del comando A. Otra característica que convendría poder variar es el tamaño. Pues bien, esto se consigue por medio del cambio de escala.
El cambio de escala se efectúa mediante el nuevo comando-letra S (de Scale). La nueva escala se indica con un entero tras

SUBCOMANDOS DEL MACROLENGUAJE GRAFICO DE MICROSOFT

U<n>	Mueve el cursor de gráficos n pixels hacia arriba
D<n>	Mueve el cursor de gráficos n pixels hacia abajo
R<n>	Mueve el cursor de gráficos n pixels hacia la derecha
L<n>	Mueve el cursor de gráficos n pixels hacia la izquierda
E<n>	Mueve el cursor en la diagonal arriba-derecha
F<n>	Mueve el cursor en la diagonal abajo-derecha
G<n>	Mueve el cursor en la diagonal abajo-izquierda
H<n>	Mueve el cursor en la diagonal arriba-izquierda
M<h>,<v>	Mueve el cursor al punto de coordenadas absolutas <h> y <v>
M±<h>,±<v>	Mueve el cursor al punto de coordenadas relativas <h> y <v>
B	Prefijo que evita que se marque la trayectoria del cursor
N	Prefijo que, tras trazar un segmento, deja el cursor en el punto de partida
A<n>	Impone una rotación a la figura que se plasme a continuación
S<n>	Permite dibujar una figura a escalas diferentes
C<n>	Indica el color que ostentarán los siguientes trazos en la pantalla

la letra S. El entero puede variar de 1 a 255, siendo la escala habitual la que corresponde a S4. En realidad, el factor de escala se obtiene de la división del entero propuesto entre 4. El dato así obtenido es el que se multiplicará a las longitudes de trazo especificadas. De esta manera, al dividir 4 entre 4 da como resultado 1, con lo que la escala es la natural (medida en pixels). Ello permite reducir la figura hasta cuatro veces (S1 de un factor de 1/4) o ampliarle hasta 63 veces (252/4=63).
A continuación se muestra un ejemplo en el que se trazan cuatro cuadrados, cada uno en una escala diferente. El resultado es el de cuatro cuadrados uno dentro de otro.

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM100,100"
40 DRAW A$
50 DRAW "S1"
60 DRAW A$
70 DRAW "S2"
80 DRAW A$
90 DRAW "S3"
100 DRAW A$
200 GOTO 200
```



Dirección en la que actuará U con cada una de las cuatro orientaciones posibles.

Al igual que el comando A, el de cambio de escala S queda fijado hasta que se vuelve a indicar una nueva escala. Su acción sólo afecta a los comandos de desplazamiento relativo, ya que las coordenadas no son susceptibles de alteración.

COLOR

Las líneas trazadas hasta el momento adoptan el color del primer plano. Color que habrá sido estipulado mediante la ejecución del apropiado comando COLOR (véase el capítulo precedente de gráficos en color). Si el aparato dispone de más colores éstos se podrán elegir en el seno de una cadena ejecutable asociada a DRAW. Para ello se emplea un nuevo comando-letra. Se trata en esta ocasión de la letra C (de Color), la cual deberá ir seguida del código correspodiente al pigmento deseado.

El siguiente ejemplo traza cuatro cuadrados, cada uno en un color diferente:

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM20,20"
35 DRAW "C0"
40 DRAW A$
50 DRAW "BM20,100"
55 DRAW "C1"
60 DRAW A$
70 DRAW "BM100,20"
75 DRAW "C2"
80 DRAW A$
90 DRAW "BM100,100"
95 DRAW "C3"
100 DRAW A$
200 GOTO 200
```

Los códigos correspondientes a cada color dependerán del aparato en particular.
A la hora de elegir el color es preciso tener en cuenta el color de fondo. Si se dibuja una figura en el mismo color que el fondo, ésta no será visible.
Una vez cambiado el color del trazo, este permanecerá como color en curso hasta que vuelva a cambiar. Para volver al color de primer término inicial puede emplearse tanto la opción C como el comando COLOR.

TABLA DE CONVERSION

ORDENADOR	Comando BASIC	Subcomandos del macrolenguaje gráfico					
	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
APPLE II (APPLESOFT)	—	—	—	—	—	—	—
APRICOT (M-BASIC)	—	—	—	—	—	—	—
ATARI	—	—	—	—	—	—	—
CBN 64	—	—	—	—	—	—	—
DRAGON	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
EQUIPOS MSX	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
HP-150	—	—	—	—	—	—	—
IBN PC	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
MPF	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—	—
ORIC	—	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—
SPECTRAVIDEO	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
ZX-SPECTRUM	—	—	—	—	—	—	—

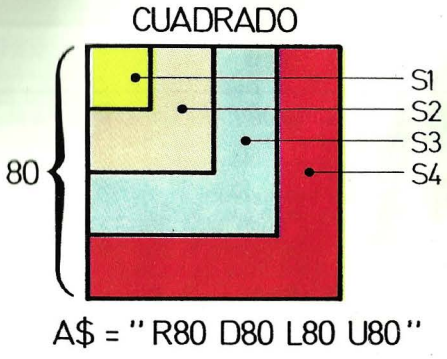
Las tres últimas características pueden ser mezcladas de forma que se obtengan figuras bien distintas partiendo de una única cadena ejecutable. En el programa que sigue se utilizan las tres posibilidades mencionadas.

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM100,100A0S4C0"
40 DRAW A$
50 DRAW "A1S1C1"
60 DRAW A$
70 DRAW "A2S2C2"
80 DRAW A$
90 DRAW "A3S3C3"
100 DRAW A$
200 GOTO 200
```

Cada cuadrado mostrara una orientación, una escala y un color diferentes. La orden de cambio de color se puede incluir antes y después de cada trazo. De esta forma se permite un color diferente por cada lado. Al contrario de los comandos de cambio de escala y orientación, el cambio de color si que afecta a las líneas trazadas con la colaboración del comando absoluto M.

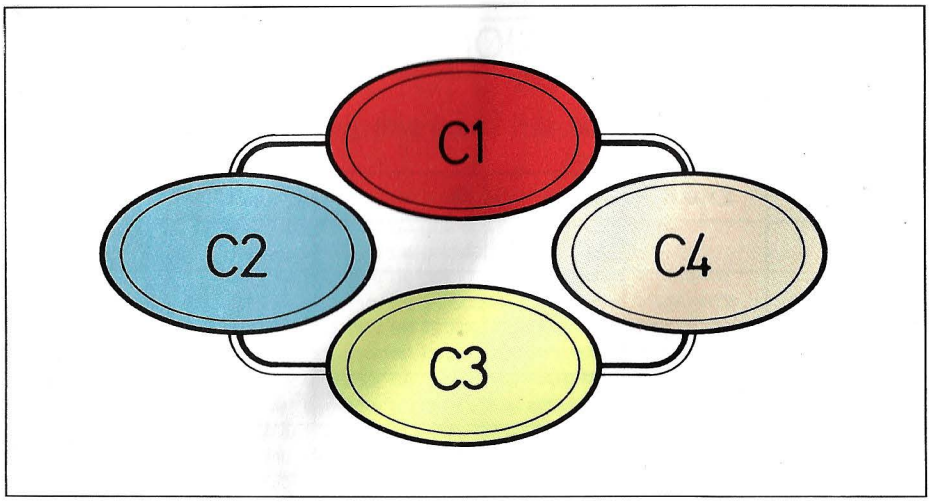
LA UTILIZACION DE DATOS VARIABLES

La cadena ejecutable situada en el argumento de DRAW puede ser constante. También puede emplearse una variable de cadena creada al efecto. En el primer caso, la figura será siempre fija. Con el empleo de una variable de cadena, por el contrario, se puede variar la figura trazada por un comando DRAW. Para ello basta con asignar una nueva cadena a la variable precisa.

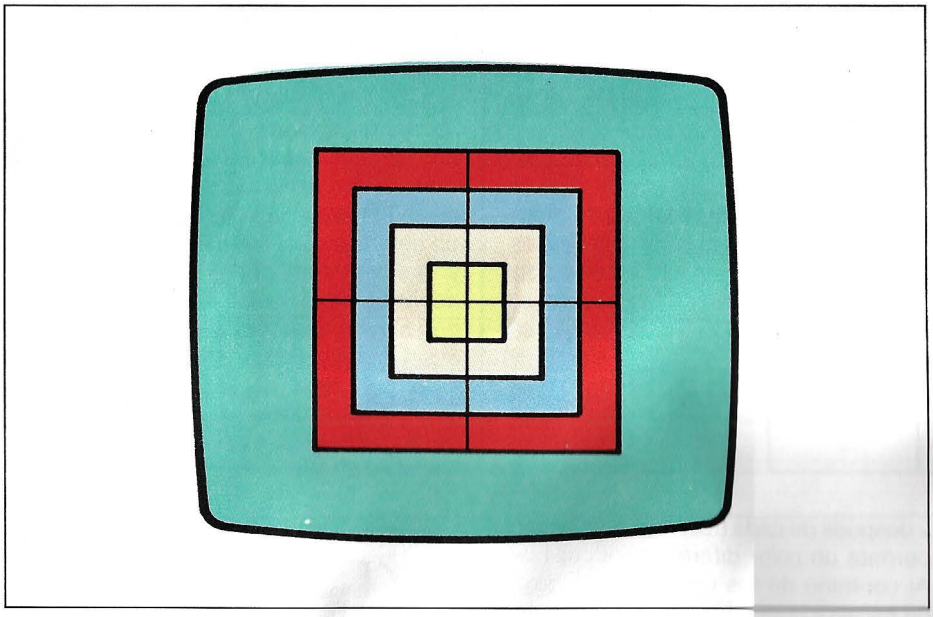


Una misma figura se puede representar en distintos tamaños, para ello basta con cambiar la escala por medio del comando S.

Al principio del capitulo se vio cómo se puede manipular una cadena para alterar



Por medio del comando C se puede especificar el color que ha de tomar la figura a trazar.



¡Y ahora todo junto! Resultado de la ejecución del último ejemplo.

la forma de la figura inicial. El método consistía en tratar la cadena mediante las funciones específicas de manejo de cadenas (LEFT$,MID$, etc.). Sin embargo, este proceso implica la creación de una rutina especial de manejo de cadenas. Dicha rutina puede resultar excesivamente compleja en ciertos casos.

Por ejemplo, para inspeccionar una cadena y alterar los valores numéricos de la misma se necesitan una serie de pasos. En primer lugar se precisa recorrer la cadena hasta el primer carácter numérico. Una vez en ese punto, se han de recoger los datos numéricos siguientes y tratarlos como un solo número (haciendo uso de la función VAL). Tras las oportunas operaciones aritméticas se deberá depositar el resultado en una nueva cadena ejecutable (en esta ocasión se precisa emplear STR$). El último paso consiste en la ejecución de la cadena obtenida.

El proceso indicado ha de ser codificado en un programa bastante complejo. Y es posible que no se consiga el efecto deseado.

Sin embargo, el procedimiento expuesto no es siempre necesario. El comando DRAW contempla la posibilidad de incluir variables en el interior de la cadena que utiliza como argumento. Estas variables se dividen por el tipo de dato que atesoran: alfanuméricas y numéricas.

Para el empleo de una variable alfanumérica dentro de la cadena ejecutable se emplea la letra-comando X. Tras ella se sitúa el nombre de la variable que contiene la subcadena deseada. Como ejemplo de su uso se muestra una nueva versión del último programa propuesto:

```
 10 SCREEN 2
 20 A$="R40D40L40U40"
 30 DRAW "BM100,100A0S4C0XA$;A1S1C1
    XA$;A2S2C2XA$;A3S3C3XA$;"
200 GOTO 200
```

En este caso se han agrupado todas las líneas que contenían un comando DRAW en una sola. Se puede ver que tras cada subcadena se pone un signo de punto y coma. Ello es obligado y sirve para identificar el final del nombre de la variable.

Esta es, pues, la forma de añadir subcadenas variables a una cadena gráfica. Para introducir variables numéricas se emplea otro método. Una variable numérica se utilizará para alterar los valores de la escala, el color, etc. Cuando se quiera hacer uso de una variable numérica se pondrá un signo igual tras el adecuado comando-letra. Inmediatamente después se colocará el nombre de la variable y se cerrará el conjunto con un punto y coma. El siguiente ejemplo hace uso de esta característica para trazar diferentes cuadrados:

```
 10 SCREEN 2
 20 A$="R40D40L40U40"
 30 DRAW "BM100,100"
 40 FOR X=1 TO 4
 50 FOR I=0 TO 3
 60 DRAW "S=X;C=I;A=X;XA$;"
 70 NEXT I
 80 NEXT X
100 GOTO 100
```

Este ejemplo utiliza dos bucles FOR para recorrer los valores de las cuatro orientaciones produciendo cuadrados de cuatro tamaños y colores.

En los ejemplos se han utilizado cuadrados, pero las figuras a representar pueden ser tan complicadas como lo desee el usuario. Cualquier figura admitirá las opciones de cambio de escala, orientación y color, tal como aquí se ha explicado.

El lenguaje C (2)

Los primeros pasos

En el capítulo precedente se esbozó la forma general de un programa en C a través de un sencillo ejemplo. Ahora es el momento de profundizar en el conocimiento su sintaxis y sus estructuras

UN NUEVO EJEMPLO

```
main ( )
{
int b;
int a=1;

b=mi__función (a);
printf ("El resultado de mi función es %d\n",b);
}
mi__función (i)
int i;
{
return (i + 1);
}
```

Programa ejemplo que ayudará a profundizar algo en "C".

En la figura aparece otro ejemplo de programa en C. Como ya se señaló en el anterior capítulo, destaca la presencia de "main" al principio del listado. En el cuerpo del mismo se encuentran dos variables, "a" y "b", ambas de tipo entero (integer), que se declaran con la palabra clave "int". Esta palabra, como otras pocas que se estudiarán, es de tipo reservado; es decir, el usuario no puede utilizar variables ni funciones que se llamen "int". Además, se observa que, en C, a la vez que se declara se puede inicializar una variable, como se hace con "a". La parte de declaración de las variables podría haberse escrito como sigue:

```
int b;
int a;
a=1;
```

O bien como:

```
int a,b;
a=1;
```

O incluso como:

```
int b,a=1;
```

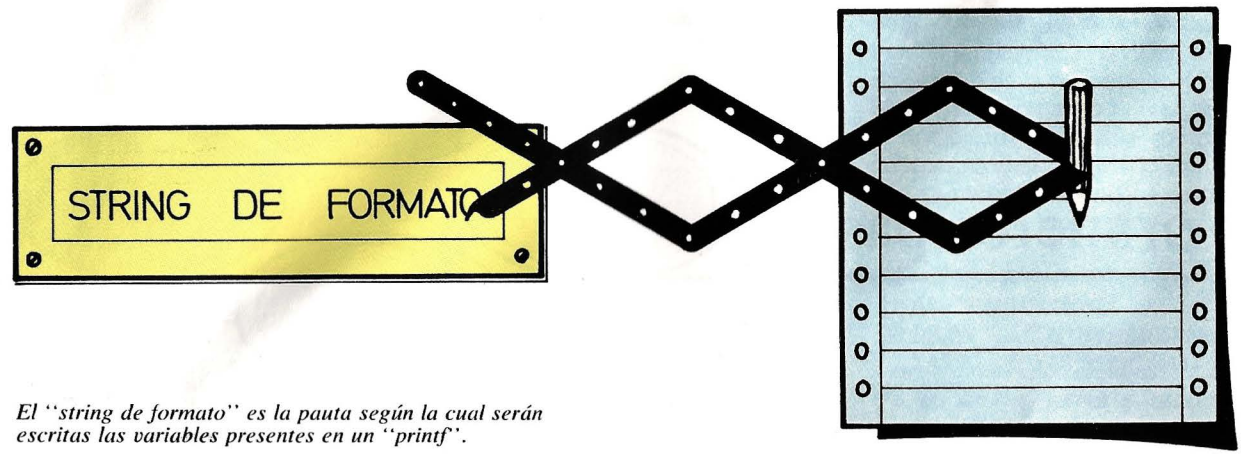
siendo las cuatro formas totalmente equivalentes. En todo momento se eligirá aquella que supongamos más adecuada para la interpretación de lo que se desea expresar.
En la siguiente línea se asigna a la variable "b" el valor de la función "mi__función" cuando el parámetro de tal función vale 1. La forma en la que las funciones utilizan los parámetros y devuelven valores es análogo a como se hace en PASCAL. Cuando el programa llega a:

PRINTF (STRING DE FORMATO , ARGUMENTO , ARGUMENTO , ...);

Forma general de la función "printf".

ESPECIFICADORES DE CAMPO PARA "PRINTF"

Valores enteros:	%d	escribir argumento como valor decimal
	%o	escribir argumento como valor octal
	%x	escribir argumento como valor hexadecimal
Valores reales:	%f	escribir el argumento en la forma ddd.ddd
	%e	escribir el argumento en la forma d.ddd+dd
caracteres:	%c	escribir un carácter
cadenas:	%s	escribir un string de caracteres



El "string de formato" es la pauta según la cual serán escritas las variables presentes en un "printf".

```
mi_función (i)
int i;
{
int cte;
cte=3;
return (i+cte);
}
```

Programa en "C" en el que se hace uso de una variable automática (cte) que sólo puede ser referida dentro de "mi_función".

```
1)
main ()
{
incrementar;
incrementar;
incrementar;
}
incrementar ()
{
int x=0;
x=x+1;
printf("%d\n",x);
}

SALIDA:
1
1
1
—
```

```
2)
main ()
{
incrementar;
incrementar;
incrementar;
}
incrementar ()
{
static int x=0;
x=x+1;
printf("%d\n",x);
}

SALIDA:
1
2
3
—
```

En el segundo ejemplo, la variable "x" ha sido declarada como estática, por lo que su valor sigue existiendo después de haber abandonado "incrementar". Observar que la inicialización de "x" sólo ocurre en la primera pasada por la citada función.

b= mi_función (a);

el valor que en ese momento tiene la variable "a" (en nuestro caso es 1) es copiado sobre la variable "i" que está en la declaración de "mi_función". En el cuerpo de "mi_función" nos encontramos una sola sentencia:

return (i+1);

que, como se intuye, hace que "mi_función" tome el valor "i+1" (en nuestro caso 2) y vuelva al punto de "main" desde donde fue llamada, asignando a "b" el valor 2.

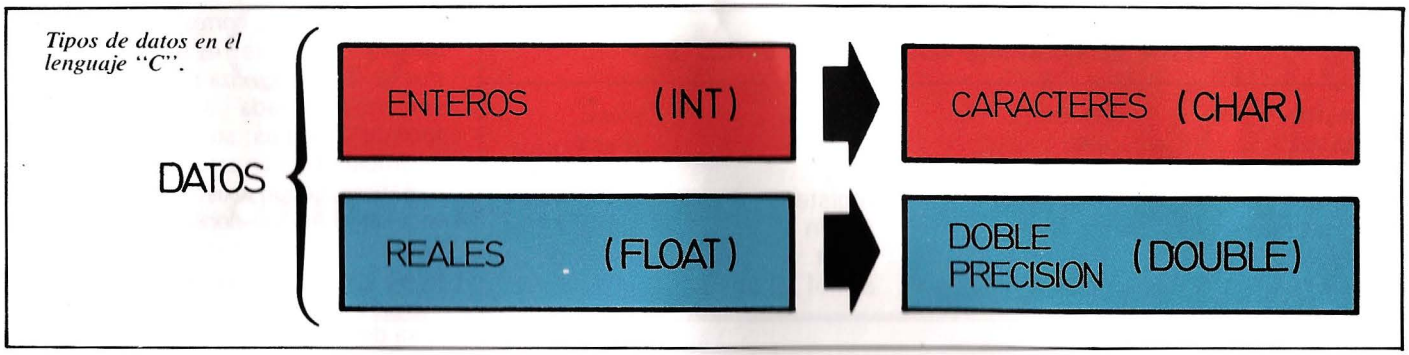
En el segmento de programa correspondiente a "mi_función", y antes de comenzar el cuerpo de la misma (delimitado por { }), se encuentra la declaración:

int i;

Al igual que en PASCAL es preciso declarar los tipos de los parámetros de las funciones. La declaración equivalente en PASCAL de nuestra función sería:

FUNCTION mi_función (i: integer): integer;

En nuestro caso sólo ha sido necesario hacer la declaración del parámetro, ya que en C una declaración de función que no lleve explícito el tipo se supone que devuelve un valor entero. Si por cualquier otra causa, "mi_función" devolviera un valor real a partir de un entero, la declaración sería:



Tipos de datos en el lenguaje "C".

```
float mi__función (i)
int i;
```

donde "float" —también palabra clave— significa "floating point", que se corresponde con la declaración "real" de PASCAL.

MAS SOBRE "PRINTF"

Volviendo a "main", una vez que a "b" se le ha asignado su valor, aparece un "printf", el cual merece un comentario.
Como se indicó en el ejemplo incluido en el primer capítulo, "printf" escribirá los caracteres que van apareciendo entre las dobles comillas hasta llegar al "%". En este punto sustituirá el grupo "%d" por el valor de la primera variable que se encuentre a continuación del citado texto entre comillas, en nuestro caso el valor de "b"; luego posicionará el cursor al principio de la siguiente línea.
Nuestro programa dará por tanto la siguiente salida:

```
El resultado de mi función es 2
—
```

La forma general de "printf" se observa en la figura adjunta. El "string de formato" irá entre dobles comillas y será lo que aparezca en la pantalla, además de los valores de los argumentos en los lugares indicados por los símbolos "%"; estos símbolos van siempre seguidos por una letra (juntos forman un "especificador de campo") que indica el tipo de argumento que se va a escribir. Las distintas especificaciones se dan en la correspondiente figura.

```
char x='f';
main ()
{
printf ("%c\n",x);
}
```

Programa ejemplo en el que se hace uso de una variable externa.

LOS TIPOS DE DATOS

En C existen cuatro tipos básicos de datos. De ellos sólo "int" y "float" son realmente básicos; los otros surgen como derivaciones o ampliaciones. Con estos cuatro tipos, y de manera parecida a como se hace en PASCAL, pueden formarse estructuras más complejas, tales como arrays (equivalente a los "strings" de C) o records ("struct" en C).
Para declarar variables se anteceden és-

Las constantes simbólicas

C permite el uso de las llamadas "contantes simbólicas" para que los listados sean inteligibles y los programas fácilmente modificables.
Al igual que en la sección CONST de Pascal, una constante simbólica permite, con la sola modificación de una línea, variar la acción del programa para que pueda ser utilizado en distintas aplicaciones.
Las constantes simbólicas se pueden colocar en cualquier parte del programa, aunque suelen situarse al principio del mismo. El ejemplo universal de constante simbólica es el número PI.
En Pascal se incluiría:

```
CONST PI=3.14159;
```

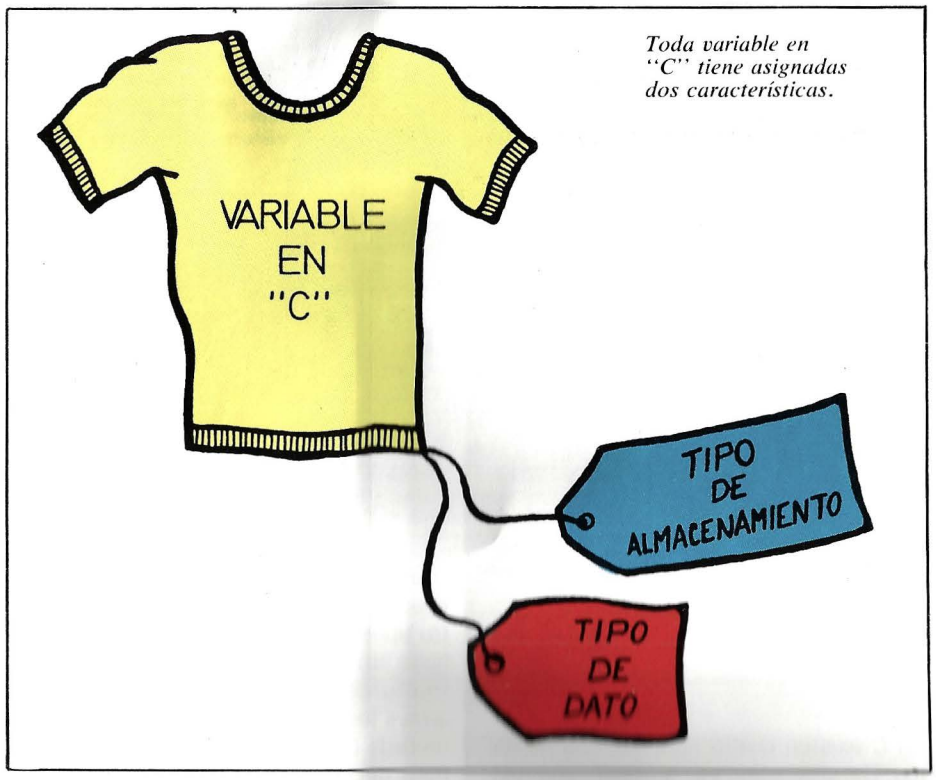
En C, una constante se especifica de la siguiente forma:

```
#define PI 3.14159
(cuidado con poner el ; )
```

La misión del preprocesador, del que se habló en el capítulo precedente, consiste en analizar el programa para encontrar la palabra PI y sustituirla por todo aquello que sigue al segundo espacio en blanco de la declaración "#define". De ahí que no haya que incluir el signo ";" en tal declaración.

LISTA DE PALABRAS CLAVE DEL LENGUAJE "C"

auto	double	if	static
break	else	int	struct
case	entry	long	switch
char	extern	register	typedef
continue	for	return	union
default	float	short	unsigned
do	goto	sizeof	while



Toda variable en "C" tiene asignadas dos características.

tas con la palabra clave correspondiente a su tipo, seguida por los nombres de las variables. Las palabras clave son:

int: entero
char: carácter
float: números reales
double: números reales en doble precisión

Un valor real en doble precisión es un valor tipo "float" (real) especificado con mayor exactitud, aunque no tiene por qué ser el doble de preciso.
Los caracteres se representan entre comillas simples, por ejemplo:

```
char primera, última;
primera='A';
última='z';
```

La diferencia entre lo que significa 'A' y "A" se evidenciará al tratar de los strings.

TIPOS DE ALMACENAMIENTO

En C una variable tiene asignado, aparte de un tipo de dato, un tipo de almacenamiento. Para obtener una idea al respecto, cabe pensar en la diferencia que existe entre variables locales y globales de PASCAL.

Variables automáticas
Suponga que se modifica "mi función" tal como muestra la correspondiente figura. En este caso, "cte" es una variable automática. Se caracteriza porque sólo puede ser referenciada dentro de "mi_función", esto es: su utilización fuera de tal ámbito daría un error. Las variables automáticas tienen existencia mientras se está en el bloque donde son declaradas, nada más. De esta forma sólo ocupan memoria cuando son necesarias, lo cual es a menudo muy ventajoso. La forma ortodoxa de declarar "cte" es la que sigue:

```
auto int cte;
```

Si bien, la palabra clave "auto" se suele omitir por la frecuencia con la que aparecen este tipo de variables. El compilador supone que toda variable sin especificar el tipo de almacenamiento es automática.

Variables de registro
Su utilización y características son análogas a los del tipo automático. En realidad se trata de indicar al compilador que tal variable será utilizada con mucha frecuencia y conviene tenerla almacenada en un lugar al que se pueda acceder con facilidad, como es un registro de la CPU, en vez de tenerla en una posición de memoria. La forma de declararlas es:

```
register int cte;
```

Variables estáticas
Este tipo de variables, al contrario que los tipos anteriores, no dejan de existir al abandonar el ámbito donde son declaradas. Al igual que los tipos anteriores sólo se pueden referenciar en el ámbito de declaración, pero el valor que tenían al abandonar el bloque lo volveremos a encontrar al entrar de nuevo en él. La figura adjunta incluye un ejemplo clarificador.

Variables externas
Una variable externa es accesible desde cualquier función del programa. Se declaran al principio del listado. Por ejemplo, en la figura 7: "x" es global a "main" y a cualquier otra función que hubiera. Estrictamente, al principio de "main" debería aparecer la declaración:

```
extern int x;
```

El uso de este tipo de declaraciones es optativo, aunque es conveniente incluirlas si el programa es muy complicado.

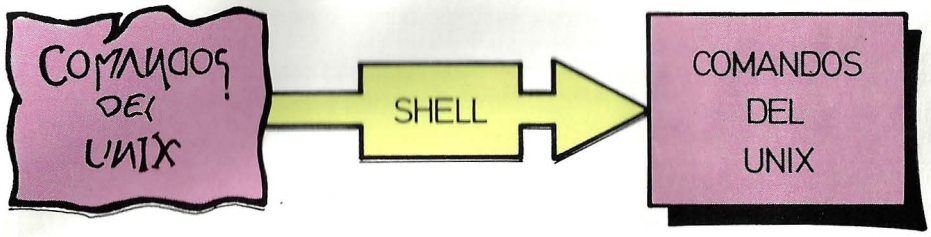
Unix (y 5)

El shell como lenguaje de programación

En el capítulo anterior se indicó que el Shell es simultáneamente un intérprete de comandos y un lenguaje de programación de alto nivel. Como intérprete de comandos procesa los que le son introducidos en respuesta a la señal que evidencia que está dispuesto a aceptarlos. Cuando actúa como lenguaje de programación procesa grupos de comandos almacenados en ficheros denominados "Shell scripts". En los próximos párrafos se tratará de todo lo referente a este tipo de operación del Shell, empleando para ello los esquemas referidos al Bourne Shell que, en gran medida, son similares los del C Shell.

Una particularidad de los ficheros "Shell scripts" es permitir que diferentes líneas de comandos puedan ser agrupadas de forma que un único comando pueda ejecutarlas. Por medio de este tipo de estructuras es posible que un usuario del sistema operativo UNIX ejecute tareas complejas, y en muchas ocasiones largas y tediosas, de una forma simple.

A través de este tipo de comandos es posible copiar varios ficheros, cambiar su nombre y borrar otros, cosa común en un proceso de actualización y reajuste de los contenidos de la memoria de un ordenador por parte del administrador del sistema, por medio de un único comando. Igualmente, es posible redirigir la entrada y salida estándar dentro de un Shell Script, de manera que puedan combinarse programas de utilidad UNIX en forma adecuada a las necesidades del usuario.



El Shell permite la cómoda creación de estructuras de comandos.

FICHEROS EJECUTABLES

Se conoce como fichero ejecutable a aquel fichero que cualquier usuario es capaz de ejecutar, o mejor dicho, que cualquier usuario tiene permiso para ejecutar. Normalmente, contiene un programa compilado o un Shell Script.

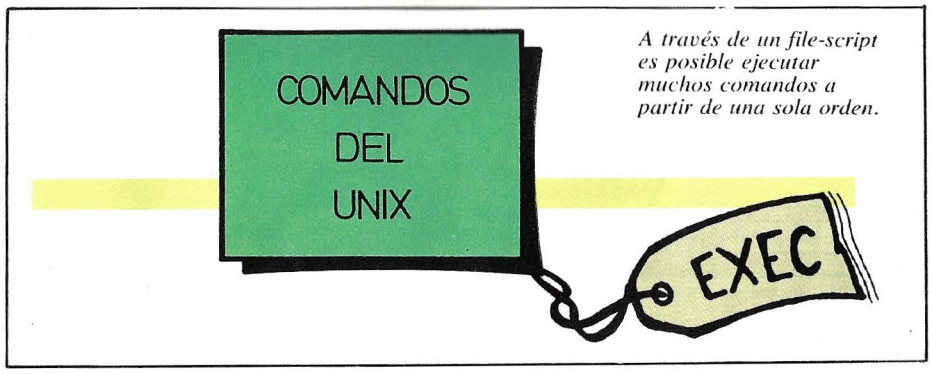
Para tener acceso a estos ficheros es necesario, por una parte, que a través del editor sea posible acceder a los mismos, y por otro lado es preciso tener derechos de ejecución sobre ellos. Caso de no ser así, por medio del comando *chmod* es posible cambiar los privilegios de acceso asociados al fichero y hacer que un usuario tenga privilegios de acceso sobre el mismo. Una vez que el usuario ha adquirido estos derechos, tecleando el nombre del fichero directamente éste será reconocido como un comando y ejecutado de inmediato. Si son varios los usuarios del comando, será preciso establecer estas prioridades y derechos de acceso para cada uno de ellos. Estos ficheros, además de estructuras de control, admiten variables como las señaladas seguidamente.

VARIABLES

El Shell es capaz de aceptar variables de cadena (variables que son capaces de tomar el valor de una cadena de caracteres) de forma que puedan ser almacenados números y texto. De estas variables existen tres tipos distintos:

— Variables de usuario.
— Variables de Shell.
— Variables de Shell de solo lectura.

Las variables de usuario pueden ser decla-



A través de un file-script es posible ejecutar muchos comandos a partir de una sola orden.

radas, inicializadas, leídas o modificadas desde la línea de comandos o bien desde un Shell Script. El Shell se ocupa de declarar e inicializar las variables de Shell, pero el usuario tiene la facultad de leerlas y modificarlas; en el caso de las variables de Shell de sólo lectura, él mismo las declara e inicializa pero no permite el cambio, aunque sí la lectura por parte del usuario.

- *Variables de usuario.*

Cualquier secuencia de caracteres distintos de espacios en blanco puede ser declarada como nombre de una variable, inicializándola seguidamente con un valor determinado. Así, por ejemplo:

```
nombre=Juan
```

En este caso se ha declarado una variable bajo la denominación nombre, variable que ha sido inicializada con el valor Juan. Un dato a resaltar es el hecho de que el signo igual en el comando que acabamos de señalar no puede ir precedido ni seguido por un espacio en blanco. Caso de desear que en la asignación de valores a la variable aparezcan espacios en blanco será necesario encerrar a la misma entre comillas.

```
nombre="Juan Martín"
```

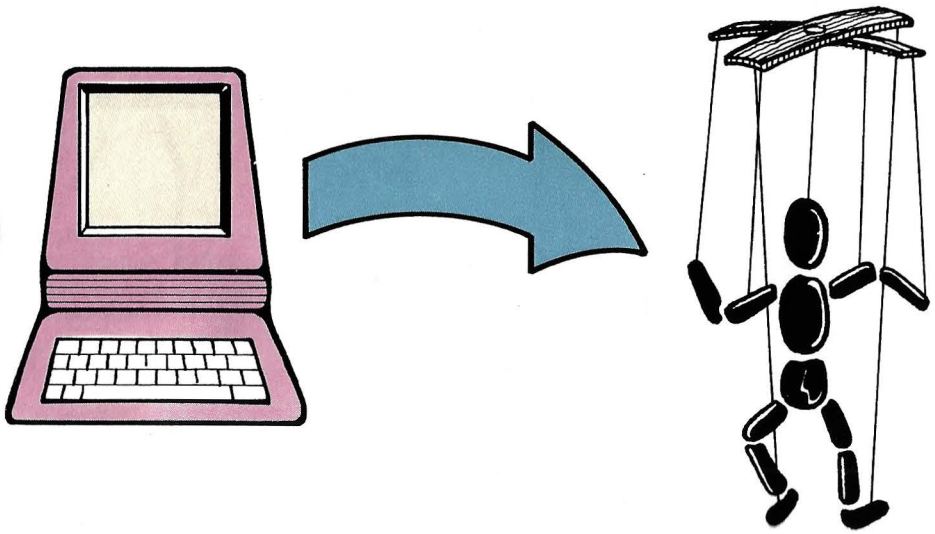
Si el usuario deseara conocer el contenido de una variable tendría que hacer uso del comando *echo*. Sin embargo, el uso del comando *echo* directamente no daría resultado. Por ejemplo:

```
echo      nombre
nombre
echo  $   nombre
Juan
```

De la primera línea de comando resulta que *echo* aplicado directamente produce tan sólo la visualización del nombre de la variable. Si bien, cuando el nombre de la variable se expresa precedido por un signo $ es cuando se presenta la cadena nombre con el valor Juan. La función del signo $ es la de comunicar al Shell que la palabra que sigue al signo $ es el nombre de una variable. En estas condiciones, el Shell toma el nombre de la variable, lo sustituye por su valor, y pasa éste para ser procesado.

El usuario tiene la posibilidad de inhibir este proceso simplemente colocando entre comillas el nombre de la variable precedido por el signo $, con lo cual no se producirá la asignación de valor a la variable. Así, por ejemplo:

```
echo  "$ nombre"
nombre
```

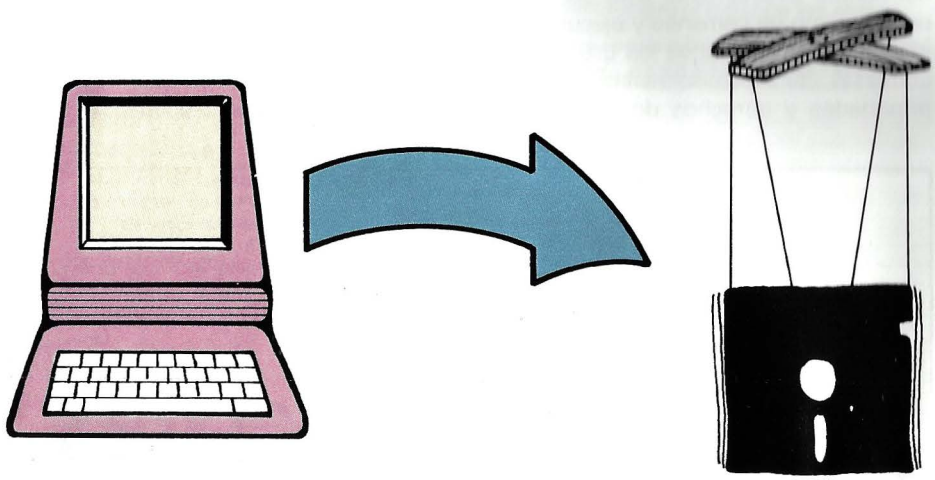


Las variables de usuario sólo son controladas por el mismo.

- *Variables de Shell de sólo lectura.*

Las variables de Shell de esta índole son variables que se declaran como de sólo lectura y a las que ha de asignarse un valor antes de ser declaradas como de este tipo. Una vez producida la declaración, este valor ya no puede ser cambiado; de intentarlo, el Shell producirá un mensaje de error. La asignación de valores a este tipo de variables es llevada a cabo directamente por el Shell, y la declaración de su carácter se realiza de la forma indicada en el ejemplo: especificando el nombre de dicha variable y anteponiendo la sentencia *read only*.

```
read only      nombre
```



Las variables de Shell quedan bajo el control exclusivo del sistema operativo.

- *Variables de Shell*

El Shell declara e inicializa una serie de variables a través de las cuales es posible definir la forma de *prompt* del sistema que el usuario va a recibir, la trayectoria de la búsqueda que el Shell sigue cuando recibe un comando, o el directorio base del usuario. Estas variables no son fijas sino que son actualizables, bien sea desde

la línea de comando o desde el fichero profile situado en el directorio base del usuario. A continuación vamos a señalar cuáles son estas variables así como su funcionamiento básico.

- *Variable Home*

Esta variable se emplea para almacenar en la misma el nombre del directorio de trabajo en el cual el usuario va a operar. Por defecto, el directorio base es el directorio de trabajo cuando se conecta el usuario al sistema por vez primera; este valor queda almacenado en la variable HOME. Si su valor se modifica especificando en ella un nuevo directorio, cuando se ejecute el comando CD para cambiar de directorio —si no se ha indicado ningún parámetro en el mismo— se producirá el cambio de directorio actual al señalado en la variable HOME. Así, por ejemplo:

```
echo  $  HOME
/usr/  JUAN
cd
pwd
/usr/  JUAN
```

Por medio de los ficheros "Shell script", el usuario puede ejecutar secuencias de múltiples comandos activando un sola orden.

- *Variable PATH*

Cuando al Shell se le da un comando, este ejecuta una búsqueda a través de la estructura de ficheros para localizar el que contiene el programa que se desea ejecutar; en su búsqueda controla diferentes directorios hasta encontrar el programa, comenzando por el directorio de trabajo y

Traducción por ordenador

En la mentalidad popular el ordenador es una máquina orientada esencialmente a la ejecución de cálculos matemáticos, y como tal cabría clasificarlo como una máquina matemática. Sin embargo, la realidad es que se trata de una máquina lingüística ya que su misión básica es la de interpretar una serie de símbolos lingüísticos que constituyen los programas que da vida a la máquina; símbolos lingüísticos llenos por tanto de significado. Un objetivo envidiable sería poder emplear el propio lenguaje hablado convencional para poder llevar a cabo estas tareas. Ello se ha intentado repetidamente; un botón de muestra se encuentra en las actuales máquinas traductoras, las cuales permiten el paso de un lenguaje natural a otro. Dichas máquinas han sido un subproducto de un problema informático muy superior.

La posibilidad de la traducción por ordenador surgió en el año 1949, cuando los pocos ordenadores existentes en el mundo se encontraban en instalaciones militares. En esta fecha, el matemático Warren Weaver estableció que las técnicas desarrolladas para la descripción de códigos podían ser aplicadas a la traducción mecánica.

El proceso en sí parece simple, aunque una primera dificultad ha sido la de conseguir un diccionario de palabras lo bastante grande como para no tener traducciones sincopadas. Una vez hecho esto, ha aparecido el gran problema: las ambigüedades. Una palabra puede tener dos significados y la frase puede no dar pistas acerca de cuál es el verdadero. En una primera aproximación, el trance se ha solventado a base de recurrir a la memoria y ver si alguno de los significados aparece previamente. Sin embargo, queda algo mucho más difícil de controlar, la llamada ambigüedad de estructura profunda: dos frases iguales pero con distinto significado. Este problema es el que se intenta resolver y mientras se consigue veremos traductores humanos en las conferencias internacionales.

Las variables de solo lectura no pueden ser modificados.

La función del PATH es almacenar los directorios especificando de esta forma el orden en el que han de ser rastreados; dicho orden es el correspondiente a los nombres de los diferentes directorios contenidos en el PATH, leídos de izquierda a derecha.

$ PATH = : /usr/Juan : /usr/bin : /bin

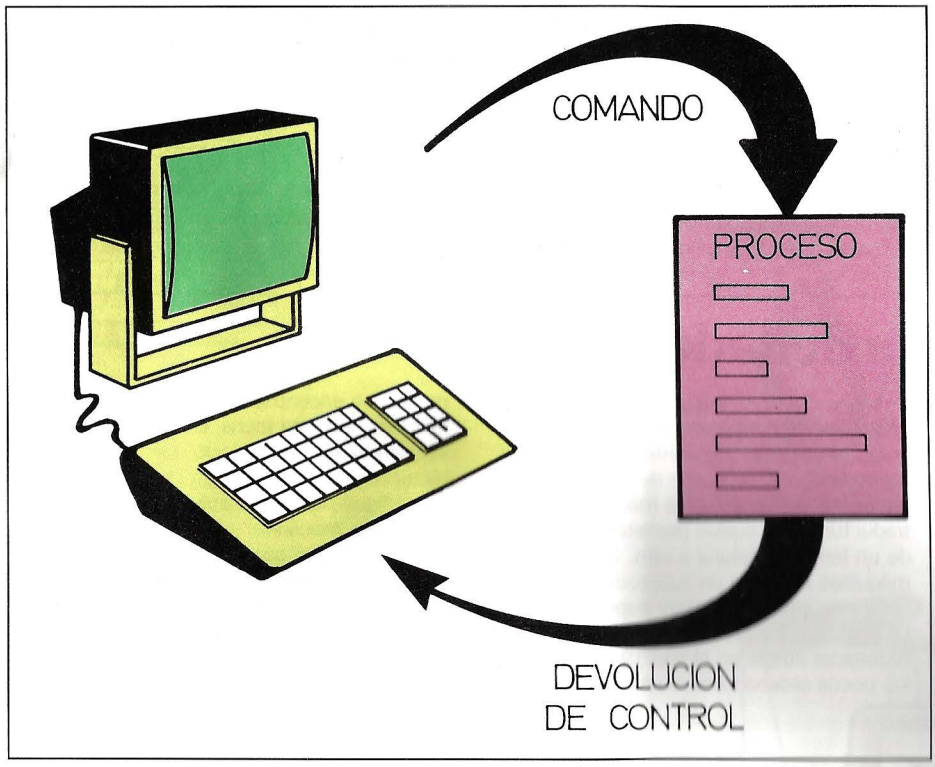
Si a cada usuario se le asigna un valor de PATH diferente, cada uno ejecutará un programa distinto aunque introduzca la misma orden.

- *Variables PS1 y PS2*

Cuando el Shell está en disposición de recibir un comando presenta un signo en la pantalla que revela este estado. Este signo no es fijo, sino que puede ser variado por el usuario y es almacenado en forma de cadena en la variable PS1. Cada vez que el valor de esta variable se modifique así lo hará el prompt del sistema. Caso de que un comando no esté completo en una línea y haya necesidad de continuar en la siguiente se produce, aparecerá en ésta el prompt o indicador secundario. El prompt secundario se almacena en la variable PS2 y, al igual que sucede con la variable PS1, cada vez que es modificada el prompt secundario cambia en consecuencia.

- *Variable IFS*

Normalmente, cuando se escribe un comando o sus argumentos se emplean espacios en blanco para separar los mismos. Sin embargo, por medio de la variable IFS es posible hacer que el símbolo almacenado en la misma se convierta en un separador de campos. Así, si IFS se hace igual a ":", un conjunto como pudiera ser a:b:c:d se convierte en cuatro datos distintos (a,b,c,d) en lugar de un conjunto de caracteres único (a:b:c:d).



Cada vez que se lanza la ejecución de un comando se inicia un proceso.

continuando con los directorios / bin y / usr / bin. En el caso de que la búsqueda sea infructuosa en estos directorios, el Shell informaría el usuario de que no ha podido localizar el programa para que éste pueda tomar la acción correctiva pertinente.

PROCESOS

Los procesos son los medios a través de los cuales el sistema operativo UNIX ejecuta los comandos, iniciándose un proceso cada vez que se da una orden, aunque esto también puede suceder a instancias del sistema operativo.

La estructura de los procesos en el sistema operativo UNIX es jerárquica, al igual que las estructuras de ficheros. Existe una raíz que está constituida por un proceso de "login" para terminal, que se inicializa cuando el usuario se conecta al terminal. Cuando se da un comando al sistema operativo, éste genera un proceso hijo para ejecutar el mismo y, mientras tanto, el proceso que podríamos denominar padre permanece en estado durmiente o inactivo. Cuando el proceso hijo finaliza, el proceso padre despierta y toma el control, indicando al usuario que está dispuesto para recibir nuevas instrucciones. En el caso de que un proceso deba actuar sin intervención del usuario, se genera el proceso hijo, pero sin pasar el proceso padre a estado durmiente y manteniéndose activo. Para lograr un control adecuado, el sistema operativo UNIX asigna a cada proceso un número por medio del cual los procesos se identifican en tanto en cuanto estén operativos.

Auto CAD (1)

Un programa para el diseño asistido por ordenador

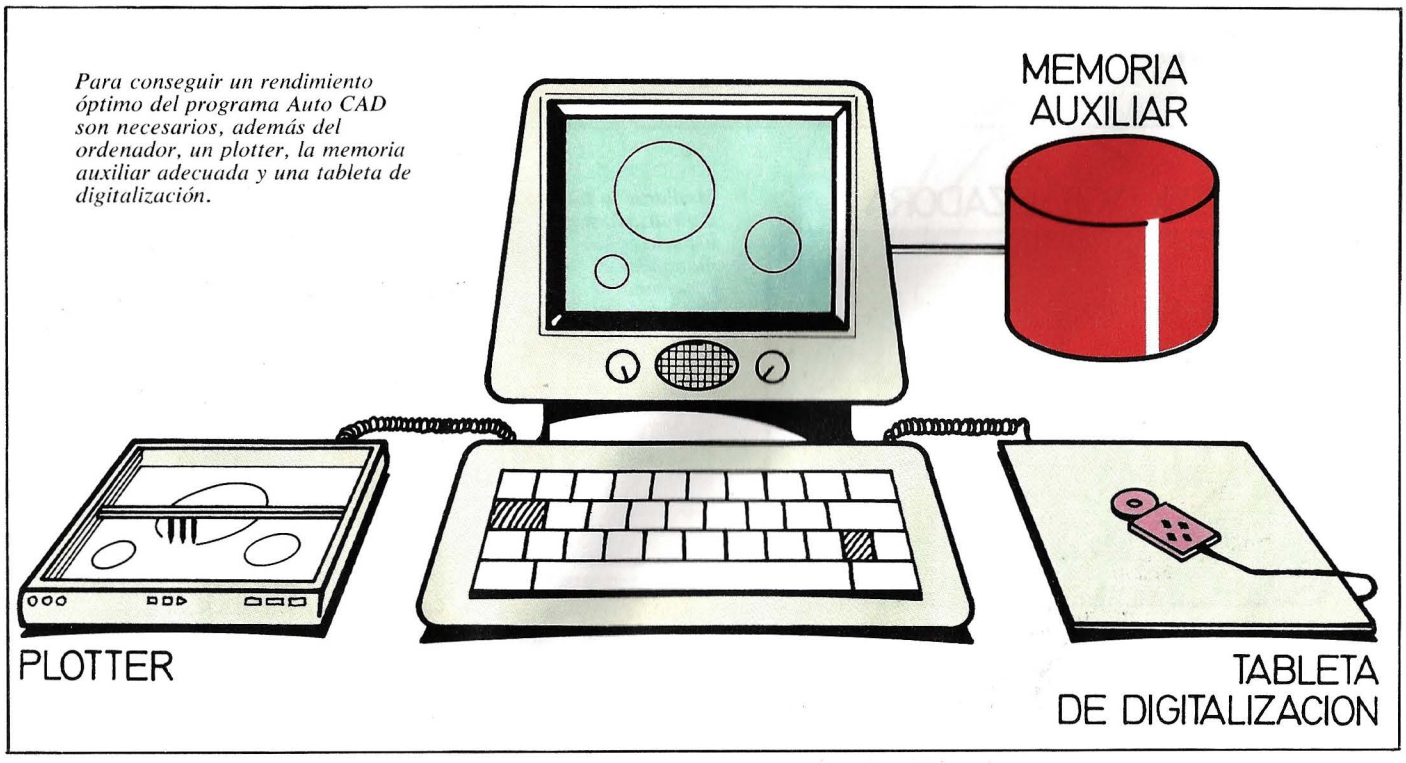
Dentro de los programas dedicados a la producción de gráficos podemos distinguir dos grandes grupos perfectamente diferenciados: los destinados a la obtención de diagramas de gestión y los destinados a la producción gráfica en general. En otras páginas de esta misma sección ya se han estudiado paquetes gráficos de gestión; en cambio, hasta ahora, no se había descrito ningún sistema CAD (Computer Aided Design). Así es como se suele denominar a los paquetes gráficos de carácter general. En este capítulo comenzaremos a detallar el funcionamiento de Auto CAD; una aplicación que, como su propio nombre indica, encaja plenamente como programa de esta naturaleza.

CARACTERISTICAS GENERALES

Tradicionalmente, el diseño gráfico realizado por ordenador ha estado reservado a grandes equipos que, en algunos casos, se especializaban únicamente en esta tarea, resultando inútiles para otros procesos de tipo convencional. En la actualidad se ha conseguido simplificar al máximo la estructura de estos programas, de forma que pueden funcionar en ordenadores no especializados e incluso en ordenadores personales.
Auto CAD es un ejemplo de esta simplificación conceptual. La compañía de software Auto-desk Inc. tiene el copyright de este potente programa gráfico utilizable en equipos IBM/PC y compatibles.
Dentro de las posibles aplicaciones de auto-CAD, podemos destacar las siguientes:

- Diseños de todo tipo aplicados a la Arquitectura.
- Producción de planos para el diseño de interiores.
- Diagramas de flujo y organizativos.
- Diseño de ingeniería Electrónica, Química, Civil y Mecánica.
- Representación de funciones matemáticas.
- Diseños de dibujo artístico.
- Y, en general, producción de cualquier tipo de gráficos.

Para conseguir un rendimiento óptimo del programa Auto CAD son necesarios, además del ordenador, un plotter, la memoria auxiliar adecuada y una tableta de digitalización.



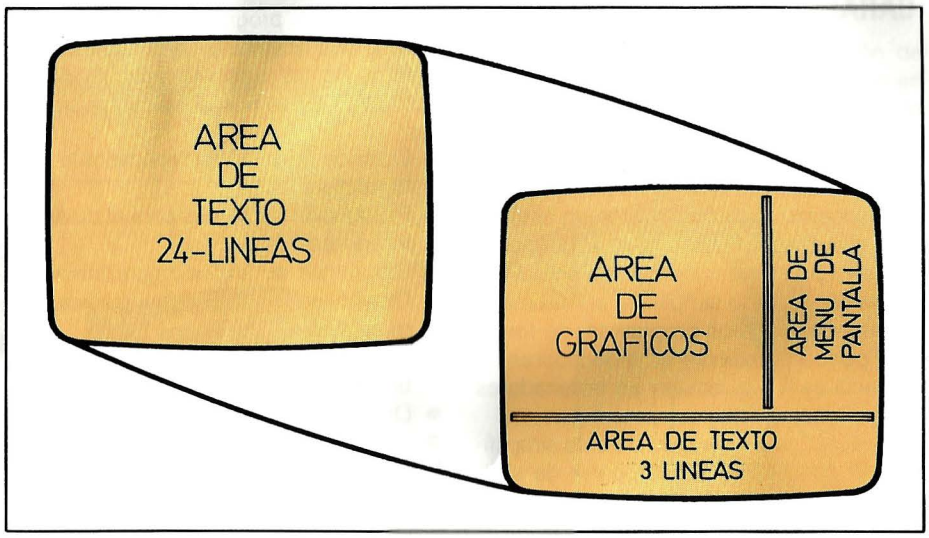
Evidentemente, además de los componentes tradicionales de cualquier ordenador personal, para obtener un máximo beneficio de Auto CAD resulta necesario disponer de una pantalla con buena resolución y un plotter para "imprimir" los diseños. En principio, las pantallas estándar de los ordenadores personales pueden servir para la explotación de programas CAD, aunque en algunos casos resulta interesante incorporar pantallas especiales (con más resolución y con varios colores). En cuanto al plotter, puede servir cualquiera aunque, evidentemente, cuanto más calidad tenga el plotter, mejores reproducciones se obtendrán.

CONCEPTOS BASICOS

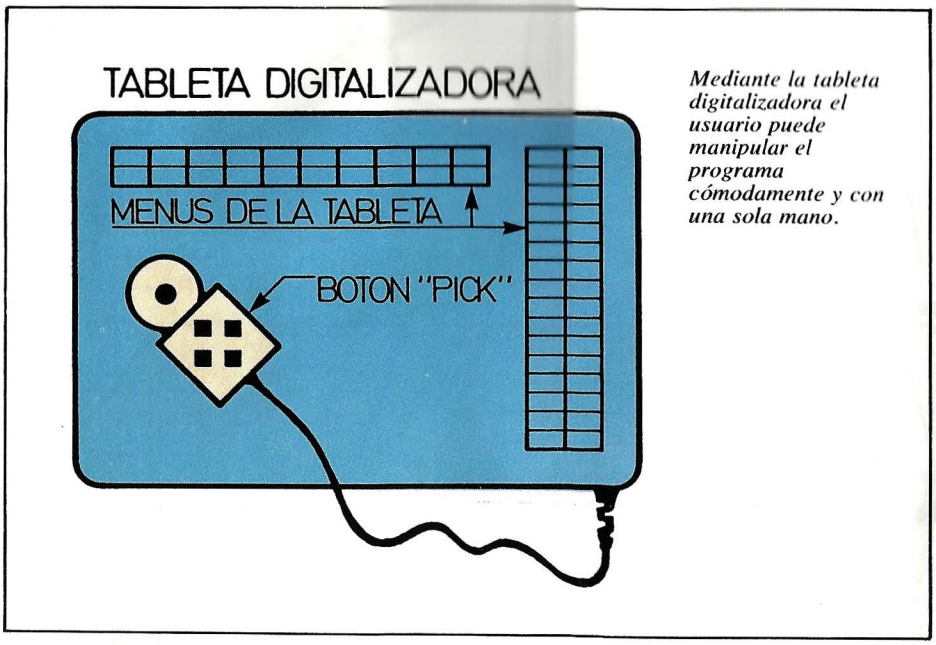
Auto CAD provee de dos útiles para el diseño: entidades y comandos.
Las entidades se utilizan para construir los gráficos y pueden ser definidas como elementos dibujables, es decir: líneas, círculos, textos, etc. Por su parte, los comandos permiten al usuario manejar las entidades y, por lo tanto, ubicarlas y dimensionarlas convenientemente para obtener el resultado deseado. La ejecución de un comando se puede desencadenar de dos formas diferentes:

- Tecleando su denominación y pulsando la tecla RETURN.
- Pulsando un botón sobre una tableta digitalizadora.

En ambos casos de selección de los comandos se realiza sobre un menú que el programa escribe en una zona de la pantalla reservada al efecto. A veces la ejecución de un comando implica la necesidad de "entrar" algunos parámetros, para ello el programa mostrará un mensaje solicitando la introducción del valor.
A continuación, el usuario, bien sea mediante el teclado o bien mediante la tableta, contestará a la pregunta. Por ejemplo, si se utiliza un comando para dibujar un círculo, Auto CAD solicitará al usuario información sobre el punto central y el radio (o el diámetro).
Los parámetros solicitados pueden ser de distintos tipos: puntos del plano, cantidades, tamaños, ángulos rotación, etc. En cualquier caso, después de haber ejecutado un comando y haber contestado a todas las preguntas, en el monitor se dibujará la entidad deseada. A partir de ese momento se puede ejecutar otro comando que dibuje otra entidad sobre el monitor, y así sucesivamente hasta tener finalizado el diseño.
Auto CAD dispone también de algunas funciones para la modificación de entidades ya dibujadas; así, una entidad puede ser borrada, desplazada de lugar, copiada muchas veces...



El programa Auto CAD divide la pantalla del ordenador en tres áreas distintas: una para los menús, otra para los mensajes y una última para los gráficos.



Mediante la tableta digitalizadora el usuario puede manipular el programa cómodamente y con una sola mano.

AYUDAS DE AUTO CAD

Manejando las entidades, los comandos y las funciones descritas de los párrafos anteriores, el usuario puede diseñar sobre la pantalla del ordenador prácticamente cualquier tipo de dibujo.
No obstante, el programa incluye una serie de ayudas que facilitan la misión del operador, entre ellas podemos destacar un sistema de menús que orientan continuamente al usuario. El menú principal de

este sistema ofrece ocho posibilidades distintas:

0—Finalizar la Sesión de Trabajo.
1—Comenzar un nuevo dibujo.
2—Editar un dibujo ya existente.
3—Imprimir un dibujo mediante el plotter.
4—Configurar Auto CAD.
5—Utilidades para ficheros.
6—Compilar un fichero.
7—Convertir un fichero.

En este caso, el menú principal ocupará toda la pantalla, pero el resto de los menús del sistema ocuparán la parte lateral derecha del monitor, la zona inferior estará reservada para mensajes y el resto de la pantalla se reservará como zona gráfica. Otra de las ayudas más poderosas, aunque no imprescindible, para explotar Auto CAD, consiste en una tableta de digitalización. Mediante un dispositivo situado sobre la tableta, el usuario puede dibujar cómodamente y con una sola mano. Sin más que desplazar el dispositivo sobre la tableta podrá seleccionar puntos de la pantalla o comandos del menú; de esta forma resulta extremadamente cómodo dibujar ya que las operaciones a realizar por el usuario son similares a las que efectuaría para dibujar manualmente. No obstante, en algunos casos, aún disponiendo de tableta digitalizadora, se utiliza el teclado para la entrada de algunos parámetros; esto es debido a la mayor precisión obtenida de esta manera. Si, por ejemplo, deseamos trazar una recta entre los puntos del plano (3.1752,−4.9145) y (−7.3195,−2.1763), evidentemente la mejor forma es introducir los valores mediante el teclado en vez de digitalizar ambos puntos.

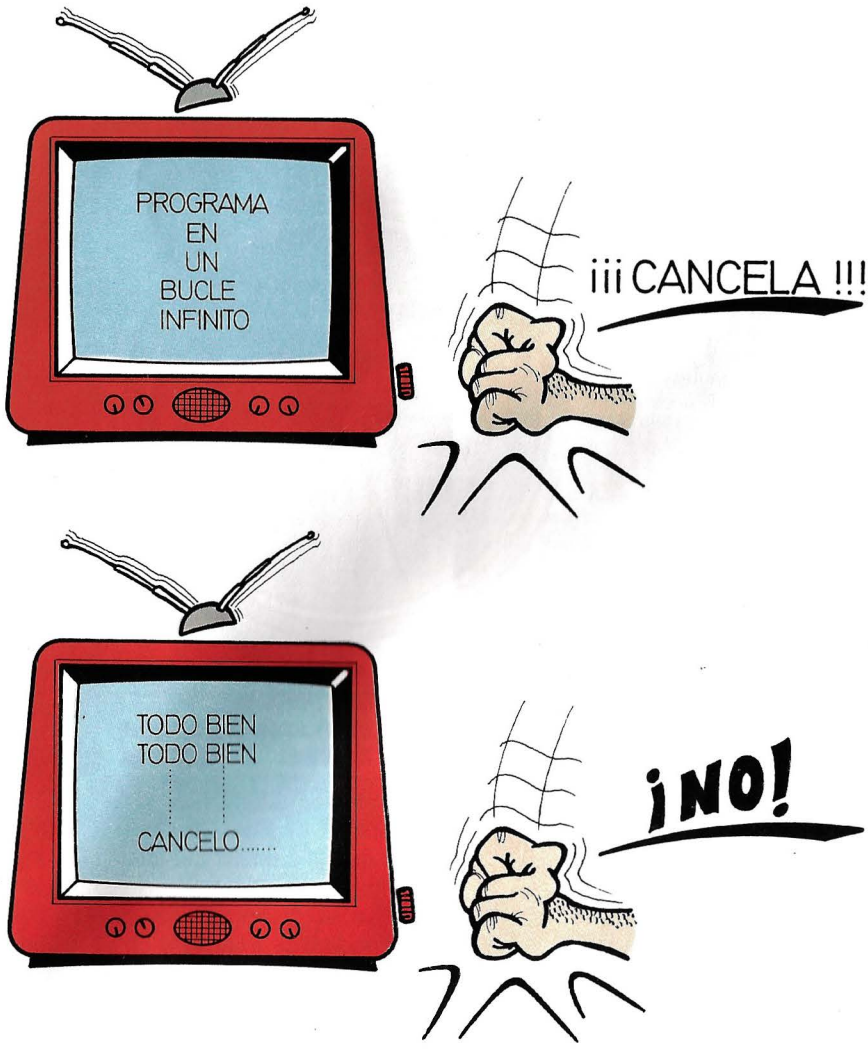
"Cancelar", palabra odiada y amada

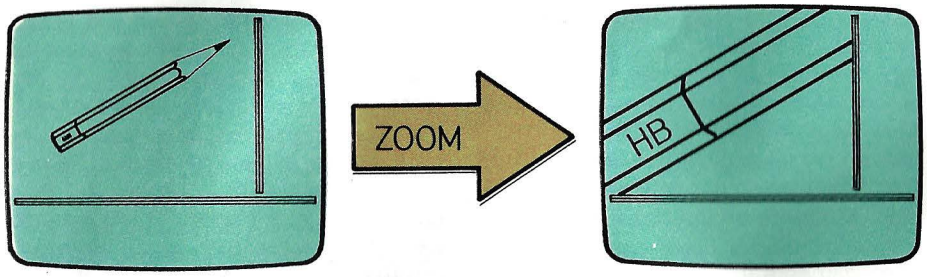
Si nos detenemos ante un usuario de informática que esté trabajando delante de un terminal, seguro que le oiremos gritar dos frases aparentemente contradictorias:
1. "¿Es que este programa no va a cancelar nunca?"
2. "¿Por qué me habrá cancelado este programa?"

Desde luego hemos suavizado las frases; en su versión original contendría ciertas palabras mal sonantes acompañadas de pequeños golpes o empujones a la pantalla del ordenador.
En efecto, la cancelación de un programa en ejecución puede ser deseada en ciertos momentos y odiada en otros. Veamos un ejemplo de cada caso:

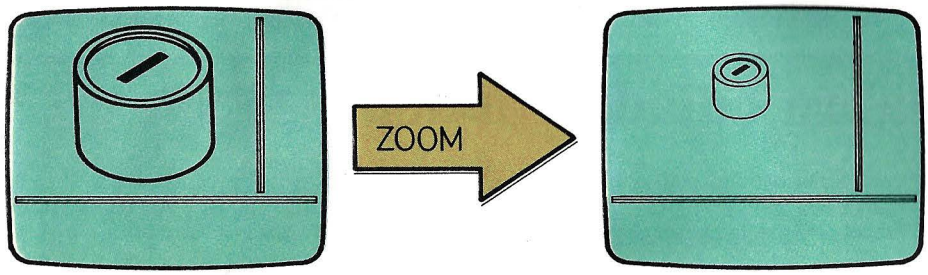
1. Supongamos que nos encontramos trabajando con un ordenador, ajeno y se nos cobra cierto dinero por cada segundo en el que nuestro programa reside en la CPU. Suponga también que después de haber ordenado la ejecución del programa nos damos cuenta de que los datos de entrada son erróneos; o aún peor, que debido a los datos, el programa entra en un bucle infinito.
Sin duda desearemos con vehemencia que el programa "cancele".

2. Cambiemos un poco la situación. Seguimos en el mismo ordenador donde nos facturan por segundo de CPU consumida, y en este caso estamos ejecutando un programa muy largo y, consecuentemente, muy caro de ejecutar. El programa ha finalizado ya el 98% de los procesos que debe realizar y por los mensajes visualizados en la pantalla sabemos que todo va bien; ya sólo le queda por realizar un último cálculo para que nos muestre el dato que esperamos con ansiedad...En ese preciso instante y por un motivo desconocido, el programa "cancela".
Probablemente, la cancela provocará un ataque de histeria en el usuario haciéndole comportarse como un auténtico grosero, aunque su educación sea exquisita.



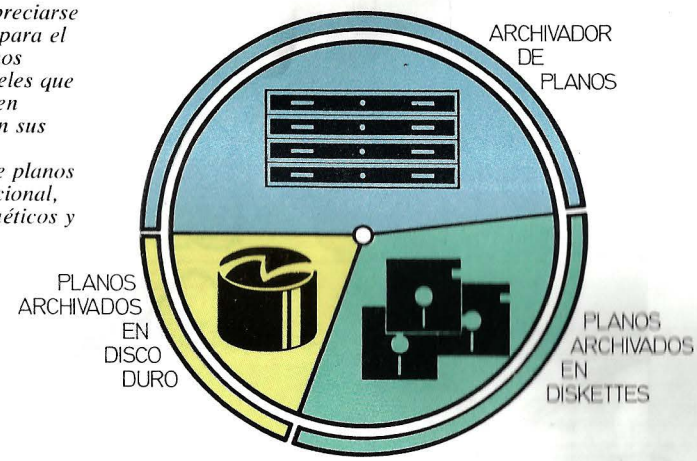


El comando ZOOM puede utilizarse para ampliar parte de una figura, de esta forma se consigue ver con gran detalle las distintas entidades incluidas en la zona ampliada.



También puede utilizarse el comando ZOOM para disminuir el tamaño de una figura. En resumen, se puede afirmar que ZOOM modifica escalas en ambos sentidos: positivo y negativo.

Como puede apreciarse en esta figura, para el archivo de planos existen tres niveles que citamos en orden creciente, según sus prestaciones: archivadores de planos de tipo convencional, disquetes magnéticos y discos duros.



REMATES FINALES

Para finalizar esta introducción al programa gráfico Auto CAD, vamos a describir las restantes características que le dotan de una gran potencia de dibujo:

- ZOOM

Normalmente los planos o diseños a producir con Auto CAD son mucho más grandes que la pantalla del ordenador. Por lo tanto, si no se desea visualizar todo el diseño simultáneamente, la escala hará que todas las entidades se vean reducidas; en algunos casos esto no origina ningún problema, pero en la mayoría sí, ya que resulta imposible dibujar con cierta precisión al tener que hacerlo en tamaños reducidos. En esta situación se puede utilizar en el comando ZOOM, cuyo objetivo consiste en aumentar determinadas porciones del plano de forma que se pueda trabajar con gran precisión. Mediante este comando los usuarios pueden preparar distintas vistas del plano a producir y trabajar con detalle en cada una de ellas. La potencia de este comando es tan grande que en algunos casos se lleva de tal forma el aumento que en la pantalla sólo se visualizará lo que en el plano final apenas ocupará unos pocos centímetros cuadrados.

- PLOT

Otro de los comandos fundamentales de Auto CAD es el que permite producir copias en papel de los diseños realizados mediante el ordenador. La sencillez del comando PLOT resulta de gran importancia.

- ALMACENAMIENTO

Ya hemos comentado como se diseña un dibujo y cómo se obtienen copias de mismo en papel. El último proceso que tradicionalmente se realiza con un plano en su almacenamiento. Para ello se pueden utilizar enormes archivadores en los que se amontonarán los planos; si bien a los pocos meses de haber sido archivado un plano su localización será complicada, a los pocos años de localización será casi imposible y, en el caso de tener la fortuna de dar con el plano, nos encontraremos con un documento amarillento y arrugado. Hay que admitir que la anterior "historia" sobre el archivo manual de planos, está un poco exagerada; no obstante, no cabe duda de que el almacenamiento de los planos en memorias auxiliares de un ordenador resulta mucho más cómoco y seguro. Auto CAD permite que el usuario cargue en un disco rígido o en un disquete los diseños que haya realizado durante la sesión de trabajo, de esta forma su localización queda garantizada; el espacio necesario como archivo se reduce notoriamente y, por muchos años que pasen, los planos jamás se arrugarán.

En el próximo capítulo se estudiarán con más detalle los principales comandos de Auto CAD. Debido a la falta de espacio, la lista de comandos no podrá ser exhaustiva, si bien, intentaremos destacar los más sencillos y frecuentes para la producción de diseños.

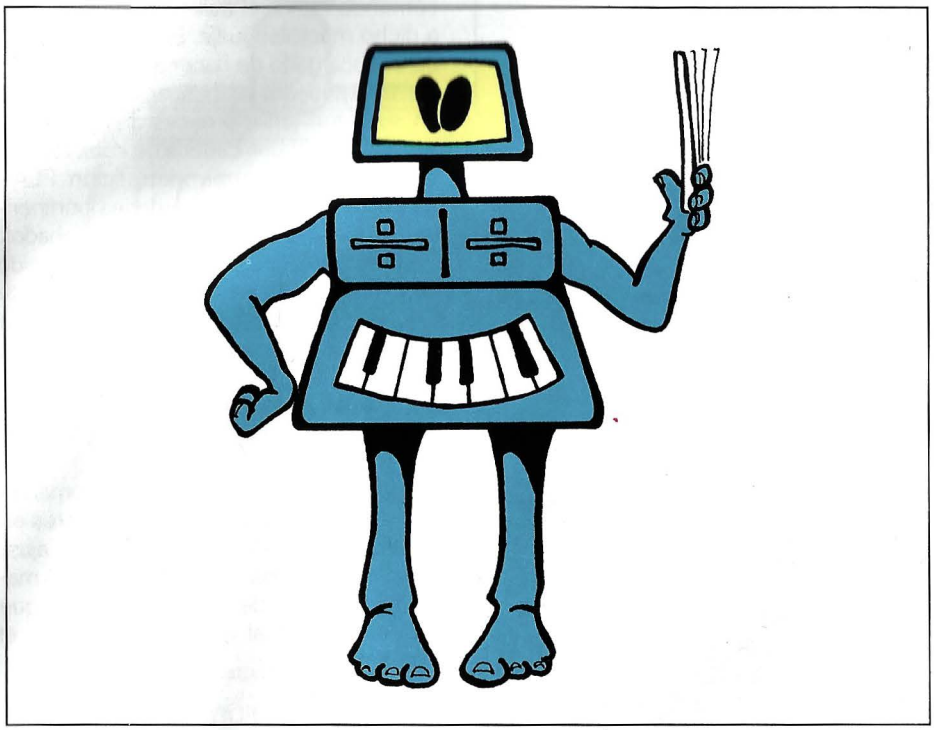
El sonido en BASIC

El macrolenguaje musical

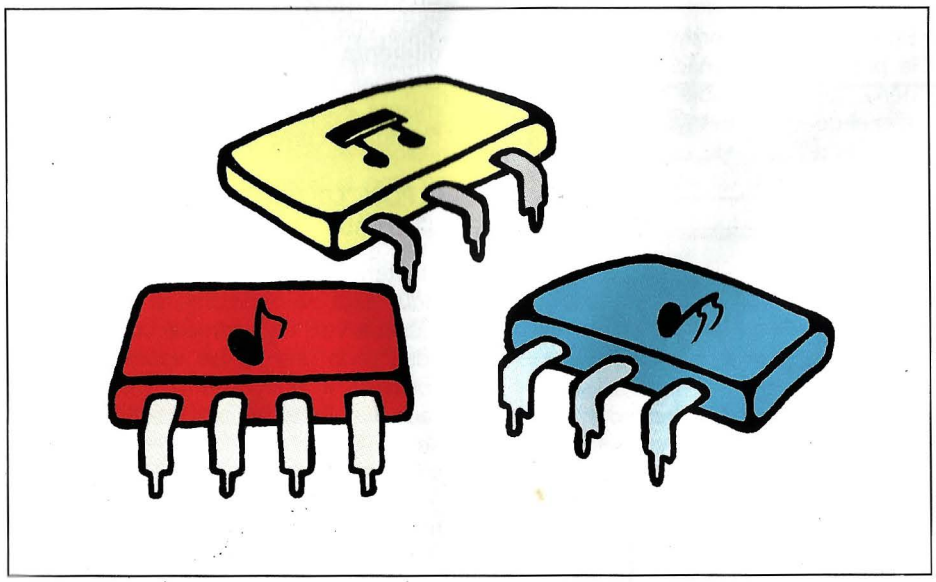
En un capítulo precedente se inició el tema de la generación de sonidos en el ordenador. En él se comentaron los comandos más simples, habitualmente utilizados para gobernar dicha facultad. Aquellos comandos se limitaban a acceder directamente al generador de sonido del ordenador. A pesar de ser el método más habitual, el acceso al generador de sonido complica excesivamente el uso de las capacidades sonoras del aparato.

El control del sonido, al igual que el de los gráficos, no posee un tratamiento estandarizado en todos los dialectos del BASIC. La razón hay que achacarla a los distintos criterios que adoptan los fabricantes a la hora de elegir el circuito generador de sonido. Dicho circuito suele admitir un número limitado de órdenes elementales que son, precisamente, las indicadas en el primer capítulo dedicado al sonido.

El circuito sonoro del ordenador puede ser programado para producir una gran variedad de tonos y melodías. Algunos aparatos incorporan un pequeño repertorio de sonidos preprogramados. Estos sonidos son accesibles mediante sencillos comandos BASIC, con lo que el empleo de la "voz del ordenador" se simplifica en gran medida.



El presente capítulo está dedicado a una peculiar forma de utilizar sonidos preprogramados. Se trata del denominado "macrolenguaje musical", presente en la mayor parte de los ordenadores que emplean el BASIC de Microsoft. El macrolenguaje musical es una potente herramienta para la creación de las más variadas melodías.



El manejo del sonido en el ordenador depende en gran medida del circuito especializado que incluya el equipo.

CORRESPONDENCIA ENTRE LAS NOMENCLATURAS DE LA ESCALA CROMATICA

DO	C
RE	D
MI	E
FA	F
SOL	G
LA	A
SI	B
DO	C
RE	D
MI	E
FA	F
SOL	G

SEPARACION ENTRE NOTAS

DO-RE	un tono
RE-MI	un tono
MI-FA	un semitono
FA-SOL	un tono
SOL-LA	un tono
LA-SI	un tono
SI-DO	un semitono

ESCALA DE SEMITONOS

DO
—
RE
—
MI
FA
—
SOL
—
LA
—
SI
DO

Su estructura y forma de uso es similar a la del comentado macrolenguaje gráfico.

MACROLENGUAJE MUSICAL Y COMANDO PLAY

El macrolenguaje musical va asociado al comando PLAY, el cual permite el acceso a dicho macrolenguaje. El comando PLAY es el encargado de hacer que se ejecuten las órdenes del macrolenguaje incluidas en su argumento; órdenes que se expresan dentro de una cadena de caracteres.
La ejecución de una instrucción PLAY hace que se envíen las órdenes pertinentes al circuito de sonido del ordenador. Una vez recibidas, el generador de sonido se pone a trabajar independientemente del resto del ordenador. Esto significa que mientras se genera el sonido se puede seguir ejecutando el programa principal. Con ello se evita que la ejecución del programa se vea retardada por efecto de la emisión de sonido.
Como ya se ha comentado, el comando PLAY admite una cadena de caracteres en su argumento. Dicha cadena debe ajustarse a las normas impuestas por el macrolenguaje. En definitiva, el formato que ha de mostrar el comando PLAY es el siguiente:

(Número de línea) PLAY <cadena1>
[, <cadena2>...]

En el formato general de PLAY se aprecia la posibilidad de añadir más de una cadena. Ello es posible cuando el aparato con el que se está trabajando dispone de más de un canal de sonido; esto es, permite emitir más de una voz al tiempo. En algunos ordenadores es corriente el empleo de hasta tres canales de voz, lo que hará posible la creación de una melodía con hasta dos acompañamientos.

DURACIONES DE LAS NOTAS

Símbolo	Duración	Selección con el comando L
redonda	duración unidad	L1 (duración unidad)
blanca	media redonda	L2 (1/2 redonda)
negra	media blanca (1/4 redonda)	L4 (1/4 redonda)
corchea	media negra (1/8 redonda)	L8 (1/8 redonda)
semicorchea	media corchea (1/16 redonda)	L16 (1/16 redonda)
fusa	media semicorchea (1/32 redonda)	L32 (1/32 redonda)
semifusa	media fusa (1/64 redonda)	L64 (1/64 redonda)

PRIMEROS PASOS: LAS NOTAS

El macrolenguaje musical de Microsoft adopta una filosofía que será del agrado de aquellos que posean algunos conocimientos de solfeo. En primer lugar, los sonidos se indican por sus correspondientes notas. De todos son conocidas las siete notas de la escala musical: DO, RE, MI, FA, SOL, LA y SI. Sin embargo, esta no es la nomenclatura utilizada aquí. El macrolenguaje musical emplea la signatura aglosajona, que tiene una correspondiencia biunívoca con la anterior. Esta nomenclatura hace uso de las siete primeras letras del abecedario inglés (las mismas del castellano sin la "ch"). La relación entre ambas formulaciones se muestra en la correspondiente tabla.
Como se puede observar en la tabla adjunta, la nueva nomenclatura comienza en la nota LA, siguiendo luego en el orden habitual. Así pues, para la interpretación de una escala de DO a SI basta con ejecutar la línea que se indica a continuación.

```
10 PLAY "CDEFGAB"
```

En el caso de disponer de varios canales, cada uno de ellos admitirá una cadena diferente. En ese supuesto, las distintas cadenas irán separadas por comas. La misma escala entonada a tres voces se formularía como sigue:

```
10 PLAY "CDEFGAB", "CDEFGAB",
"CDEFGAB"
```

Al interpretar las tres voces la misma melodía, ésta se oirá como si se tratara de una sola voz. Para apreciar con mayor claridad cada una de las voces o canales, conviene indicar distintas melodías para cada una. Ejecutando el siguiente ejemplo se emitirán notas diferentes por cada canal:

```
10 PLAY "CDEFGAB", "DEFGABC",
EFGABCD"
```

SOSTENIDOS Y BEMOLES

Las siete notas de la escala cromática no son las únicas existentes. En realidad, la separación entre las sucesivas notas sigue una regla un tanto extraña. La diferencia de frecuencia entre dos notas recibe el nombre de tono o semitono dependiendo de las notas que sean. Un tono indica una separación en frecuencia fija, de la cual el semitono es justamente la mitad. Para aclarar conceptos será necesario consultar la tabla de separación entre notas.

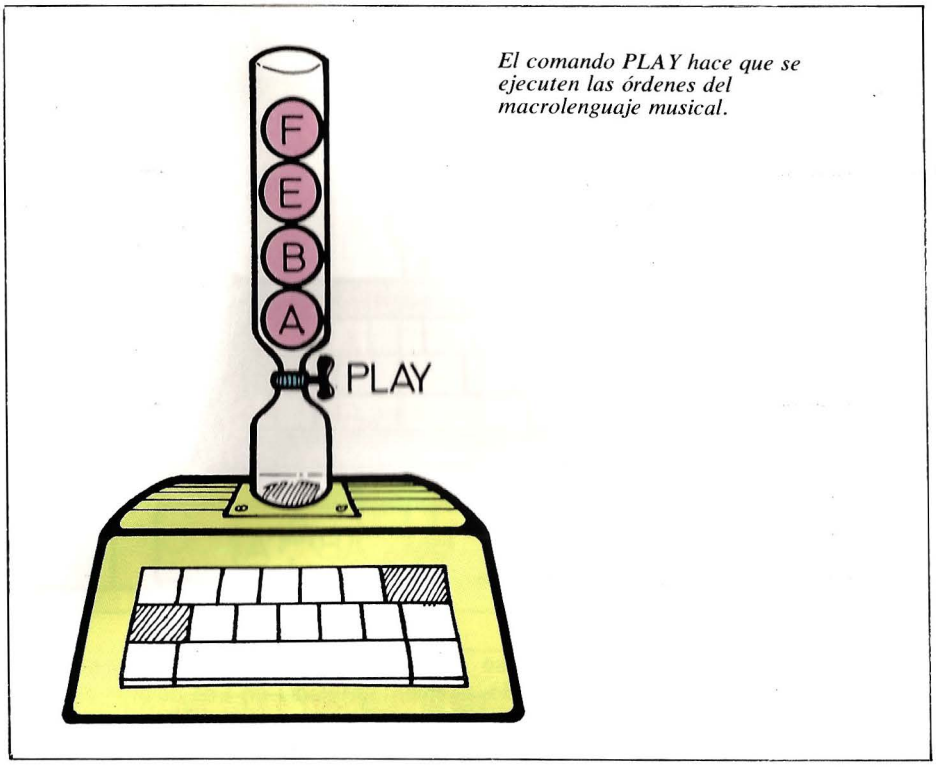
Dicho de otra forma, si se separan las notas tomando como unidad el semitono, quedarán huecos libres. Téngase en cuenta que un tono equivale a dos semitonos. La tabla adjunta muestra la escala de semitonos completa.

En ella se observa que quedan posiciones sin ocupar después de cada nota, excepto entre MI/FA y SI/DO. Estos puntos, marcados en la tabla con guiones, corresponden a notas intermedias. Dichas notas se nombran en relación a la nota anterior o posterior. Así, el primer guión de la tabla corresponde al DO sostenido, el segundo al RE sostenido, etc. Esta nomenclatura hace referencia a la nota inmediatamente anterior. Si se desea referenciar las notas intermedias en relación a las que las siguen, se utiliza el término 'bemol''. De esta forma, el primero guión se puede denominar DO sostenido o RE bemol, mientras que el segundo corresponde tanto al RE sostenido como al MI bemol.

Las referidas notas intermedias también pueden ser interpretadas mediante el macrolenguaje musical. Para ello se emplean los signos + y – detrás de la nota en cuestión. Por ejemplo, el mencionado primer guión se formularía como C+ (DO sostenido) o D– (RE bemol). Por supuesto, existen dos notas que no admiten sostenido: MI y SI, ya que sus sostenidos corresponden a FA y DO respectivamente. De la misma forma, ni FA ni DO admiten el correspondiente bemol.

Empleando estas notas intermedias se puede recorrer la escala completa por medio de la siguiente instrucción:

```
10 PLAY "CC+DD+EFF+GG+AA+B"
```

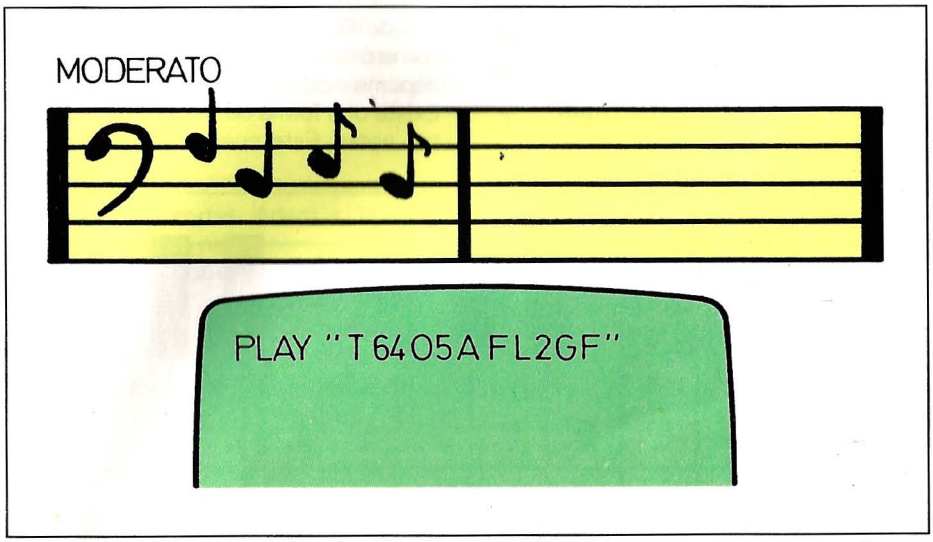


El comando PLAY hace que se ejecuten las órdenes del macrolenguaje musical.

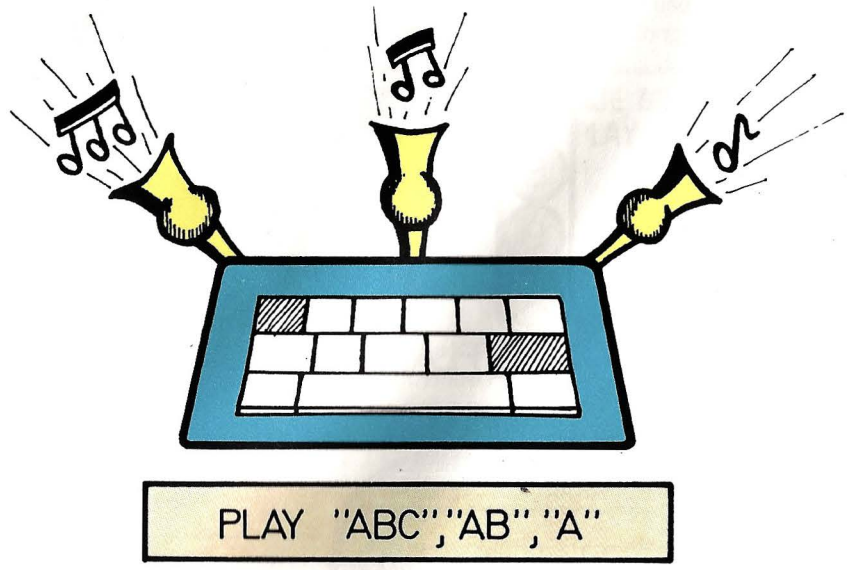
MAS ESCALAS: LAS OCTAVAS

Hasta ahora se ha visto la forma de utilizar las notas de una escala. Se sabe que, tras la última nota de una escala, SI, viene el DO de la escala siguiente. En realidad, la secuencia de notas se puede repetir infinitamente, tanto hacia arriba (sonidos agudos) como hacia abajo (sonidos graves). El único límite lo impone la capacidad auditiva del oído humano.

El oído medio es capaz de percibir una gama de frecuencias que supera con creces la extensión de la escala básica. Por ello se añaden más escalas, identificán-



El macrolenguaje musical adopta una nomenclatura muy cercana al cifrado clásico de la música.



Algunos aparatos permiten el uso simultáneo de tres voces o canales. Ello se consigue añadiendo 3 cadenas en el argumento de PLAY.

dose cada una por la posición en la que se encuentra. Cada escala individual se denomina "octava". Con esto, un sonido determinado se reconoce por la octava en la que se encuentra, y dentro de la octava, por la nota a la que corresponde.

Los modernos ordenadores incluyen circuitos integrados generadores de sonido capaces de abarcar ocho o más octavas. El macrolenguaje musical contempla esta posibilidad, permitiendo elegir la octava. Para ello se emplea el comando-letra 0 seguido del número identificativo de la octava deseada. Cuando no se indica la octava (como en los ejemplos anteriores) se utiliza la octava central del espectro. Como ejemplo se muestra el efecto asociado al uso de diferentes octavas para cada canal:

```
10 PLAY "01CDEFGAB", "04CDEFGAB",
"07CDEFGAB"
```

Esta sencilla línea hará que suene la misma melodía en tres octavas distintas. Cada canal emplea una octava diferente, sonando las tres al tiempo. Con ello se aprecia tanto la diferencia de octava, como la diferencia de canal. Si se desea que suene cada octava separada (empleando una sola voz) se ha de introducir y ejecutar la siguiente línea:

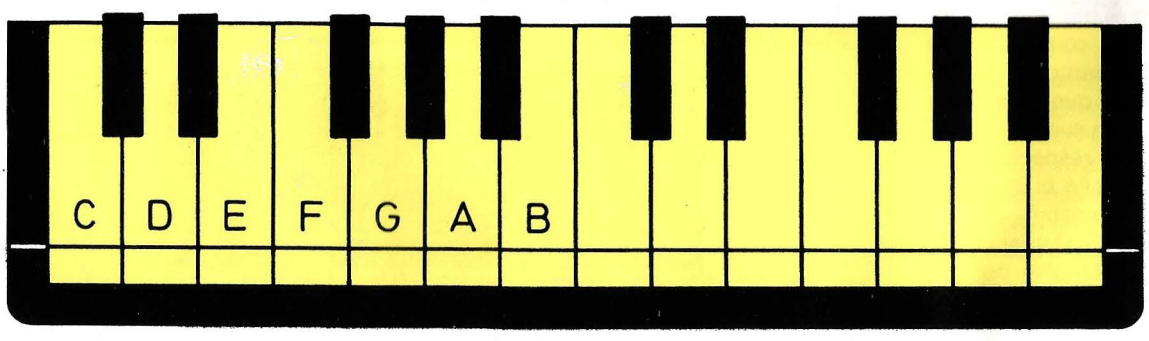
```
10 PLAY "01CDEFGAB04CDEFGAB07CDEFGAB"
```

Tras la ejecución de un comando 0, la octava queda fijada hasta que se vuelva a variar por medio de otro comando 0. Esto quiere decir que si se ejecuta otro comando PLAY a continuación del incluido como último ejemplo, se comenzará en la séptima octava (07).

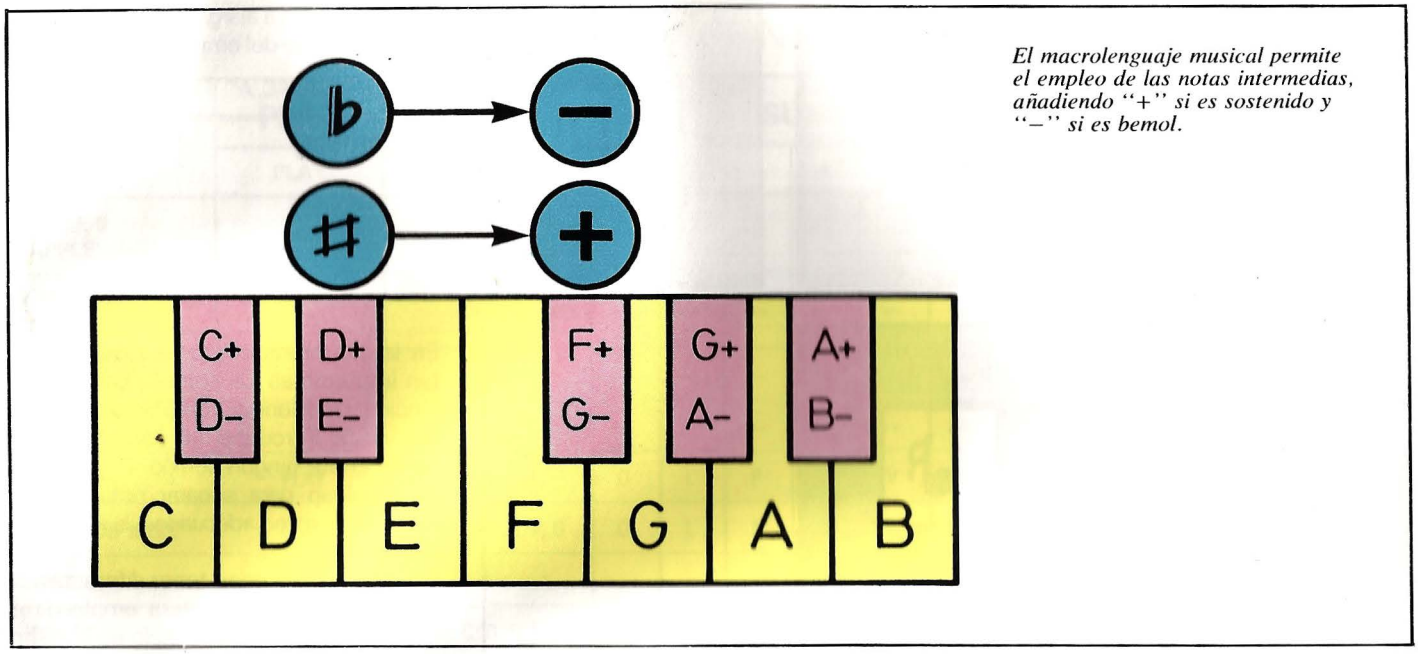
Existe otra forma de acceder a las diferentes notas. Este nuevo método no utiliza las escalas habituales, sino que numera las notas consecutivamente. Contando las notas naturales y las intermedias, cada octava tiene doce notas. Si se dispone de un total de ocho octavas, el número de notas permitidas es de 12×8=96.

Las 96 notas pueden identificarse mediante su número de orden. Ese es precisamente el otro modo de acceder a cada nota. Para el empleo de esta notación se hace imprescindible un nuevo comando-letra. Se trata en este caso de la letra N, que seguida por un número del 0 al 95 permite la emisión de la nota correspondiente. En esta notación, N0 corresponde al DO de la primera octava, y N95 al SI de la octava número ocho.

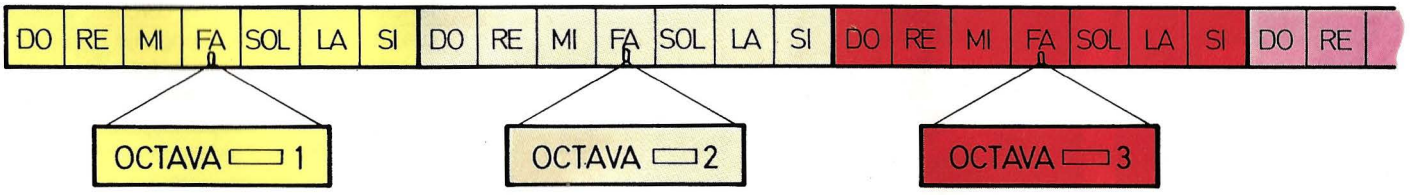
Las dos instrucciones siguientes proporcionan el mismo resultado. La primera emplea la notación habitual, mientras que la segunda hace uso de la notación absoluta:



La notación empleada para identificar las notas es la denominada "anglosajona".



El macrolenguaje musical permite el empleo de las notas intermedias, añadiendo "+" si es sostenido y "−" si es bemol.



La repetición cíclica de las escalas obliga a identificar la octava a la que corresponde una determinada nota.

```
10 PLAY "01CC+DD+EFF+GG+AA+B"
10 PLAY "N0N1N2N3N4N5N6N7N8N9N10N11
   N12"
```

Siguiendo la notación absoluta, el DO de la octava inicial (cuarta octava: 04) se formula como N36.

DURACION DE LAS NOTAS

En la interpretación de una melodía la duración de todas las notas no es la misma. Una nota puede mantenerse sonando más o menos tiempo que las demás. Por regla general, a cada nota de una melodía le corresponderá una duración propia. En terminología musical, las duraciones se especifican como submúltiplos de la duración máxima. Cada duración inferior es exactamente la mitad de la anterior. La nomenclatura empleada para las duraciones de las notas es la que se muestra en la tabla adjunta.

La duración empleada hasta ahora es la de "negra". Esta es la duración que el ordenador toma inicialmente. Para hacer que la duración de una determinada nota aumente puede recurrirse a la repetición de la misma. Por ejemplo, la siguiente línea interpreta DO en negra, RE en blanca y MI en redonda.

```
30 PLAY "CDDEEEE"
```

Sin embargo, este método no permite duraciones más cortas. Además, la repetición de una nota no proporciona el efecto exacto: se percibe la separación entre cada dos notas.

En el macrolenguaje musical está prevista la variación de la duración de las notas. Para ello se emplea la letra-comando L. A continuación de la misma se añade un número que puede variar de 1 a 32. Dicho número indica la fracción de "redonda" que ha de emplearse. La correspondiente tabla relaciona las figuras clásicas con la notación del macrolenguaje.

El comando L permite también especificar duraciones no estándar, como por ejemplo L5 ó L6. A continuación, se muestra un ejemplo en el que la escala se interpreta en corcheas:

```
20 PLAY "L8CDEFGAB"
```

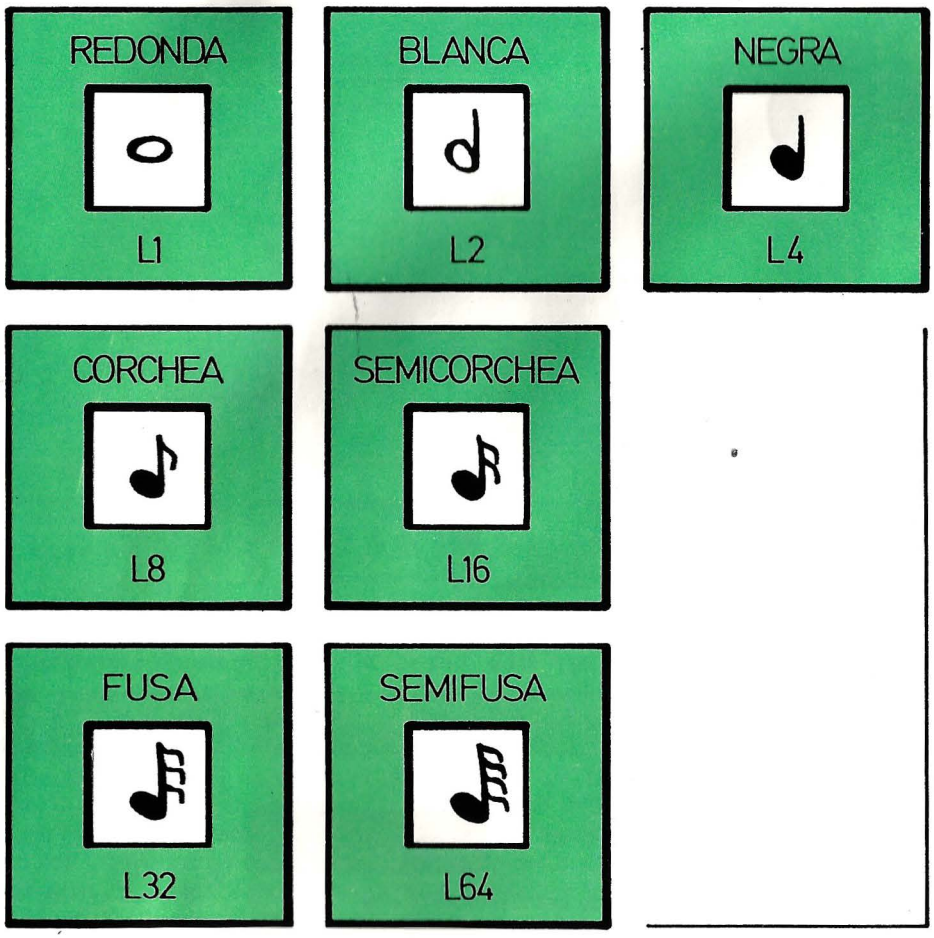
En el ejemplo se ve que la acción del comando L queda fijada. Al igual que el comando que especifica la octava (0) su efecto se extiende hasta el punto en el que se ejecuta otro comando análogo. Así, la línea siguiente:

```
30 PLAY "CDEFGAB"
```

tendrá una duración distinta dependiendo de la instrucción, de entre las siguientes, que la preceda:

```
20 PLAY "L1"
20 PLAY "L4"
20 PLAY "L16"
```

Para hacer que cada nota tenga su propia duración puede optarse por preceder cada una de ellas con el correspondiente comando L. En el siguiente ejemplo se inter-



Repertorio de duraciones estándar de las notas.

preta la escala en corcheas, salvo el FA, que tiene duración de semicorchea:

```
20 PLAY "L8CDEL16FL8GAB"
```

Las tres primeras notas están afectadas por el primer comando L8, siendo por lo tanto corcheas. La nota FA se hace semicorchea por medio de L16. Para que el resto de notas siga con la duración inicial, es preciso restaurar ésta con un nuevo L8. En este caso, el proceso es demasiado engorroso para variar la duración de una sola nota. Afortunadamente, el macrolenguaje permite realizar esto mismo de una forma más sencilla.

Si lo que se pretende es alterar la duración de una sola nota no es preciso el uso del comando L. Como se ha visto, el comando L fija la duración de todas las notas que le siguen. Para especificar la duración de una sola nota basta con indicar esa duración a continuación de la propia nota (se emplea al efecto el número que se utilizaría con el comando L). Haciendo uso de esta nueva facilidad, el anterior ejemplo se reduciría a lo siguiente:

```
20 PLAY "L8CDEF16GAB"
```

Se observa en el ejemplo que la variación en la duración sólo afecta a FA. Las restantes notas seguirán con la duración marcada por el último comando L.

La duración de una nota puede ser variada de una tercera forma. En música se emplean puntos para alargar la duración de una nota. Un punto situado inmediatamente después de la nota alarga ésta en la mitad de su duración. Un segundo punto la alargará en la mitad de la mitad, etc. Esto mismo puede realizarse en el ordenador. El método es idéntico al mencionado: basta con añadir uno o dos puntos tras la nota a alargar. El siguiente es un ejemplo válido del empleo de los puntos:

```
40 PLAY "CAC.AC..A"
```

SILENCIOS

En la interpretación de una melodía son tan importantes los sonidos como los silencios. Un silencio, como su propio nombre indica, introduce una pausa en la que no se emite ningún sonido. Los silencios se emplean para separar notas, dando, además, el ritmo adecuado a la interpretación.

Los silencios pueden tener diferentes duraciones. La nomenclatura empleada en las duraciones de un silencio es la misma que la indicada para las notas. Así, un silencio puede durar una corchea, una negra, etc. En el macrolenguaje musical, el silencio se indica con la letra R seguida por un número. El número que acompaña a este comando adopta las mismas reglas que el parámetro asociado al comando L. A continuación, se muestra un ejemplo del uso de silencios:

```
10 PLAY "CDEE"
20 PLAY "CR64DR64ER64E"
```

Las dos melodías poseen las mismas notas con idénticas duraciones. Sin embargo, el empleo de silencios en la segunda hace que las notas se interpreten ligeramente separadas. Concretamente, las dos últimas notas parecen una sola en la primera ejecución.

VOLUMEN Y MOVIMIENTO

Existen dos características que repercuten en la totalidad de la melodía a interpretar. Estas son el volumen y el movimiento. El volumen se refiere a la intensidad del sonido. Por regla general, el volumen se mantendrá fijo durante toda la emisión sonora. Esto significa que bastará con definirlo al principio.

Por otra parte, el movimiento indica la ve-

TABLA DE CONVERSION

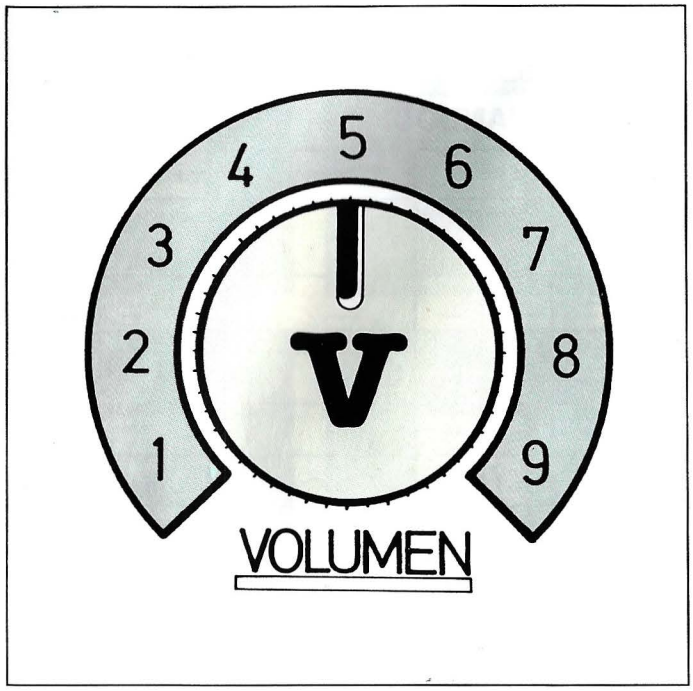
ORDENADOR	PLAY	SUBCOMANDOS								
	PLAY	A, B, C, D, E, F, G	O	L	R	N	V	T	X <var>;	<com>=<var>;
APPLE II (APPLESOFT)	—	—	—	—	—	—	—	—	—	—
APRICOT (M-BASIC)	—	—	—	—	—	—	—	—	—	—
ATARI	—	—	—	—	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—	—	—	—	—
DRAGON	PLAY	A, B, C, D, E, F, G	O	L	P	—	V	T	X <var>;	—
EQUIPOS MSX	PLAY	A, B, C, D, E, F, G	O	L	R	N	V	T	X <var>;	<com>=<var>;
HP-150	—	—	—	—	—	—	—	—	—	—
IBM PC	PLAY	A, B, C, D, E, F, G	O	L	P	N	V	T	X <var>;	<com>=<var>;
MPF	—	—	—	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—	—	—	—	—
ORIC	—	—	—	—	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—	—	—	—
SPECTRAVIDEO	PLAY	A, B, C, D, E, F, G	O	L	R	—	V	T	X <var>;	<com>=<var>;
ZX-SPECTRUM	—	—	—	—	—	—	—	—	—	—

locidad a la que se toca la melodía. Dicha velocidad marca la duración real de las notas. El movimiento viene a especificar el número de redondas por minuto.
El comando que selecciona el volumen en el macrolenguaje que se está analizando es V. Este comando irá seguido por un número que indica la magnitud de dicho volumen. La citada cantidad varía desde cero (pianissimo) hasta 15 (fortissimo). En el siguiente ejemplo se hace sonar una nota en todos los volúmenes posibles:

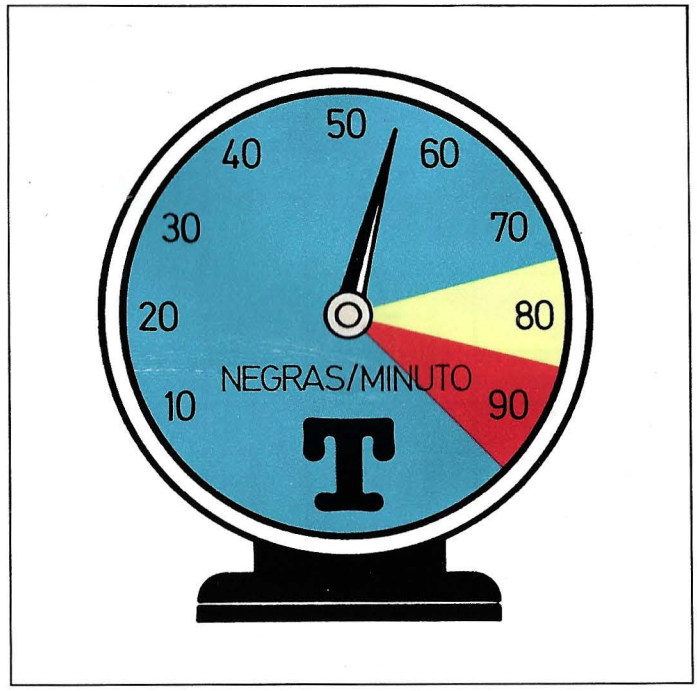
```
10 PLAY "V0CV1CV2CV3CV4CV5CV6CV7C"
20 PLAY "V8CV9CV10CV11CV12CV13CV14
   CV15C"
```

Por su parte, el movimiento se indica por medio del comando T. El número que

Los silencios son también importantes. El subcomando R es el encargado de proporcionar los silencios.



El volumen del sonido emitido se controla por medio del subcomando V.



El subcomando T permite variar la velocidad de ejecución de la pieza, prefijando el número de negras por minuto a interpretar.

acompaña a esta letra indicará la cantidad de negras por minuto que han de sonar. La variación que admite este comando cubre desde 32 (32 negras por minuto) hasta 255 (255 negras por minuto). De esta forma se calibra la velocidad de ejecución de la obra, desde "lento" hasta "vivace", pasando por "adagio", "andante" y "allegro".

EL EMPLEO DE VARIABLES

En algunas ocasiones se emplea una misma serie de notas en distintos puntos de una melodía. En estos casos resulta tedioso repetir las mismas notas una y otra vez. El macrolenguaje musical permite almacenar esas listas de notas de forma muy flexible. Como se ha visto, los conjuntos de órdenes del macrolenguaje se manipulan en forma de cadenas de caracteres. Pues bien, también se puede hacer uso de variables de ese mismo tipo. Un posible ejemplo es el que se muestra a continuación:

```
100 LET A$="CDECDFCDG"
110 PLAY A$
```

Esta facilidad permite interpretar varias veces una misma melodía sin tener que teclear de nuevo toda su formulación. Pero no es ésta la única posibilidad del empleo de variables. Se pueden insertar variables en el interior de una cadena ejecutable, conteniendo éstas fragmentos de la melodía completa. Para ello se utiliza el comando X, seguido del nombre de variable de cadena y de un punto y coma. Siguiendo con el ejemplo anterior, ésta es una ampliación del mismo:

```
120 PLAY "03XA$;06XA$;"
```

En esta última línea se ha repetido dos veces el anterior fragmento. En cada una de las ejecuciones se ha utilizado una octava distinta, para diferenciarlas.
Las variables de cadena no sólo pueden contener notas, sino también indicaciones de octava, duración, etc.
Además de las variables de cadena (para listas de órdenes) se puede hacer uso de variables de tipo numérico. Estas últimas permiten almacenar los datos que se adjuntan con los comandos O, V, T, etc.: datos numéricos. Las variables se añaden tras el comando adecuado, precedidas por un signo "=" y seguidas del preceptivo punto y coma. El mismo ejemplo de más arriba se completa con la posibilidad de variar el volumen:

```
10 INPUT "VOLUMEN=";VOL
100 LET A$="CDECDFCDG"
110 PLAY "V=VOL;"
120 PLAY "03XA$;06XA$;"
```

En este caso se recoge el lado de volumen en la instrucción INPUT de la línea 10. Ese valor se almacena en la variable VOL, variable empleada en la línea 110 para fijar el volumen.

El lenguaje C (3)

Estructuras de control y operadores

Antes de entrar en los aspectos más significativos del lenguaje C, conviene conocer las estructuras que permiten dirigir la ejecución del programa hacia diversos puntos (selección) y las que, en ocasiones, obligan a apretar apresuradamente el botón reset del ordenador (bucles o iteraciones).

LA ESTRUCTURA IF-THEN-ELSE

En C, esta estructura tiene el siguiente aspecto:

```
if   (esta condición es cierta)
     ejecutar esto;
else
     ejecutar esto otro;
```

Se observa que no hay "then". Al contrario que en otros lenguajes, "then" no es palabra reservada en C, por lo que es posible declarar variables con dicho nombre.

OPERADORES RELACIONALES

A == B	A ES IGUAL A B
A < B	A ES MENOR QUE B
A > B	A ES MAYOR QUE B
A >= B	A ES MAYOR O IGUAL QUE B
A <= B	A ES MENOR O IGUAL QUE B
A != B	A ES DISTINTO DE B

OPERADORES LOGICOS

&&	'AND' LOGICO ("Y")
||	'OR' LOGICO ("O")

Las expresiones booleanas se forman a través de operadores relacionales y lógicos.

Las selecciones "ejecutar esto" y "ejecutar esto otro" pueden estar formadas por una única sentencia o por un grupo de ellas encerradas entre llaves. Esto mismo es lo que se hacía en PASCAL sin más que sustituir las llaves por BEGIN-ENDs.
La condición que está entre paréntesis a continuación del "if" se forma a base de "operadores relacionales" y "operadores lógicos"; ambos tipos reflejados en el cuadro adjunto.
Los operadores relacionales son más o menos familiares para el aficionado, excepto el último, que en otros lenguajes se representa con el símbolo "< >". Al examinar la lista de operadores lógicos nos puede surgir una pregunta: ¿por qué && y no simplemente &? La razón es que los operadores simples como & o | están reservados para operaciones con bits, de las que se hablará sucintamente en otros capítulos. En cuanto al orden de evaluación, los operadores relacionales se evalúan siempre antes que los lógicos, por lo que:

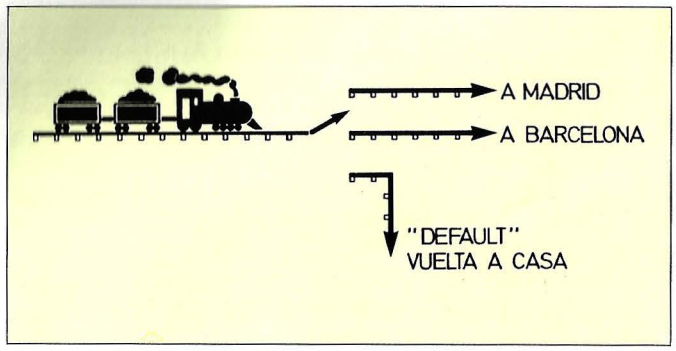
a<b && b==c

es totalmente equivalente a:

(a<b) && (b==c)

Dentro de los lógicos tienen mayor preferencia las expresiones relacionadas por && que las que lo están por ||.

```
switch   (<expr. entera o variable char>)
{
case     <cte. entera o carácter 1>:
         haz esto;
case     <cte. entera o carácter 2>:
         haz esto;
         .
         .
         .
default:
{        haz esto;
```

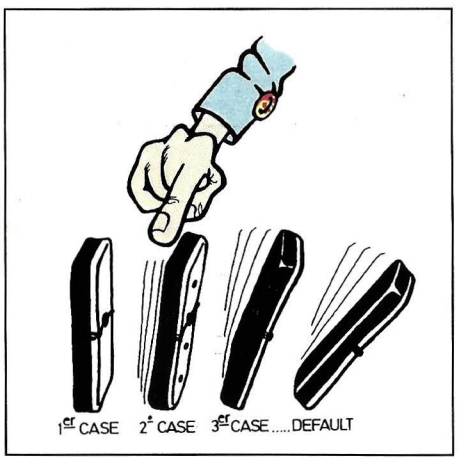


PROGRAMA 1: El "switch" permite elegir una acción entre varias.

```
main ()
{
   int i=2;

   switch (i)
   {
      case 1;
            printf("Estamos en el caso 1\n");
      case 2;
            printf("Estamos en el caso 2\n");
      case 3;
            printf("Estamos en el caso 3\n");
      default:
            printf("Estamos en default\n");
         {
         {
```

```
ESTAMOS EN EL CASO 2
ESTAMOS EN EL CASO 3
ESTAMOS EN DEFAULT
```

PROGRAMA 2: Un ejemplo del curioso comportamiento de la estructura "switch".



Un "switch" que no contenga ningún "break" ejecutará la secuencia asociada al "case" (en la figura es el segundo de ellos) y todas aquellas que la sigan hasta el final del mismo.

LA ESTRUCTURA SWITCH

No siempre son tan sencillas las cosas para que existan sólo dos alternativas a una pregunta. Para estos casos más complicados se dispone de la estructura "swicht" (equivalente al CASE del PASCAL). Su aspecto queda reflejado en la figura adjunta.

Al llegar a un "swicht" se evalúa la expresión que está entre paréntesis y se exploran secuencialmente los "case" hasta encontrar uno cuya constante coincida con el resultado obtenido al evaluar la expresión. Si no hay ninguna coincidencia, se ejecutará la acción que sigue a "default". Hay que resaltar que, al contrario que en el CASE de PASCAL, una vez que ha ocurrido una coincidencia con algún "case", se ejecutará la acción asociada a él y el resto de las opciones que lo siguen, hasta llegar al final de la estructura, señalado por el }.

Esta forma de operar del "swicht" puede sorprender hasta el punto de dudar de su utilidad. Para obviar este problema se hace uso de la sentencia "break", la cual obliga a pasar el control, justo a continuación del bloque delimitado por {} en el que se encuentra el programa, al aparecer dicha sentencia. Al incluirla en un "swicht" obligará inmediatamente a pasar el control

```
main()
{
   char c="b";
   switch (c)
   {
      case 'a':
            printf("Estamos en el caso A\n");
             break;
      case 'b':
            printf("Estamos en el caso B\n");
            break;
      case 'c':
            printf("Estamos en el caso C\n");
            break;
      default:
            printf("Estamos en default\n");
         {
         {
```

PROGRAMA 3: En el ejemplo se observa cómo solucionar el problema planteado en el programa 2.

```
ESTAMOS EN EL CASO B
```

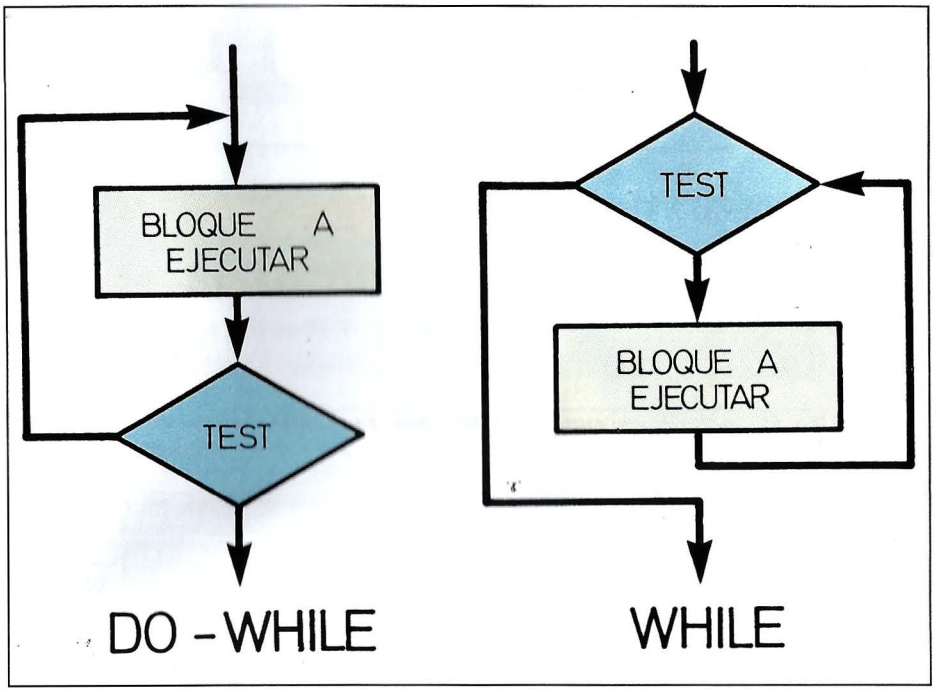
al punto donde está el "}" que indica el final del "swicht", tal y como se observa en la correspondiente figura.

LOS BUCLES EN C

El lenguaje C brinda tres tipos de estructuras para realizar bucles, que tienen sus equivalentes en los FOR-DO, REPEAT-UNTIL y WHILE-DO del PASCAL.

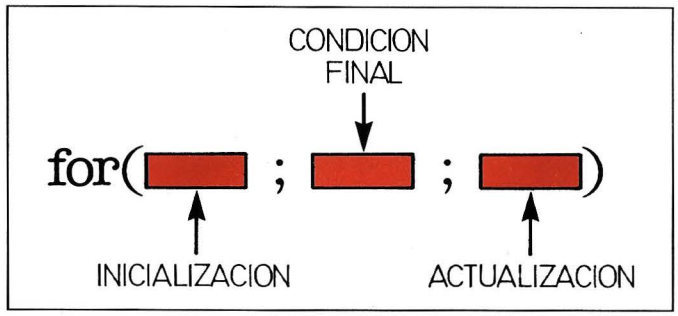
• *El bucle "for"*

En la figura adjunta se incluye un programa cuya acción es escribir en columna los diez primeros números naturales. Como se observa, a continuación de la palabra clave "for" se encuentra un paréntesis con tres parámetros que controlan el bucle. El primero de ellos es el de



La diferencia entre un "do-while" y un "while" la establece el distinto lugar en el que se realiza la comprobación de la certeza o falsedad de la condición del bucle.

```
main()
 {
     int i;

     for (i=0;i<=9;i=i+1)
         printf ("%d\n",i);
     {
```



PROGRAMA 4: La estructura FOR es, sin duda, el método básico de realizar un bucle en "C".

inicialización y sólo se ejecuta en la primera entrada al mismo. El central expresa la condición para la cual el bucle seguirá ejecutándose. El último proporciona la actualización de la variable de control del bucle.

En el ejemplo, tal actualización es un simple incremento de una variable. Para realizar incrementos de variables en C podemos recurrir a una de las siguientes expresiones:

```
i=i + 1;
i ++;
++ i;
i += 1;
```

Mientras que para decrementar una varia-

Lo cierto y lo falso en el "C"

Una constatación inmediata es que en el lengaje C no existe el tipo booleano en sí. Esto se debe a que para el C todo aquello que sea distinto de cero es cierto. Suponga el siguiente segmento de programa:

```
if (es_cierto)
     función1();
else
     función2();
```

donde la variable "es_cierto" se ha declarado de tipo "int". Si al llegar al "if" su valor es cero, se ejecutará la función 2 ("funcion2"). Por contra, si su valor es cualquier otro excepto el mencionado, se ejecutará "función1". Otra forma de tratar variables booleanas es a través de un par de sentencias "#define":

```
#define TRUE  1
#define FALSE 0
```

Es posible definir "TRUE" con cualquier valor distinto de cero, aunque se suele tomar el valor "1" por simplicidad.

```
main()
 {
    int i;

    for (i=0;i<=9;i=++i)
    {
        printf ("%d",i);
        printf ("\n");
    {
    {
```

PROGRAMA 5: Aspecto de una estructura FOR cuando en su cuerpo hay que incluir más de una sentencia.

```
main()
 {
    int i, n;

    for (i=2; n<=1000 ; ++n)
    {
        for (i=2; i<n ; ++i)  /* esto es el */
            if (n%i==0)       /*   bucle    */
                break;        /*  interno   */
        if (i==n)
            printf ("%d\n",n);
    {
    {
```

PROGRAMA 6: ¡Por fin un programa útil en "C"!

ble se pueden utilizar las mismas expresiones cambiando "+" por "−".
La diferencia entre la segunda y la tercera de las formas indicadas reside en que, en la segunda, primero se utiliza la variable y luego se incrementa, mientras que las cosas ocurren en orden contrario en la tercera. Aunque en el ejemplo propuesto ambas formas darán el mismo resultado, se verán otros ejemplos en los que el elegir una u otra forma es muy importante. La cuarta forma puede ser útil cuando haya que incrementar una variable con un nombre muy largo, o una expresión compleja, ya que sólo hay que escribirla una vez. Como es habitual, si el cuerpo del bucle está formado por más de una sentencia hay que encerrar a éstas entre {} (ver figura).

- *El bucle "while"*

Su forma es:

```
while (esto sea cierto)
  hacer esto;
```

Nuevamente, el cuerpo del "while" debe ir encerrado entre llaves si está formado por más de una sentencia.

- *El bucle "do-while"*

Su equivalente es el REPEAT-UNTIL del PASCAL y presente el siguiente aspecto:

```
do
  esto;
while (esta condición sea cierta);
```

La diferencia entre este y el anterior reside en que el bucle "do-while" siempre es ejecutado al menos una vez (hay que observar dónde se produce el "test" para decidir si se ejecuta de nuevo o no), mientras que el "while" puede no ser ejecutado ni una sola vez.
Las condiciones que tanto en uno como en otro caso aparecen a continuación de la palabra "while" se forman de manera análoga como se hace con las de la estructura "if-then-else".

UN EJEMPLO FINAL

A modo de resumen de las estructuras descritas cabe proponer el ejemplo práctico de la figura adjunta.
Como se vio en el primer capítulo, el lenguaje C está todavía en su más tierna infancia y hay muchos aspectos del mismo que son claramente mejorables. Tal sería el caso de los operadores lógicos, reduciendo el símbolo doble a uno simple por ejemplo. Además, se ha visto que el "swicht" de C es mucho menos potente que su equivalente en el PASCAL. Como sucede en otros campos, cuanto más se aligere el trabajo de la máquina, más se tendrá que cargar de trabajo el hombre; con el C se pretende tener un lenguaje compacto, cuyo compilador sea fácil de diseñar y ocupe poca memoria. Haciendo que sea el programador quien se ocupe de ciertas tareas que debieran estar a cargo del compilador (caso del "swicht") se puede conseguir el objetivo arriba citado.
Volviendo al ejemplo, se trata de un programa que calcula los números primos entre 2 y 1000. Tal vez sea conveniente recordar que un número primo es aquél que sólo es divisible por sí mismo y por la unidad; y esta será la propiedad que se utilizará en el programa. Se parte de dos bucles, uno que generará candidatos a primos y otros que se dedicará a comprobar la citada propiedad. Observando el bucle interno de la figura, se deduce que si se ha detectado que "n" (candidato) es divisible por un número, se abandona dicho bucle para comprobar si el divisor encontrado es el propio "n", en cuyo caso será un número primo; si no es así, se retorna al bucle exterior para asignar a "n" un nuevo candidato.
Salta a la vista que el uso de "break" no se restringe al ámbito de los "swicht". Además, se ha introducido el operador "%", que calcula el resto de la división entre dos números. Si este resto es cero, el primer número es divisible por el segundo.

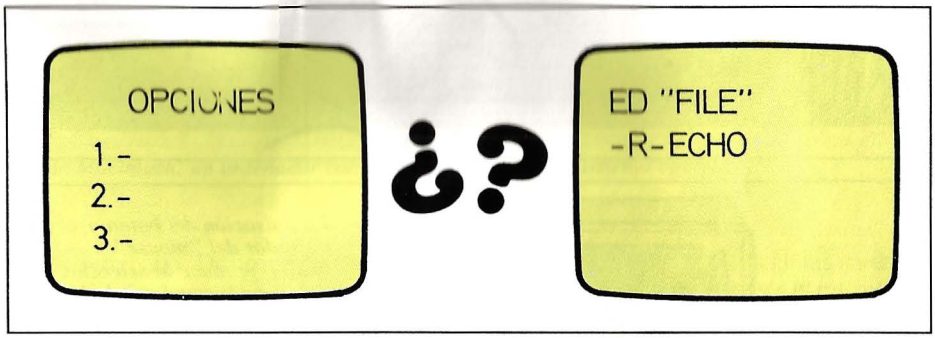
Apple Macintosh (1)

Un nuevo concepto de sistema operativo

Los diversos sistemas operativos están orientados o resultan más adecuados para un determinado ámbito de operación. Hay sistemas operativos cuya estructura es tal que permite el uso eficaz del ordenador por parte de personal muy poco versado en informática, toda vez que hacen un uso exhaustivo de menús para el desarrollo de cualquier tarea.

Otros sistemas operativos, por el contrario, son más adecuados para su empleo por personal experto, al poseer una estructura de tipo abierto, con multitud de comandos y, dentro de estos, con distintos parámetros que matizan su operatividad.

La gran disyuntiva aparece ahora. ¿Qué tipo de interface o método de diálogo interactivo sistema operativo-usuario es más adecuada?



El usuario ha de tener en cuenta sus necesidades y conocimientos técnicos a la hora de escoger un sistema operativo.

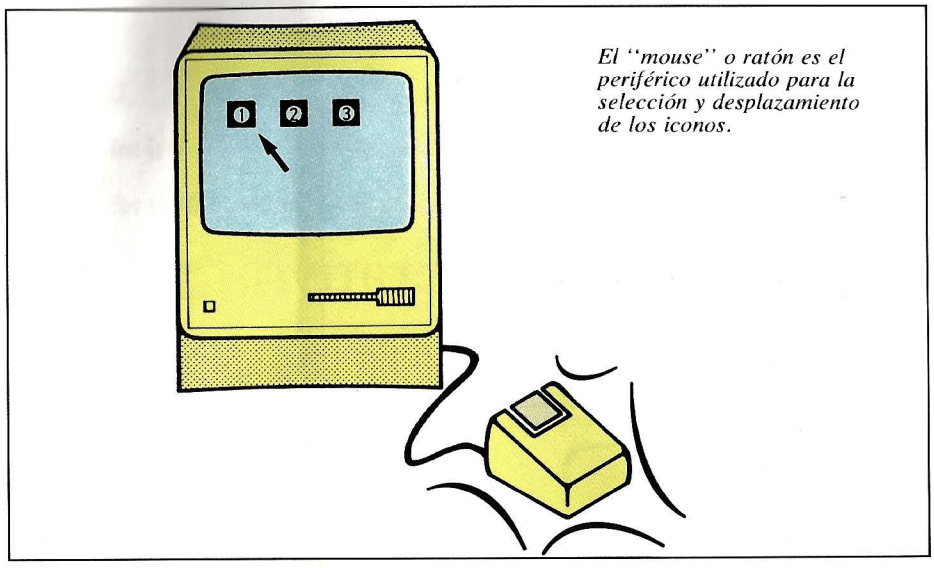
LA PROBLEMATICA DE LA INTERFACE

Para responder a la pregunta anterior hay que tener muy presente el tipo de ordenador, la tarea a la que éste se dedica, así como el usuario que lo utiliza. Un ordenador destinado al desarrollo de programas y aplicaciones, será normalmente empleado por personal experto en informática, con lo cual la interface no tiene porqué ser simple y cabe el empleo de multitud de comandos.

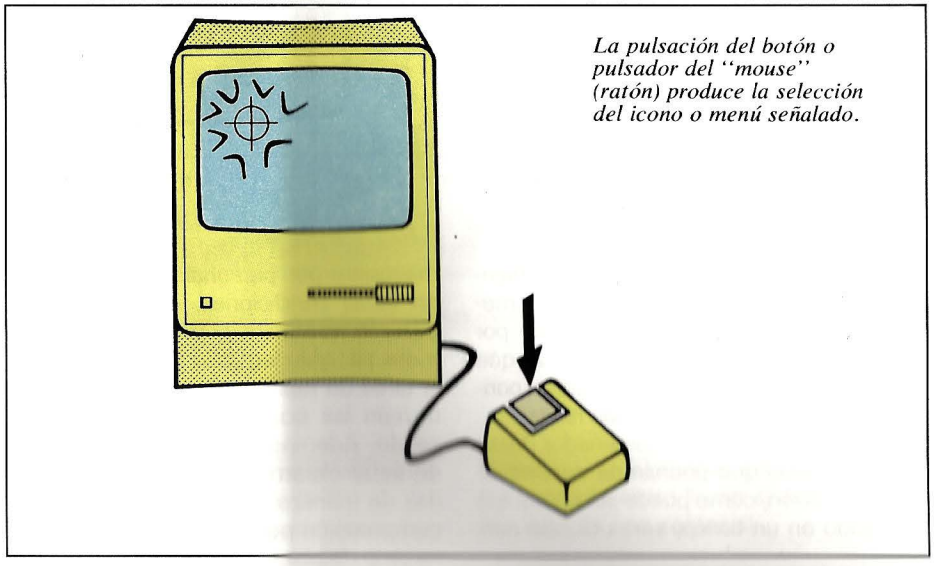
Sin embargo, este hecho es importante y hasta cierto punto beneficioso en dicho entorno, ya que se producen importantes ahorros de tiempo en el desarrollo de procesos. En efecto, teclear el comando directamente conduce a una respuesta inmediata a los deseos del usuario, mientras que de utilizar una estructura de menús sería necesario ir descendiendo por los distintos menús hasta llegar al que contiene entre sus opciones la correspondiente a la tarea que se desea realizar.

En el caso contrario, un ordenador destinado a tareas que podríamos denominar de producción, como puede ser un equipo instalado en un banco, será utilizado normalmente por personal no especializado; en estas condiciones, interesará que el sistema operativo coloque el mayor número posible de pasos entre el usuario y la tarea de realizar, de manera que se reduzcan las posibilidades de error indeseado. Además, las referidas tareas han de estar claramente definidas y expresadas de manera que resulten de muy fácil comprensión para el usuario.

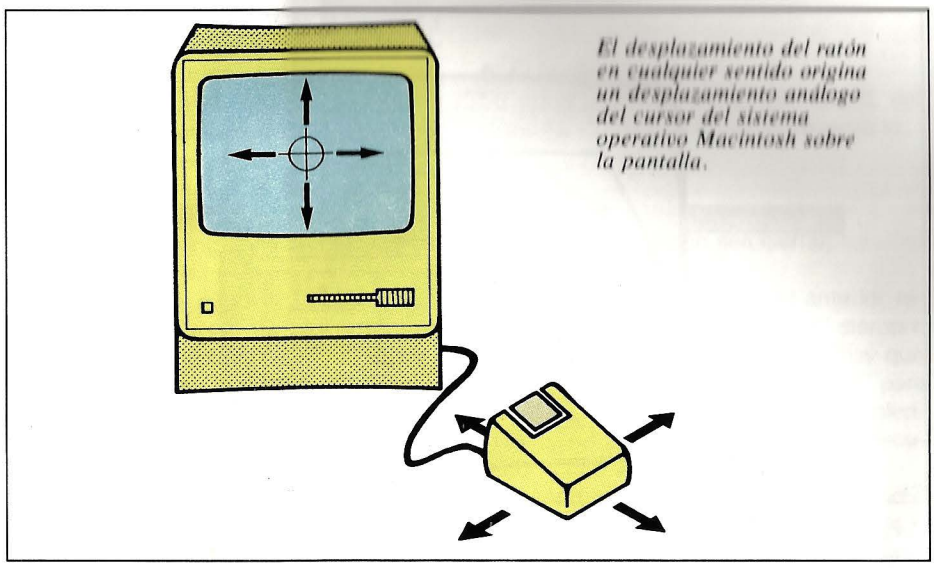
La pantalla controlada por el sistema operativo del Apple Macintosh se estructura de forma similar a la mesa de una oficina.



El "mouse" o ratón es el periférico utilizado para la selección y desplazamiento de los iconos.



La pulsación del botón o pulsador del "mouse" (ratón) produce la selección del icono o menú señalado.

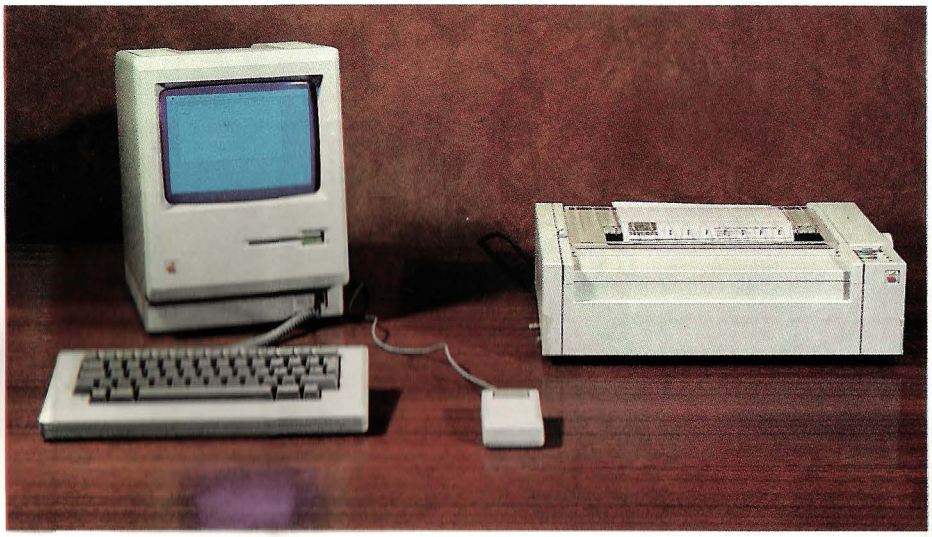


El desplazamiento del ratón en cualquier sentido origina un desplazamiento análogo del cursor del sistema operativo Macintosh sobre la pantalla.

EL MACINTOSH: SUS ORIGENES

En la actualidad el ordenador se ha expandido mucho más allá de las fronteras en las que, inicialmente, se pensaba quedaría constreñido. En su expansión ha invadido el hogar y las oficinas de todo tipo de empresas, las cuales hace pocos años ni tan siquiera intuían la posibilidad de utilizar ordenadores para controlar su gestión. Estos campos comerciales han resultado sumamente fértiles para los fabricantes de ordenadores, sobre todo personales, quienes han invadido el mercado con productos destinados al mismo; ahí están los IBM-PC, DEC-Rainbow, NCR-DMV, etc.
Todos estos productos, ordenadores de pequeño tamaño del tipo denominado "personal", hacen uso de sistemas operativos ya estandarizados en el mercado, como pueden ser el MS-DOS o el sistema operativo CP/M en sus distintas versiones (80,86), por citar quizás los más conocidos y universales. Este tipo de sistemas operativos pertenecen al género de los señalados en el apartado anterior en primer lugar; es decir, se trata de sistemas cuyo empleo exige al usuario recordar distintas secuencias de comandos. En consecuencia, pues, no son precisamente sistemas operativos muy "amigables" para el usuario. Por lo demás, hay que tener en cuenta que en este nuevo mercado la mayor parte de usuarios son personas carentes de formación informática y deseosas de obtener un rápido rendimiento de sus equipos. La compañía americana Apple Computers tomó una aproximación a este problema totalmente radical fruto de la cual es el sistema operativo que equipa a su ordenador personal Apple Macintosh. Este sistema es, a efecto del ordenador que lo porta, grande, complejo y exigente respecto al microprocesador en términos de ciclos de máquina. La razón es que ya no existen largas listas de comandos que deba memorizar el usuario, sino que todo tipo de tareas se ven reflejadas con pleno detalle a través de una serie de símbolos denominados *iconos* y menús desplegables de tipo "persiana".
El origen de este sistema operativo se encuentra en ciertas investigaciones realizadas en los laboratorios de Xerox en Palo

Alto (California). En estos laboratorios se estaba intentando desarrollar un sistema operativo por medio del cual el ordenador se adaptara al usuario y no el usuario al ordenador. Apple trasladó esta idea a los nuevos ordenadores que estaba desarrollando, destinados a competir con los señalados anteriormente en el terreno de la informática personal.
Estos equipos, debido al tamaño del sistema operativo y a sus necesidades de CPU, se basan en un microprocesador de 32 bits: el Motorola 68000. Con esta potencia de PCU es ya posible obtener lo que ofrece este sistema operativo.



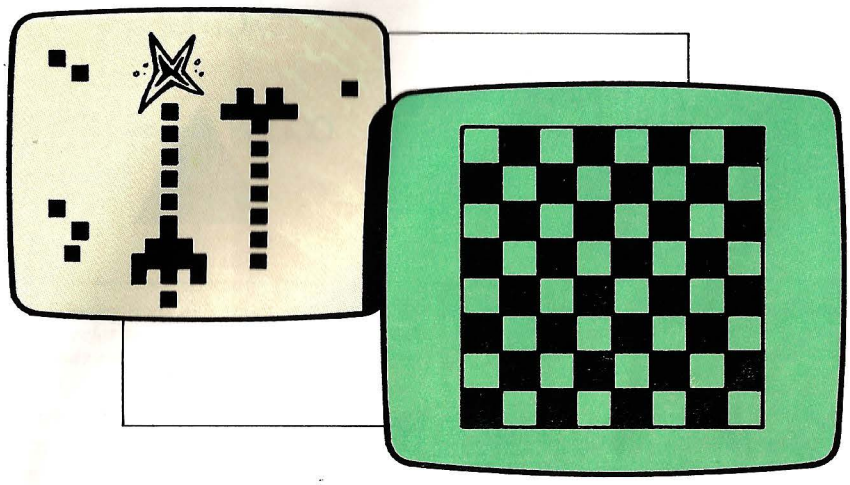
Apple Macintosh, un ordenador con un nuevo concepto de sistema operativo.

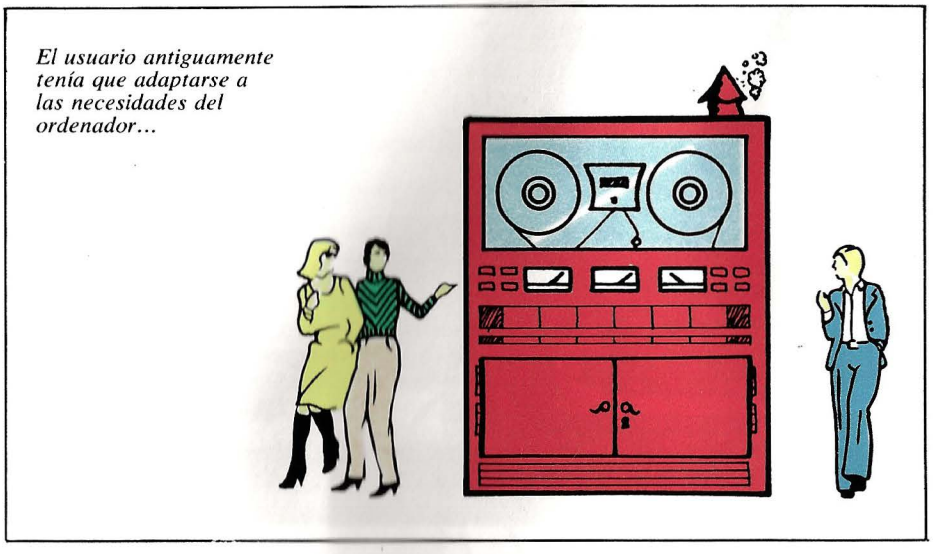
LA INTERFACE DEL MACINTOSH

La característica básica de la interface máquina/usuario de este sistema operativo se basa en el hecho de que está orientada al objetivo y no sigue un procedimiento predeterminado. Dicho de forma simple, el usuario únicamente ha de preocuparse del trabajo a realizar y no de la forma en que ha de realizarse. La pantalla en el caso del sistema operativo Macintosh se distribuye como si se tratara de un escritorio normal de oficina, con distintas tareas a realizar y documentos sobre el mismo, las cuales pueden ser desarrolladas y consultadas de igual forma que operaría una persona ante esta situación.
Bajo estas premisas es posible editar do-

Inteligencia en los juegos por ordenador

El ordenador ha invadido en su desarrollo fulgurante muchas de las actividades humanas como una eficaz herramienta de apoyo a las mismas. Uno de los ámbitos en los que se manifiesta su amplia presencia es el del ocio. Los juegos de ordenador pueden clasificarse en dos categorías; cada una de ellas presenta al programador y creador de los mismos diversos problemas.
En el primer grupo se encuentran los juegos cuya base esencial es la presentación gráfica. De ellos, el ejemplo más sobresaliente lo constituyen los populares "juegos de marcianos", en los que se produce una confrontación entre el usuario y el ordenador, operando con unas reglas invariables muy normalmente rígidas y repetitivas. La dificultad de estos juegos para el usuario estriba normalmente en que las dificultades se presentan con creciente rapidez y disminuyen el tiempo de preparación y posicionamiento del jugador humano.
La segunda categoría la integran los juegos en los que el ordenador ha de realizar una serie de operaciones que de ser efectuadas por un ser humano serían calificadas de inteligentes. Dentro de este grupo se encontrarían aquellos programas que simulan juegos como el ajedrez y el backgammon. En estos programas, las reglas de juego son idénticas para el ordenador y el usuario, no teniendo el primero el control último de los acontecimientos del juego ni su gestión. En tales condiciones y de cara a mantener el desarrollo del juego, el programa ha de evaluar de manera inteligente las situaciones que se presentan, de forma que adquiera ventaja sobre su contrincante... Las características de estos programas los acercan a las premisas de la inteligencia artificial, con lo cual su importancia va más allá de la simple experiencia lúdica.





El usuario antiguamente tenía que adaptarse a las necesidades del ordenador...

cumentos, pasar partes de uno a otro empleando una técnica similar a la de recortar y pegar encima que se utilizaría en la realidad (de hecho el símbolo que se emplea para llevar a cabo esta tarea es el de recortar y pegar), etc. Todas estas tareas se realizan por medio de la activación de una serie de símbolos en la pantalla, a los que se denomina iconos; entre ellos se encuentra una papelera, los correspondientes a distintos tipos de ficheros, los representativos de los discos utilizados... Estos iconos actúan como elementos de conexión hacia otras opciones, de forma similar a las opciones de un menú primario. La selección y activación de los inconos no se realiza mediante el teclado, sino por medio de un periférico que apareció asociado al sistema operativo Macintosh: el "mouse" o ratón. Este elemento consiste en una pequeña caja deslizante, de tamaño semejante a un paquete de cigarrillos. Su desplazamiento sobre la mesa de trabajo, realizado en una dirección determinada, produce un desplazamiento equivalente del indicador sobre la pantalla. Cuando este cursor se encuentra situado sobre el icono correspondiente a la tarea deseada, la pulsación de un botón situado en el ratón selecciona a éste como opción de trabajo. Toda la gestión se realiza a través del ratón, de forma que el teclado sólo es necesario para introducir textos y números.

El Macintosh hace que el usuario no tenga que adaptarse a la complejidad de la máquina.

COMPARACION CON OTROS SISTEMAS OPERATIVOS

De la breve exposición anterior se deduce que la operación del Macintosh difiere totalmente de la propia de cualquuier otro sistema operativo, toda vez que su facilidad de empleo es inigualable. De hecho, hay usuarios que se jactan de no haber tenido necesidad de desempaquetar sus manuales de usuario.

Sin embargo, cabe preguntarse si este sistema operativo es un paso en la dirección correcta por lo que se refiere a la portabilidad de aplicaciones y a la comunicación entre equipos, características que parecen dominantes en el mundo actual de la informática. Por un lado se encuentra el hecho de que cualquier programa desarrollado para este sistema operativo no puede ser empleado en ninguna otra máquina, debido a la plena y total interacción existente entre el hardware y el software. Al mismo tiempo, el hecho de que cualquier programa que se prepare deba hacer uso del método de interfase a través de iconos, hace que dichos programas no sean fáciles de desarrollar y únicamente sean rentables en el caso de existir una gran difusión comercial del ordenador al que se destinan.

Pese a ello, hay que hablar del sistema operativo del Macintosh como una verdadera revolución en el terreno de los sistemas operativos. Una tarea compleja que ha necesitado para su puesta a punto la friolera de 200 años de programador.

A pesar del hecho de que el S.O. del Macintosh es un voraz consumidor de recursos de CPU en comparación con otros sistemas que emplean el mismo microprocesador, con respecto a los cuales es más lento, no cabe la menor duda de que el sistema operativo Macintosh es el sueño de cualquier persona que necesita ponerse a trabajar con un ordenador sin demasiadas complicaciones.

Auto CAD (y 2)

Diseñando con el programa Auto CAD

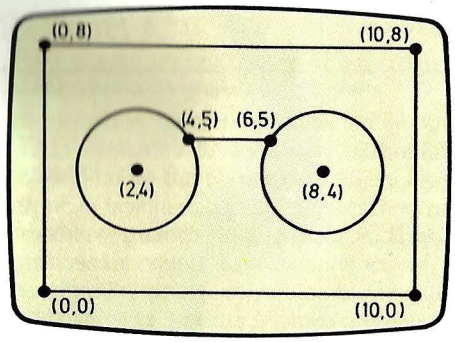
El presente capítulo pone punto final al estudio del programa Auto CAD. En primer lugar se detalla el sistema de coordenadas que permite al usuario posicionarse y dibujar en cualquier punto del plano. A continuación, describiremos los principales comandos del Auto CAD, para concluir describiendo una sencilla sesión de trabajo en la que realizaremos un diseño elemental.

COORDENADAS

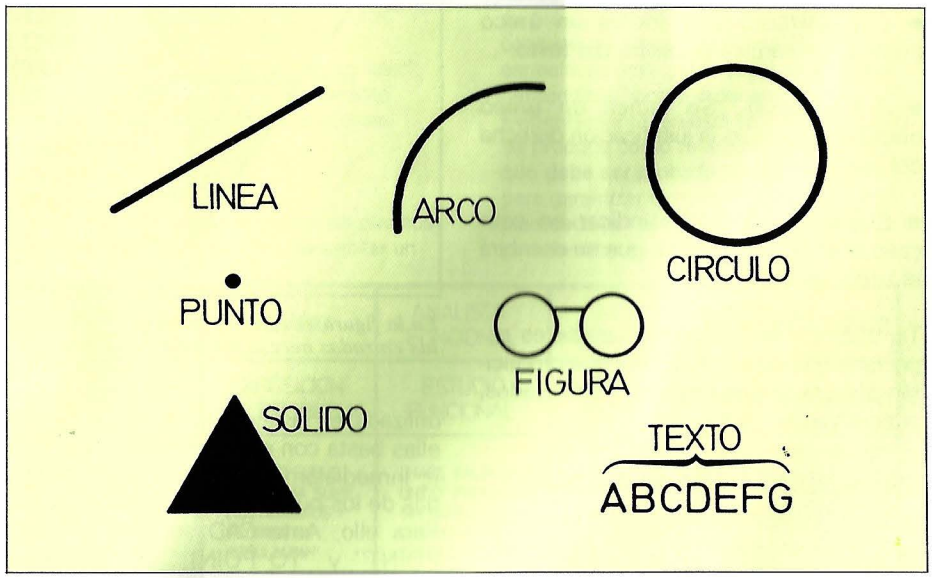
Los distintos puntos del plano se pueden localizar por medio de un sistema de coordenadas cartesianas. Mediante él se deben especificar dos coordenadas (X,Y): con la primera se define la localización horizontal, mientras que con la segunda se indica la localización vertical del punto.
Auto CAD sitúa normalmente el punto (0,0) en la parte inferior izquierda del dibujo, aunque el usuario puede fijar el origen en otro lugar del diseño. El sistema de coordenadas resulta fundamental para situar las entidades. Así, si se desea dibujar una línea, habrá de indicar sus dos puntos extremos, e inmediatamente el programa trazará la línea sobre la pantalla.
Inicialmente los límites del dibujo vienen marcados por los puntos situados en las cuatro esquinas; estos son: (0,0), (0,8), (10,8) y (10,0). No obstante, el usuario puede decidir unos límites sin más que indicar los nuevos puntos inferior izquierdo y superior derecho.
En resumen, el usuario puede realizar cualquier tipo de diseño sin más que ir indicando los distintos puntos del sistema de coordenadas entre los que se representarán las entidades.

TIPOS DE ENTIDADES

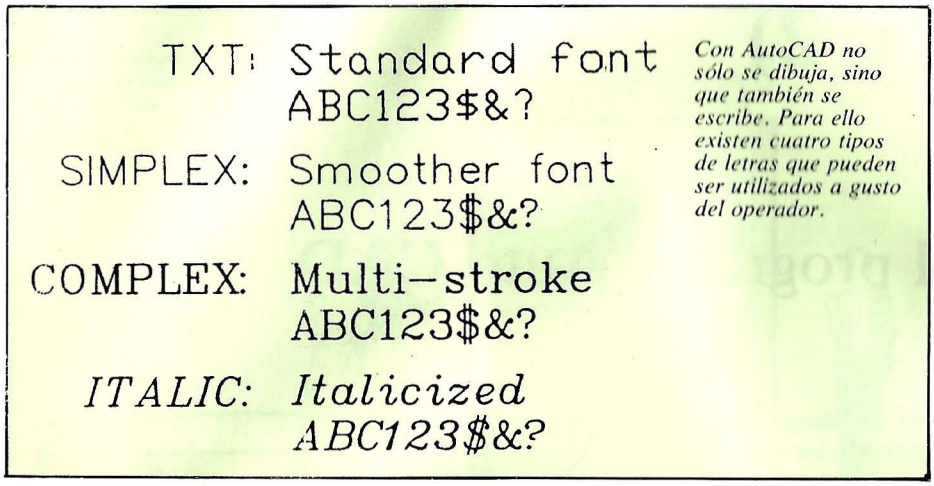
Auto CAD es capaz de representar ocho tipos de entidades distintas: líneas, trazos, puntos, círculos, arcos, textos, sólidos y figuras creadas por el propio operador. Con este último tipo de entidades se consigue que en el caso de tener que repetir un objeto muchas veces, dentro de un mismo diseño, no sea necesario dibujar más que una vez y definirlo como entidad, de esta forma se simplifica notoriamente el trabajo del operador.
En cuanto al tipo de entidad texto, ésta se



El sistema de coordenadas resulta de vital importancia en todo programa CAD. Aunque éstas no aparezcan en el diseño final, deben ser especificadas por el usuario para incluir las entidades en el plano.



Los principales tipos de entidades de AutoCAD son las que aparecen en esta figura. Como se puede apreciar todas ellas resultan muy sencillas, pero al concatenarlas pueden dar lugar a diseños muy complejos.



Con AutoCAD no sólo se dibuja, sino que también se escribe. Para ello existen cuatro tipos de letras que pueden ser utilizados a gusto del operador.

puede considerar un tanto especial; es manejable mediante el comando TEXT. Para incluir un texto dentro de un diseño, en primer lugar habrá que indicar el punto del dibujo donde debe comenzar el texto y, a continuación, se deben especificar cuatro valores con los que se indicarán las siguientes características:

- A (Alineamiento). Se especificarán dos puntos que marcarán dos líneas imaginarias que servirán de guías para el texto.

- C (Centrado). Se indica un único punto que marcará el centro del texto.

- R (Derecha). Se indica un único punto que marcará la justificación derecha del texto.

- S (Estilo). Se debe indicar en este caso el estilo de letra con que se escribirá el texto en el dibujo.

También pueden incluirse caracteres especiales y códigos de control que producirán efectos especiales como subrayados, superrayados, etc.

COMANDOS PARA EL DIBUJO DE ENTIDADES

Dado que el número de comandos que incluye el programa Auto CAD desborda la posibilidad del espacio disponible para su exposición, nos vamos a limitar a presentar los dedicados al dibujo de entidades. Precisamente estos forman el grupo más característico dentro de la aplicación:

- **LINE**

Por supuesto, la entidad elemental y más utilizada es la línea. Para dibujar una de ellas basta con ejecutar el comando LINE e, inmediatamente, indicar las coordenadas de los puntos inicial y final de la línea. Para ello, Auto CAD solicitará: "FROM POINT" y "TO POINT", limitándose el usuario a teclear los puntos o marcarlos con la tableta de digitalización.

- **LIN: UNDO**

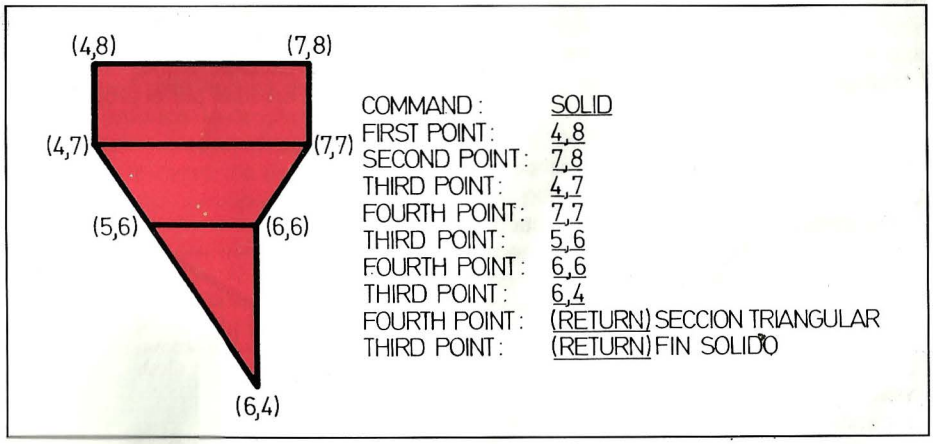
Cuando se desee dibujar varias líneas consecutivas, es decir cuando el punto inicial de la segunda línea sea el final de la primera, el inicial de la tercera, el final de la segunda, etc., se debe ejecutar el comando LINE UNDO que solicitará un único punto inicial; "FROM POINT" y, a continuación, irá pidiendo los sucesivos puntos finales: "TO POINT". En el caso de que la última línea deba finalizar en el punto inicial de la primera, vale con teclear C (CLOSE/CERRAR) para que Auto CAD finalice cerrando el polígono.

- **POINT**

Otra entidad también muy sencilla es el punto. Para que Auto CAD dibuje un punto en el plano de diseño, basta con ejecutar el comando POINT y teclear las coordenadas en las que se desea ubicar el punto.

- **CIRCLE**

Existen diversas formas de dibujar un círculo en el plano, todas ellas comienzan con la ejecución del comando CIRCLE; a continuación, Auto CAD solicitará al usuario que elija entre las siguientes opciones:



En la figura se reproduce el sólido conseguido mediante la utilización del comando SOLID y las entradas necesarias para su confección.

1. Centro y radio

El círculo se determinará indicando las coordenadas de su centro y el radio.

2. Centro y diámetro

Análogo al anterior, pero indicando su diámetro en vez de su radio.

3. Tres puntos

En este caso el programa solicitará tres puntos del plano y dibujará automáticamente el círculo que pasa por ellos (un principio básico en Geometría es que por tres puntos solo pasa un único círculo).

El programa AutoCAD no sólo sirve para el diseño de planos técnicos, de arquitectura o ingeniería, sino que también es perfectamente utilizable como herramienta para el dibujo artístico. El escenario de trabajo se traslada de la mesa de dibujo o del lienzo a la pantalla del ordenador.

4. Dos puntos
Si se elige esta opción, el usuario debe indicar las coordenadas de los puntos inicial y final del diámetro del círculo.

5. Especificación dinámica
La última forma, y probablemente la más espectacular, de dibujar círculos, consiste en indicar un punto fijo donde se situará el centro e ir variando dinámicamente el radio, con lo que se irá desplazando el círculo sobre el plano.

- **ARC**

Para dibujar un arco sobre el plano. Auto CAD ofrece diversas posibilidades:

1. Indicar tres puntos por los que debe pasar el arco.

2. Indicar dos puntos inicial y final, y un centro de arco.

3. Indicar un punto inicial, un centro y un ángulo.

Puesto de trabajo informáticos (1)

Cualquier actividad informática implica la participación de distintas categorías de técnicos. En este cuadro vamos a describir los puestos de trabajo relacionados con el desarrollo de un nuevo programa, y en el próximo capítulo detallaremos los puestos de trabajo necesarios para la explotación de programas ya desarrollados.

DESARROLLO DE NUEVOS PROGRAMAS

- USARIO

La iniciativa para el desarrollo de un nuevo programa suele recaer casi siempre en su usuario final; su misión consiste en detallar y describir con toda precisión los resultados que desea conseguir con el programa. Evidentemente, el enfoque dado por el usuario no debe prejuzgar las características técnicas del programa final, pero sí debe delimitar perfectamente su alcance y la casuística que se presentará en los problemas que deba resolver.

- ANALISTA FUNCIONAL

Tiene como principal misión servir de apoyo al usuario para la realización de un primer análisis del programa a desarrollar. El producto final de su trabajo se suele denominar "Especificaciones funcionales".

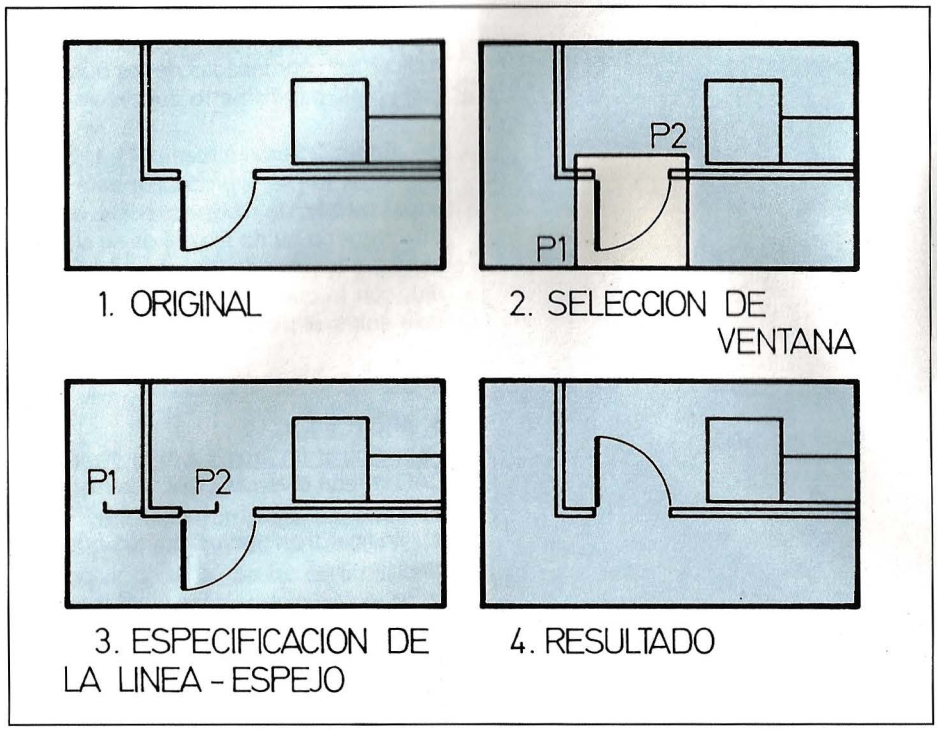
- ANALISTA ORGANICO

A partir de las especificaciones funcionales, el analista orgánico se encarga de estudiar la estructura técnica del programa y los datos qe se manejarán. El documento producido por este analista se suele denominar "Especificaciones orgánicas"; en todo caso debe ser un documento consecuente con las especificaciones funcionales.

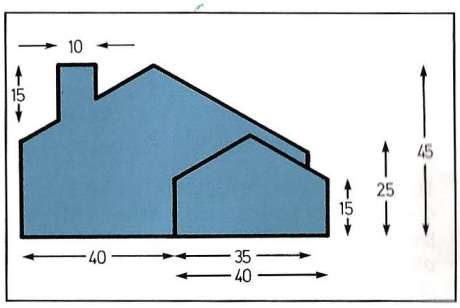
- PROGRAMADOR

El último eslabón de la cadena de puestos de trabajo necesarios para desarrollar un nuevo programa es el programador. Como su propio nombre indica, su misión consiste en programar (traducir) las especificaciones orgánicas a un lenguaje "entendible" por el ordenador. Evidentemente, en esta fase del desarrollo se produce como resultado un programa que debe ser probado exhaustivamente, para garantizar que resuelve la casuística planteada por el usuario, y documentado para facilitar su manejo.

PUESTO DE TRABAJO	USUARIO	ANALISTA FUNCIONAL	ANALISTA ORGANICO	PROGRAMADOR
MISION	EXPOSICION DEL PROBLEMA	ESTUDIO FUNCIONAL	ESTUDIO ORGANICO	PROGRAMACION
PRODUCTO	"QUIERO QUE EL PROGRAMA SUME DOS NUMEROS Y ESCRIBA EL RESULTADO"	"HACE FALTA UN UNICO PROGRAMA SIN RUTINAS DEBIDO A LA SENCILLEZ"	E LEER A LEER B X = A + B ESCRIBIR X F	10 INPUT A 20 INPUT B 30 LET X = A+B 40 PRINT X 50 STOP



Mediante la utilización de ventanas se pueden conseguir efectos muy interesantes y cómodos. En la figura se observa cómo se puede cambiar el sentido de apertura de una puerta.



En algunos casos resulta necesario acotar los diseños producidos. AutoCAD dispone de una utilidad con la que la acotación resulta muy cómoda.

4. Indicar un punto inicial, un centro y la longitud del arco.

5. Indicar los puntos inicial y final y el radio del círculo.

6. Indicar el punto inicial, el punto final y un ángulo.

7. También existe la posibilidad de concatenar el arco con una línea dibujada previamente.

Según el diseño que se esté realizando y las medidas que se conozcan, el usuario eligirá la forma que más sencilla le resulte para dibujar el arco.

- **TRACE**

En algunos casos resulta necesario incluir en el diseño líneas gruesas que representan a elementos sólidos; en terminología Auto CAD se denominan trazos. Para dibujar trazos sobre el plano basta con ejecutar el comando TRACE y, a continuación, indicar la anchura del trazo, el punto inicial y los sucesivos puntos por donde pasará. Para indicar al programa que el trazo ha finalizado, se pulsará la tecla RETURN sin señalar previamente ningún nuevo punto.

- **SOLID**

Este comando puede utilizarse para dibujar regiones sólidas, es decir, secciones cuadriláteras o triangulares. Cuando el usuario ordena la ejecución del comando SOLID, Auto CAD le solicitará cuatro puntos sucesivos que representan a los cuatro vértices del cuadrilátero a dibujar (exactamente en este orden: vértice superior izquierdo, superior derecho, inferior izquierdo e inferior derecho). En el caso de que se desee representar una sección triangular, vale con introducir como vértices superiores (o inferiores) el mismo punto, de forma que al coincidir den lugar a un triángulo.
Una vez dibujado el sólido, el programa solicita de nuevo los puntos tercero y cuarto de un nuevo sólido que se concatenerán con los puntos tercero y cuarto del sólido dibujado previamente. Como en todos los comandos repetitivos de Auto CAD, la forma de indicar el fin de su ejecución consiste en pulsar RETURN sin haber introducido previamente ningún dato.

- **TEXT**

Este comando ya ha sido presentado en el párrafo dedicado a los distintos tipos de entidades de Auto CAD.

LA UTILIDAD DE AUTO CAD

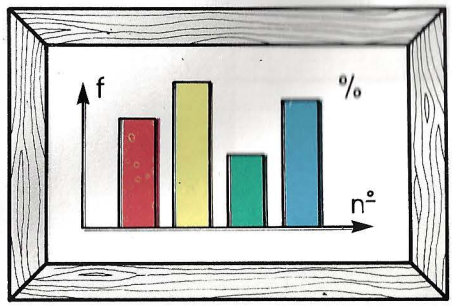
De todo lo expuesto hasta ahora sobre este programa se puede deducir erróneamente que Auto CAD tan sólo sirve para el diseño de planos técnicos propios de Ingeniería, Arquitectura, etc. Nada más lejos de la realidad: Auto CAD puede utilizarse perfectamente como herramienta para realizar dibujo artístico. Es importante no confundir esta faceta artística de Auto CAD con la posibilidad de generación de dibujos artísticos automáticos en un ordenador. En el primer caso, es decir mediante un programa CAD, el protagonismo del artista humano es total; en el fondo el dibujante se limita a utilizar una nueva herramienta artística: el ordenador y un programa CAD. En cambio, en el segundo caso, la generación automática de dibujos artísticos tiene ciertas componentes aleatorias y, aún dependiendo fundamentalmente de la inspiración del suario, no encajan dentro de lo que podíamos llamar dibujo artístico manual.
Como consecuencia a lo expuesto anteriormente, a realizar una sesión artística con Auto CAD no solo hace falta un ordenador personal y el programa, sino también una persona que conjugue un dominio perfecto sobre el manejo de programas CAD y sobre el dibujo artístico en sí mismo.

El basic científico

Estadísticas por ordenador

El empleo del ordenador en la sociedad actual se fundamenta en su gran potencia de cálculo. La mayor parte de las actividades en las que se hace necesario el uso del ordenador implican manipular un enorme volumen de datos. Para el manejo eficiente de grandes cantidades de datos es obligado una velocidad de cálculo también grande. Es ahí donde entra en juego el ordenador. Su capacidad de tratar muchos datos en poco tiempo lo convierte en una herramienta casi imprescindible.

En este capítulo se va a desarrollar un programa que hace uso de la capacidad anteriormente comentada. Se trata de un sencillo programa estadístico. Con él se pretende, a un tiempo, ejercitarse en la programación de tareas matemáticas y obtener un ejemplo de lo que se puede hacer con un ordenador.



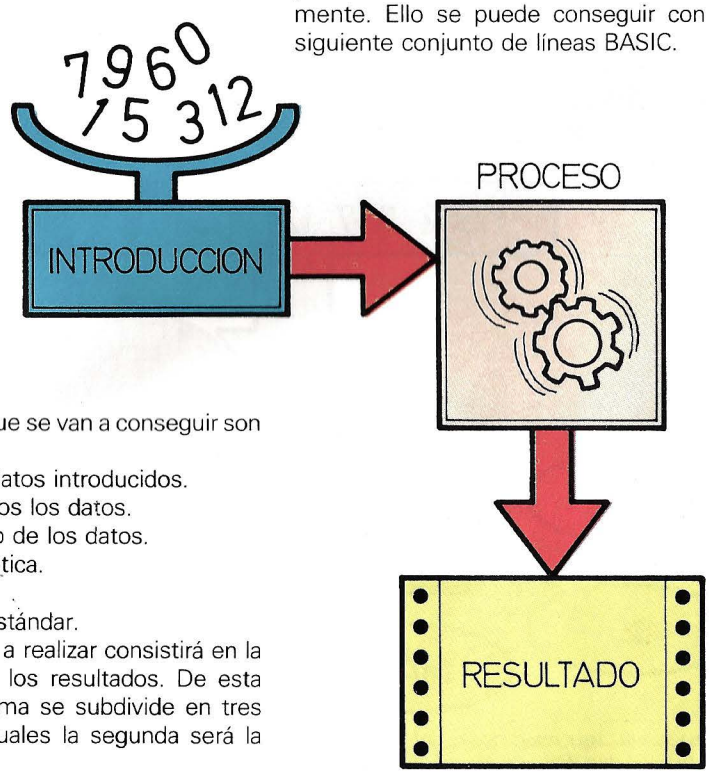
El ordenador puede servir como una gran herramienta estadística.

ESTRATEGIA DE PROGRAMACION

A la hora de crear un programa hay que empezar realizando un análisis del problema a tratar. En este caso, se sentarán las bases de lo que se desea que haga el programa. En principio, la estadística trabaja con un número grande de datos. Esos datos serán introducidos por el operador para tomar parte, más tarde, en los cálculos pertinentes. Para un manejo más adecuado de los datos habrá que contar con variables de conjunto o arrays. Así pues, el conjunto de datos a analizar se almacenará en un array.

Una vez determinada la estructura de datos, el siguiente paso consiste en definir los procesos a realizar con los mismos. El proceso inicial ha de materializarse en la introducción de los datos. Tras esa introducción de datos se procederá al cálculo de los distintos resultados.

El problema se puede subdividir en tres partes: introducción de datos, proceso de los mismos y salida de resultados.



Los resultados que se van a conseguir son los siguientes:

— Número de datos introducidos.
— Suma de todos los datos.
— Valor máximo de los datos.
— Media aritmética.
— Varianza.
— Desviación estándar.

La última acción a realizar consistirá en la presentación de los resultados. De esta forma, el problema se subdivide en tres partes, de las cuales la segunda será la más compleja.

INTRODUCCION DE LOS DATOS

Antes de proceder a la introducción de los datos es necesario dimensionar la matriz que ha de contenerlos. Como ejemplo haremos uso de una matriz o array de 100 elementos. En el caso de desear un mayor número de datos, bastará con alterar esta primera línea del programa.

```
10 DIN A(100)
```

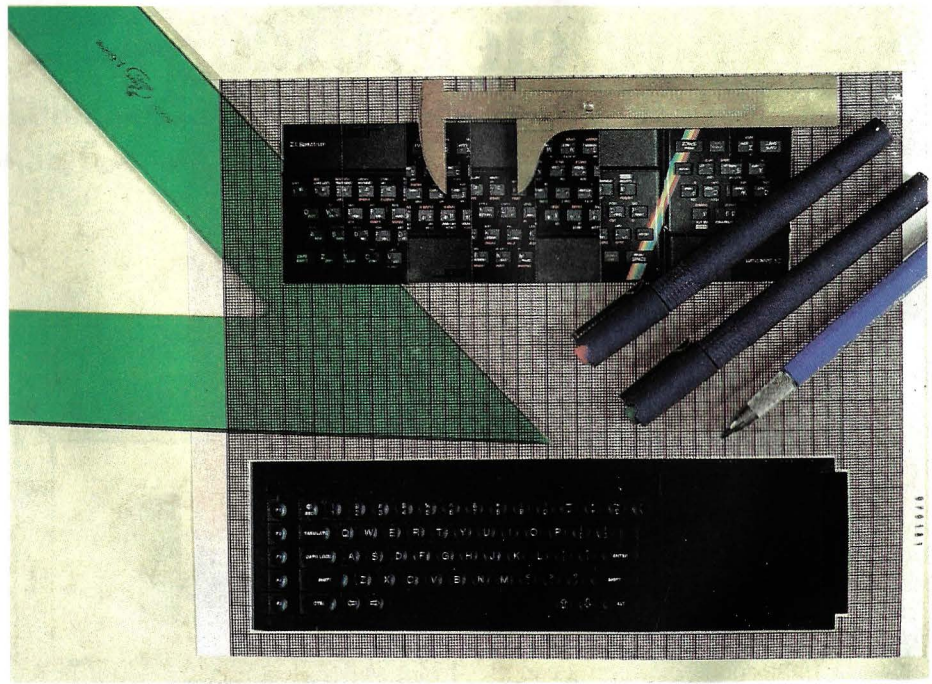
Para la introducción de los datos se empleará una instrucción INPUT reiteradamente. Ello se puede conseguir con el siguiente conjunto de líneas BASIC.

```
100 FOR I=1 TO 100
110 INPUT A(I)
120 NEXT I
```

Estas líneas permiten la introducción de los datos; sin embargo, obligan a teclear exactamente 100 números. Si lo que se desea es el análisis de una cantidad de datos, las líneas mencionadas no servirán. Lo correcto en este caso sería ir introduciendo datos hasta finalizar, y una vez introducido el último dato indicar al ordenador que se ha completado la fase de entrada de datos. Esto se puede realizar de la siguiente forma:

```
100 LET I=1
110 INPUT B
120 IF B=99 THEN GOTO 200
130 LET A(I)=B
140 LET I=I+1
150 GOTO 110
```

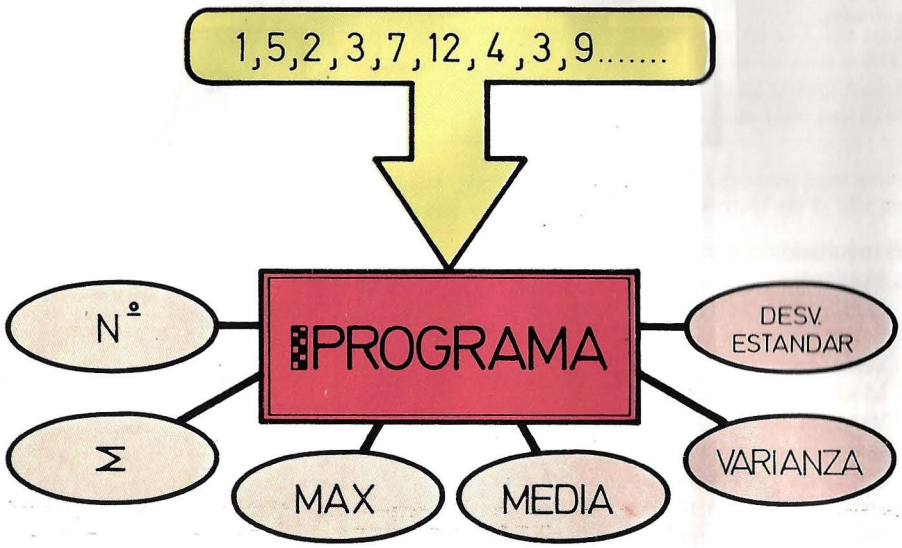
Ahora se irán recogiendo los datos en la variable B. Esta será inspeccionada para ver si contiene el dato 99. Este dato es el que servirá para indicar la finalización de los datos. Si no se ha llegado al final, se pasa el valor de B a la matriz, se incrementa el índice I, y se vuelve a recoger otro dato.
Este método funciona correctamente, salvo por el hecho de que no permite introducir el dato 99 como uno más del conjunto a analizar. Esto se puede solventar de otra forma: utilizando como marca de final de datos un carácter no numérico. Ello hará que se complique el programa un poco más. En primer lugar, será necesario emplear una variable alfanumérica en lugar de B. A esta variable habrá de aplicársele una función que la convierta en dato numérico para su posterior tratamiento aritmético.
La solución más sencilla consiste en utilizar B$ en lugar de B y aplicarle luego la función VAL. El carácter que señale el fin de la introducción puede ser cualquiera no numérico, aunque aquí se utilizará la letra F (de fin). Este sería el aspecto de la nueva rutina:



El lenguaje BASIC brinda todas las herramientas necesarias para crear programas de tipo científico o técnico ejecutables en ordenadores personales.

```
100 LET I=1
110 INPUT B$
120 IF B$="F" THEN GOTO 200
130 LET A(I)=VAL(B$)
140 LET I=I+1
150 GOTO 110
```

CALCULOS



El programa, tras analizar los datos de entrada, proporciona una serie de parámetros estadísticos que dan una idea de las características del conjunto.

Tras la introducción de los datos, el programa ha de proceder al cálculo de los resultados. En este apartado se analizan las rutinas que proporcionan dichos cálculos.
El primer resultado a calcular es el número de datos introducido. Este valor será útil para el cálculo de otros parámetros. El número de datos se extrae directamente de la rutina de entrada. En ella se ha empleado un contador para variar el índice de

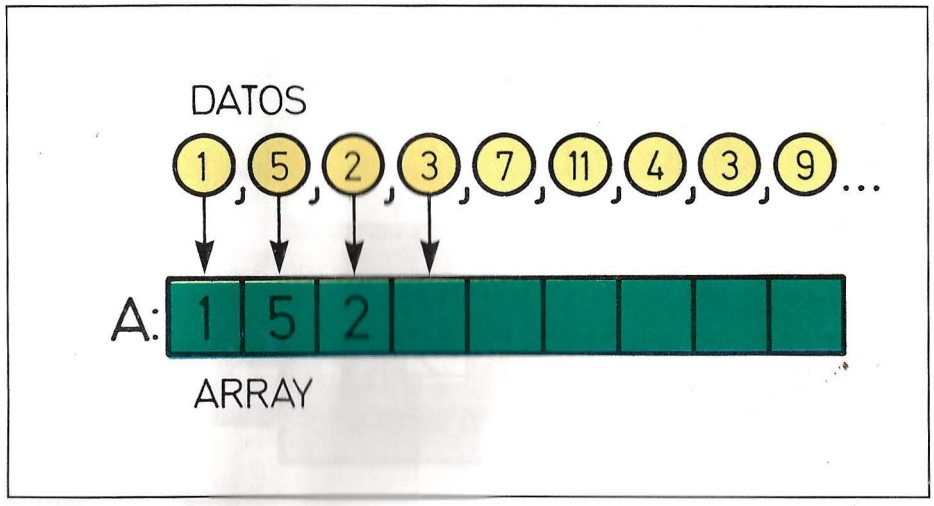
la matriz A. Pues bien, al finalizar la introducción, dicho contador contendrá el número de datos que se han recogido. Así pues, la variable I contiene este primer dato. Para una mayor claridad del programa se traspasará ese dato a una variable específica que llevará el nombre de ND (número de datos):

```
200 LET ND=1
```

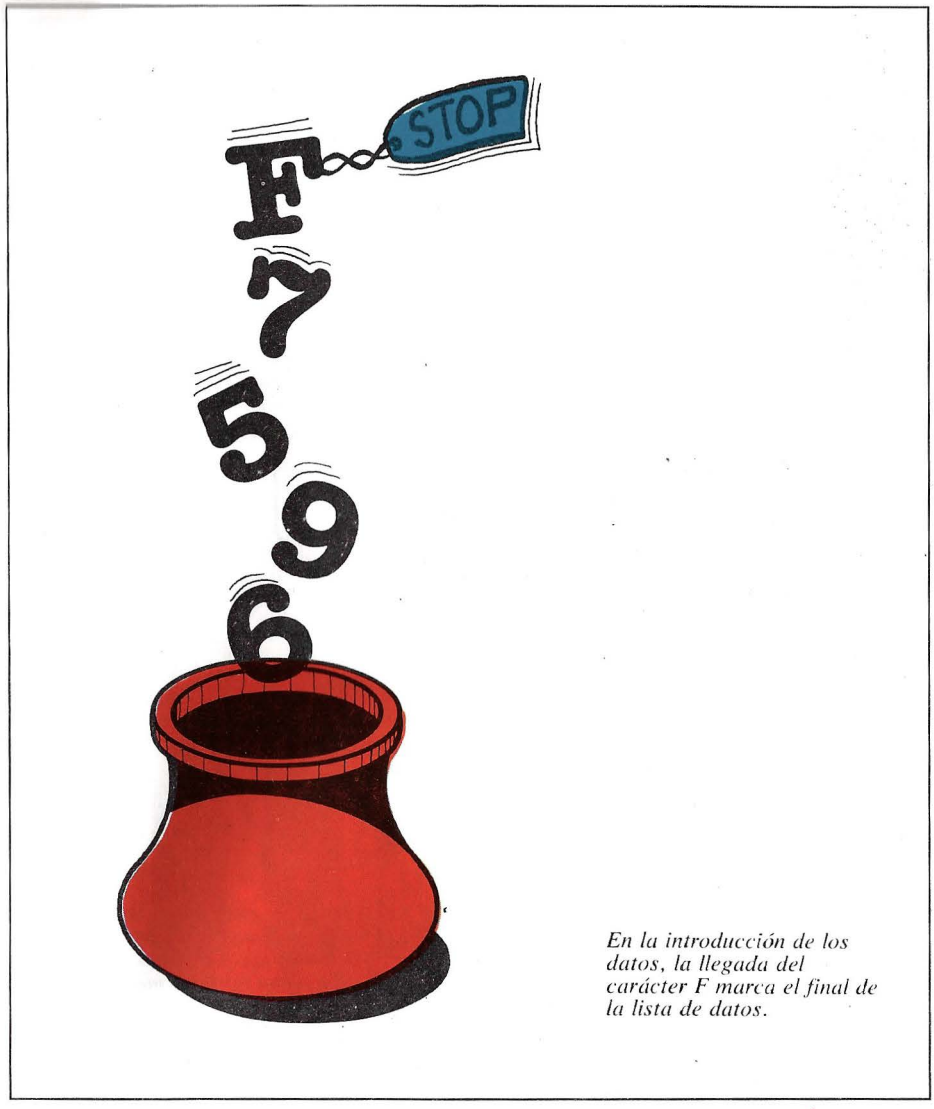
El resto de los resultados necesitan de unos cálculos más complejos que el expuesto. La suma de los datos se ha de realizar empleando un bucle. Los límites de ese bucle son los índices mayor y menor de la matriz. El menor va a ser siempre 1, ya que se empieza a rellenar desde el primer elemento. Sin embargo, el índice mayor dependerá de la cantidad de datos introducidos; o lo que es lo mismo: viene indicado por el recientemente calculado ND.
Una vez conocidos los límites se procede a diseñar el núcleo del bucle. Como se trata de hallar la suma, bastará con ir sumando cada dato en una variable que recoja así el total. Dicha variable recibirá el nombre de SUM. Y la línea que la actualice se limitará a sumar a SUM el valor del dato en curso:

```
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
```

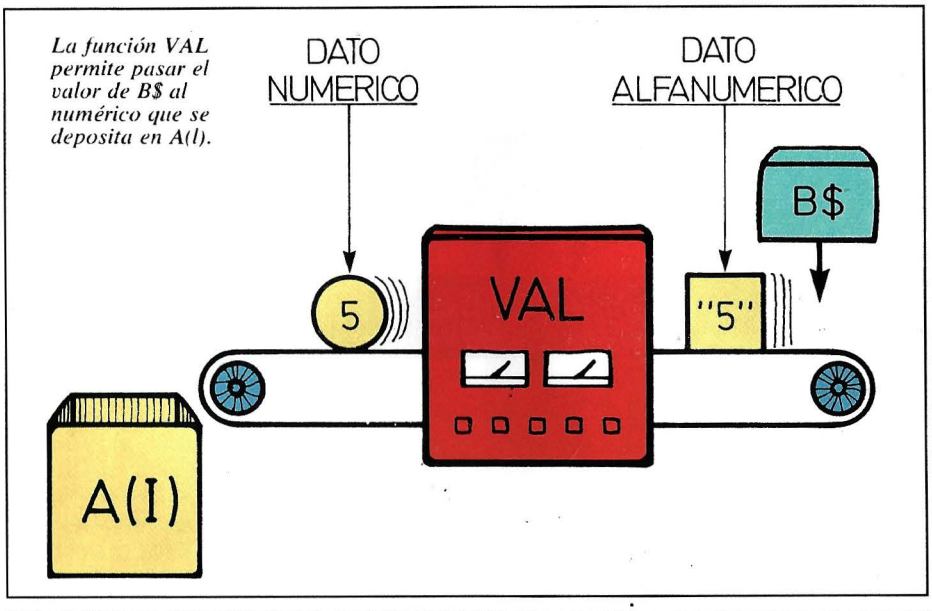
En este bucle se emplea de nuevo la variable I como contador. Ello no plantea ningún problema ya que su valor anterior ha sido almacenado en ND (línea 200). El uso de la misma variable en diferentes zonas del programa (y para usos bien distintos) proporciona un ahorro de memoria. Si se hubiera utilizado la variable J, por ejemplo, como contador en este bucle el ordenador habría reservado espacio para dos variables: I y J. Tal como se ha realizado aquí se ahorra el espacio correspodiente a la supuesta variable J.
La rutina de suma no está del todo finalizada. Es necesario cerciorarse de que se suman todos los datos y nada más que los datos. Que se suman todos los datos se comprueba por el hecho de que el índice I recorre todos los elementos introducidos en A. Para evitar que se sume algo más es preciso verificar que SUM vale cero en la primera pasada del bucle. Si no fuera así,



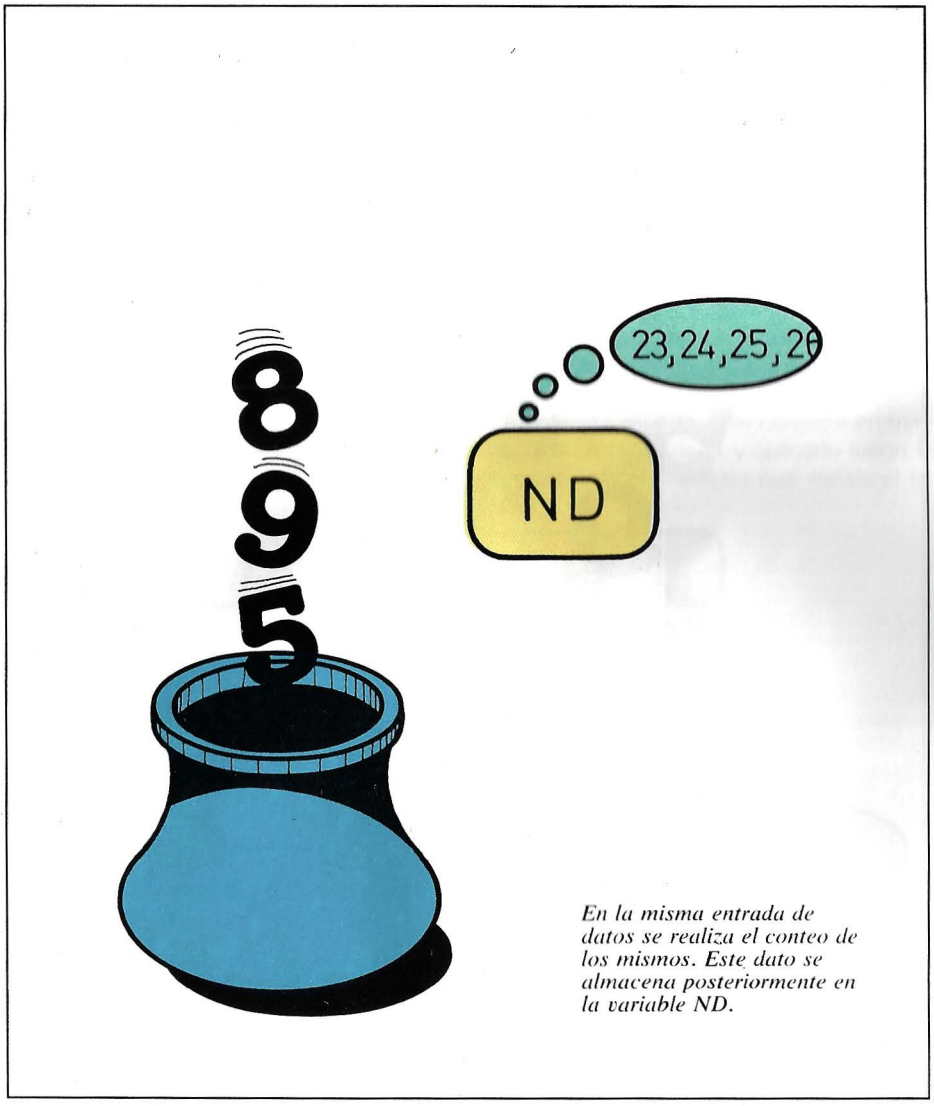
Los datos de partida se introducen en un array para un manejo más cómodo por parte del programa.



En la introducción de los datos, la llegada del carácter F marca el final de la lista de datos.



La función VAL permite pasar el valor de B$ al numérico que se deposita en A(I).



En la misma entrada de datos se realiza el conteo de los mismos. Este dato se almacena posteriormente en la variable ND.

se estaría sumando una cantidad supérflua al total. Así pues, la rutina completa quedaría de la siguiente forma:

```
300 LET SUM=0
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
```

La ejecución de esta rutina dejará el valor de la suma de los datos en la variable SUM.
El siguiente resultado a calcular es el máximo de entre los datos proporcionados. La forma de hallar ese dato consiste en la comparación de los datos dos a dos. Tras cada comparación se retiene el mayor de los dos datos, y es ese dato retenido el que se compara con el siguiente. Tras recorrer todos los datos, no cabe la menor duda de que el dato arrastrado será mayor o igual que cada uno de los que componen el conjunto a analizar. La formulación en BASIC del método indicado es la siguiente:

```
410 FOR I=1 TO ND
420 IF A(I)>MAX T  EN MAX=A(I)
430 NEXT I
```

Como en el caso anterior se ha hecho uso de nuevo de la variable I, ahorrándose así el espacio de una nueva variable.
En esta rutina puede surgir un problema: si el valor inicial de MAX es mayor que cualquiera de los del conjunto, se producirá un error de cálculo. En efecto, el máximo calculado no corresponderá a ninguno de los datos introducidos. Una primera solución consiste en la inicialización adecuada de la variable MAX, por ejemplo a cero. La rutina quedaría de esta forma:

```
400 LET MAX=0
410 FOR I=1 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
```

Pero esta solución no resulta satisfactoria en la totalidad de los casos. Si da la casualidad de que todos los datos introducidos son negativos, el cero será mayor que todos ellos, reproduciéndose el error comentado. Por ello, la mejor solución sería inicializar MAX con un valor de los contenidos en la matriz. De esta forma no se introduciría la posibilidad de comparar con un valor inexistente dentro del conjunto a analizar. Se inicializará MAX con el pri-

mero de los datos introducidos. Esto significa que no es preciso comparar con el primero y, por lo tanto, el bucle puede comenzar en 2. De nuevo se muestra el aspecto de la rutina mejorada:

```
400 LET MAX  A(1)
410 FOR I=2 TO ND
420 IF A(I)  MAX THEN MAX=A(I)
430 NEXT I
```

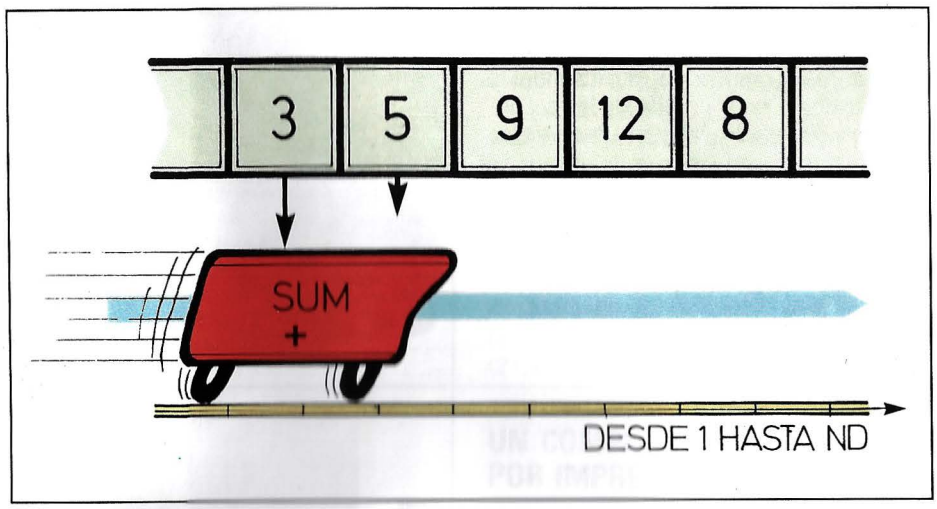
Esta sería, pues, la rutina correcta. El mismo método puede emplearse para calcular el mínimo de los datos. En este caso se habrá de cambiar la comparación y, por supuesto, el nombre de la variable empleada. La rutina capaz de calcular el mínimo es la siguiente:

```
450 LET MIN=A(1)
460 FOR I=2 TO ND
470 IF A(I)<MIN THEN MIN=A(I)
480 NEXT I
```

Los siguientes resultados son los más puramente estadísticos. Para su cálculo se hace uso de algunos de los resultados ya hallados. El primero y más inmediato de los parámetros estadísticos de un conjunto de datos es la media. La media se define en base al número de datos y la suma de los mismos. Matemáticamente coincide con el cociente entre la suma y el número de datos. Su cálculo resulta, pues, muy sencillo partiendo de los datos que ya se conocen.

```
500 MED=SUM/ND
```

Otro parámetro importante es la desviación media. El valor medio (media) recién calculado indica el valor alrededor del cual se distribuye el conjunto de datos. Se llama desviación de un dato a la distancia que le separa de la media. Pues bien, la desviación media es la media de las desviaciones de los datos del conjunto. La desviación media proporciona una medida de la dispersión de los datos.
Para calcular la desviación media es preciso calcular las desviaciones individuales de los datos. Tras ello se ha de proceder al cálculo de la media de esos valores. Esto es, a sumarlos y dividir dicha suma por el número de datos. El cálculo de la desviación y la suma de ese valor al total puede realizarse de una sentada, en el interior del bucle que halla la suma total. He aquí la rutina apropiada:



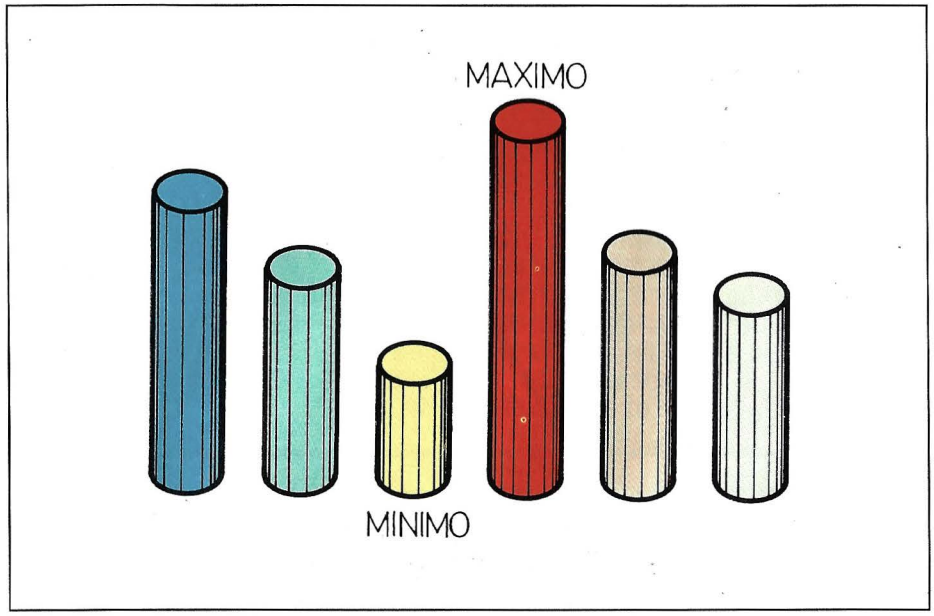
La suma de todos los datos se halla recorriendo la matriz desde el primero hasta el último de los elementos. El total queda en SUM.

```
550 LET SD=0
560 FOR I=1 TO ND
570 SD=SD+ABS(MED-A(I))
580 NEXT I
■
```

El cálculo de la desviación individual se realiza mediante la resta MED−A(I). A ese valor se le aplica la función valor absoluto (ABS) para eliminar el signo, ya que el dato se puede encontrar tanto por encima como por debajo. Al dato así obtenido se le suma en la variable totalizadora SD (suma de desviaciones). En esta última variable quedará retenido el valor de la suma de las desviaciones. Para obtener la desviación media bastará con dividir el valor hallado por el número de datos, tal como se hace en la siguiente línea BASIC:

```
590 LET DM=SD/ND
```

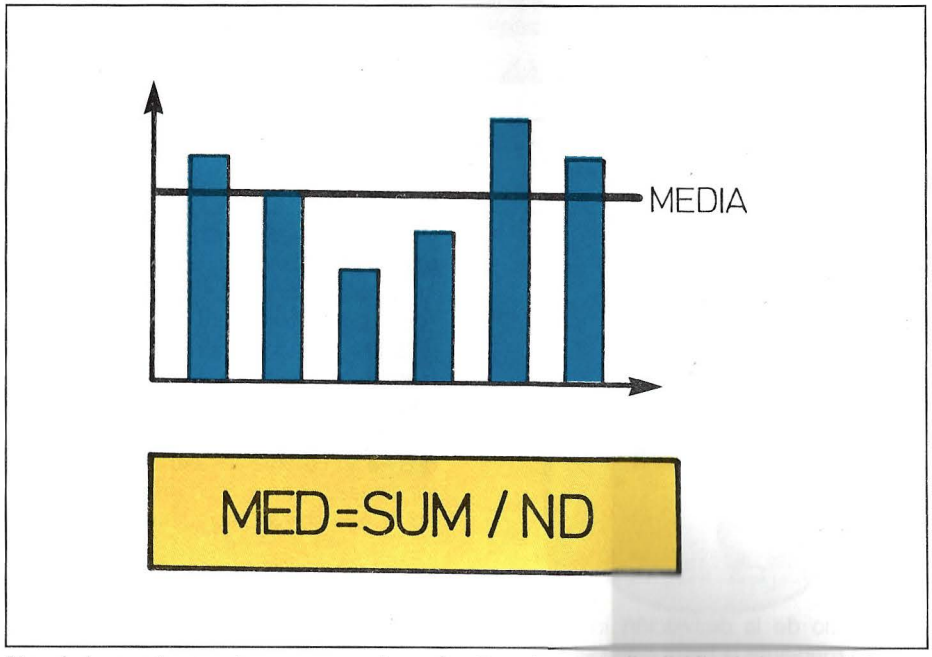
La varianza es otro parámetro que da una idea de la dispersión de los datos. Sin



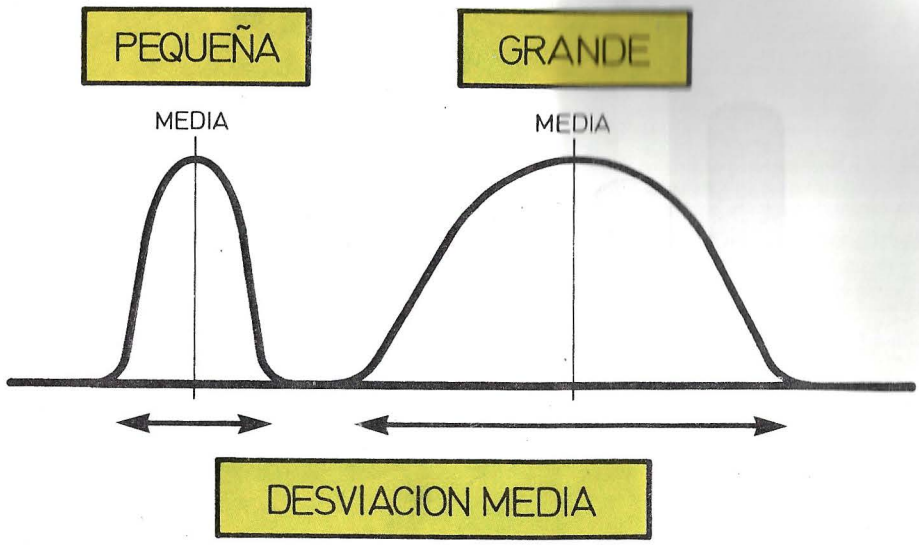
Algunos datos interesantes son el máximo y el mínimo de la distribución.

embargo su cálculo es ligeramente diferente al de la desviación media. Lo que se halla en este caso no es la media de las desviaciones, sino la media de los cuadrados de las desviaciones. Esto requiere decir que la rutina de cálculo de la varianza será muy similar a la de la desviación media. En realidad el único cambio consiste en la sustitución de la función ABS por la operación de elevar al cuadrado la desviación. La rutina tomaría el siguiente aspecto:

```
600 LET SD=0
610 FOR I=1 TO ND
620 SD=SD+(MED-A(I))2
630 NEXT I
640 LET V=SD/ND
```



Uno de los parámetros importantes es la media. Este dato se halla dividiendo la suma de los datos por el número total de los mismos.



La desviación media indica la dispersión de los datos alrededor de la media.

Como se puede apreciar, se ha empleado la misma variable SD para almacenar la suma de los cuadrados. Ello no importa ya que el anterior valor de SD no es ya necesario, y en esta nueva rutina se inicializa de nuevo dicha variable.

El último de los datos a calcular es la desviación estándar. Este valor se asemeja en gran manera a la desviación media. Si la varianza daba un valor basado en los cuadrados de las desviaciones (y no en las propias desviaciones, como la desviación media) la desviación estándar corrige esa diferencia. La corrección consiste en extraer la raíz cuadrada del resultado ofrecido por la varianza. Así pues, la desviación estándar no es más que el resultado de hallar la raíz cuadrada de la varianza. Esto se codifica en BASIC con una sencilla línea:

```
650 LET DE=SQR(V)
```

Y con esto se completa la zona de proceso de los datos. En este momento el programa calcula los resultados situándolos cada uno en una variable. El siguiente paso consiste en mostrar los resultados obtenidos. Este es el tema del siguiente apartado.

PRESENTACION DE LOS RESULTADOS

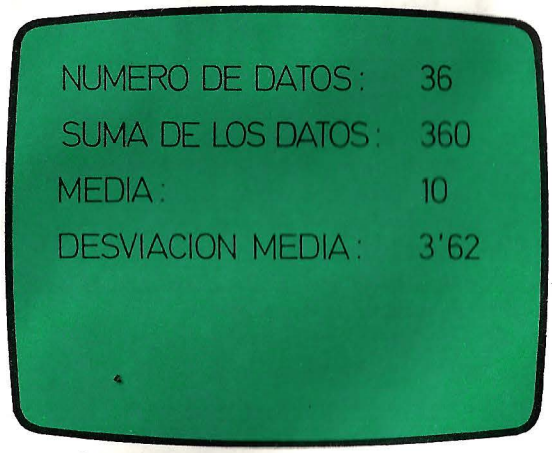
A lo largo de toda la obra se ha comentado la importancia de una buena presentación de los datos. En el presente caso se dispone de los siguientes datos a representar:

- ND, SUM, MAX, MIN, MED, DM, V y DE

Estos siete datos se pueden representar en pantalla por medio del comando PRINT. La forma más sencilla de hacerlo es a través de un conjunto de líneas de este estilo:

(número de línea) PRINT "<nombre del resultado>: ";<variable>

En el argumento de PRINT se sitúa el comentario que identifica el dato a pre-



La presentación de los resultados se limita a la visualización de la cantidad calculada para cada parámetro.

sentar. Tras esa cadena se mostrará el contenido de la correspondiente variable. Entre dato y dato convendrá dejar una línea en blanco, para dar una mayor claridad a la presentación. En conjunto, la rutina de salida de datos puede quedar de la forma siguiente:

```
1000 CLS
1010 PRINT
1020 PRINT "NUMERO DE DATOS: ";ND
1030 PRINT
1040 PRINT "SUMA DE LOS DATOS: ";SUM
1050 PRINT
1060 PRINT "DATO MAXIMO: ";MAX
1070 PRINT
1080 PRINT "DATO MINIMO: ";MIN
1090 PRINT
1100 PRINT "MEDIA ARITMETICA: ";MED
1110 PRINT
1120 PRINT "DESVIACION MEDIA: ";DM
1130 PRINT
1140 PRINT "VARIANZA: ";V
1150 PRINT
1160 PRINT "DESVIACION ESTANDAR: ";DE
```

Esta presentación puede reducirse con el uso de la partícula AT en el argumento de PRINT. Ello, asimismo, permite colocar los datos numéricos en una misma columna. Con esa nueva facilidad la rutina quedaría como sigue:

```
1000 CLS
1020 PRINT AT(1,2) "NUMERO DE DATOS: ";
     AT(21,2) ND
1040 PRINT AT(1,4) "SUMA DE LOS DATOS:
     "; AT(21,4) SUM
1060 PRINT AT(1,6) "DATO MAXIMO: ";
     AT(21,6) MAX
1080 PRINT AT(1,8) "DATO MINIMO: ";
     AT(21,8) MIN
1100 PRINT AT(1,10) "MEDIA ARITMETICA:
     "; AT(21,10) MED
1120 PRINT AT(1,12) "DESVIACION MEDIA:
     "; AT(21,12) DM
1140 PRINT AT(1,14) "VARIANZA: ";
     AT(21,14) V
1160 PRINT AT(1,16) "DESVIACION ESTAN-
     DAR: "; AT(21,16) DE
```

Los enunciados se colocan todos comenzando en la primera columna por medio de AT(1,Y). También se sitúan en la misma columna los datos; esta vez en la columna 21, siendo el argumento de AT (21,Y).

El programa permite la salida de resultados por impresora. Ello evita la necesidad de tomar nota «a mano».

UN COMPLEMENTO: SALIDA POR IMPRESORA

En programas de este tipo es frecuente la necesidad de apuntar los resultados obtenidos. Ello permite conservarlos y evita tener que ejecutar el programa cada vez que se desea volver a verlos. El método más sencillo de almacenamiento de los resultados consiste en plasmarlos en papel, a través de una impresora. Como complemento al programa expuesto se añade una rutina de impresión de los datos.

En principio, la rutina de impresión puede ser opcional. Así pues, el programa ha de preguntar al operador si desea esa impresión. Para ello se ha de incluir una rutina que recoja la respuesta del usuario. He aquí una posible rutina:

```
2000 PRINT
2010 PRINT "DESEA SALIDA IMPRESA
     (S/N)?"
2020 LET B$=INKEY$
2030 IF B$="N" THEN END
2040 IF B$<>"S" THEN GOTO 2020
2100 LPRINT "ANALISIS ESTADISTICO"
2110 LPRINT:LPRI  T "LOS DATOS INTRO-
     DUCIDOS SON:"
2120 FOR I=1 TO ND
2130 LPRINT A(I);
2140 NEXT I
2150 LPRINT:LPRINT "RESULTADOS CALCU-
     LADOS:"

3010 LPRINT
3020 LPRINT "NUMERO DE DATOS: ";ND
3030 LPRINT
```

```
5 REM ESTADISTICA/SOFT
10 DIM A(100)
99 REM ENTRADA DE DATOS
100 LET I=1
110 INPUT B$
120 IF B$="F" THEN GOTO 200
130 LET A(I)=VAL(B$)
140 LET I=I+1
150 GOTO 110
199 REM NUMERO DE DATOS
200 LET ND=I
299 REM SUMA
300 LET SUM=0
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
399 REM MAXIMO
400 LET MAX=A(1)
410 FOR I=2 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
449 REM MINIMO
450 LET MIN=A(1)
460 FOR I=2 TO ND
470 IF A(I)<MIN THEN MIN=A(I)
480 NEXT I
499 REM MEDIA
500 MED=SUM/ND
549 REM DESVIACION MEDIA
550 LET SD=0
560 FOR I=1 TO ND
570 SD=SD+ABS(MED-A(I))
580 NEXT I
590 LET DM=SD/ND
599 REM VARIANZA
600 LET SD=0
610 FOR I=1 TO ND
620 SD=SD+(MED-A(I))2
630 NEXT I
640 LET V=SD/ND
649 REM DESVIACION ESTANDAR
650 LET DE=SQR(V)
999 REM PRESENTACION EN PANTALLA
1000 CLS
1020 PRINT AT(1,2) "NUMERO DE DATOS: "; AT(21,2) ND
1040 PRINT AT(1,4) "SUMA DE LOS DATOS: "; AT(21,4) SUM
1060 PRINT AT(1,6) "DATO MAXIMO: "; AT(21,6) MAX
1080 PRINT AT(1,8) "DATO MINIMO: "; AT(21,8) MIN
1100 PRINT AT(1,10) "MEDIA ARITMETICA: "; AT(21,10) MED
1120 PRINT AT(1,12) "DESVIACION MEDIA: "; AT(21,12) DM
1140 PRINT AT(1,14) "VARIANZA: "; AT(21,14) V
1160 PRINT AT(1,16) "DESVIACION ESTANDAR: "; AT(21,16) DE
1999 REM PRESENTACION EN IMPRESORA
2000 PRINT
2010 PRINT "DESEA SALIDA IMPRESA (S/N)?"
2020 LET B$=INKEY$
2030 IF B$="N" THEN END
2040 IF B$<>"S" THEN GOTO 2020
2100 LPRINT "ANALISIS ESTADISTICO"
2110 LPRINT:LPRINT "LOS DATOS INTRODUCIDOS SON:"
2120 FOR I=1 TO ND
2130 LPRINT A(I);
2140 NEXT I
2150 LPRINT:LPRINT "RESULTADOS CALCULADOS:"
3010 LPRINT
3020 LPRINT "NUMERO DE DATOS: ";ND
3030 LPRINT
3040 LPRINT "SUMA DE LOS DATOS: ";SUM
3050 LPRINT
3060 LPRINT "DATO MAXIMO: ";MAX
3070 LPRINT
3080 LPRINT "DATO MINIMO: ";MIN
3090 LPRINT
3100 LPRINT "MEDIA ARITMETICA: ";MED
3110 LPRINT
3120 LPRINT "DESVIACION MEDIA: ";DM
3130 LPRINT
3140 LPRINT "VARIANZA: ";V
3150 LPRINT
3160 LPRINT "DESVIACION ESTANDAR: ";DE
```

Listado completo del PROGRAMA.

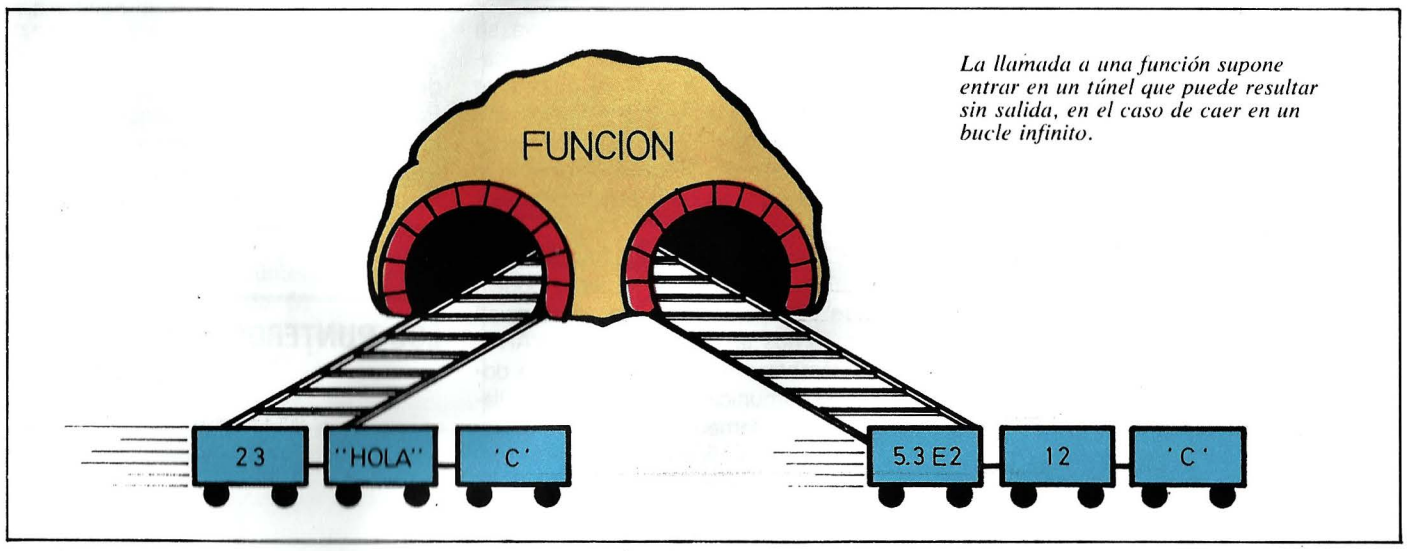
```
3040 LPRINT "SUMA DE LOS DATOS:
     ";SUM
3050 LPRINT
3060 LPRINT "DATO MAXIMO: ";MAX
3070 LPRINT
3080 LPRINT "DATO MINIMO: ";MIN
3090 LPRINT
3100 LPRINT "MEDIA ARITMETICA: ";MED
3110 LPRINT
3120 LPRINT "DESVIACION MEDIA: ";DM
3130 LPRINT
3140 LPRINT "VARIANZA: ";V
3150 LPRINT
3160 LPRINT "DESVIACION ESTANDAR: ";DE
```

En esta rutina se ha añadido la impresión de los datos introducidos (líneas 2100 a 2140). Ello permite tener juntos datos y resultados. El resto de la rutina es, en esencia, idéntica a la de presentación en pantalla. La única variación se manifiesta en el uso de LPRINT en lugar de PRINT.

El lenguaje C (4)

Funciones y punteros

Un elemento básico asociado a la estructura fundamental del lenguaje C —la función— es el puntero. Los conocedores del PASCAL saben que los punteros son un tema difícil, aunque bien es cierto que sin ellos se pueden hacer muchas cosas en PASCAL. En C, por el contrario, es muy raro encontrar un programa que no haga uso de ellos.



La llamada a una función supone entrar en un túnel que puede resultar sin salida, en el caso de caer en un bucle infinito.

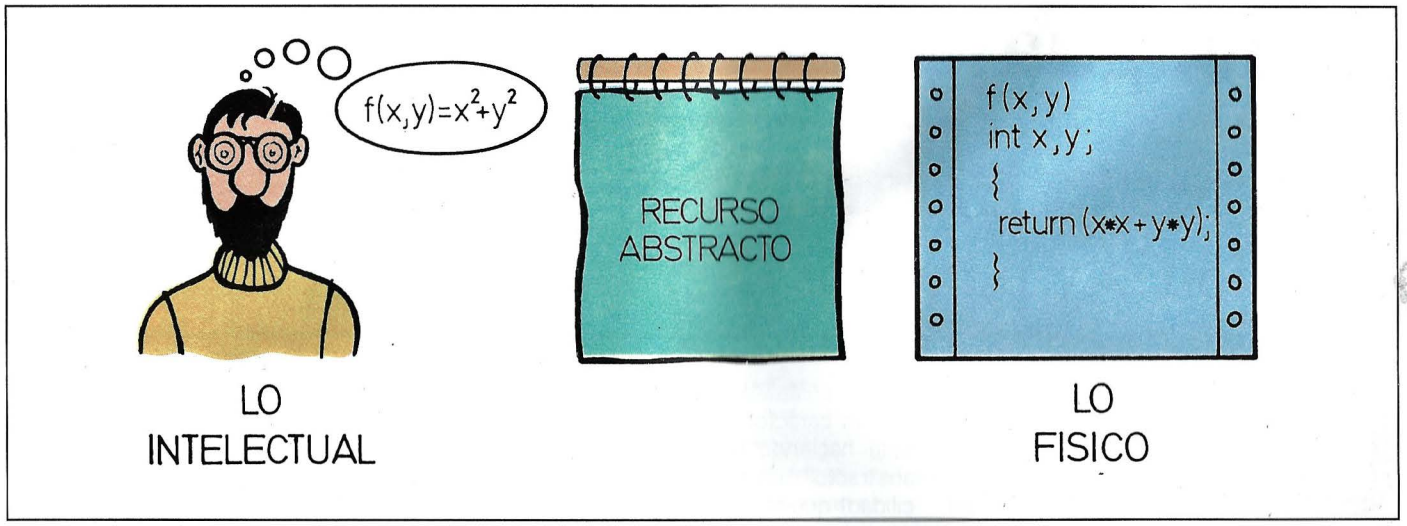
ALGO MAS SOBRE FUNCIONES: EL "RECURSO ABSTRACTO"

Una función puede compararse con una "caja negra", en la que se introducen ciertos valores, y la cual devuelve otros valores de salida como resultado de procesar los primeros. Siempre que se crea una función, aparte de estar contribuyendo a dar un carácter modular al programa, se está haciendo referencia a un "recurso abstracto": un recurso (un medio, una facilidad) que no posee la máquina en sí y que el usuario le entrega a partir de ese momento.

Veamos un ejemplo:
En C no existe operador de potenciación, así que si se quieren realizar elevaciones al cuadrado, habrá que dotar al ordenador de un recurso (abstracto, porque en realidad no lo tiene) tal como éste:

```
#define E 2.718281828
main()
{
    double número,su_cuadrado;
    double al_cuadrado();

    número= E;
    su_cuadrado= al_cuadrado
    (número);
    printf("%f\n",su_cuadrado);
}
double al_cuadrado (x)
double x;
{
    return (x*x);
}
```

¡Atención con las declaraciones de funciones! El programa constituye un ejemplo correcto de declaración.



El «recurso abstracto» es la barrera que separa y une a la vez dos campos distintos.

```
double al_cuadrado (x)
double x;
{
        return (x*x);
}
```

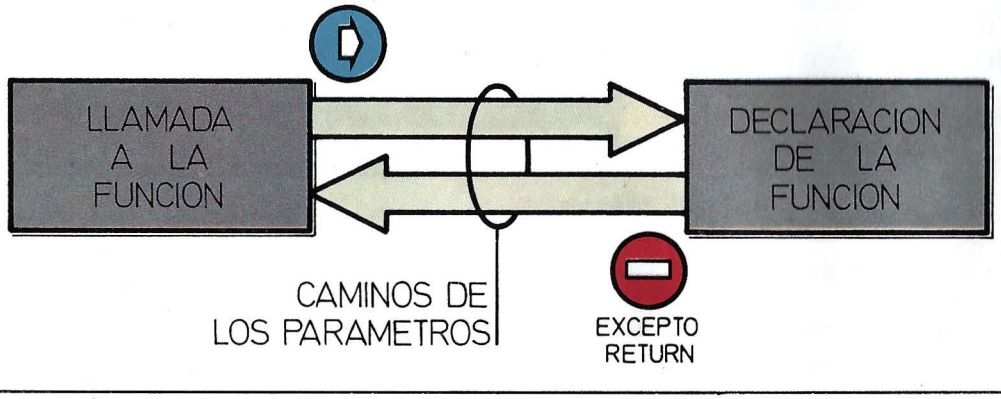
Aun a pesar de sus connotaciones filosóficas, no hay que perder de vista que los recursos abstractos son pilares fundamentales sobre los que descansa la metodología de la programación estructurada.
Unas líneas más arriba se ha declarado una función que devuelve un valor en doble precisión. En una de las figuras aparece un programa que la utiliza. Como se observa, si se utiliza en un ámbito ("main" en nuestro caso) una función que no devuelve un entero, es preciso declarar en tal ámbito el tipo de la función:

```
double al_cuadrado ();
```

Esto es necesario porque, como ya se comentó en otro capítulo, todo identificador del que no se especifica el tipo se supone entero. Si no se hubiera declarado "al_ cuadrado" en "main", el compilador supondría que iba a devolver un entero, lo cual está en desacuerdo con la posterior declaración.
Otro punto importante: en PASCAL, los parámetros de un procedimiento podían pasarse "por valor" o "por referencia" (estos últimos también llamados "VAR"). Los parámetros VAR establecen una doble vía de comunicación entre la parte "llamadora" y la "llamada", ya que el parámetro actual es modificado al finalizar la ejecución del procedimiento. En C todos los parámetros pasados lo son por valor, es decir, el único medio que hasta ahora conocemos de comunicación en C entre una función y su "llamador" es a través de "return". Como luego se verá, la forma de "imitar" a un parámetro VAR es a través de los punteros.
Para finalizar, hay que mencionar que al igual que en PASCAL, una función puede llamarse a sí misma recursivamente. La recursividad es una propiedad muy importante y que sirve para muchísimas más cosas que para calcular factoriales.

LOS PUNTEROS DEL C

C es un lenguaje que hace un uso intensivo de punteros. Hay ocasiones en las



En el lenguaje «C», el camino hacia las funciones es de una sola dirección.

que la única manera de expresar un cálculo es a través de punteros. Otras veces el uso de punteros reduce el tamaño del código objeto de los programas, aumentando su velocidad de ejecución. No obstante, el uso indiscriminado de punteros puede conducir a la creación de monstruos (programas) irreconocibles incluso para su propia madre (el programador). Hay que procurar, por lo tanto, ser cuidadosos con su uso.

Conviene recordar en este punto que, por muy inspirados que sean los nombres otorgados a las variables y por muy estructurado que resulte el programa, tras la compilación sólo quedarán direcciones y datos. Es bueno, por tanto, saber qué pasa con las direcciones y los datos.

Un puntero es una variable que no contiene un dato, sino que se utiliza para apuntar a una variable; es decir, contiene la dirección de memoria en la que se encuentra la variable. Puede accederse al dato contenido en la variable directamente, a través de su nombre, o indirectamente, a través de un puntero que la señale.

Con un puntero se pueden hacer básicamente dos cosas:

1.° Almacenar en el puntero la dirección de una variable (operador &).
2.° Acceder al valor contenido en la dirección señalada (operador *).

Para comenzar, tomemos la siguiente declaración de puntero:

```
char *punt;
```

Ello significa que "punt" señalará hacia una posición de memoria que contiene un carácter o, de otra manera, que "*punt" contendrá un carácter. Sustituyendo "char" por otros declaradores de tipo se tendrán punteros a distintos tipos. Ahora, podemos declarar e inicializar algunas variables:

```
char letra1,letra2;
letra1= 'y' ;
```

Al hacer:

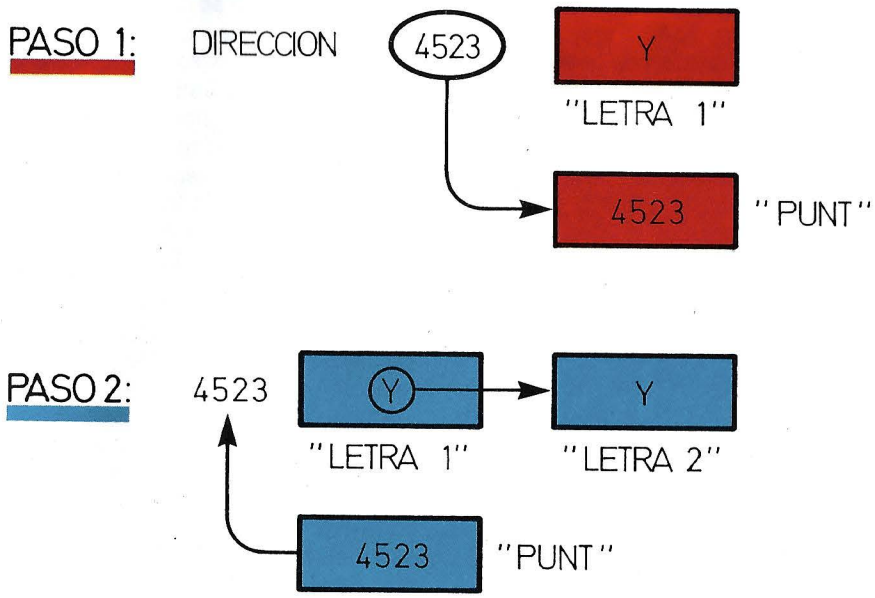
```
punt=&letra1;
```

la variable "punt" está apuntando hacia "letra1", y al hacer:

```
main()
{
    int x=6;

    printf("%d\n",factorial(x));
}
factorial (n)
int n;
{
    if (n==1)
        return (1);
    else
        return (n*factorial(n_1));
}
```

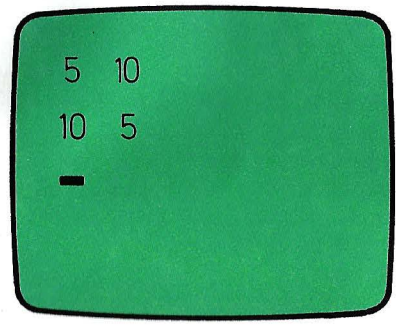
Un ejemplo de función recursiva.

```
main()
{
    char *punt;
    char letra1,letra2;

    letra1= 'y';
    punt= &letra1; /* paso1 */
    letra2= *punt; /* paso2 */
    printf ("%c\n",letra2);
}
```



Los punteros: viaje al país de los trabalenguas.

```
main ()
{
    int a=5,b=10;

    printf("%d %d \n",a,b);
    cambia (&a,&b);
    printf("%d %d \n",a,b);
}
cambia (px,py)
int *px,*py
{
    int temporal;

    temporal= *px;
    *px= *py;
    *py= temporal;
}
```



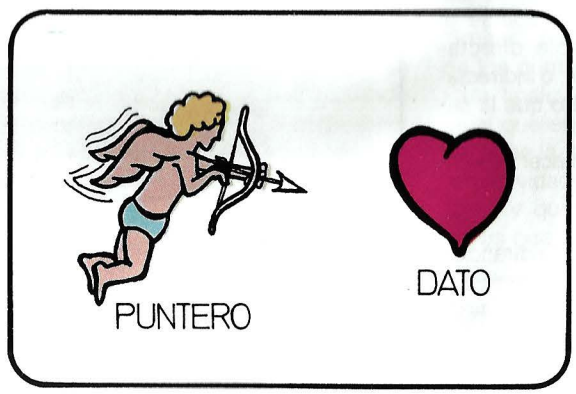
Un parámetro al estilo VAR se pasa a través de la dirección de la variable, no de su valor concreto. La pantalla adjunta muestra el resultado de la ejecución del presente programa.

letra2= *punt;

"letra2" contiene el dato al que apuntaba "punt". Todos estos pasos se ven reflejados en la correspondiente figura.
El resultado de todo este ir y venir es análogo al que se hubiera obtenido haciendo:

letra2=letra1;

una vez que se asignó a "letra1" el valor ỳ.



Los punteros son un recurso muy frecuente en la programación en lenguaje «C».

ARGUMENTOS DE FUNCIONES Y PUNTEROS

En este punto, se está ya en disposición de tratar el problema del paso de argumentos por referencia. Se puede plantear el problema de requerir una función que devuelva dos valores; esto no puede lograrse tal como se venía haciendo hasta ahora, a través de "return". Suponga una función que debe intercambiar los valores de dos variables enteras. Si ya se tienen las variables declaradas y asignadas, no bastaría con escribir:

cambia (a,b);

en donde "cambia" ha sido definido como sigue:

```
cambia (x,y)
int x,y;
{
        int temporal;

        temporal=x;
        x=y;
        y=temporal;
}
```

ya que como en C los parámetros son pasados por valor, a "x" e "y" les son asignados los valores de "a" y "b", pero al finalizar "cambia" las variables originales permanecen inalteradas.
La solución consiste en pasar las direcciones de "a" y "b" en vez de sus valores, realizando en "cambia" un intercambio de contenido de direcciones tal como se observa en la ilustración. Al llamar a "cambia" desde "main" se asignan a "px" y a "py" las direcciones de "a" y "b", no sus valores como se venía haciendo hasta ahora. Una vez en el cuerpo de "cambia", cada vez que se escribe "*px" o "*py" se hace referencia a los contenidos de las direcciones a las que apuntan "px" y "py", que son las de "a" y "b". Aunque resulte complicado a primera vista, se recomienda al lector que trabaje sobre este ejemplo y se dé cuenta de que realmente se consigue el efecto deseado.
Cabe hacer algún comentario adicional sobre las funciones en C. En primer lugar, las funciones no tienen por qué llevar obligatoriamente un "return"; al igual que en el ejemplo, puede pensarse en lo innecesario de un "return" en una función que simplemente ejecuta un bucle para producir un retardo. Por lo demás, como consecuencia de la no existencia del "return", el tipo de la función es irrelevante, ya que no es asignado el resultado a ningún valor, como ocurre en la llamada a "cambia" que se produce en "main".

Apple Macintosh (2)

Una original filosofía de trabajo

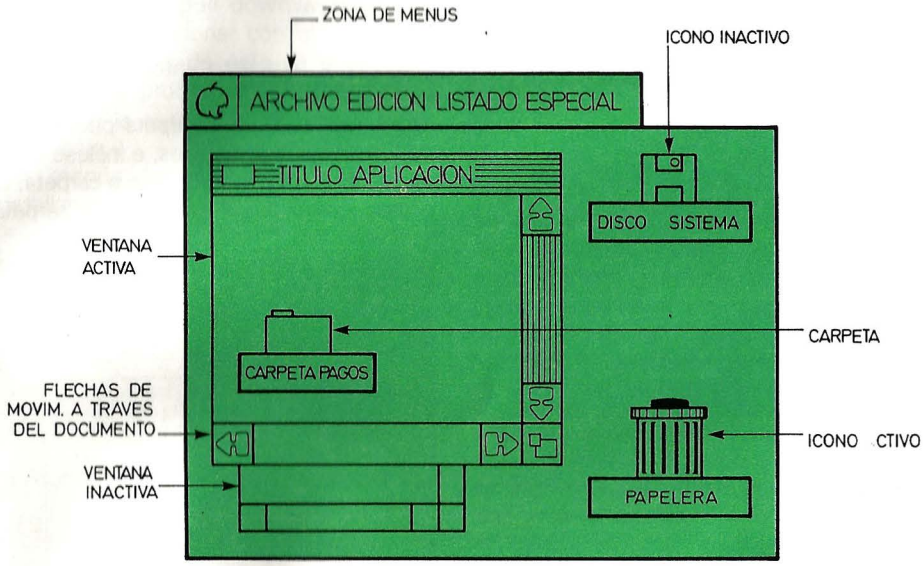
La aparición en el mercado del ordenador Apple Macintosh introdujo una nueva concepción de la zona del sistema operativo encargada de "contactar" con el usuario. La inclusión de imágenes (también llamadas *iconos)* representativas de los distintos elementos y funciones, contribuye en gran medida a crear una interface hombre-máquina de manejo sencillo y cómodo que simplifica la comunicación del sistema operativo con el usuario.

Está comprobado que las interfaces "amistosas" con el usuario son preferidas por la mayoría de las personas que han tenido que enfrentarse con un sistema operativo, frente a las clásicas interfaces alfanuméricas. No cabe duda que el trabajo con ratón ("mouse"), ventanas, iconos y menús de tipo persiana resulta más cómodo, ameno e incluso menos propenso a errores.

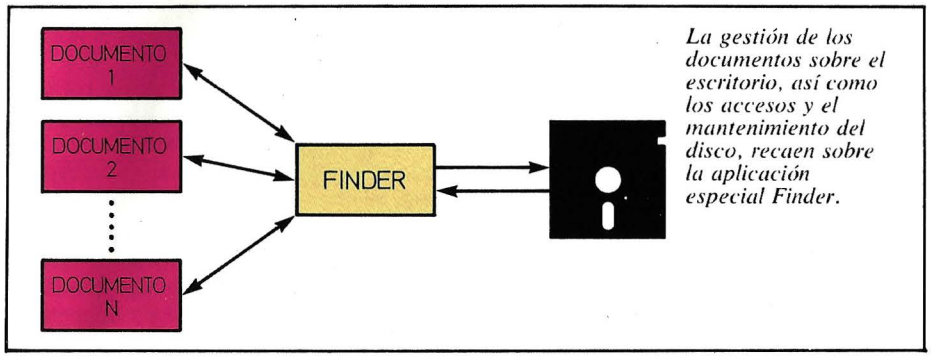
La interface con el usuario del sistema operativo que equipa al Macintosh trata de reproducir las condiciones que reinan en una oficina de corte tradicional. La pantalla representa un escritorio sobre el cual pueden extenderse diversas carpetas y documentos, accesibles a través de ventanas que se abren o se cierran.

Las ventanas pueden moverse de un lugar a otro de la pantalla, ampliar su superficie o reducirla a voluntad del usuario. Este puede, en consecuencia, redistribuir su escritorio y elegir la zona del mismo en la que desea emplazar el documento que estime oportuno. Cada documento pertenecerá o será generado desde una aplicación, la cual permitirá su gestión.

La agrupación de documentos en carpetas con el fin de ordenarlos convenientemente, adoptada por el Macintosh, tiene su contrapartida en el agrupamiento de los ficheros en directorios propio de la mayoría de los sistemas operativos. Una ventaja esencial del Macintosh por lo que se refiere al manejo de documentos, consiste en la utilización de un "ratón": dispositivo que se desliza sobre la mesa y dirige el cursor a través de la pantalla. El ratón permite posicionar el cursor sobre los iconos representativos de archivos, programas, utilidades y volúmenes de almacenamiento, así como en zonas especiales de las ventanas para actuar sobre estos elementos. También la selección de menús y opciones internas se realiza por medio de esta técnica, para cuya puesta en práctica no es necesario el aprendizaje de un elevado número de comandos; ello posibilita un uso casi instantáneo de las aplicaciones. Los países de habla anglosajona describen a este modo de trabajo con las siglas WYSIWYG (what you see is what you get): lo que se ve es lo que se realiza.



El escritorio contiene los objetos sobre los que se va a actuar: iconos, documentos, ventanas, menús...



La gestión de los documentos sobre el escritorio, así como los accesos y el mantenimiento del disco, recaen sobre la aplicación especial Finder.

EL FINDER

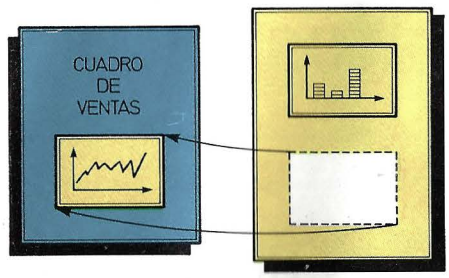
El Finder es la aplicación genérica de que dispone el Macintosh para manejar documentos y sobre la que recae la responsabilidad del intercambio de información entre el usuario y la máquina; de ahí que pueda ser considerado como el verdadero sistema operativo.
La totalidad de las acciones a realizar sobre un documento están contempladas en esta aplicación; así pues, el Finder puede compararse con el pasillo de una casa, a través del cual se puede acceder a las diferentes habitaciones (aplicaciones) y comunicarse con el mundo exterior (disco, impresora, etc.). Al utilizar otras aplicaciones distintas del Finder se sigue disponiendo de funciones propias de éste: abrir nuevos documentos, examinar documentos ya existentes, imprimir documentos, guardar documentos en disco, etc. La actuación del Finder se centra sobre las siguientes entidades:

• Discos

Para visualizar el contenido de un disco en particular es necesario activar el icono que lo representa (disco blanco para disco insertado, gris para disco ya expulsado y negro para disco seleccionado). Una vez activado el disco se abre una ventana que muestra los diversos elementos que residen en él (documentos, aplicaciones y carpetas).
Cada uno de estos elementos está representado a su vez por un icono que lo identifica, y puede ser seleccionado o desplazado a través del escritorio. El Finder conserva el contenido de un disco aunque éste haya sido expulsado de la unidad de lectura/escritura, siendo perfectamente admisible activar algún documento o aplicación de dicho disco; el propio Finder se encargará de pedir al usuario la inserción del disco correspondiente si ello es necesario.

• Carpetas

Las carpetas en el Macintosh tienen la misma misión que una carpeta de oficina: contener documentos y aplicaciones ordenados jerárquicamente. La visualización de su contenido se realiza de un modo



La creación y modificación de documentos se realiza con las utilidades del menú de edición. Sus funciones (entre ellas las de «cortar» y «pegar») permiten mezclar todo tipo de documentos, tanto alfanuméricos como gráficos.

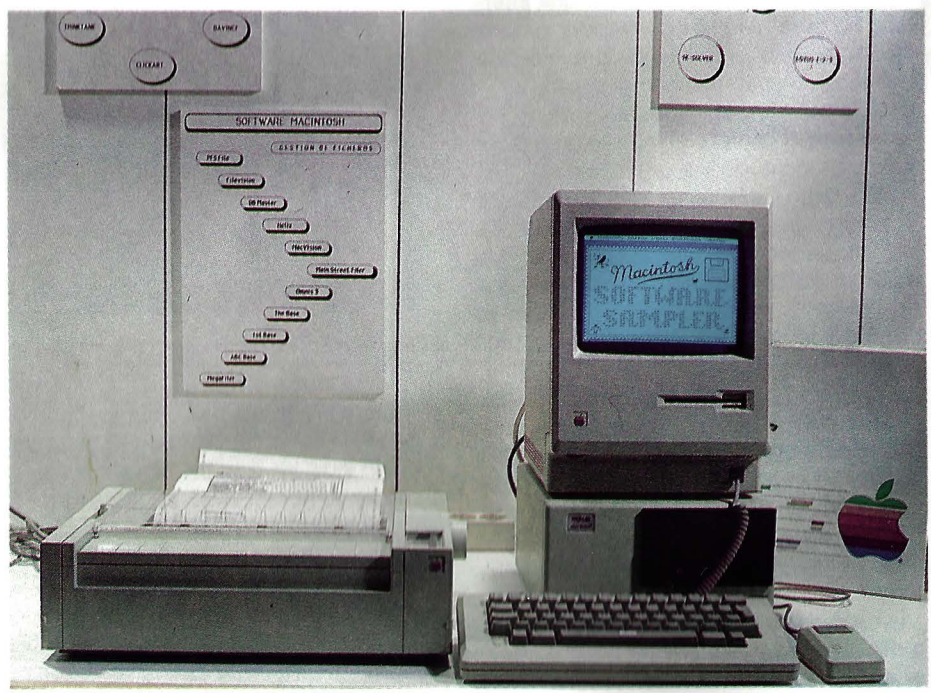
análogo al expuesto anteriormente para un disco.
El contenido de una carpeta pueden ser documentos, aplicaciones, e incluso otras carpetas. Para crear una nueva carpeta, el método más usual es copiar la carpeta vacía contenida en el disco del sistema y darle un nombre adecuado. No está limitado el número de carpetas utilizables.
Para rellenar una carpeta basta con arrastrar el icono seleccionado hasta colocarlo sobre ésta. Tan sólo habrá que seguir el proceso inverso en el caso de que la acción deseada sea eliminar un documento de la carpeta.

• Escritorio

Como se ha mencionado con anterioridad, el escritorio es el lugar adecuado para colocar los iconos temporalmente. La selección de iconos se realiza por medio del ratón y sobre la ventana en la que se encuentren. Puede actuarse sobre un único icono o sobre un grupo de ellos, según se desee.

• Papelera

El comando DELETE (o ERASE) común en casi todos los sistemas operativos y utilizado para borrar información, ha sido sustituido en el Macintosh por un concepto mucho más gráfico: la papelera.
A la papelera van a parar (mediante arrastre) todos los documentos y carpetas que el usuario desee eliminar.
En todo momento se puede activar la papelera y consultar su contenido, de tal forma que incluso está permitido retirar un documento de la papelera para devolverlo al escritorio.
La papelera se vacía automáticamente cada vez que se pone en marcha una aplicación, cada vez que se expulse un disco, o cada vez que se graba algún documento



El Macintosh de Apple está especialmente diseñado para el trabajo de oficina, siguiendo el mismo esquema de actuación que se presenta en una oficina tradicional.

en un disco; en estas situaciones se pierde irremediablemente su contenido.

- **Portapapeles**

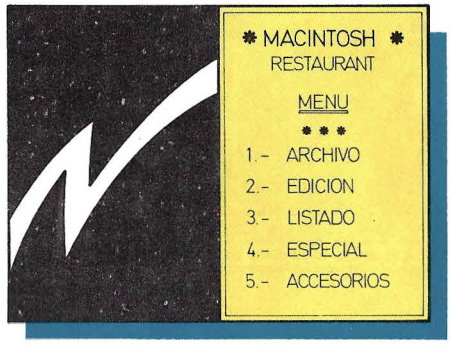
Los datos que se mueven o copian de un documento en proceso de creación o modificación son reservados en una zona especial llamada portapapeles. Los comandos de Pegar, Copiar y Cortar del menú de Edición permiten transferir los datos del portapapeles a la ventana activa y viceversa; así pues, por medio del portapapeles pueden transferirse datos a otras zonas del documento activo o incluso a otros documentos.

- **Documentos**

Las acciones del Finder sobre documentos se centran en su apertura, cierre, copia, eliminación, desplazamiento y cambio de nombre.

MENUS DEL FINDER

Las diversas tareas a realizar en el escritorio se activan a través de varios menús de tipo persiana ("pull down"). Para ello, basta con seleccionar con el "ratón" el menú y, dentro de éste, la opción deseada. Este procedimiento facilita enormemente la asimilación de los métodos de trabajo.
Los menús disponibles en el Finder se detallan a continuación:



La manipulación de los documentos se realiza con el apoyo de los menús del Finder; su comodidad de uso y activación obvia el aprendizaje de una larga lista de comandos.

- **Menú Apple**

Contiene los accesorios de escritorio disponibles en el Macintosh. Estos son:
— Calculadora: permite realizar las mis-

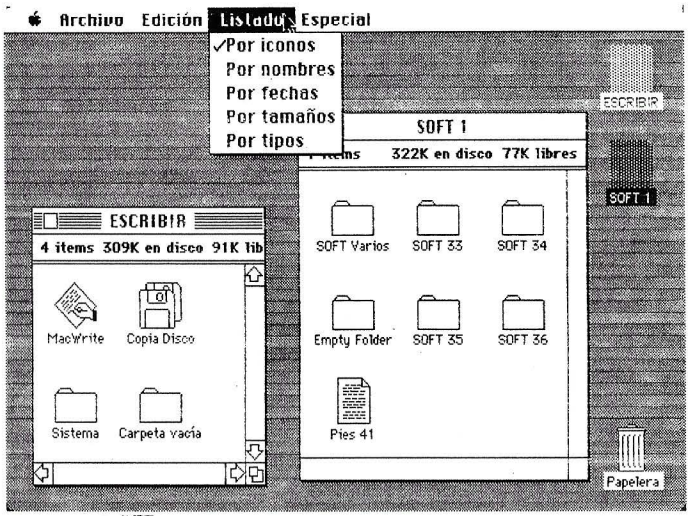
La voz del ordenador

La conexión al ordenador de una pantalla de tubo de rayos catódicos (CRT) y de un teclado similar al de una máquina de escribir convencional, supuso un gran avance en la comunicación hombre-máquina, frente a los arcaicos métodos de introducción de información en el ordenador por medio de interruptores y, más tarde, de tarjetas perforadas.

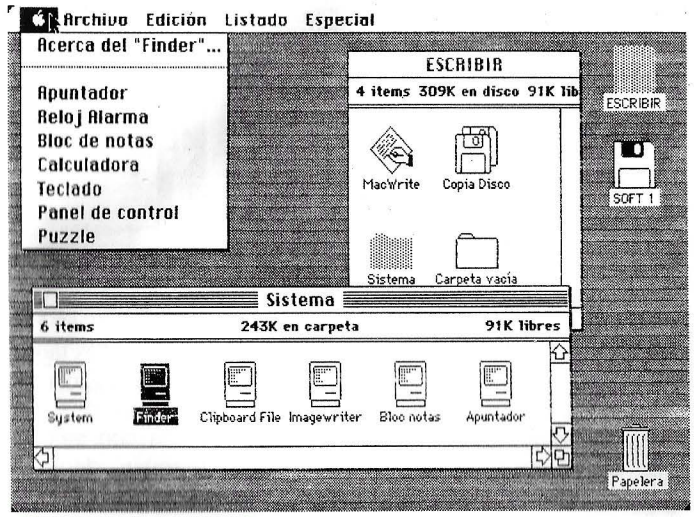
No obstante, el uso del teclado y la pantalla sigue coartando al hombre en su comunicación con la máquina, ya que ésta no se realiza en el modo más natural y expresivo de que dispone: el habla.
Actualmente, la generación de voz por ordenador se logra a partir de dos métodos diferentes: codificación y síntesis.

El primer método pasa por digitalizar las ondas sonoras que constituyen las palabras, almacenando en un soporte permanente (disco, disquete, cinta, etc.) el resultado de la digitalización. De esta manera, el ordenador dispone de un banco de palabras para ser utilizadas en el momento oportuno. Esta técnica genera vocabularios de una gran calidad, ya que el proceso de digitalización de la voz introduce poco ruido que distorsione la señal sonora. Por contra, la cantidad de palabras disponibles suele ser bastante limitada puesto que la aplicación de este método exige una gran cantidad de memoria.
La síntesis de voz se produce a partir de chips especiales que son capaces de generar un amplio número de fonemas independientes que al combinarlos adecuadamente forman palabras. Así pues, el número de palabras generable es mucho más amplio que en el caso anterior, al estar limitadas las posibles combinaciones fonéticas sólo por unas pocas reglas; como contrapartida, la calidad de la voz no es excesivamente brillante.
La otra cara de la moneda la representa el reconocimiento de voz por parte del ordenador. Esta es una tarea que dada su complejidad dista mucho de estar concluida. El principal inconveniente reside en que el lenguaje humano es intrínsecamente ambiguo, lo que complica en gran medida su tratamiento por parte del ordenador.
A pesar de todo se han realizado logros sustanciales en el reconocimiento de la voz con vocabularios amplios, aunque con una forma gramatical simplificada resultante de eliminar las palabras menos significativas de las frases. Las diferentes entonaciones de varios interlocutores suponen igualmente un grave problema, de manera que los sistemas de reconocimiento de voz están fuertemente ligados a la persona que previamente proporciona los patrones de comparación, resultando problemático su uso por otras personas.
Aunque el camino por recorrer todavía es largo, el esfuerzo y los medios dedicados a este tema, conjuntamente con la inteligencia artificial y sistemas complejos de visión, por Universidades y centros de investigación de grandes multinacionales auguran próximos desarrollos rápidos y fructíferos.

La incorporación de voz a los ordenadores es ya un hecho palpable en algunos sistemas informáticos. Por contra, el reconocimiento de voz es un campo que aún se encuentra en sus albores.



Un aspecto de la pantalla del Macintosh bajo el control del Finder. En ella se observa que el menú de listados aparece «desplegado» y seleccionada la opción de listado «Por iconos».



En la ventana que contiene los programas que constituyen el sistema operativo del Apple Macintosh aparece destacado el Finder. La zona superior izquierda muestra las opciones del menú «Apple».

mas operaciones que cualquier calculadora de cuatro funciones, e intercambiar datos y resultados con los documentos.
— Reloj alarma: muestra la fecha y hora, disponiendo de una alarma sonora.
— Teclado: permite examinar los caracteres seleccionables a partir del teclado.
— Puzzle: a utilizar en caso de aburrimiento extremo.
— Bloc de notas: consta de ocho páginas y se presta para que el usuario haga en él anotaciones de todo tipo.
— Apuntador: está previsto para almacenar en él textos e imágenes utilizadas frecuentemente (cabeceras, anagramas, etc.).
— Panel de control: permite regular las características del teclado, altavoz, "ratón", escritorio, etc.

• Menú de archivo

Sus opciones actúan sobre los iconos y ventanas. Estas son:

— Abrir: abre el icono seleccionado.
— Duplicar: duplica elementos seleccionados en un mismo disco.
— Obtener datos: da información acerca de lo que representa un icono.
— Devolver: archiva los documentos, aplicaciones o carpetas seleccionadas en el disco de donde se han extraído.
— Cerrar: cierra una ventana activa, transformándola en el icono correspondiente.
— Cerrar todo: cierra todas las ventanas.
— Imprimir: imprime el documento representado por el icono.
— Expulsar: expulsa el disco seleccionado.

• Menú de Edición

Permite editar los nombres de iconos, el texto que aparece en la ventana de información y textos e imágenes en los accesorios del escritorio.
Las opciones disponibles son:

— Deshacer: anula la última operación de edición realizada.
— Cortar: sirve para eliminar un elemento seleccionado y colocarlo en el portapapeles.
— Copiar: copia un elemento seleccionado en el portapapeles.
— Fijar: coloca una copia del contenido seleccionado en el portapapeles, en el punto de inserción indicado.
— Borrar: suprime el elemento seleccionado sin colocarlo en el portapapeles.
— Seleccionar todo: selecciona todos los iconos de la ventana activa.
— Mostrar portapapeles: enseña el contenido del portapapeles.

• Menú de listado

Permite examinar los directorios o, lo que es lo mismo, el contenido de los discos, de las carpetas o de la papelera en un orden de clasificación diferente. Las opciones de ordenación disponibles son las siguientes:

— Por iconos: muestra el contenido de una ventana de directorio en forma de iconos.
— Por nombres: enumera alfabéticamente los nombres contenidos en una ventana del directorio.
— Por fechas: muestra el contenido de una ventana de directorio ordenado por fechas crecientes.
— Por tamaños: ordena de mayor a menor el contenido de una ventana del directorio.
— Por tipos: muestra el contenido de una ventana de directorio indicando si se trata de documentos, aplicaciones o carpetas.

• Menú Especial

Contiene una miscelánea de funciones:

— Ordenar: ordena todos los iconos de una ventana en filas y columnas, o también ordena y acomoda el escritorio.
— Vaciar la papelera: suprime el contenido de la papelera.
— Borrar disco: sirve para reinicializar el disco.
— Arranque: activa la aplicación cuyo icono esté seleccionado como proceso de arranque.

ALFA. UNO (1)

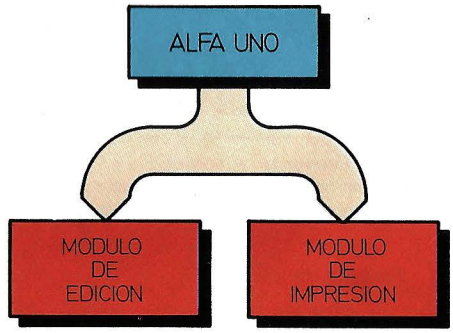
Un procesador de textos español

La compañía APLICACIONES UNO, S. A., con sede en Madrid, ha desarrollado y comercializado un paquete para el tratamiento de textos denominado ALFA UNO. Un programa desarrollado íntegramente en España y cuya notable calidad nada tiene que envidiar de los restantes programas para el tratamiento de textos ya descritos en esta obra.

LOS ELEMENTOS DEL ALFA UNO

Como ocurre con todo programa encuadrable dentro del software horizontal, en este caso en la categoría de los procesadores de texto, el ALFA UNO tiene sus propios elementos y su propia terminología. Estos factores determinan la filosofía del programa, la cual le permitirá diferenciarse, favorable o desfavorablemente, de otros programas del mismo tipo. La estructura de ALFA UNO se puede considerar modular; dentro del mismo existen dos entornos perfectamente diferenciados y, hasta cierto punto, totalmente independientes: *entorno de edición* y *entorno de impresión*. El primero de ellos sirve para editar documentos, esto es: para introducir su texto inicial o para modificar su contenido. El entorno de impresión tiene, como misión producir copias escritas y paginadas de los documentos, de forma automática; realizando labores de formateo, como numerado de páginas, ubicación de cabeceras y pies de página, etc. Otro elemento característico del ALFA UNO es la disposición que adopta la pantalla del ordenador en una sesión de trabajo; en ella se pueden distinguir cuatro zonas distintas:

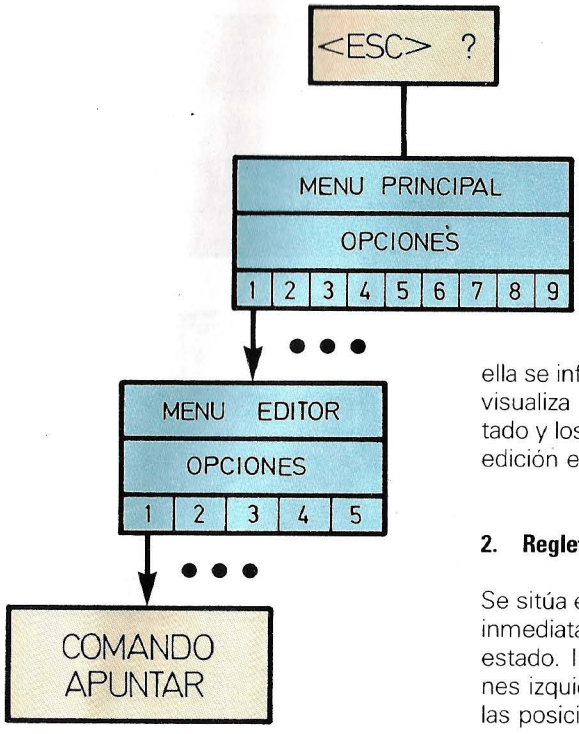
Dentro de lo que podemos denominar pantalla típica de una sesión de trabajo con ALFA UNO existen cuatro zonas distintas. La primera indica el estado del programa, la segunda está formada por una regleta de formatos, la tercera es la pantalla de edición y la cuarta, y última, sirve para contener menús y mensajes.



La estructura modular del programa ALFA UNO permite distinguir dos entornos de trabajo perfectamente diferenciados: edición de documentos e impresión de los mismos.

1. Línea de estado

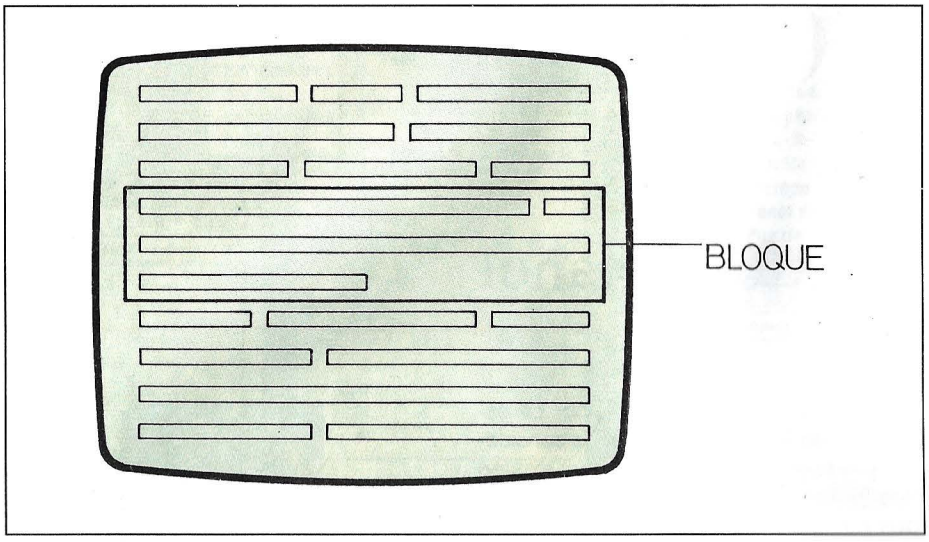
Ocupa la parte superior de la pantalla y en ella se informa sobre el editor, es decir, se visualiza el nombre del documento editado y los modos activos en el proceso de edición en curso.



La estructura de menús de ALFA UNO es de tipo arborescente. Para «entrar» en ella basta con pulsar <ESC>? A continuación, el usuario puede optar entre las distintas posibilidades que aparecerán en la última línea de la pantalla.

2. Regleta de formatos

Se sitúa en la segunda línea de la pantalla, inmediatamente por debajo de la línea de estado. Indica la situación de los márgenes izquierdo y derecho del documento y las posiciones de tabulación.



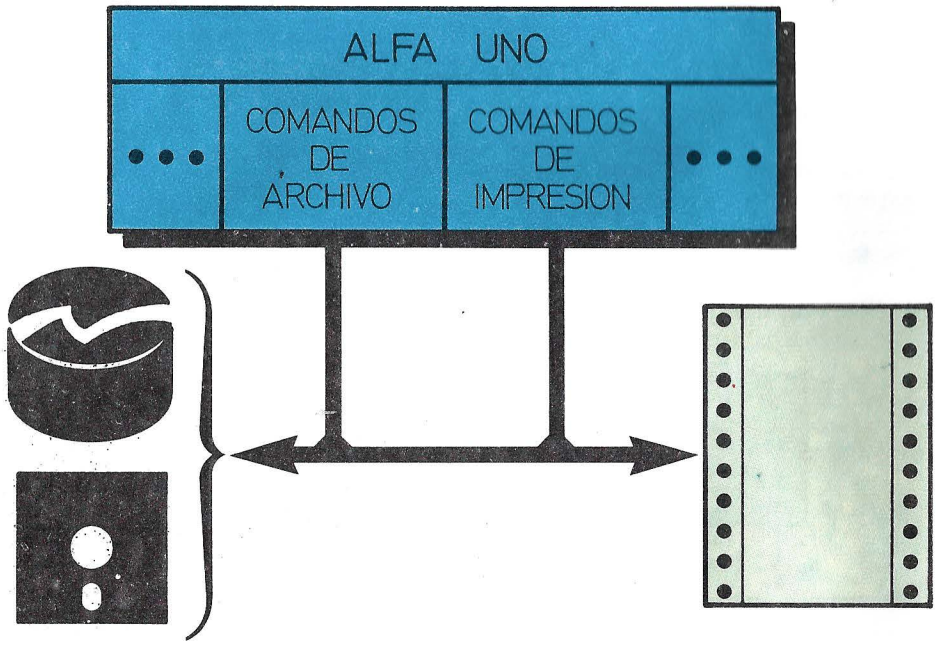
Aunque el elemento fundamental de proceso para ALFA UNO es la palabra, en algunos casos resulta muy útil poder definir bloques, de forma que se puedan realizar determinadas operaciones sobre un conjunto de palabras.

3. Pantalla de edición

Comprende las 20 líneas situadas debajo de la regleta de formatos. En realidad, la pantalla de edición es una ventana a través de la que se puede examinar el contenido del documento editado; evidentemente, esta pantalla puede desplazarse en todas las direcciones para salvar la dificultad que representa el inferior tamaño de la pantalla respecto al documento.

4. Líneas de menús y mensajes

Las dos últimas líneas de la pantalla se reservan para que el programa se comunique con el usuario; es decir, para que presente menús de ayuda y para cualquier otro tipo de mensaje, advirtiendo de errores o solicitando información adicional.
Cuando el usuario "arranca" al programa ALFA UNO, en la pantalla sólo se visualizarán las tres primeras zonas; en la línea de estado se visualizará ALFA UNO, en la regleta de formatos se marcarán los márgenes por defecto y en la pantalla de edición aparecerán 20 líneas en blanco. Si el operador desea que aparezca el menú principal, debe pulsar <ESC>? para ello pulsará la tecla [ESC] seguida de la tecla [?].



Mediante los comandos de archivo, el usuario de ALFA UNO puede «traer» o «llevar» documentos a soportes de memoria externos. Mediante los comandos de impresión se especifican las características del documento impreso.

SISTEMA DE MENUS DEL ALFA UNO

El menú principal del sistema está formado por nueve opciones distintas. De algunas de ellas "cuelga" directamente una orden, en cambio, de otras "cuelga" un nuevo menú con más opciones entre las que el usuario puede elegir.
A continuación, se describe someramente el objetivo de las distintas opciones del menú principal:

1. Editor

Ofrece ayudas para que el usuario pueda realizar operaciones sobre un documento.

2. Bloques

ALFA UNO puede trabajar con caracteres, palabras o líneas, pero en algunos casos puede ser interesante gestionar bloques de líneas; ese es precisamente el objetivo de la segunda opción del menú principal.

3. Modos

Permite al usuario optar entre distintos modos para la entrada de documentos:

- LINEA/CONTINUO

Este modo permite elegir entre el paso manual de una línea a otra (modo LINEA) o el paso automático (modo CONTINUO).

- GUIONES

El modo guiones sólo se puede activar estando previamente en modo CONTINUO. Como su propio nombre indica, al activar el modo GUIONES el programa partirá automáticamente las palabras fina-

les que no quepan en una línea mediante guiones de continuación.

- INSERCION/REEMPLAZO

Si el usuario activa el modo INSERCION cuando se introduzca un carácter, los situados sobre el cursor y los de su derecha se desplazarán una posición; en cambio, si opta por el modo REEMPLAZO, al teclear un nuevo carácter, éste simplemente reemplazará al existente en la posición del cursor.

- ADELANTE/ATRAS

Este modo se puede utilizar para que las búsquedas, sustitución, etc., se realicen desde el cursor en ADELANTE o desde el cursor hacia ATRAS.

4. Archivos

La cuarta opción del menú principal sirve para facilitar al usuario las operaciones de almacenamiento de documentos en memoria auxiliar; para ello le permite traer información, guardarla, cambiar de nombre un documento, borrarlo, etc.

5. Formatos

Esta opción del menú principal permite que el usuario determine el aspecto estético del documento final, independientemente del formato con el que se haya introducido. ALFA UNO permite ajustar los textos a uno o ambos márgenes de forma interactiva.

6. Letras

Si el usuario selecciona la opción LETRAS del menú principal estará en disposición de modificar el tipo de letra en parte del texto. De esta forma se podrán destacar determinadas palabras o frases poniéndolas en negrita, cursiva, subrayadas, etc.

7. Imprimir

Las órdenes de impresión de ALFA UNO permiten imprimir copias sobre papel de cualquier documento elaborado con el procesador de texto. En el entorno de edición no se tienen en cuenta las características finales del documento a imprimir, pero en la séptima opción del menú prin-

Puestos de trabajo informático (y 2)

Como continuación del cuadro publicado en el capítulo anterior, se definen seguidamente otros puestos de trabajo relacionados con la informática. Como consecuencia de la utilización masiva de programas en cualquier disciplina, el número de posibles trabajos relacionados con el ordenador ha crecido espectacularmente. Aquí nos limitaremos a describir los más caracterizados.

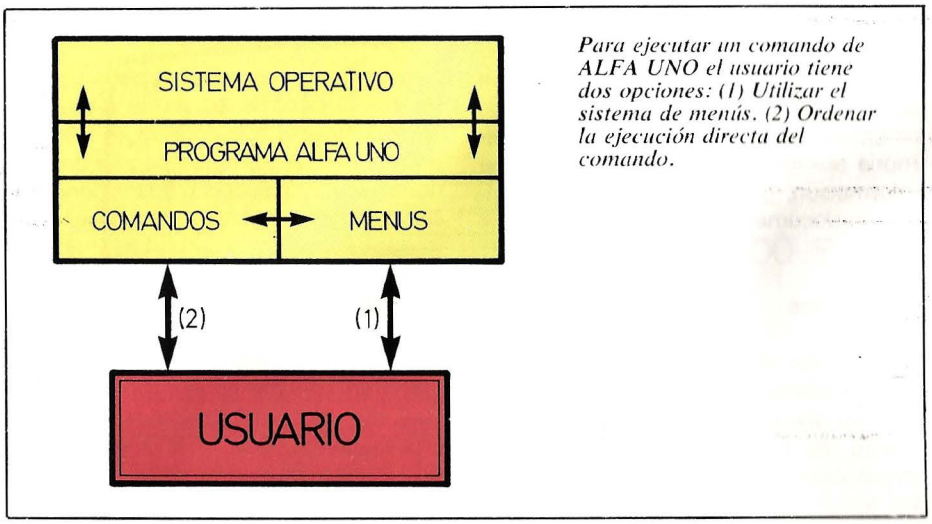
EXPLOTACION DE PROGRAMAS

- USUARIO. La participación del usuario resulta imprescindible tanto en el desarrollo de un programa como en su posterior explotación. En este último caso se limitará a utilizar el programa desarrollado para resolver los problemas que él mismo planteó.

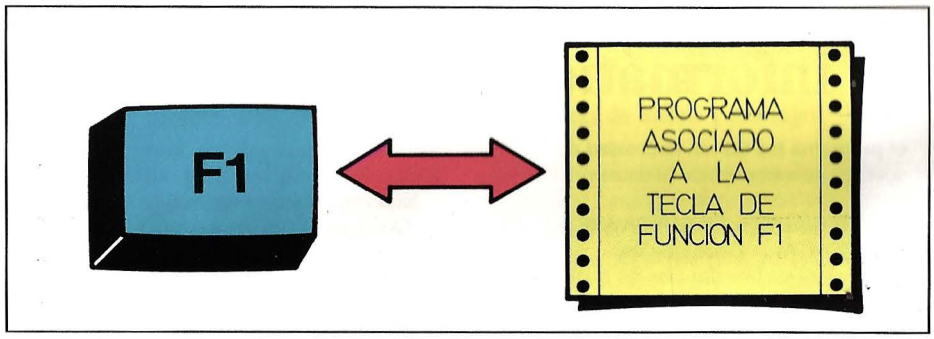
- OPERADOR. Cuando un programa utiliza un equipo informático relativamente complejo suele aparecer la figura del operador. Su trabajo consiste en atender al ordenador, vigilando la existencia de papel continuo en la impresora, la colocación de cintas en sus unidades correspondientes y contestando a todos los requerimientos del sistema operativo y del propio programa.

- GRABADOR. Si la explotación de un programa obliga a la introducción de muchos datos de entrada, puede resultar útil la creación de un puesto de trabajo de grabador. Su misión consistirá en grabar todos los datos en un soporte auxiliar de almacenamiento para que, a continuación, el programa los lea sin necesidad de solicitar su introducción al usuario.

- ESPECIALISTA EN SOFTWARE HORIZONTAL. Dado que las posibilidades ofrecidas por los paquetes horizontales cada vez son mayores, resulta frecuente encontrar puestos de trabajo tales como: especialista en bases de datos, especialista en proceso de textos, especialista en programas CAD, etc.



Para ejecutar un comando de ALFA UNO el usuario tiene dos opciones: (1) Utilizar el sistema de menús. (2) Ordenar la ejecución directa del comando.



ALFA UNO aporta un sistema para la programación de teclas de función. De esta forma, sin más que pulsar una tecla se desencadenará la ejecución del programa asociado.

cipal el usuario puede especificar parámetros en los que indicará: el número de líneas que se escribirán en cada página, el espacio libre entre cada par de líneas, los márgenes superior e inferior de cada página, la numeración de las páginas, etc.

8. Salir

Esta opción del menú principal del sistema indica el final de la ejecución del programa; al ejecutarla, el sistema operativo del ordenador vuelve a tomar el control. Antes de producirse la salida definida, ALFA UNO dará opción al usuario para que almacene el documento utilizado durante la sesión de trabajo.

9. Cancelar

La novena y última opción del menú principal sirve, como su propio nombre indica, para cancelar el sistema de menús.

EJECUCION DE ORDENES

En el párrafo anterior se han descrito las principales opciones aportadas por el sistema de menús. En el fondo, este sistema puede compararse con un árbol de opciones cuyas hojas son los comandos del ALFA UNO. Existen dos posibilidades para la ejecución de una orden:

a. Ejecución directa.
Desde el teclado se pueden pulsar las teclas oportunas para que el programa ejecute directamente un comando.

b. Ejecución por menús.
Entrando en el sistema de menús y seleccionando opciones hasta llegar al comando deseado.

Evidentemente, la primera alternativa resulta mucho más rápida que la segunda; a cambio, esta última es mucho más cómoda y segura. Generalmente el usuario "novato" optará por trabajar exclusivamente con el sistema de menús, mientras que el "veterano" en algunos casos desencadenará la ejecución directa y en otros recurrirá al sistema de menús.

TECLAS DE FUNCION

La mayoría de los ordenadores disponen de teclas especiales que pueden ser programadas por el usuario; de esta forma, sin más que pulsar una tecla se desencadenará automáticamente la ejecución del "programa" asociado a dicha tecla.
ALFA UNO ofrece un amplio abanico de posibilidades para que el usuario programe las teclas de función con gran comodidad:

1. Inicialización de las teclas

Es frecuente que cada vez que el usuario se conecte al ordenador deba realizar la programación de las teclas de función. ALFA UNO dispone de un brillante sistema de inicialización basado en un fichero donde están almacenados los programas estándar para cada tecla; así, más que ejecutar el comando INIFUN, las teclas quedarán programadas automáticamente.

2. Desprogramación de las teclas

En algunos casos es necesario anular las funciones previamente asignadas, de forma que al pulsarlas no se produzca ningún efecto. ALFA UNO dispone del comando DESFUN para realizar esta desprogramación.

3. Programación de las teclas

Aparte de la inicialización automática de las teclas de función, el usuario puede programar a su gusto cualquiera de ellas. Para que el operador pueda programar las teclas, ALFA UNO dispone de un comando denominado PROFUN.

Archivos aleatorios (2)

Ejercicio práctico con un archivo de acceso directo

En este segundo capítulo dedicado a los archivos aleatorios o de acceso directo, el objetivo es profundizar en el estudio de su estructura y forma de empleo. Para el eficaz desarrollo de esta tarea se confeccionará un programa que reflejará los aspectos prácticos.

El ejemplo a desarrollar se concreta en la versión programada del archivo de una biblioteca, en el cual se introducirá información relativa a los libros existentes en la biblioteca. Esta información constituye en su conjunto lo que denominamos archivo.

La información a almacenar puede clasificarse en diversos bloques. Cada uno de dichos bloques constituye una unidad de información denominada registro. Como ya se indicó en el primer capítulo dedicado a los archivos de acceso directo, la estructura de los registros ha de ser fija. Cada registro, a su vez, consta de una serie de campos que pueden considerarse como unidades elementales de información.

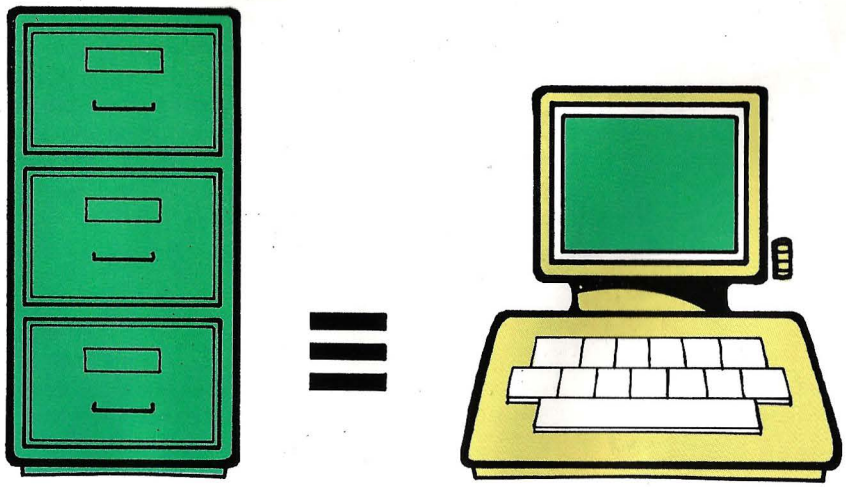
DEFINICION DE CARACTERISTICAS

La primera operación a realizar, antes de empezar el trabajo con un archivo de acceso directo, consiste en definir la estructura de los registros. Esta estructura queda determinada por tres parámetros fundamentales: número de campos que lo componen, longitud de cada uno de los campos y naturaleza de los campos. En nuestro caso, se desea crear una estructura cuyos registros contengan los siguientes campos:

—Título.
—Autor.
—Editorial.
—Número de páginas.
—Año.
—Código.
—Signatura.

A continuación es imprescindible estudiar cada uno de los campos para decidir cuál será la estructura y naturaleza de cada uno de ellos.

En primer lugar cabe suponer que el campo "Título" puede poseer una longitud muy variable; el número de caracteres necesarios para almacenar el título de un libro puede oscilar entre unos pocos de caracteres y 100 ó más. En todo caso, no es corriente que supere los 30 caracteres. Hay que tener en cuenta que si se otorga una longitud muy grande a este campo, surgirá el problema de que cada registro ocupará un excesivo espacio en el disco. Por el contrario, si se elige una longitud pequeña serán pocos los títulos que podrán introducirse sin proceder a su simplificación.



Las bases de datos informatizadas presentan notables ventajas frente a los archivos tradicionales. Las principales contrapartidas se encuentran en ahorro de espacio, tiempo de manipulación y versatilidad.

En definitiva, se ha de optar por una solución de compromiso intermedia entre ambas opciones. Por ejemplo, en nuestro caso podría ser razonable asignar a este campo una longitud de 40 caracteres.

Donde no cabe duda alguna es en lo que se refiere a la naturaleza del archivo: se trata de un campo alfanumérico ya que, por lo general, se compondrá de caracteres alfabéticos y, ocasionalmente, de algún número.

El siguiente campo es el de Autor. En él se desean almacenar los nombres y apellidos de los autores de los libros. Aquí surgen varias posibilidades a contemplar; la primera es que algunas veces un libro tiene más de un autor, en cuyo caso resultará difícil incluir los dos nombres en el espacio reservado para un solo nombre. En nuestro ejemplo se reservarán 30 caracteres para almacenar esta información; longitud que parece suficiente. Por supuesto, su naturaleza va a ser también alfanumérica.

El tercer campo es el destinado a la Editorial. Las características y salvedades a contemplar en este campo son muy semejantes a las ya señaladas para los campos anteriores. Ahora puede ser suficiente con una longitud de 20 caracteres para almacenar el nombre de la editorial. De nuevo, su naturaleza será alfanumérica. El número de páginas del libro es el primer campo numérico con el que tropezamos, de ahí que los problemas que plantea sean algo diferentes. Hay que decidir el tipo de variable a utilizar.

Puede que resulte un poco extraño el hecho de tener que seleccionar el tipo de variable numérica a emplear, ya que estos campos se almacenan en forma de cadenas de caracteres. Sin embargo, la explicación es sencilla. En el primer capítulo dedicado a este tema se precisó que los números se almacenan a modo de cadenas de caracteres. Pasando por alto el empleo de las funciones STR$ y VAL que resulta poco eficiente, se dispone de las funciones del tipo MKI$, MKS$ y MKD$ y de las CVI, CVS y CVD. Las primeras permiten convertir un número en una cadena, recurriendo para ello a un formato especial que permite ahorrar memoria; el segundo bloque de funciones permite recuperar los números originales. Las funciones relacionadas se diferencian entre sí en el tipo de variables numéricas con las que trabajan. Evidentemente, es necesario conocer el tipo de variables a emplear y, por supuesto, hay que ser consecuentes con la elección y realizar las transformaciones de números a cadenas y viceversa con las funciones adecuadas.

Para almacenar el número de páginas es conveniente recurrir a variables de tipo entero; en primer lugar ello permite ahorrar memoria y, desde luego, un libro no suele tener medias hojas y tampoco suele ser tan largo como para superar las 65.000 páginas. Así pues, dicho campo se almacenará con la precisión de un número entero. El número de caracteres que emplea la función MKI$ para almacenar una variable de tipo entero es de dos bytes, por lo que la longitud del campo es de dos caracteres.

Tampoco cabe la menor duda en la elección del tipo de campo para almacenar el Año, ya que se trata de un dato numérico. Este puede almacenarse como tal en una variable entera, o bien a modo de cadena de caracteres. En el primer caso serán necesarios campos con una longitud de dos caracteres, y en el segundo de cuatro caracteres. Para el ejemplo que nos ocupa seleccionaremos el tipo numérico y, dentro de éste, la precisión de entero.

En cuanto al Código y a la Signatura, como pueden estar constituidos por caracteres alfabéticos y numéricos, será preciso almacenarlos directamente en una cadena; al efecto se reservarán 10 caracteres para cada uno de ambos campos. En resumen, los diferentes campos de cada registro, así como su naturaleza y longitudes respectivas quedan reflejados en la tabla adjunta.

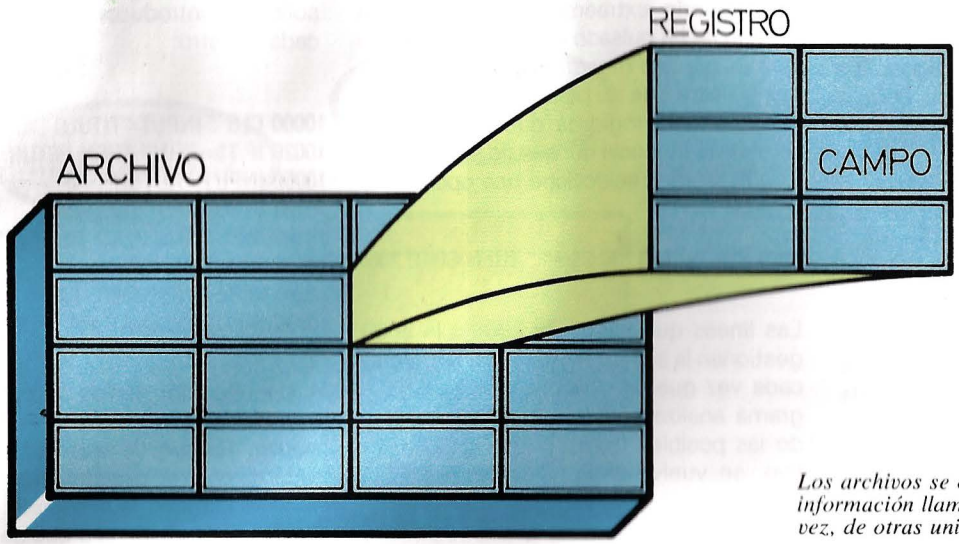
Una vez definida la estructura de los campos, es necesario elegir el número de canal a emplear. Ciertamente, este no es un factor trascendente sino que tan sólo debe contemplar que su número no coincida con el de un canal ya abierto.

A la hora de abrir el canal es preciso conocer el nombre del archivo que se desea abrir y la longitud de cada uno de los campos. En nuestro caso, la orden para efectuar esta operación es la siguiente:

```
OPEN "R",#1,"DATOS",114
```

Este formato puede parecer en principio un tanto rígido; aunque no lo es tanto. Así, por ejemplo, no es necesario conocer previamente el nombre del archivo que se desea abrir; este parámetro, que en nuestro caso aparece como una constante de cadena, puede ser sustituido por una variable del mismo tipo. Lo mismo cabe afir-

Un ejercicio práctico como el realizado en el presente capítulo puede concluir con una útil herramienta para la gestión personal. Tal es el caso del programa confeccionado, el cual se puede utilizar para controlar el archivo bibliográfico.



Los archivos se componen de unidades de información llamadas registros, y éstos, a su vez, de otras unidades llamadas campos.

mar de los restantes parámetros de esta instrucción.
El segundo paso a realizar es la definición de la estructura de los registros, para lo cual se utiliza la instrucción FIELD. Por ejemplo:

```
FIELD#1,40 AS títulos, 30 AS autor$, 20
    AS editor$, 2 AS numpag$, 2 AS ano$,
    10 AS codi$, 10 AS signat$
```

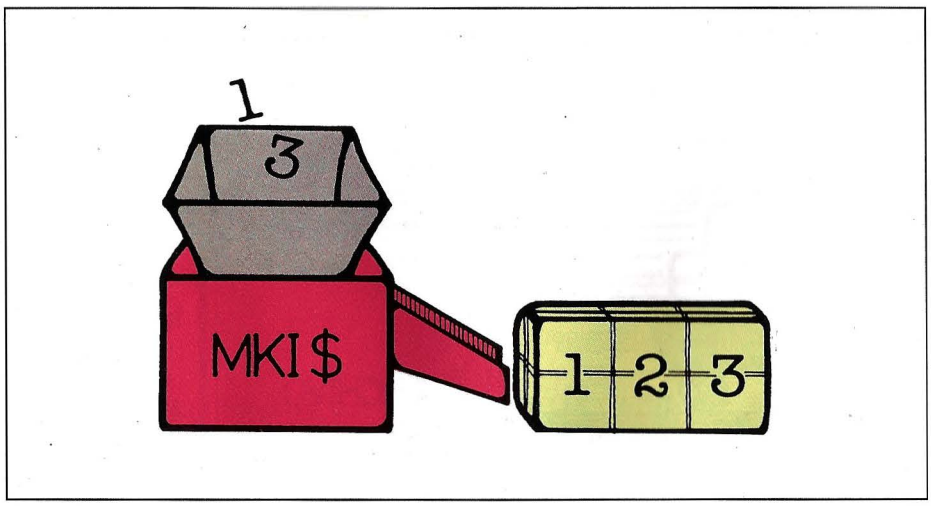
Esta especificación es necesaria, ya que para trabajar con uno de estos archivos el ordenador necesita crear una zona de memoria asociada a cada uno de los canales. Esta zona de memoria temporal (buffer) actúa como intermediario para la transferencia de información entre la memoria del ordenador y el archivo externo. En el buffer están reservadas zonas para cada uno de los campos que se van a emplear. Cada una de estas zonas puede manejarse de forma independiente, lo cual significa que pueden introducirse los datos que se deseen y en el orden que parezca más conveniente.

CREANDO UN PROGRAMA PARA MANEJAR ARCHIVOS DE ACCESO DIRECTO

Una vez estudiado el método para la apertura del archivo y decidida su estructura, se está en condiciones de crear un programa para su tratamiento.
Antes de empezar la codificación del programa, es necesario decidir qué operaciones deseamos que éste sea capaz de realizar.
En nuestro caso, crearemos un programa que realice las operaciones básicas en un archivo de acceso directo; estas operaciones son las de creación, actualización y consulta.
La estructura de nuestro programa consistirá en una zona central, encargada de la apertura del archivo y de la selección de la operación a realizar, y de un surtido de subrutinas especializadas.
Cada operación a realizar será ejecutada por una subrutina distinta creada al efecto; tras ejecutarse, la secuencia del programa volverá a la zona de selección de las siguientes operaciones a realizar.
A continuación se confeccionará el cuerpo principal del programa que incluye el menú para la selección de la opción deseada.
Las dos primeras líneas del programa (1000 y 1010) abren el archivo y crean el buffer de trabajo:

```
1000 OPEN"R",#1,"DATOS", 114
1010 FIELD#1, 40 AS titulo$, 30 AS autor$, 20
     AS editor$, 2 AS numpag$, 2 AS ano$, 10
     AS codi$, 10 AS signat$
```

Acto seguido llega el momento de borrar la pantalla e inicializar las variables NUM% y NMAX% cuya misión es actuar como contadores. En el caso de que el ordena-



La función MKI$ se encarga de empaquetar un número en una cadena, permitiendo con ello ahorrar memoria frente al uso de la función STR$.

dor sea del tipo IBM-PC o MSX, también será preciso eliminar de la pantalla los recordatorios de las funciones asignadas a las teclas de función. Todo ello corre a cargo de las siguientes líneas de programa:

```
1020 NUM%=0 : NMAX=0
1900 CLS : KEY OFF
```

La zona comprendida entre las líneas 2000 a la 2040 presentan por pantalla el menú de selección. La visualización de las opciones se realiza mediante una serie de instrucciones PRINT, anteponiendo a cada una de las operaciones que se pueden realizar un número que permitirá seleccionarlas:

```
1950 PRINT
2000 PRINT TAB(28);"MENU DE SELECCION"
     : PRINT : PRINT
2010 PRINT TAB(25);"1— CREACION DEL
     ARCHIVO" : PRINT
2020 PRINT TAB(25);"2— LECTURA DE UN
     REGISTRO" : PRINT
2030 PRINT TAB(25);"3— MODIFICACION DE
     UN REGISTRO" : PRINT
2040 PRINT TAB(25);"4— LISTADO DEL
     ARCHIVO" : PRINT
```

Con este menú a la vista, el usuario debe seleccionar la operación que desea realizar. Para que tal selección sea efectiva sin más que pulsar una tecla, es necesario recurrir a la función INKEY$.
Nuestro programa recoge el carácter pulsado en la variable B$ y, acto seguido, se comprueba si dicha variable tiene algún contenido. De no poseer contenido alguno puede extraerse la conclusión de que no se ha pulsado ninguna tecla, por lo que es preciso muestrear de nuevo el teclado hasta que se pulse una tecla. En tal situación el programa queda encerrado dentro de un bucle del que no saldrá hasta que el usuario seleccione una opción.

```
2050 B$=INKEY$ : IF B$=" " THEN GOTO 2050
```

Las líneas que van de la 2060 a la 2080 gestionan la selección realizada. Al efecto, cada vez que se pulsa una tecla, el programa analiza si ésta corresponde a una de las posibles opciones. En caso negativo, se vuelve a examinar de nuevo el teclado.
Si la tecla pulsada corresponde a una de las posibles opciones, la instrucción ON GOSUB de la línea 2080 ordenará el salto a la rutina adecuada.
La línea 2090 provoca el retorno al menú una vez que concluye la ejecución de la subrutina a la que se ha bifurcado.

```
2060 caracter=ASC(B$)—48
2070 IF caracter < 1 OR caracter > 4 THEN
     GOTO 2050
2080 ON caracter GOSUB 10000, 20000, 30000,
     40000
2090 GOTO 1900
```

Aquí concluye el cuerpo principal del programa. A partir de este punto se encuentran las subrutinas encargadas de procesar las distintas opciones.
Si se elige la opción 1 (creación del archivo), se ejecutará la siguiente rutina que solicita la introducción de los datos para cada registro:

```
10000 CLS : INPUT "TITULO DEL LIBRO"; T$
10020 IF T$="FIN" THEN RETURN
10050 INPUT "AUTOR"; A$
10100 INPUT "EDITOR"; E$
10150 INPUT "NUMERO DE PAGINAS"; N$
10200 INPUT "ANO"; Y$
10250 INPUT "CODIGO"; C$
10300 INPUT "SIGNATURA"; S$
```

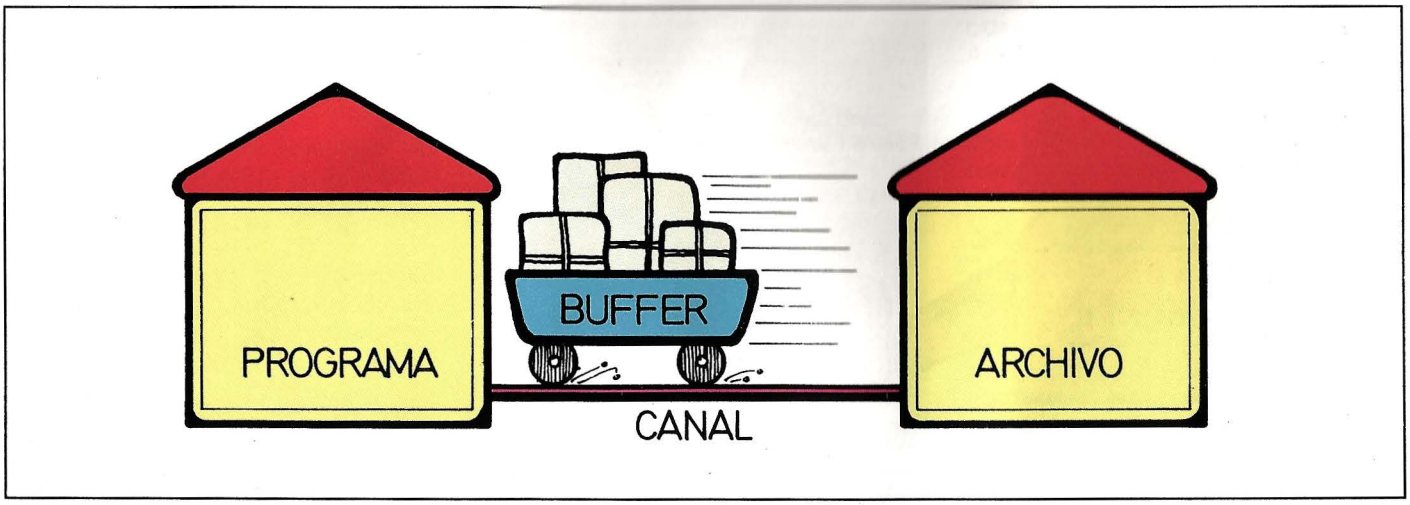
A continuación estos datos deben ser transferidos al buffer, realizando, al mismo tiempo, la operación de ajustar cada dato a las características de su respectivo campo; ello supone adecuar el dato al tipo de variable que se utilice:

```
10310 LSET titulo$=T$
10320 LSET autor$=A$
10330 LSET editor$=E$
10340 RSET numpag$=MKI$(VAL(N$))
10350 RSET ano$=MKI$(VAL(Y$))
10360 LSET codi$=C$
10370 LSET signa$=S$
```

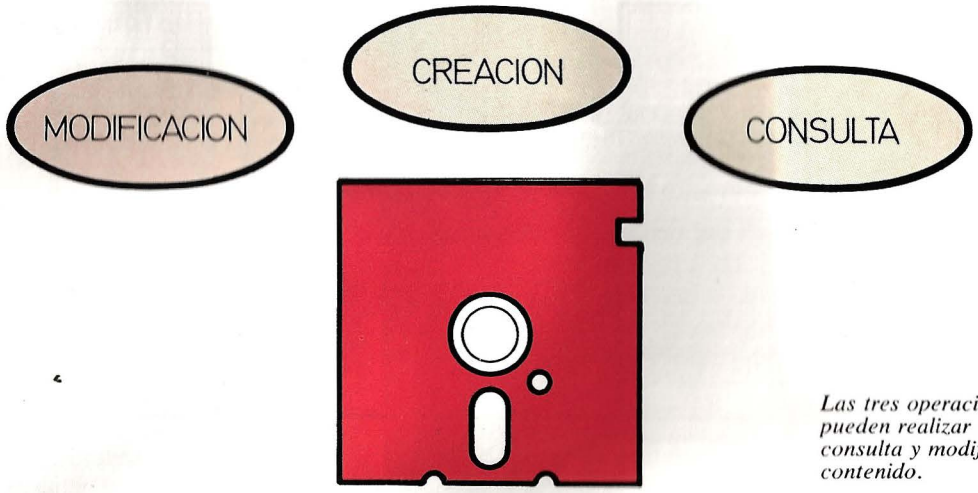
Es importante ir actualizando el contador que sirve para conocer qué posición relativa ocupa el registro dentro del archivo que se está creando. Dicha posición viene dada por el valor de la variable NUM%:

```
10380 NUM%=NUM%+1
```

También es preciso saber cuántos registros han sido creados para, por ejemplo,



El canal actúa a modo de ente intermedio a través del que los datos pueden pasar del programa o proceso en curso hacia el archivo.



Las tres operaciones fundamentales que se pueden realizar con un archivo son: creación, consulta y modificación o actualización del contenido.

cuando sea necesario listarlos, realizar la operación sin que se produzca un error:

```
10390 NMAX%=NMAX%+1
```

Para transferir el contenido del registro a la posición deseada del archivo se emplea una instrucción PUT. En ella se especifica el número de canal asociado al archivo en cuestión y la posición del registro dentro del archivo:

```
10400 PUT#1, NUM%
```

Tras ello hay que pasar a la fase de introducción de otro registro, de lo cual se ocupa la instrucción 10410.
A la vista de las instrucciones descritas puede parecer que esta rutina no tiene salida, y que constituye un bucle cerrado por completo. Sin embargo se ha previsto una salida.
Esta se encuentra en la línea 10020; en ella se examina el campo de título sobre la variable T$, y en caso de que se haya introducido la palabra Fin, la secuencia de ejecución volverá al menú principal del programa.

```
10410 GOTO 10000
10420 RETURN
```

En la línea 10420 concluye con el comando RETURN la subrutina de creación del archivo.
La próxima subrutina a crear es la de consulta de registros del archivo, la cual está asociada a la opción Z del menú principal.

Pantalla de entrada al programa con el menú de opciones. Su presentación corre a cargo de la rutina o cuerpo principal del programa.

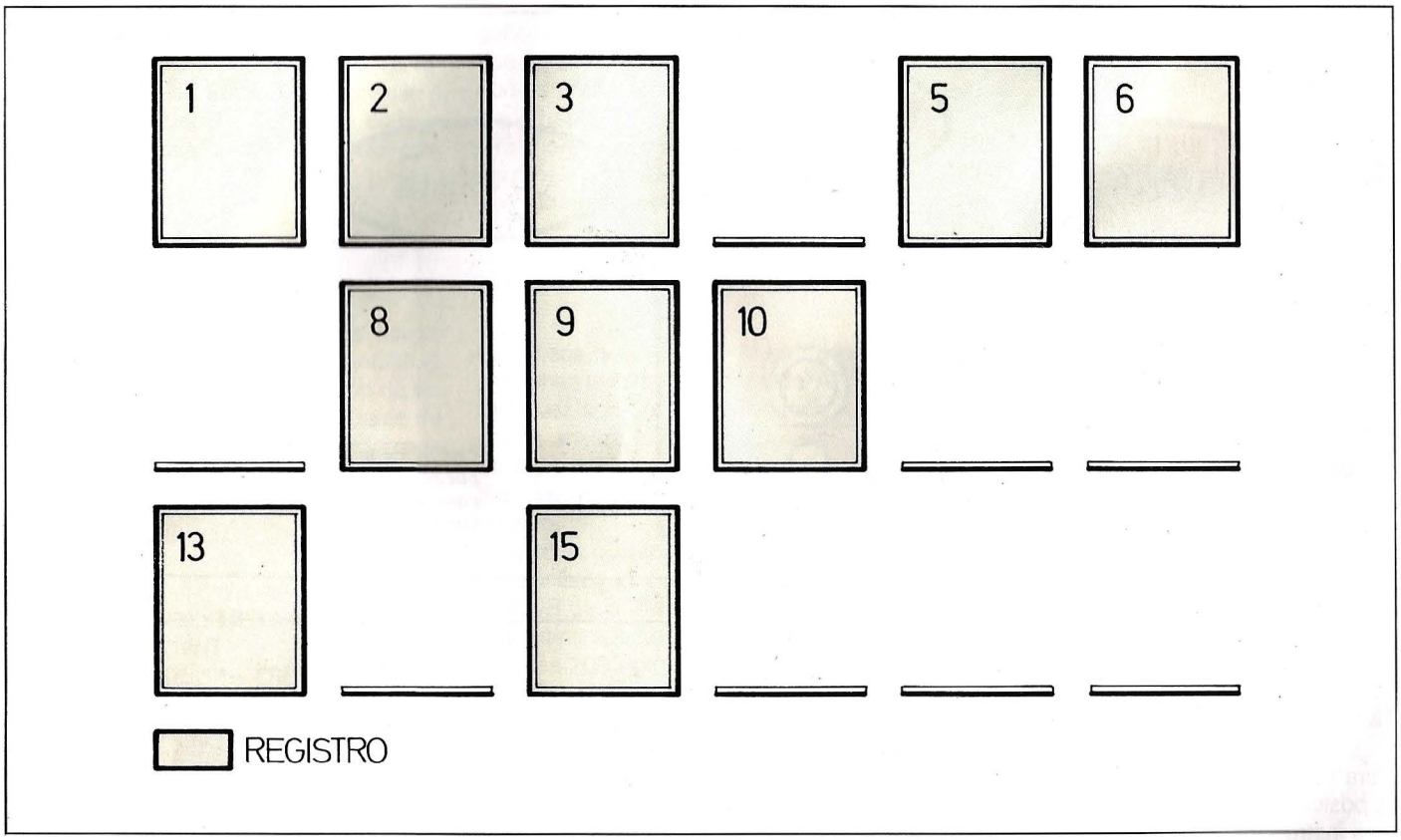
La primera opción que debe realizar esta subrutina es la de consultar al usuario cuál es el registro que se desea leer:

```
20000 CLS : INPUT "QUE REGISTRO DESEAS
      LEER"; NUM%
```

A continuación, y una vez que se conoce el registro que se desea consultar, se ejecutará una instrucción GET, la cual accede al disco y lee el registro cuyo número se ha solicitado:

```
20010 GET#1, NUM%
```

Y por fin, una vez que el registro a pasado al buffer, la siguiente operación es mostrar por pantalla el contenido del mismo. Esta tarea se puede realizar sencillamente



En los archivos de acceso directo, el ordenador se encarga de ir almacenando los distintos registros en el orden definido por su clave. Desde luego, queda abierta la posibilidad de incluir nuevos registros cuyo número de clave esté comprendido entre otros dos.

Pantalla asociada a la opción de modificación de un registro. Los contenidos que aparecen tras los diversos campos proceden de un ejemplo de ficha bibliográfica.

mediante una secuencia de instrucciones PRINT:

```
20020 PRINT "1—TITULO"; titulo$
20030 PRINT "2—AUTOR"; autor$
20040 PRINT "3—EDITOR"; editor$
20050 PRINT "4—NUMERO DE PAGINAS";
CVI
      (numpag$)
20060 PRINT "5—ANO"; CVI(ano$)
20070 PRINT "6—CODIGO"; codi$
20080 PRINT "7—SIGNATURA"; signa$
20090 LET B$=INKEY$ : IF B$=" " THEN
      GOTO 20090
```

Tras presentar en la pantalla el contenido del registro, hay que regresar al programa principal por medio del correspondiente RETURN:

```
20100 RETURN
```

La subrutina asociada a la opción 3 (modificación de un registro), muestra en pantalla el contenido del registro que se solicite a través de una llamada a la subrutina de lectura de un registro.

ESTRUCTURA DE CADA REGISTRO

Nombre del campo	Número de caracteres	Tipo de campo
Título	40	Alfanumérico
Autor	30	Alfanumérico
Editorial	20	Alfanumérico
Número pág.	2	Número entero
Año	2	Número entero
Código	10	Alfanumérico
Signatura	10	Alfanumérico

Número total de caracteres en cada registro: 114

El salto a la subrutina de lectura se produce al ejecutar la instrucción GOSUB 20010 de la línea 30010.
Acto seguido se ofrece al usuario la posibilidad de renunciar a la modificación de campos, sin más que responder con una acción sobre la tecla 8. En todo caso, la situación habitual en este punto será la de responder a la pregunta "QUE CAMPO DESEAS MODIFICAR" con el número del campo oportuno. Este número, almacenado en la variable C, será utilizado para la instrucción ON GOSUB de la línea 30030 para bifurcar la secuencia de ejecución hacia la zona adecuada.
La zona de programa que sigue coincide con la asociada al tratamiento de la opción descrita:

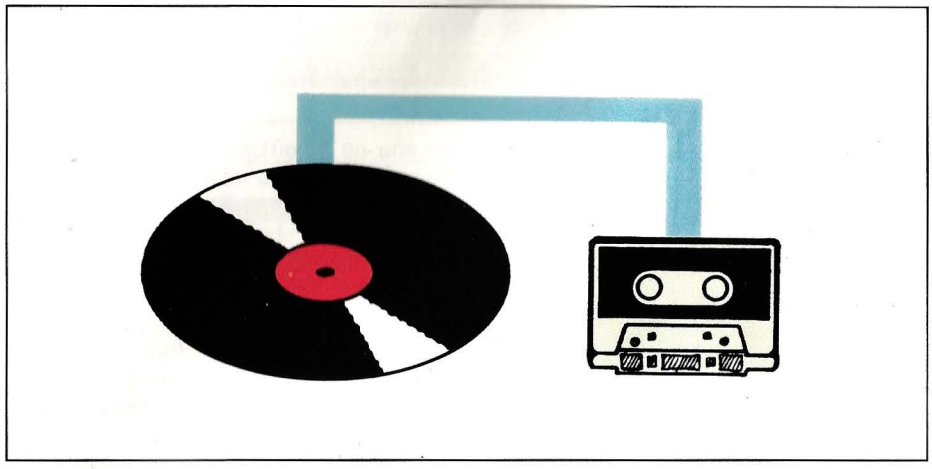
```
30000 CLS : INPUT "QUE REGISTRO DESEAS
      MODIFICAR"; NUM%
30010 GOSUB 20010
30015 PRINT "SI NO DESEAS MODIFICAR
      NINGUNO PULSA 8"
30020 INPUT "QUE CAMPO DESEAS
      MODIFICAR"; C
30030 ON C GOSUB 30100, 30200, 30300,
      30400, 30500, 30600, 30700
30040 RETURN
30100 INPUT "TITULO"; T$
30110 LSET titulo$=T$
30120 PUT#1, NUM% : RETURN
30200 INPUT "AUTOR"; A$
30210 LSET autor$=A$
30220 PUT#1, NUM% : RETURN
30300 INPUT "EDITOR"; E$
30310 LSET editor$=E$
30320 RETURN
30400 INPUT "NUMERO DE PAGINAS"; N$
30410 RSET numpag$=N$
30420 RETURN
30500 INPUT "ANO"; Y$
30510 RSET ano$=Y$
```

La estructura de un archivo de acceso directo guarda un cierto paralelismo con la organización de los archivos tradicionales. La información se agrupa en bloques denominados registros y éstos, a su vez, engloban a un conjunto de campos.

```
30520 RETURN
30600 INPUT "CODIGO"; C$
30610 LSET codi$=C$
30620 RETURN
30700 INPT "SIGNATURA"; S$
30710 LSET signa$=S$
30720 RETURN
```

La cuarta y última opción del menú da paso al listado de un archivo. Su puesta en práctica se fundamenta en sucesivas llamadas a la subrutina de lectura de un registro. Ello tiene lugar dentro del siguiente bucle:

```
40000 CLS : FOR NUM%=1 TO NMAX%
40010 GOSUB 20010
40020 NEXT I
40030 RETURN
```



Las diferencias de acceso que presenta un archivo de acceso directo frente a un archivo secuencial son perceptibles en las diferencias que se manifiestan entre un disco de audio y una casete.

```
1000 OPEN"R",#1, "DATOS", 114
1010 FIELD#1, 40 AS titulo$, 30 AS autor$, 20 AS editor$,
2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
1020 NUM%=0 : NMAX=0
1900 CLS : KEY OFF
1950 PRINT
2000 PRINT TAB(28); "MENU DE SELECCION" : PRINT : PRINT
2010 PRINT TAB(25); "1- CREACION DEL ARCHIVO" : PRINT
2020 PRINT TAB(25); "2- LECTURA DE UN REGISTRO" : PRINT
2030 PRINT TAB(25); "3- MODIFICACION DE UN REGISTRO" : PRINT
2040 PRINT TAB(25); "4- LISTADO DEL ARCHIVO" : PRINT
2050 B$=INKEY$ : IF B$=" " THEN GOTO 2050
2060 caracter=ASC(B$)-48
2070 IF caracter < 1 OR caracter > 4 THEN GOTO 2050
2080 ON caracter GOSUB 10000, 20000, 30000, 40000
2090 GOTO 1900
10000 CLS : INPUT "TITULO DEL LIBRO"; T$
10020 IF T$="FIN" THEN RETURN
10050 INPUT "AUTOR"; A$
10100 INPUT "EDITOR"; E$
10150 INPUT "NUMERO DE PAGINAS"; N$
10200 INPUT "ANO"; Y$
10250 INPUT "CODIGO"; C$
10300 INPUT "SIGNATURA"; S$
10310 LSET titulo$=T$
10320 LSET autor$=A$
10330 LSET editor$=E$
10340 RSET numpag$=MKI$(VAL(N$))
10350 RSET ano$=MKI$(VAL(Y$))
10360 LSET codi$=C$
10370 LSET signa$=S$
10380 NUM%=NUM%+1
10390 NMAX%=NMAX%+1
10400 PUT#1, NUM%
10410 GOTO 10000
10420 RETURN
20000 CLS : INPUT "QUE REGISTRO DESEAS LEER"; NUM%
20010 GET#1, NUM%
20020 PRINT "1-TITULO"; titulo$
20030 PRINT "2-AUTOR"; autor$
20040 PRINT "3-EDITOR"; editor$
20050 PRINT "4-NUMERO DE PAGINAS"; CVI(numpag$)
20060 PRINT "5-ANO"; CVI(ano$)
20070 PRINT "6-CODIGO"; codi$
20080 PRINT "7-SIGNATURA"; signa$
20090 LET B$=INKEY$ : IF B$=" " THEN GOTO 20090
20100 RETURN
30000 CLS : INPUT "QUE REGISTRO DESEAS MODIFICAR"; NUM%
30010 GOSUB 20010
30015 PRINT "SI NO DESEAS MODIFICAR NINGUNO PULSA 8"
30020 INPUT "QUE CAMPO DESEAS MODIFICAR"; C
30030 ON C GOSUB 30100, 30200, 30300, 30400, 30500, 30600, 30700
30040 RETURN
30100 INPUT "TITULO"; T$
30110 LSET titulo$=T$
30120 PUT#1, NUM% : RETURN
30200 INPUT "AUTOR"; A$
30210 LSET autor$=A$
30220 PUT#1, NUM% : RETURN
30300 INPUT "EDITOR"; E$
30310 LSET editor$=E$
30320 RETURN
30400 INPUT "NUMERO DE PAGINAS"; N$
30410 RSET numpag$=N$
30420 RETURN
30500 INPUT "ANO"; Y$
30510 RSET ano$=Y$
30520 RETURN
30600 INPUT "CODIGO"; C$
30610 LSET codi$=C$
30620 RETURN
30700 INPUT "SIGNATURA"; S$
30710 LSET signa$=S$
30720 RETURN
40000 CLS: FOR NUM%=1 TO NMAX%
40010 GOSUB 20010
40020 NEXT I
40030 RETURN
```

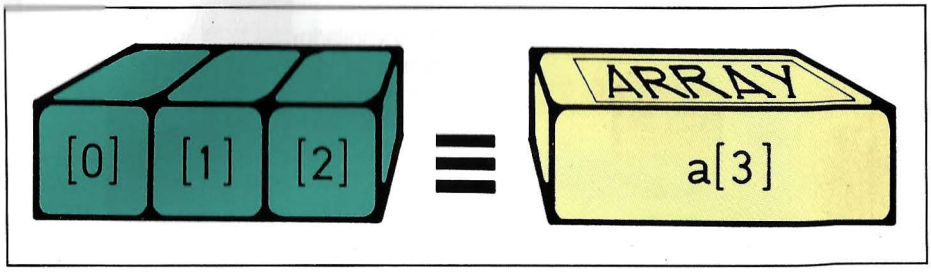
Listado del PROGRAMA.

El listado adjunto muestra el programa en su totalidad. Tal como cabe deducir de las descripciones realizadas, la creación y tratamiento de archivos aleatorios o de acceso directo no presenta mayores complicaciones de las derivadas del propio conocimiento de los comandos y funciones BASIC especializadas en su manipulación. Al respecto, cabe precisar que en el ejemplo propuesto se ha utilizado un repertorio específico que estará sujeto a variaciones según el dialecto BASIC que equipe al ordenador utilizado. Por ello, antes de proceder a la instrucción de este programa, habrá que revisar la equivalencia de comandos y funciones de su dialecto BASIC con respecto al repertorio empleado en nuestro ejemplo práctico.

El lenguaje C (5)

Estructuras de datos: los arrays

El objetivo del presente capítulo lo constituye el estudio de los arrays, estructuras de datos que resultarán muy familiares a los conocedores del lenguaje PASCAL. Como ya se ha comentado en algún capítulo anterior, con el "C" se pretendía obtener un lenguaje compacto y fácilmente implementable en casi cualquier ordenador. Por esta razón, las estructuras de datos de este lenguaje destacan por su sencillez, aunque resultan suficientes para la mayoría de las aplicaciones. Antes de comenzar con su estudio es conveniente ampliar conocimientos relativos a punteros.



En "C", un array declarado como "int a[3];" no contiene al elemento a[3]. Dicho array contendrá los elementos a[0], a[1] y a[2]. La razón se comprenderá plenamente al estudiar la relación entre punteros y arrays. De momento, basta con considerar que los arrays comienzan con el elemento cero (a[0]) en vez de con el uno (a[1]).

PUNTEROS Y ARITMETICA

Los punteros son piezas clave en el desarrollo de casi cualquier programa. Por esta razón, las operaciones con y entre punteros no se reducen a lo ya descrito en capítulos precedentes, sino que además está permitido:
— Incrementar o decrementar un puntero.
— Sumar o restar un entero a un puntero.
— Comparar y substraer dos punteros, en el supuesto de que ambos señalen al mismo tipo de objeto.
Cuando se incrementa o decrementa un puntero, el lenguaje C tiene en consideración el hecho de que no todos los tipos de datos ocupan igual número de posiciones de memoria, tal como se observa en el cuadro adjunto.
Cabe recordar que toda la información que contenga el ordenador ha de quedar reducida a ristras de ceros y de unos, reunidos en grupos de ocho. Suponga una sentencia como la que sigue:

```
char *caractpunt;
```

la cual declara a "caractpunt" como puntero a un carácter. Haciendo referencia a la zona de memoria representada en el cuadro "Los datos en memoria", puede lograrse que "caractpunt" apunte a la dirección del primer carácter de la misma a través del operador "&", tal como se expuso en el capítulo precedente.
Si ahora se ejecuta la orden:

```
++ caractpunt;
```

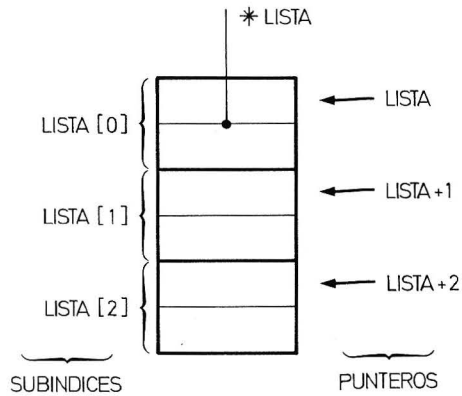
"caractpunt" se incrementa en una unidad, y pasa a apuntar al siguiente carácter almacenado.
De forma análoga, puede lograrse que un puntero señale al primer entero de la zona de memoria (posición 1125), declarando:

```
int *enteropunt;
```

esta vez, al hacer:

```
++ enteropunt;
```

su valor no se incrementará en una unidad, sino en dos, para apuntar al siguiente entero.
Análogamente, si aparece una sentencia en la que hay que incrementar el valor de un puntero para señalar hacia un dato de tipo real, el incremento debe ser el necesario para "saltar" la zona de memoria ocupada por el dato y quedar apuntando al inmediato posterior.
Todo lo expuesto para las operaciones de incrementar o decrementar es aplicable al caso de una suma o diferencia de un puntero y un número entero. Si se tiene un



El nombre de un array es, de hecho, un puntero orientado al elemento cero del mismo.

puntero que señala a un valor entero y se ejecuta:

enteropunt = enteropunt + 3;

el verdadero incremento que sufrirá "enteropunt" será de seis unidades y su efecto será el de saltar tres enteros consecutivos de la zona de datos sobre la que actúe.
En los próximos apartados se analizarán ejemplos concretos de la aritmética de punteros.

LA ESTRUCTURA ARRAY

Si hay una forma sencilla de agrupar una serie de datos es por medio de un array. La manipulación de los arrays en C es muy parecida a la habitual en los restantes lenguajes de alto nivel; sus peculiaridades aparecen al relacionar los arrays con los punteros.
Veamos en primera instancia cómo se declara un array. En el lenguaje C no existe una palabra clave al efecto, sino que, por ejemplo:

int a[6];

es la declaración de un array de nombre "a" que contendrá seis elementos enteros. De igual forma:

float num[100];

declara un array de cien elementos que serán números expresados en punto flotante. Para referenciar un elemento concreto del array se hace seguir al nombre de un subíndice; este subíndice puede ser una constante, una variable, o una expresión cuya evaluación sea un número entero, encerrado entre corchetes. Así, volviendo a la declaración de "a", para asignar el valor 10 al tercer elemento se procederá como sigue:

a[2]=10;

¿Hemos dicho tercer elemento? Más bien parece el segundo. En la figura adjunta aparece la justificación de esta aparente incongruencia.

REPRESENTACION INTERNA

Suponga un segmento de programa como el siguiente:

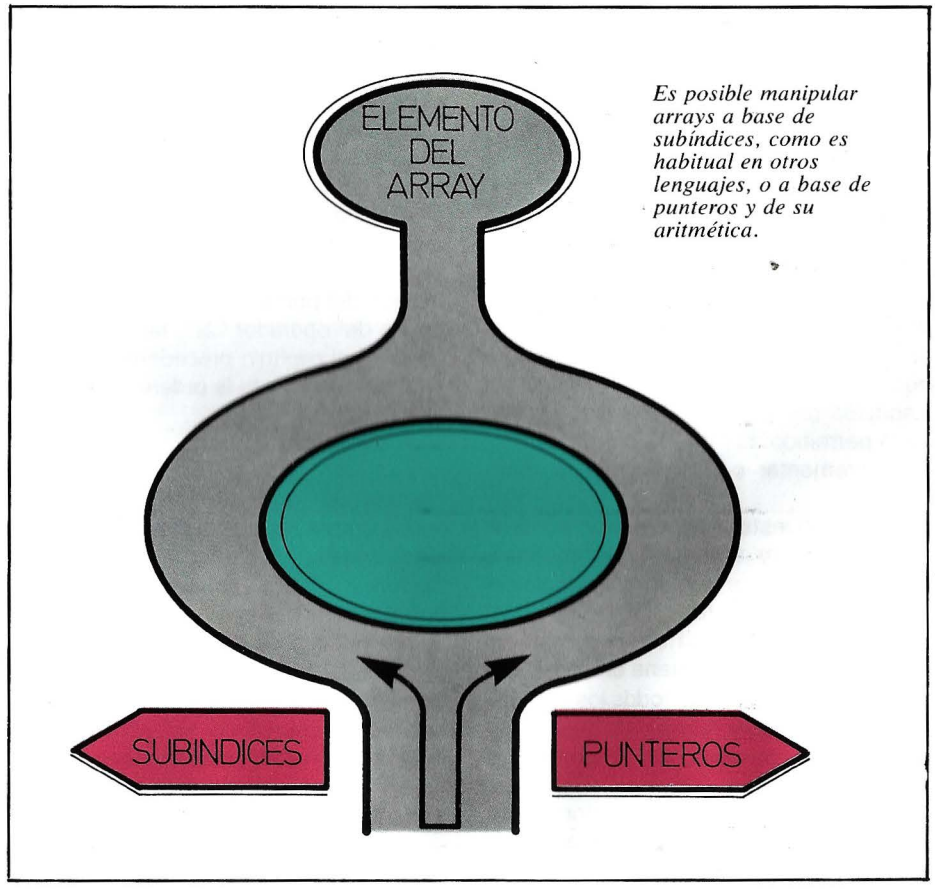
int lista[3];

lista[0]=2;
lista[1]=5;
lista[2]=−3;

El compilador de "C" al reconocer la declaración de "lista" en la primera línea, reserva tres espacios de memoria consecutivos a partir de una dirección de comienzo. Según esto, el acceso a un array puede interpretarse como tomar la dirección de comienzo y sumarle el desplazamiento necesario para acceder al elemento designado: para el elemento cero el desplazamiento es cero, para el primero es uno, y así sucesivamente.
¿No tiene este método un gran paralelismo con la técnica de los punteros? De hecho, en el C, el propio nombre de un array ("lista" en nuestro caso) constituye un puntero que señala hacia el elemento cero del mismo.
Por esta razón, y considerando lo dicho acerca de la aritmética de punteros, las expresiones:

lista[1]
*(lista + 1) y

son absolutamente equivalentes.
Normalmente, para referenciar al segundo elemento del array se utilizaría la expresión "lista[1]", pero como "lista" es un puntero al elemento "lista[0]", que es un entero, la expresión "lista + 1" apuntará dos posiciones más adelante (los enteros ocupan dos posiciones de memoria). Ahora puede aplicarse el operador "*" que da el contenido de la dirección a la que apunta lo que sigue a su derecha, obteniendo el mismo resultado en ambos casos. Lo descrito queda patente de un modo gráfico en la correspondiente figura.
Al igual que los otros tipos de variables, los arrays pueden ser inicializados también en el momento de su declaración. La



Es posible manipular arrays a base de subíndices, como es habitual en otros lenguajes, o a base de punteros y de su aritmética.

única restricción sobre este punto es que tal inicialización sólo se podrá llevar a cabo cuando su tipo de almacenamiento sea externo o estático.
Una inicialización de array tiene el siguiente aspecto:

```
static int a[6]= {1,2,3,4,5,6};
```

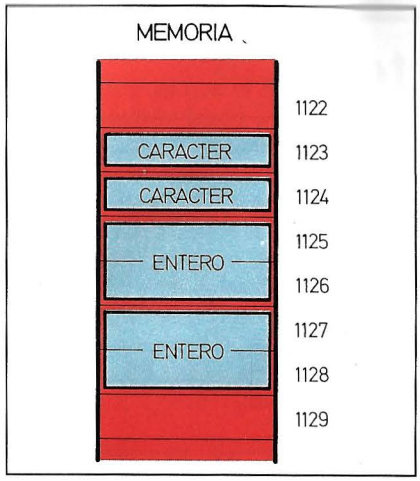
Como ejemplo del uso de arrays y de inicialización de uno de ellos con almacenamiento externo, en el cuadro adjunto se incluye un programa que calcula la suma de los elementos de un array.

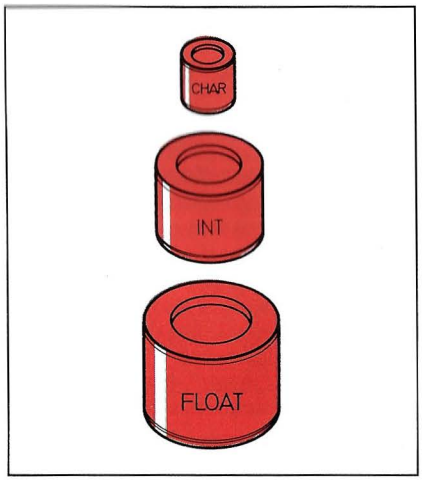
ARRAYS DE CARACTERES

Tienen su correspondencia en los llamados "string de caracteres" del BASIC; también en "C" se denominan así. Su manipulación es análoga al resto de los arrays. Se representan encerrando los ca-

Los datos en memoria

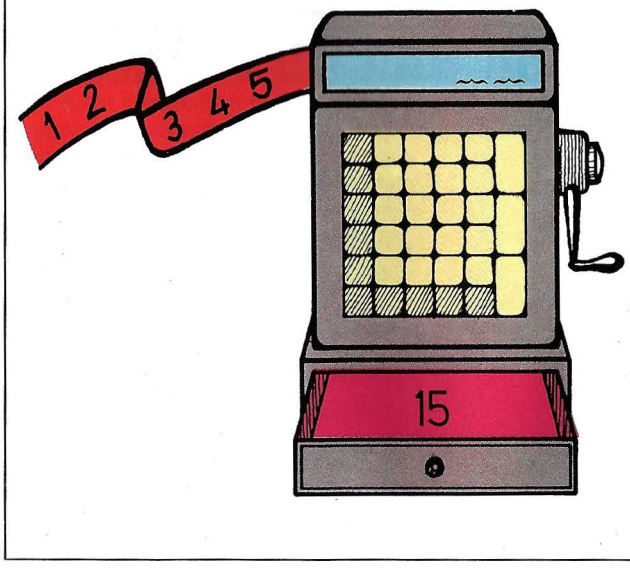
En el lenguaje "C", los caracteres ocupan una posición de memoria, mientras que los valores enteros necesitan dos para su almacenamiento. El espacio ocupado por un valor en punto flotante, en simple o doble precisión, depende de las características del compilador.
En la figura aparece una zona de memoria que contiene dos caracteres en las posiciones 1123 y 1124 y dos enteros que comienzan en las posiciones 1125 y 1127, respectivamente.
Conocer el tamaño del espacio ocupado por las variables ayudará a entender mejor la filosofía de trabajo del lenguaje "C".





Suma de los elementos de un array

En el programa adjunto, se observa que el parámetro "mtx" de la función "suma" no contiene ningún subíndice en la declaración; basta con indicar que se trata de un array de enteros. Para el lenguaje C, esta información es suficiente. La condición del bucle "while" en "suma" está para abreviar, no para confundir. El efecto sería el mismo si tal condición fuera, simplemente, "tamaño>=0" y la primera sentencia del bucle fuera "--tamaño".




```
int matriz[5]= {1,2,3,4,5};
int dim=5;

main ()
{
   int sum;

   sum=suma(matriz,dim);
   printf("La suma vale %d n",sum);
   }

suma(mtx,tamano)
int mtx[ ],tamano;
{
   int parcial=0;

   while (--tamano>=0)
      parcial=parcial+mtx[tamano];
   return(parcial);
   }
```

racteres constituyentes entre dobles comillas:

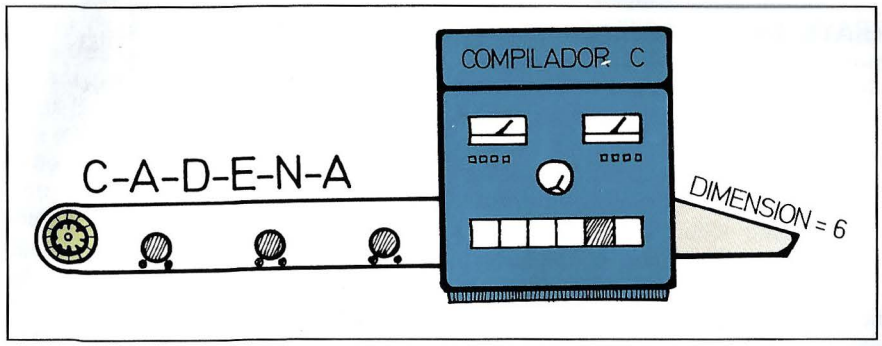
"Esto es un string"

El ejemplo propuesto coincide con un string de 18 elementos. Sin embargo, entre las comillas sólo aparecen 17 letras. Ello se debe a que se introduce un carácter nulo (que se representa por `\0` y es un sólo carácter) al final del array, lo que permite manipularlo fácilmente, como se observa en el cuadro que acompaña al texto. El programa incluido en el referido cuadro escribe en columna las letras que componen el "string" y su equivalente ASCII.

Es preciso destacar la diferencia que existe entre `x` y "x". La primera expresión representa a la letra x, mientras que la segunda es un string formado por dos caracteres: `x` seguido por `\0`. Hay que recordar también que `\0` es completamente diferente a `0`. El primero se representa en binario como 00000000, mientras que la representación binaria del segundo es 00110000. Por esta razón, se puede decir que no existe propiamente el string nulo en C; el string " " consta en realidad de un carácter, el nulo.

Strings de caracteres

En el ejemplo se observa cómo en la declaración e inicialización de "cad" no ha sido necesario incluir entre corchetes el número 6, correspondientes a su longitud. Al llevar explícita la inicialización, el compilador se encarga de asignar automáticamente la dimensión 6 a "cad". De igual manera, en el ejemplo relativo a la suma de los elementos de un array se podría haber suprimido el 5 en la declaración de "matriz". Esto también es válido para inicializaciones de arrays estáticos.




```
char cad[]="Cadena";
main()
{
   int i=0;
   while (cad[i] != `\0`)
      {
      printf("%c = %d\n",cad[i],cad[i]);
      ++i;
      }
}
```

```
C = 67
a = 97
d = 100
e = 101
n = 110
a = 97
—
```

Cuando el mundo tiene más de una dimensión...

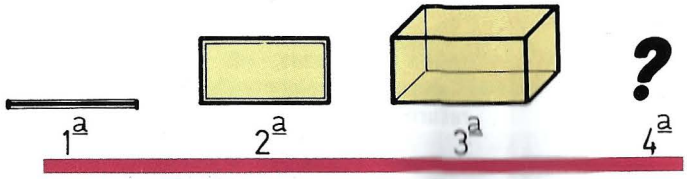
En un lenguaje como el C, cuyo campo de aplicación se centra en el desarrollo de software de sistemas, los arrays de una sola dimensión o lineales son los más utilizados, y en especial los arrays de caracteres. Sin embargo, también se pueden manipular arrays de dos o más dimensiones. Estos son tratados de igual forma que los lineales.

Un array bidimensional se declara, por ejemplo, como sigue:

int x[2] [3];

Y puede accederse al elemento (1,2) de la siguiente forma:

x[1] [2]=1;



En memoria, la representación es lineal, estando más próximos a la dirección de comienzo los elementos cuyo primer subíndice es más bajo, esto es:

00 — primer elemento
01
02
10
11
12 — último elemento

La inicialización de este tipo de arrays se realiza de la siguiente forma:

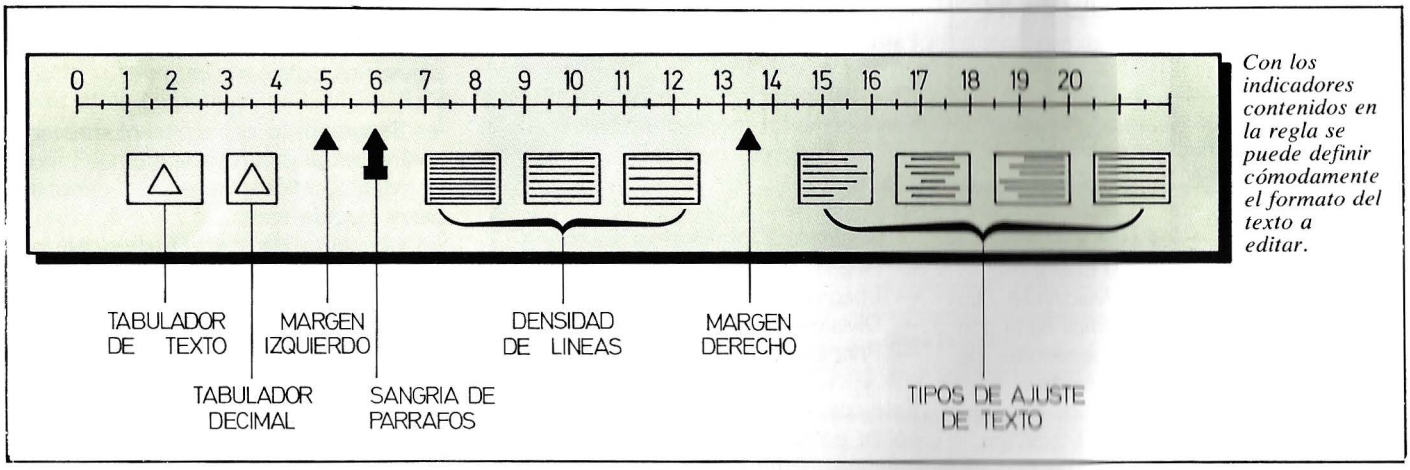
int x[2] [3]={{0,1,−3}, {2,7,150}};

Apple Macintosh (3)

El procesador de textos MacWrite

En capítulos precedentes se ha redundado en lo novedoso del método con que trata la información el microordenador Macintosh de la firma Apple Computer (emulación de escritorio, trabajo con iconos, utilización de menús, uso del ratón o "mouse", etc.), y en su gran adecuación para trabajos de oficina. En este capítulo y en el siguiente se analizarán las dos aplicaciones más características: el procesador de texto MacWrite y el programa para la creación de dibujos MacPaint; ambas entregadas junto con el equipo.

El uso combinado de las dos aplicaciones permite confeccionar documentos de una calidad extraordinaria; documentos en los que en todo momento es posible combinar texto con gráficos.



Con los indicadores contenidos en la regla se puede definir cómodamente el formato del texto a editar.

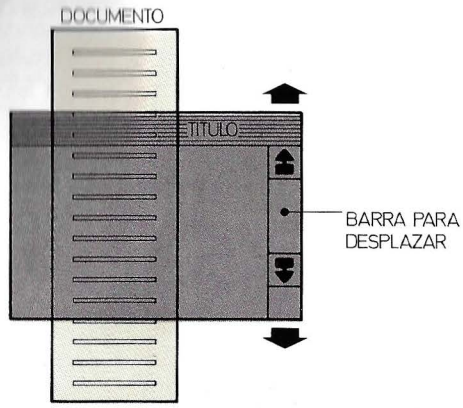
ACTIVANDO EL MACWRITE

La aplicación MacWrite se activa al abrir mediante el ratón y con el apoyo del Finder (cabe recordar que el Finder es el encargado de activar (abrir) y desactivar (cerrar) aplicaciones y documentos, además de gestionar el acceso a disco) el icono que representa a dicho procesador de textos. Desde luego, en ese instante debe estar insertado el disco que lo contiene. Transcurridos algunos segundos, el usuario estará ya en disposición de realizar todas las tareas propias de la edición de texto, creando un documento nuevo o modificando un documento ya existente. Cuando se desee terminar la sesión, se guardará en el disco el documento editado, en el caso de que se desee su conservación, y se desactivarán (cerrarán) todas las ventanas que se activaron para acceder al MacWrite.

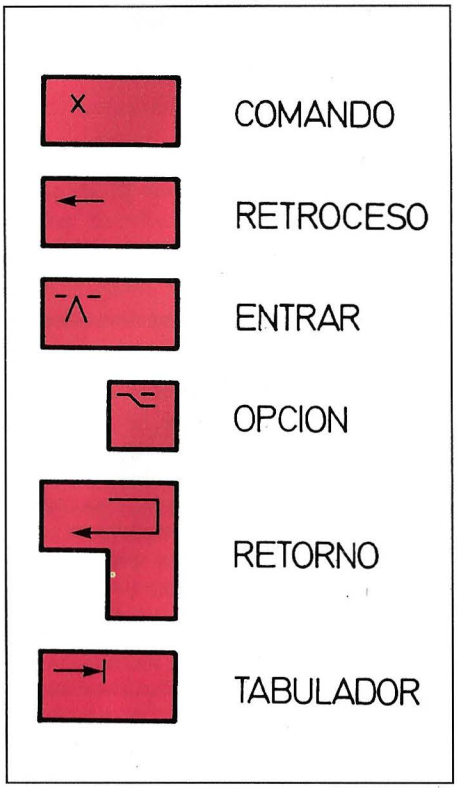
CARACTERISTICAS

Al abrir un documento con el MacWrite aparece en la pantalla una ventana con sus elementos característicos: cuadro de cierre, título, barras de desplazamiento e icono para el control del tamaño de la ventana, etc. En la ventana aparecerá también una regla que será de gran utilidad para fijar el formato del documento, como ya se explicará más adelante.

Los elementos de control que brinda el MacWrite son numerosos y diversificados, aunque todos ellos cómodamente seleccionables a través de menús de



La barra situada en el margen izquierdo de la pantalla de edición, sirve para desplazar la ventana de trabajo a lo largo del documento que por su excesivo número de líneas no es visualizable en su totalidad.



En el teclado del Macintosh existen varias teclas especiales utilizadas para ejecutar comandos y tareas de edición.

"persiana" (desplegables) o de botones accionables con una simple pulsación.
La potencia de edición del MacWrite y su comodidad de uso tienen difícil parangón entre los tradicionales procesadores de textos que existen en el mercado.
A continuación se describen los elementos básicos de trabajo con el MacWrite.

Teclas fundamentales

El teclado del Macintosh dispone de las teclas usuales de una máquina de escribir, colocadas en distribución de tipo QWERTY. Existen además otras teclas menos convencionales que son utilizadas como elementos de ayuda para la edición de documentos. Estas teclas son:

— Tecla Comando: ejecuta un comando de un menú cuando se la presiona junto con otra tecla. Por ejemplo, al accionar las teclas comando y R se obtiene un texto subrayado.
— Tecla Retroceso: retrocede para borrar caracteres.
— Tecla Entrar: confirma o da por terminada una entrada o comando.
— Tecla Opción: se utiliza para dar una interpretación alterna a otra tecla accionada. Sirve, por ejemplo, para la escritura de caracteres especiales.
— Tecla Retorno: hace que el punto de inserción de texto se mueva al principio de la línea siguiente.
— Tecla Tabulador: mueve el punto de inserción hasta el tabulador más próximo.

Párrafos

Un párrafo es considerado por el MacWrite como un conjunto de texto que esté situado entre cualquiera de los siguientes elementos:

— Acciones sobre la tecla retorno.
— Reglas
— Imágenes
— Divisiones de página
— Principio o final de documento

Este concepto difiere bastante del que se aplica en la escritura de un documento con una máquina de escribir, en donde los párrafos vienen definidos por los puntos y aparte. Así pues, la utilización de la tecla Retorno ha de hacerse cuidadosamente, ya que además de tener una función equivalente al salto de carro de la máquina de escribir, determina el fin de un párrafo.

La selección del texto sobre el que se van a efectuar tareas de edición se realiza sin más que arrastrar el puntero sobre el texto a seleccionar.

Páginas

Equivalen a una página de papel normal en la cual se escribe el texto de un documento. El tamaño se regula con el comando Ajustar del menú de archivo. Cada página puede incluir un encabezamiento y un pie de página; ambos pueden contener texto, imágenes, números de páginas, la fecha o la hora. Estos dos elementos determinan el número de líneas disponibles para la edición.

Reglas

La forma de definir el formato del texto hace uso de un método un tanto curioso: el empleo de reglas.
Para el MacWrite, una regla es un elemento que contiene una escala numerada de 0 a 21,5 (indicativo del ancho en cm. de una página estándar en USA), así como varios símbolos que sirven para definir las siguientes características:

— Sangrado del párrafo o columna en la cual se empieza un párrafo.
— Marcador de margen izquierdo (mínimo 3 cm.).
— Marcador de margen derecho (máximo 20,5 cm.).
— Tabuladores de texto: columnas a las que se llega pulsando la tecla tabulador.
— Tabuladores decimales: indica las columnas en las que se sitúa el punto decimal para listas de números.
— Espaciado de líneas: espaciado sencillo, espacio y medio y doble espacio.
— Ajuste de líneas: ajuste a la derecha, a la izquierda, centrado y ajuste derecha e izquierda (justificado).

La situación de los tabuladores y los marcadores de márgenes sobre la regla se varía arrastrando sus símbolos sobre la regla hasta el lugar adecuado, mientras que el espaciado y el ajuste de líneas se produce activando el cuadro o botón correspondiente.
El formato definido por una regla afecta a todo el texto que esté debajo de ella, siendo necesario insertar otras reglas si hubiera que tener varios formatos en un mismo documento.
La utilización de las reglas se parece tanto a los pasos necesarios a realizar en una máquina de escribir, que cualquier per-

sona familiarizada con la escritura de documentos es capaz de ajustar formatos sin ninguna dificultad.

Edición de texto

El punto de inserción de texto viene determinado por una barra vertical parpadeante. Al ir escribiendo, este punto de inserción se desplaza hacia la derecha según el flujo de escritura. Es posible variar su posición con la tecla de retroceso, con la tecla de retorno o bien con el ratón, arrastrando dicho punto hasta la posición deseada.

La corrección se realiza seleccionando el texto a corregir sin más que pasar por encima del mismo el puntero bajo el control del ratón, y manteniendo el botón pulsado hasta que se desee finalizar la selección. Acto seguido se pueden utilizar los comandos de cortar, pegar y copiar del menú de Edición.

En el caso de que alguna acción durante el proceso de edición no haya resultado satisfactoria, siempre se está a tiempo de volver atrás con objeto de anularla. Este efecto se consigue con el comando Deshacer Escritura, localizado en el menú de Archivo.

Cuando se haya finalizado el documento se seleccionará el comando Guardar del menú de Archivo, el cual grabará el documento en el disco. Es una buena práctica realizar esta operación a intervalos no muy superiores a 15 minutos, ya que de esta forma no se perderá un excesivo tiempo de trabajo ante cualquier eventualidad, tal como un corte accidental de la corriente.

MENUS DEL MACWRITE

Al igual que ocurría en el caso del Finder, tratado en el capítulo anterior, la ejecución de comandos y la elección de tipos y estilos de letra se realiza con menús desplegables cuyos títulos aparecen sobre la zona superior de la ventana de edición.

Al seleccionar cualquiera de los menús aparece una lista de opciones que pueden ser activadas para llevar a cabo la acción requerida. La elección de algunas de estas opciones o comandos ocasiona la apari-

Menús del MacWrite

Menú Apple

Apuntador	Reloj alarma	Bloc de notas
Calculadora	Teclado	Panel de control
Puzzle		

Menú Archivo

- Nuevo: Abre un documento nuevo y sin título.
- Abrir...: Abre un documento ya existente en el disco.
- Cerrar: Cierra la ventana activa.
- Guardar: Copia el documento actual al disco.
- Guardar como...: Copia el documento actual al disco con el nombre indicado.
- Ajustar página: Especifica el tamaño del papel.
- Imprimir...: Obtiene copia impresa del documento indicado.
- Salir: Se sale del MacWrite y se vuelve al Finder.

Menú Edición

- Deshacer escritura: Anula los efectos del comando anterior.
- Cortar: Elimina textos, reglas o divisiones de página, colocándolos en el portapapeles.
- Copiar: Copia textos, reglas o divisiones de página en el portapapeles.
- Pegar: Inserta el contenido del portapapeles en la posición que señale el puntero.
- Mostrar portapapeles: Visualiza el contenido del portapapeles.

Menú Búsqueda

- Buscar...: Localiza en el documento el texto indicado.
- Cambiar...: Cambia en el documento el texto indicado.

Menú Formato

- Insertar regla: Coloca una regla en el punto de inserción.
- Mostrar reglas: Vuelve visibles las reglas.
- Esconder reglas: Permite visualizar las reglas.
- Abrir encabezado: Abre la ventana de encabezado.
- Abrir pie: Abre la ventana de pie de página.
- Mostrar encabezados: Visualiza el encabezado.
- Mostrar pies: Visualiza el pie de página.
- Determinar no. página: Fija el número de página inicial.
- División de página: Indica el comienzo de página.
- Portada: Marca una página como portada.

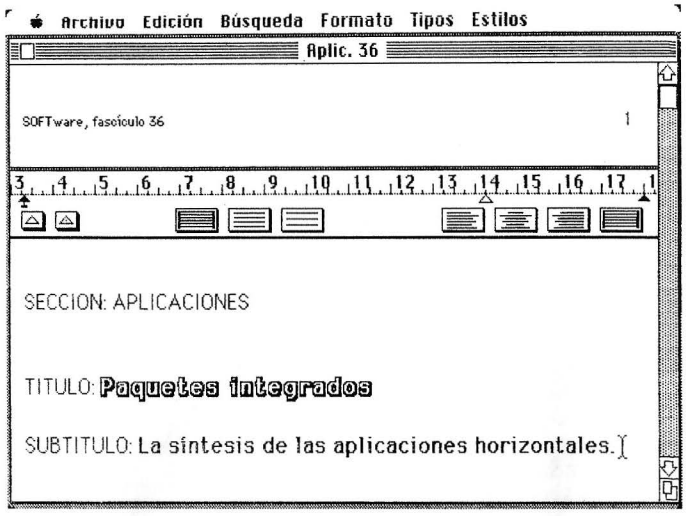
Menú Tipos

Venice	NewYork	London
Athens	Chicago	Geneva
Monaco	Toronto	...

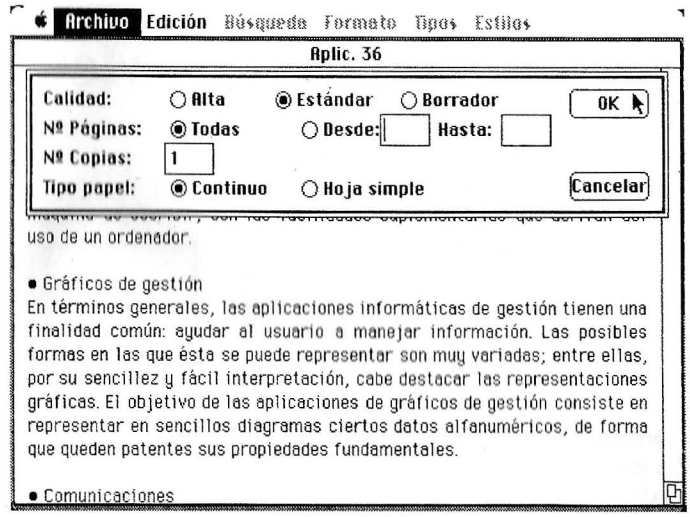
Menú Estilos

Común	Negrita	Cursiva
Subrayar	Hueca	Sombra
Indice	Subíndice	

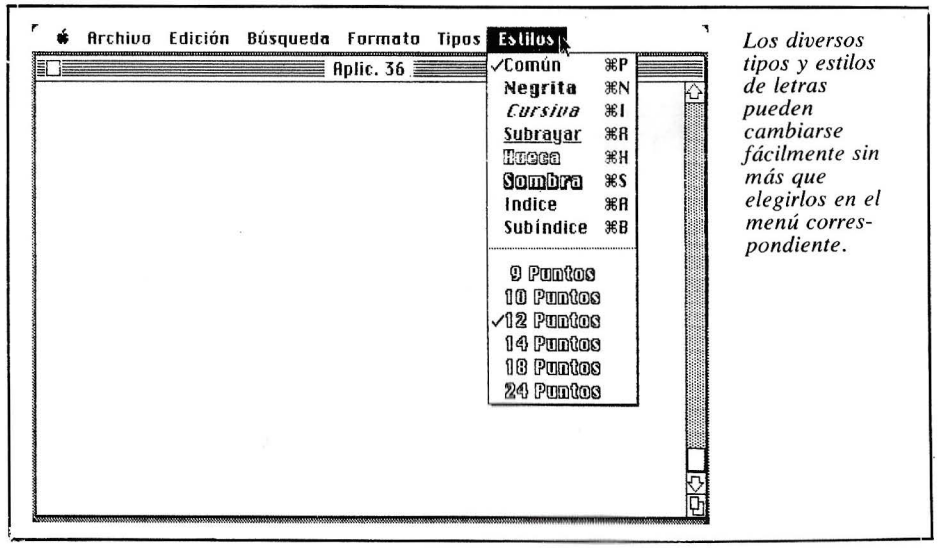
9,10,12,14,18 y 24 puntos por carácter



Aspecto de la pantalla de trabajo correspondiente al procesador de textos MacWrite.



La elección de alguna de las opciones conduce a cuadros de diálogo, en los que hay que rellenar los rectángulos con texto y activar los botones adecuados para acomodar la selección a nuestras necesidades. En la figura se reproduce el cuadro de diálogo asociado a la opción Imprimir del menú ARCHIVO.



Los diversos tipos y estilos de letras pueden cambiarse fácilmente sin más que elegirlos en el menú correspondiente.

ción de un cuadro de diálogo, con espacios reservados para introducir información, elegir alternativas, etc. En estos casos, después de cumplimentar el cuadro de diálogo se puede confirmar la ejecución del comando llevando el puntero hacia el recuadro etiquetado con OK y apretando el botón del ratón o, más simplemente, presionando la tecla Entrar. Si no se desea continuar con el comando siempre se puede volver atrás activando el recuadro "Cancelar".

Los comandos incompatibles con el estado actual de edición aparecen con un tono grisáceo atenuado (distinto del color negro habitual). Ello indica que su uso está restringido en ese instante.

Los menús disponibles en la aplicación MacWrite son los siguientes:

Menú Apple:

Contiene los accesorios de escritorio característicos del Finder.

Menú Archivo:

Sirve para abrir y cerrar documentos, crear nuevos documentos, guardar documentos en disco, imprimir, etc. En suma, gestiona el almacenamiento y el acceso a los documentos.

Menú de Edición:

Se utiliza en las operaciones de cortar, copiar y pegar textos o gráficos en un documento, además de permitir el acceso al Portapapeles.

Menú de Búsqueda:

Permite búsquedas y cambios de texto genéricos a través de todo el documento.

Menú de Formato:

Controla el formato utilizado en la edición. Desde el menú se accede a las reglas de formato, encabezado y pie de página, paginación, etc.

Menú de Tipos:

Selecciona los diversos tipos de letra existentes en el archivo de "fuentes", tipos que el usuario puede utilizar en sus documentos.

Menú de Estilos:

Permite seleccionar el estilo (común, negra, cursiva, subrayado...) y tamaño de los caracteres.

ALFA UNO (y 2)

Descripción de comandos y sesión de trabajo

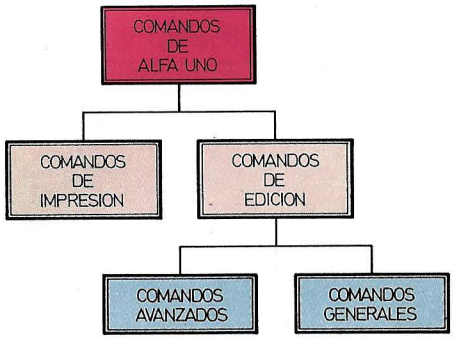
Para finalizar el estudio del programa para el tratamiento de textos ALFA UNO se describirá con rigor el funcionamiento de algunos de los principales comandos de edición e impresión. Por supuesto, las definiciones no podrán ser tan amplias como las que se incluyen en la documentación del programa, no obstante sí servirán de punto de partida para describir una sencilla sesión de trabajo con ALFA UNO.

COMANDOS BASICOS PARA LA EDICION DE DOCUMENTOS

Dentro de los comandos utilizables desde el programa ALFA UNO, cabe distinguir un primer grupo de carácter general. Su empleo en una sesión de trabajo será masivo, ya que su misión consiste en permitir al usuario dar las órdenes para la preparación del documento a editar. Entre ellos destacan los siguientes:

1. *Comandos de movimiento*

Como su propio nombre indica, sirven para mover el cursor dentro del documento editado y, de esta forma, señalar al programa el lugar del texto sobre el que se va a operar. ALFA UNO permite una gran variedad de posibilidades de mover el cursor a lo largo de toda la extensión del texto. Para ello, el usuario puede utilizar las teclas especiales del ordenador o utilizar los comandos del ALFA UNO precedidos siempre de la tecla <ESC> o <CTRL>. Existen veintiuna operaciones de movimiento en el programa ALFA UNO: desde las más sencillas como puede ser ARRIBA, ABAJO, DERECHA, IZQUIERDA; hasta las más complejas, que permiten saltar de párrafo en párrafo o utilizar un tabulador.



En el presente estudio sobre el programa ALFA UNO se han organizado los comandos según revela el esquema adjunto.

2. *Comandos de borrado*

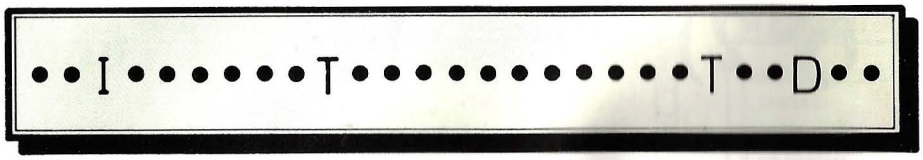
Sin duda, una de las ventajas más importantes ofrecidas por un programa para el tratamiento de textos consiste en la facilidad que brinda al usuario para el borrado de palabras y la reutilización automática del espacio sobrante. Dado que el borrado accidental de una palabra o frase puede traer consecuencias desagradables, ALFA UNO ofrece la posibilidad de recuperar el texto eliminado en la última operación de borrado.

Los comandos de borrado de ALFA UNO permiten múltiples opciones para borrar bloques de texto; así, en orden de menor a mayor, puede borrarse un carácter, una palabra, una línea, un párrafo, un bloque o incluso todo el texto.

3. *Comandos de inserción de texto*

La operación complementaria al borrado, es la inserción. Mediante ella se pueden incorporar nuevas palabras al documento editado de forma que el espacio se reorganice automáticamente. ALFA UNO ofrece distintos comandos de inserción. Se puede insertar un simple carácter o toda una línea; se puede partir una línea en dos para completar después ambas, y también se puede insertar directamente un "pantallazo"; por último, el usuario puede optar por abrir huecos y cerrarlos después de haber escrito el texto a insertar.

Las cinco teclas fundamentales para desplazar el cursor sobre un documento editado con ALFA UNO son las clásicas: hacia arriba, hacia abajo, a la derecha, a la izquierda y RETURN.

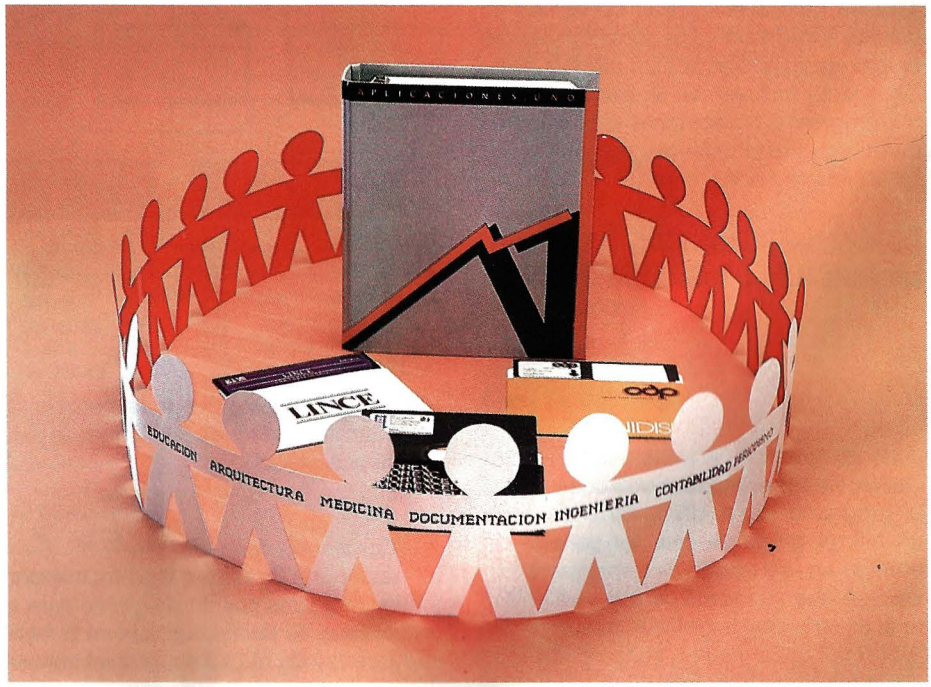


La regleta marca los límites izquierdo (I), derecho (D) y los puntos de tabulación (T). Su estructura puede ser modificada libremente por el usuario.

4. *Comandos de búsqueda*
Tanto si se desea borrar información, como si lo que se pretende es insertar nuevas palabras, existe la posibilidad de utilizar los comandos de movimiento para situarse en la posición donde se desea operar. No obstante, existe otro método mucho más rápido: buscar directamente la palabra a borrar o el origen de la inserción; para ello, ALFA UNO dispone de los llamados comandos de búsqueda. Después de haber ejecutado el comando búsqueda, el programa solicitará una cadena de caracteres y automáticamente situará el cursor en la zona del documento donde los haya localizado.
Una posibilidad adicional consiste en no sólo buscar una cadena de caracteres, sino reemplazarla por una nueva cadena en todas las ocurrencias que se encuentren sobre el documento.

5. *Comandos para modos de edición*
ALFA UNO admite diferentes modalidades para la edición de documentos. Para ello presenta un menú con cuatro modos distintos entre los que el usuario puede elegir.
Las características de cada uno de los cuatro modos quedaron definidas en el capítulo precedente, al estudiar el sistema de menús del ALFA UNO.

6. *Comandos para el tipo de letra*
Dentro del texto de un documento se pueden utilizar distintos tipos de letras. Para que el usuario pueda decidir en todo momento el tipo de letra a utilizar, ALFA UNO dispone de un menú con las siguientes posibilidades:
- SUBRAYADO: Subraya un fragmento del texto.
- GRUESA: Permite escribir parte del texto en negrilla (letra gruesa).
- CURSIVA: Activa el tipo de letra cursiva a partir de su ejecución.
- NORMAL: Tipo de letra utilizado por defecto.
- DOBLE: Aumenta el doble el ancho de los caracteres de una línea.
- MAYUSCULAS/MINUSCULAS: Transforma en minúsculas todas las letras mayúsculas de una porción de texto.
- MINUSCULAS/MAYUSCULAS: Transforma en mayúsculas todas las letras minúsculas de parte del documento.
- TECLADO: Permite seleccionar el juego de caracteres del teclado, entre ASCII o castellano.

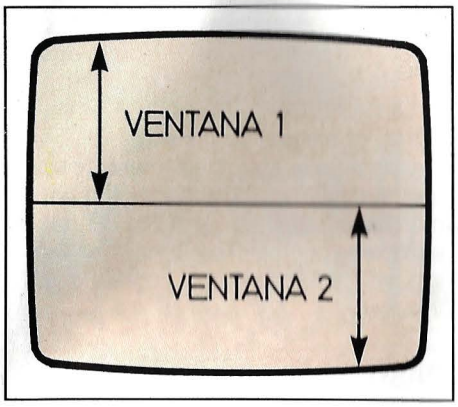


COMANDOS AVANZADOS PARA LA EDICION DE DOCUMENTOS

Además de los comandos anteriores, ALFA UNO dispone de toda una serie de órdenes para realizar operaciones más sofisticadas. Dentro de esta categoría podemos destacar los siguientes grupos de comandos.

1. *Comandos sobre bloques*
Su utilidad estriba en poder tratar porciones del documento de forma integrada, lo que, en consecuencia, facilita la reorganización del texto. Antes de realizar ninguna operación sobre un bloque, es necesario MARCAR el primero y el último de sus caracteres; a continuación se puede elegir entre las siguientes operaciones:
- QUITAR: Elimina el bloque marcado.
- PONER: Inserta el bloque marcado tras la posición del cursor.
- GUARDAR: Almacena en un soporte externo el bloque definido previamente.
- RECUPERAR: Inserta un bloque almacenado en un soporte externo tras el cursor.

2. *Comandos sobre archivos*
Facilitan el traspaso de documentos desde la memoria principal a un soporte externo o viceversa. Para ello se ofrecen las siguientes opciones:
- TRAER: Trae de soporte externo a memoria principal.
- GUARDAR: Guarda de memoria principal a soporte externo.
- RENOMBRAR: Permite cambiar el nombre de un archivo .
- BORRAR: Elimina un documento del soporte externo en el que esté almacenado.
- INCLUIR: Inserta un documento archivado en el documento en edición.
- COPIAR: Archiva una copia de un documento del soporte externo.



Mediante la opción ventana se pueden realizar dos sesiones de trabajo al tiempo, ya que es posible operar sobre dos documentos con simultaneidad.

- PROTEGER: Marca un archivo como no modificable.
- DIR: Permite visualizar el directorio de todos los documentos archivados en el soporte de memoria externa que se encuentre activo.

3. *Comandos sobre formatos*

Existe cinco comandos distintos para modificar los formatos de edición:

- JUSTIFICAR: Cambia el formato del párrafo sobre el que se encuentre el cursor según el estado de la regleta.
- REGLETAS: Modifica la regleta activa.
- ALINEAR: Alinea o centra una línea de texto.
- LINEAS: Dibuja o borra líneas verticales en el documento.

El origen español del ALFA UNO hace que tanto sus comandos como su "filosofía" sea óptima para producir documentos redactados en castellano.

Incidente en el ordenador

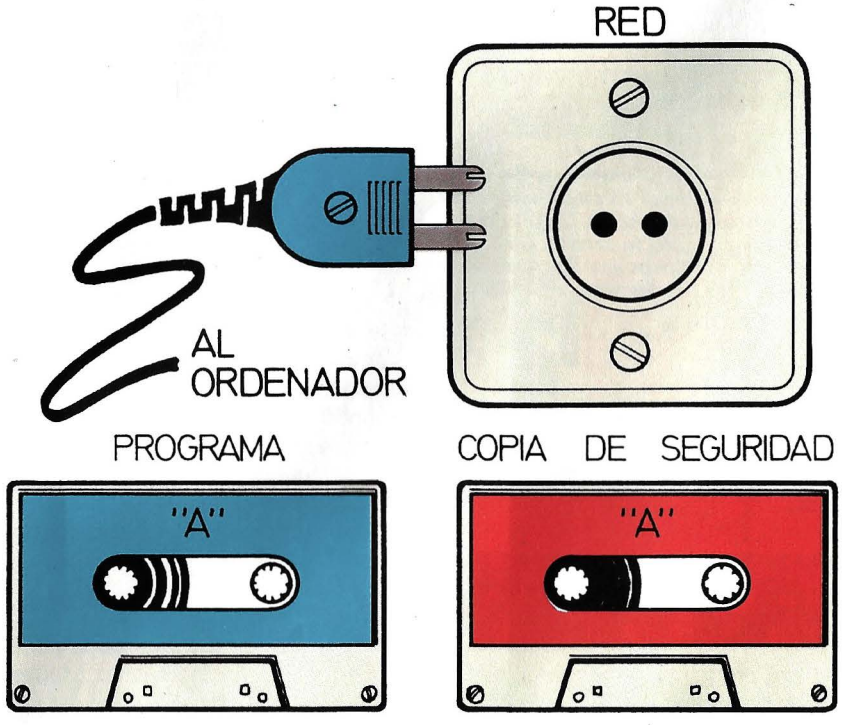
La memoria principal de los ordenadores personales es de tipo RAM y con la característica de "volátil"; es decir, su contenido se borra instantáneamente ante la falta de alimentación eléctrica. Desde luego, cuentan también con alguna zona de memoria ROM, cuyo contenido no se ve afectado por un corte en la alimentación eléctrica. No obstante, esta zona permanente de la memoria está reservada para el sistema operativo y el usuario no puede incluir en ella sus propios programas.

En resumidas cuentas, tanto los programas de aplicación como los datos necesarios se almacenan en soportes de memoria externos a la unidad central; de tal forma que al comenzar cada sesión los programas y datos necesarios se cargan en la memoria principal, y al final se copian de nuevo los datos actualizados sobre el soporte externo. Ello elimina el problema derivado del borrado de la memoria principal al apagar el ordenador. Pues bien, en algunos casos resulta imprescindible disponer de dos copias en soportes externos: una de trabajo y otra de seguridad. Como justificación, vamos a relatar un incidente grave que sucede con harta frecuencia entre los usuarios de ordenadores.

Suponga que se está desarrollando un nuevo programa de aplicación y que la dificultad de dicho programa implica que el trabajo se realice en distintas sesiones. Evidentemente, al finalizar cada una de estas sesiones se obtendrá una copia del programa en un soporte externo. Y suponga también que no se efectúa ninguna copia de seguridad.

El programa está ya prácticamente finalizado y el usuario se presta a realizar las últimas modificaciones estéticas sobre el mismo. Para ello, lo cargamos en la memoria principal, realiza las modificaciones oportunas, y termina copiándolo de nuevo sobre el soporte externo. Pero en ese preciso instante una "mano inocente" desenchufa el ordenador... Si este accidente ocurre a mitad de la grabación, se habrá perdido completamente todo el trabajo realizado durante tantos y tantos días: en la memoria principal se habrá borrado el programa, y la grabación del nuevo programa sobre la versión anterior que había empezado a realizarse provocará la destrucción de dicha versión previa.

Ante una situación como la descrita —más frecuente de lo que cabría desear—, huelga cualquier recomendación acerca de la conveniencia de obtener frecuentes copias de seguridad de la información.



COMANDOS EMBEBIDOS DEL ALFA UNO

Comando	Descripción
.LP nn	Una página impresa contendrá nn líneas
.MS nn	Emplear un margen superior de nn líneas
.MI nn	Emplear un margen inferior de nn líneas
.ES nn	Dejar nn líneas en blanco entre dos de texto
.SP nn	Saltar al inicio de la página nn.
.SC nn	Empezar una nueva página si faltan menos de nn líneas para acabar la actual
.LI nn	Situarse en la línea nn de la página
.CA nn	Las nn líneas siguientes son el texto de cabecera
.PI nn	Las nn líneas siguientes son el texto de pie de página
.CP nn	Las nn líneas siguientes son el texto de cabecera de las páginas pares
.PP nn	Las nn líneas siguientes son el texto de pie de página de las páginas pares
.CN nn	Las nn líneas siguientes son el texto de cabecera de las páginas impares
.PN nn	Las nn líneas siguientes son el texto de pie de página de las páginas impares
.AD	Ajustar el paginado para iniciar una página derecha.
.AI	Ajustar el paginado para iniciar una página izquierda
.IN "nombre"	Incluir el fichero <nombre> en el texto durante la impresión
.PA	Pausa. El impresor espera a que se pulse una tecla
.BL x	El carácter x se imprime en lo sucesivo como un blanco

Como se puede apreciar en este resumen, la sencillez de manejo del ALFA UNO es muy grande. Prácticamente, la descripción de las operaciones a realizar en una típica sesión de trabajo coinciden con los nombres de los comandos a utilizar.

● PAGINA: Permite modificar parámetros tales como: líneas por página, márgenes, etc.

4. *Comandos sobre ventanas*
ALFA UNO permite dividir la pantalla en dos ventanas diferentes, en cada una de las cuales se podrá ejecutar cualquier operación del programa; en general, suelen utilizarse para editar dos documentos distintos simultáneamente.

COMANDOS PARA IMPRESION

Los comandos de impresión de ALFA UNO permiten imprimir copias sobre papel de cualquier documento elaborado con el procesador de textos. Los módulos de edición e impresión son independientes. Cuando el usuario selecciona en el menú principal la opción IMPRIMIR, aparecerá un nuevo menú con cuatro posibilidades:

1. IMPRESORA
Su misión consiste en enviar un documento a la impresora, produciendo una copia de acuerdo a los formatos y otras características de impresión en activo.

2. ARCHIVO
Se encarga de archivar en el soporte externo de memoria un documento paginado. Es importante no confundir este comando con otros ya descritos que también se encargan de gestionar la memoria auxiliar; aquellos almacenaban documentos editables, mientras que el comando estudiado ahora archiva documentos imprimibles.

3. PANTALLA
Permite revisar a través de la pantalla el aspecto final del documento. Mediante su ejecución se evitará gastar papel inútilmente, ya que tan sólo se imprimirá el documento cuando el usuario tenga garantía de que no contiene ningún error.

4. OPCIONES
Como su propio nombre indica, al ser ejecutado presenta un nuevo menú de opciones que permiten que el usuario defina: márgenes, alineado, longitud de página, espaciado, número de copias, páginas a imprimir y selección de páginas pares e impares.

PASOS BASICOS EN UNA SESION DE TRABAJO

Para "arrancar" el programa es suficiente con teclear ALFA UNO y pulsar seguidamente la tecla RETURN; por supuesto, estando el ordenador bajo el control del sistema operativo ("prompt" en la pantalla). Al cabo de unos segundos aparecerá el nombre ALFA UNO en la parte superior izquierda de la pantalla; esta es una indicación de que el programa se encuentra disponible para comenzar a trabajar. En este momento, el usuario puede elegir entre comenzar la edición de un nuevo documento o recuperar alguno de los que estén almacenados en memoria auxiliar. Supongamos que en nuestra sesión se trata de producir un nuevo documento basado en uno de los ya existentes, la única modificación que deseamos realizar es sustituir el nombre " LUIS PEREZ RUIZ" por " JUAN LOPEZ SANCHO " en todas las líneas en que aparezca. Para ello, pulsaremos <ESC> <?> con lo que aparecerá el menú principal y, dentro de él, optaremos por ARCHIVOS; en el siguiente menú elegiremos el comando TRAER e indicaremos el nombre del fichero donde está almacenado el antiguo documento.
Después de esto retornaremos al menú principal y seleccionaremos la opción BUSQUEDA; de esta forma aparecerá un nuevo menú en el que seleccionaremos SUSTITUCION, tecleando a continuación el antiguo nombre y elegiremos la opción para que la sustitución se realice en todo el documento. Por último, teclearemos el nuevo nombre. Automáticamente se obtendrá un nuevo documento que podrá ser escrito o guardado en un soporte externo.

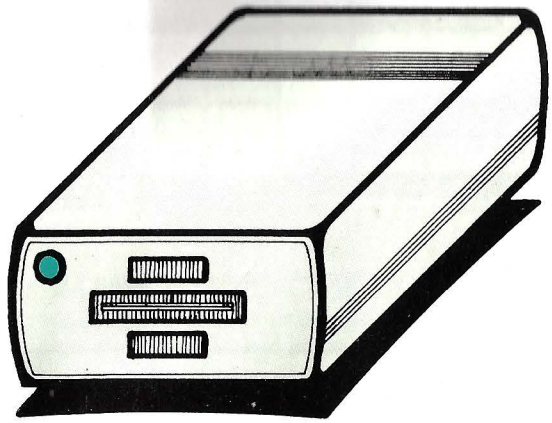
Los archivos del Commodore 64

Tratamiento de archivos en la unidad de disco 1541

La unidad de disco básica de la familia Commodore (apellidada 1541) es un periférico inteligente. En efecto, posee su propio microprocesador de 8 bits (un 6502), rodeado de 2 Kbytes de memoria RAM y de una zona de memoria ROM en la que está almacenado el sistema operativo de disco (DOS).

Esta "inteligencia" facilita el tratamiento o manipulación de los archivos almacenados en el disquete, independientemente de la tarea que en cada instante esté ejecutando el microprocesador que dirige las actividades de la unidad central. Así, por ejemplo, la unidad de disco puede dar salida por sí mismo a algunos archivos; enviando la información hacia la impresora para que ésta imprima el contenido de los mismos en papel. Mientras tanto, el ordenador puede ejecutar cualquier otra tarea, puesto que no se ve en la necesidad de atender al proceso que está en curso entre la unidad de disco y la impresora.

Las operaciones sobre los archivos almacenados en disco se desencadenan a través de "órdenes" BASIC o por medio de comandos del sistema operativo; por supuesto, esgrimiendo los números de archivos lógicos adecuados y los números de dispositivos periféricos correspondientes, cuyos formatos se estudiarán en los próximos apartados. Hay que tener en cuenta (y de ahí el motivo del presente capítulo) que las órdenes a utilizar en el caso que nos ocupa difieren de las establecidas para otros intérpretes BASIC

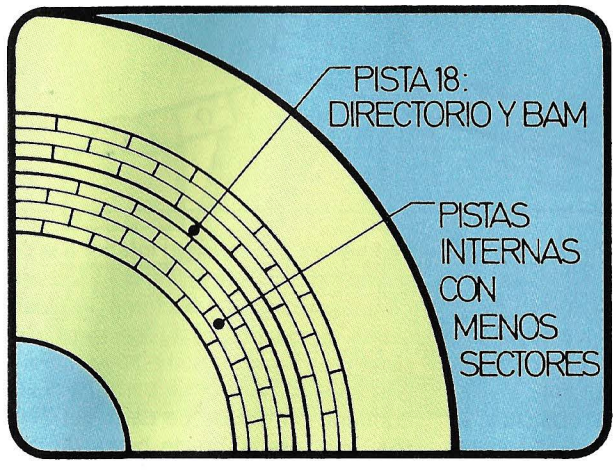


La unidad de disco 1541 de Commodore es un periférico "inteligente", que incorpora su propio microprocesador del tipo 6502.

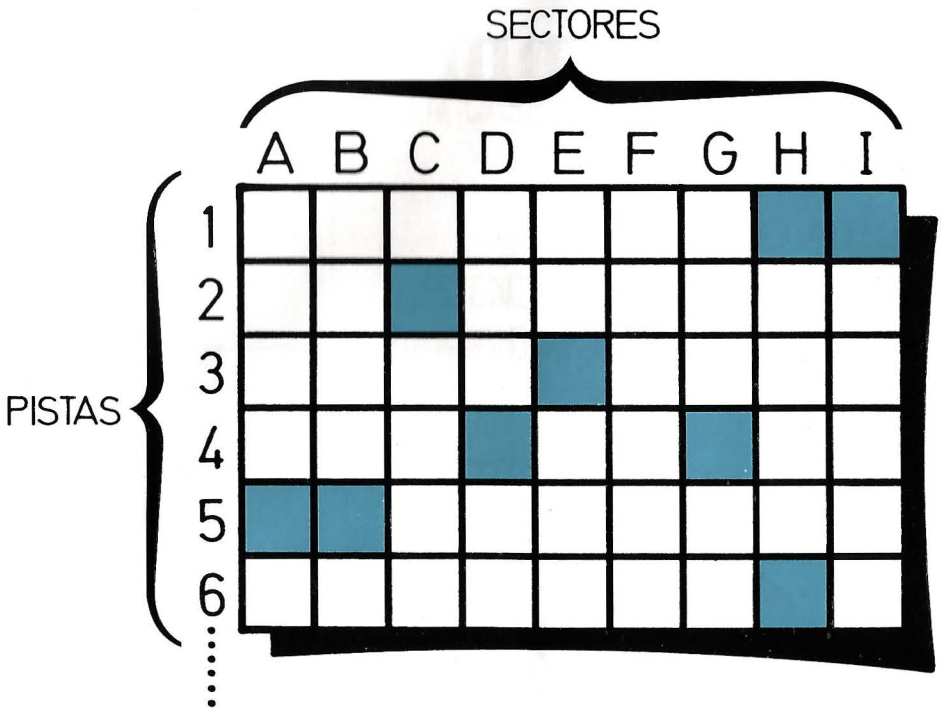
EL FORMATO DE LOS DISCOS

El sistema operativo de disco de Commodore ofrece una primera ventaja significativa: este no debe carg rse a partir de disquete, puesto que reside en la ROM incluida en la circuitería de la propia unidad de disco. Ello significa que las funciones del DOS están disponibles desde el preciso momento en el que entra en funcionamiento la unidad de disco conectada al ordenador.

Por otro lado, el DOS maneja un amplio conjunto de órdenes destinadas a que el usuario pueda codificar complicados programas que manipulen tanto los archivos



El DOS distribuye los sectores en las pistas del disquete, de tal forma que los interiores "tocan" a menos sectores por ser inferior su longitud.



El BAM lleva la cuenta de los sectores del disco que han sido ocupados por datos y de los que permanecen disponibles.

El directorio contiene información de diversa índole, relativa a los archivos almacenados en el disco.

de programas, como los archivos de datos del usuario.

Pero vayamos al principio. Como es natural en este tipo de soportes, antes de empezar el trabajo con un disco es preciso formatearlo en pistas y sectores. El formato de los disquetes que proporciona el DOS consta de 35 pistas, divididas en sectores. Dado que las pistas más internas tienen una longitud inferior a la de las pistas más externas, el DOS las divide en distinto número de sectores. Así, esta división va desde los 17 sectores para la pista más interna, hasta 21 sectores para la más externa. Por su parte cada sector contiene un bloque de datos de 256 bytes, más una serie de bytes de control necesarios para el correcto funcionamiento del sistema. De esta forma se obtiene un espacio máximo de 170 Kbytes libres por disco flexible de 5 y 1/4 pulgadas.

El formateado del disquete se completa con un mapa de disponibilidad de bloques (BAM=Block Availability Map) y un directorio de los archivos del disco. Las referidas tablas sirven para administrar la distribución de los datos en el disco. Ambas están almacenadas en la pista 18 del disquete. Más concretamente, el BAM se encuentra situado en el sector 0 de dicha pista y está constituido por 144 bytes que señalan qué bloques están libres para el almacenamiento de datos y cuáles no. El directorio del disquete se sitúa a partir del sector 1, y es una lista que puede incluir hasta 144 nombres de archivos asociados a información relativa al tipo de archivo y al número de bloques que lo constituyen.

El BAM y el directorio se van actualizando automáticamente a medida que se introducen datos en el disquete, aunque también es posible acceder a ellos directamente, como se verá más adelante.

LOS ARCHIVOS DE PROGRAMAS

Los programas BASIC se almacenan en el disquete como archivos binarios. Para ello se dispone de las tradicionales órdenes BASIC LOAD y SAVE, análogas a las utilizadas para el almacenamiento de datos en casete, aunque con ciertas diferencias en su formulación:

```
LOAD <nombre$>, <dispositivo>
  [, <comando>]
```

Donde el nombre del programa (<nombre$>) es una cadena de caracteres válida, es decir: un nombre encerrado entre comillas o el contenido de la variable alfanumérica especificada. El <dispositivo> es un número entero que identifica al periférico al que se quiere acceder. Inicialmente, este número está fijado por hardware, de forma que a la unidad de disco le corresponde el número 8; si bien, es posible modificar este número, ya sea por software o por hardware, para así poder conectar simultáneamente más de una uni-

dad de disco. El <comando> es otro número entero cuyo uso es opcional; si se omite o se hace igual a 0, el programa se carga normalmente, mientras que si le otorga el valor 1 se cargará en las mismas posiciones de memoria que ocupaba al ser traspasado al disquete. Veamos algunos ejemplos:

LOAD "Programa prueba", 8
LOAD NOMBRE$, 8, 1
LOAD N$, D, C

El comando LOAD se utiliza también para "leer" el directorio del disquete. Este recibe un tratamiento por parte del DOS como si se tratara de un programa BASIC, de forma que puede ser "cargado" en la memoria del ordenador y "listado" para examinar su contenido. El nombre asignado a este "programa" es "$", de forma que la orden completa que permite "cargar" el directorio del disquete será:

LOAD "$", 8

Y una vez cargado no queda más que ejecutar la orden LIST para que aparezca el directorio en la pantalla. Este listado contiene la siguiente información:

—Nombre del disquete.
—Identificador (de dos caracteres) del disquete.
—Hasta 144 nombres de archivos.
—Tipo de cada archivo.
—Longitud en bloques de cada archivo.
—Número de bloques libres disponibles.

Los nombres de los programas almacenados en el disquete, admiten también los clásicos "comodines" o "wildcards" ("*" y "?"). Con ello es posible que una misma orden haga referencia a múltiples archivos que compartan tan sólo alguna característica en sus respectivos nombres.
Hay que tener en cuenta que, debido al tratamiento que se le da al directorio con este método, su carga "destruye" cualquier otro programa BASIC residente en ese preciso momento en la memoria del ordenador, con lo que se pierde esa información. En todo caso, también es posible leer el directorio sin perder la información previamente almacenada en memoria; ello supone leer el archivo "$" por medio del comando GET#, como si se tratara un archivo secuencial.
El formato del comando SAVE es totalmente análogo al de la LOAD ya descrita:

LOAD

Carga el programa nombre$ desde el dispositivo especificado.

Formato: LOAD <nombre$>, <dispositivo>, <comando>

Ejemplos: LOAD "PRUEBA", 8
LOAD A$, 8, 1

SAVE

Almacena un programa en el dispositivo especificado y le asigna el nombre: nombre$.

Formato: SAVE <nombre$>, <dispositivo>, <comando>

Ejemplos: SAVE "TEST", 8, 1
SAVE B$, D, C

SAVE <nombre$>, <dispositivo>
[, <comando>]

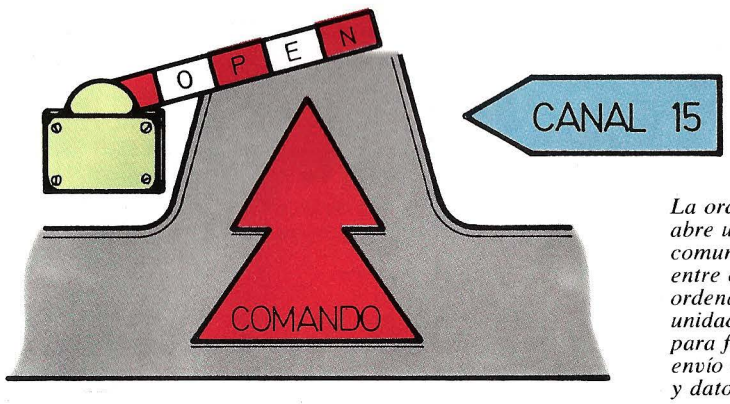
Por lo que respecta a la actuación de este comando, hay que considerar que se producirá un error en la unidad de disco cuando se intente almacenar un archivo con un nombre ya existente en el disco activo. En tal situación, será necesario leer el "canal de errores", borrar el archivo antiguo, y almacenar entonces el nuevo. Este inconveniente se puede evitar sin más que indicar en el comando SAVE lo siguiente:

SAVE "@0 : "+NOMBRE$, 8

Con lo que el nuevo archivo reemplazará automáticamente al antiguo, sin necesidad de recurrir a otros comandos adicionales.

LOS COMANDOS DEL DISCO

El cometido y principales aplicaciones de los comandos OPEN y PRINT# del lenguaje BASIC, han sido ya descritos en un capítulo anterior de esta obra. Respecto a su funcionamiento con la unidad de discos de Commodore, cabe precisar que su uso es semejante al definido como caso general. La novedad de Commodore consiste



La orden OPEN abre un canal de comunicación entre el ordenador y la unidad de disco para facilitar el envío de órdenes y datos.

OPEN

Abre un canal de comunicación entre el ordenador y el archivo contenido en un determinado dispositivo. También establece un canal de comandos.

Formato: OPEN <archivo>, <dispositivo>, <canal>, <texto$>

Ejemplos: OPEN 15, 8, 15A$
OPEN N1, N2, N3

PRINT

Envía la lista de variables que figuran en su argumento a través del canal abierto para el archivo especificado. También puede enviar comandos.

Formato: PRINT# <archivo>, <lista de variables>

Ejemplos: PRINT# 15, "NEW : DISCO1"
PRINT# 5, A$: B$; C$; D$

en que también pueden ser utilizados para abrir un canal para comandos de disco que controlen los intercambios de información entre el ordenador y la referida unidad.
El formato para el comando OPEN es:

OPEN <archivo>, <dispositivo>,
<canal>, <texto$>

Donde: <archivo> es un número entero comprendido entre 1 y 255 que identificará durante el resto del programa al archivo al que se quiere acceder. El número del <dispositivo> es, para el caso de la unidad de discos, el 8 y ya ha sido comentado con anterioridad. El <canal> debe ser un número comprendido entre 1 y 15, y se refiere al canal de comunicación entre unidad de disco y ordenador que se desea utilizar. Los números 0 y 1 ya han sido comentados y se reservan para su uso con LOAD y SAVE. Del 2 al 15, se utilizan para el trasvase de datos a archivos, y el canal 15 es a través del cual se deben enviar los comandos de disco. El <texto$> es una cadena de caracteres válida, que será enviada al archivo abierto, al igual que si se ejecutase un comando PRINT#; ello significa que pueden enviarse comandos a través del canal con el auxilio de la orden OPEN o directamente con PRINT#.

El comando PRINT# funciona análogamente al PRINT convencional, salvo que en vez de enviar el mensaje a la pantalla lo enviará por el canal especificado hacia el archivo previamente abierto. Finalmente, se debe cerrar todo archivo o canal abierto, que ya no sea necesario, mediante la ya conocida orden CLOSE.
Entre los distintos comandos disponibles, quizás el de primera necesidad sea el que se utiliza para formatear o inicializar el disquete, de lo contrario no será posible acceder al disco. Este comando es NEW, cuyo cometido es borrar completamente el soporte magnético, definir en él las marcas de control de los bloques, y crear el directorio y el BAM. Su formulación es la siguiente:

PRINT#15, "NEW:<nombre$>
[,<identificador>]"

En donde: <nombre$> es una cadena de caracteres válida que constituye el nombre que se asignará al disco y que aparecerá en la cabecera del directorio. El <identificador> constará de dos caracteres únicamente y su uso es opcional. Sirve para caracterizar a todos los ficheros de ese disquete, y evitar que se cometa algún error al intentar acceder a un archivo de otro disco. Un ejemplo de formulación es el que sigue:

PRINT#15, "NEW:disco de pruebas, LC"

Otro comando de utilidad es COPY, el cual sirve para obtener copias en el mismo disco de un programa o archivo ya residente en el mismo, aunque asignándole un nuevo nombre.

La unidad de disco 1541 es un periférico de la gama Commodore compatible con los populares VIC-20 y Commodore 64.

Por ejemplo:

PRINT#15, "COPY:COPIA=ORIGINAL"

Este mismo comando permite también crear un nuevo archivo de tipo secuencial, que sea la concatenación de hasta otros cuatro archivos secuenciales diferentes. La que sigue es su formulación más genérica:

PRINT#15, "COPY:<nombre nuevo>=
<nombre antiguo 1>[, <nombre
antiguo 2>, <nombre antiguo 3>,
<nombre antiguo 4>]"

Si sólo se desea cambiar el nombre de un archivo, sin modificar la información que contiene, hay que recurrir al comando RENAME, tal como se indica en las siguientes líneas:

PRINT#15, "RENAME:<nombre
nuevo>=<nombre antiguo>"

Un ejemplo práctico es el siguiente:

PRINT#15, "RENAME:JOSE=PEPE"

Para borrar un archivo del disco se dispone del comando SCRATCH, el cual hay que complementarlo con el nombre o nombres de los archivos a borrar.
Desde luego, en la formulación de SCRATCH está permitido el uso de las referencias ambiguas "*" y "?". Su formato general será, por tanto:

PRINT#15, "SCRATCH:<nombre1$>
[, <nombre 2$>, ...]"

Si al ejecutar algún comando se produce en el disco alguna condición de error, no será posible realizar en él una nueva acción hasta que no sea inicializada de nuevo la unidad de disco, dejándola en el mismo estado en que se encontraba al conectarla. El comando que realiza esta misión es el siguiente:

PRINT#15, "INITIALIZE"

Finalmente, hay que presentar un último comando de disco, VALIDATE, cuya función es la de reorganizar los archivos residentes en el disco flexible. La desorganización tiene su origen en las sucesivas grabaciones y borrados de archivos que darán lugar a la existencia de bloques libres entre archivos. Con la ejecución de VALIDATE también se liberan los bloques utilizados por archivos que no han sido correctamente cerrados y que, por lo tanto, no son accesibles. Este nuevo comando se introduce de la forma siguiente:

PRINT#154, "VALIDATE"

Todos los comandos de discos pueden formularse de forma abreviada mediante su inicial; así, el último comando descrito podría introducirse como sigue:

PRINT#15, "V"

La incorporación de la unidad de disco al Commodore 64 potencia las posibilidades de este microordenador, capacitándolo para ejecutar múltiples aplicaciones de gestión en las que resulta imperativo el trabajo con archivos de acceso aleatorio.

ARCHIVOS SECUENCIALES

Los archivos secuenciales se construyen en el disquete transfiriendo a éste toda la información, byte a byte, a través de un buffer. Por esta razón, deben ser escritos y leídos de principio a fin, para lo que previamente se debe establecer un canal de comunicación de datos mediante la orden OPEN:

OPEN <archivo>, <dispositivo>,
<canal>, <texto$>

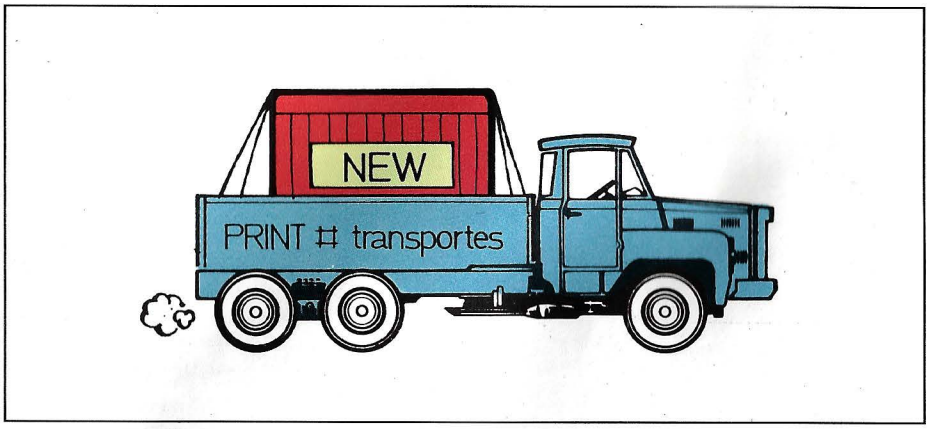
Los parámetros que acompañan a este comando son ya conocidos. Sólo varía el número del canal, que en este caso estará comprendido entre 2 y 14, y el <texto$>. Para la creación de archivos de tipo secuencial, este último parámetro debe tener el siguiente formato:

"0" :<nombre>, <tipo>, <dirección>"

La indicación "0 :", debe preceder siempre al nombre del fichero, para así poder acceder a más de dos de los buffers disponibles en la unidad de disco. El <nombre> será una cadena de caracteres válida y constituirá el nombre del archivo a crear o al que se quiere acceder; en él queda autorizado el empleo de referencias ambiguas sólo en las operaciones de lectura de datos. El <tipo> puede ser uno de los siguientes:

PRG – Archivo de programa.
SEQ – Archivo secuencial.
USR – Archivo de usuario.
REL – Archivo relativo.

Todos estos tipos de archivos se crean de la misma forma, diferenciándose únicamente en los comandos que facilitan el acceso a los mismos (recuérdese que incluso el directorio del disco era tratado como si de un archivo de programa se tratara).
La <dirección> debe ser una de las siguientes: "READ" (abreviadamente "R") o "WRITE" (abreviadamente "W"), según se quiera acceder al archivo para escribir o leer información en el mismo.

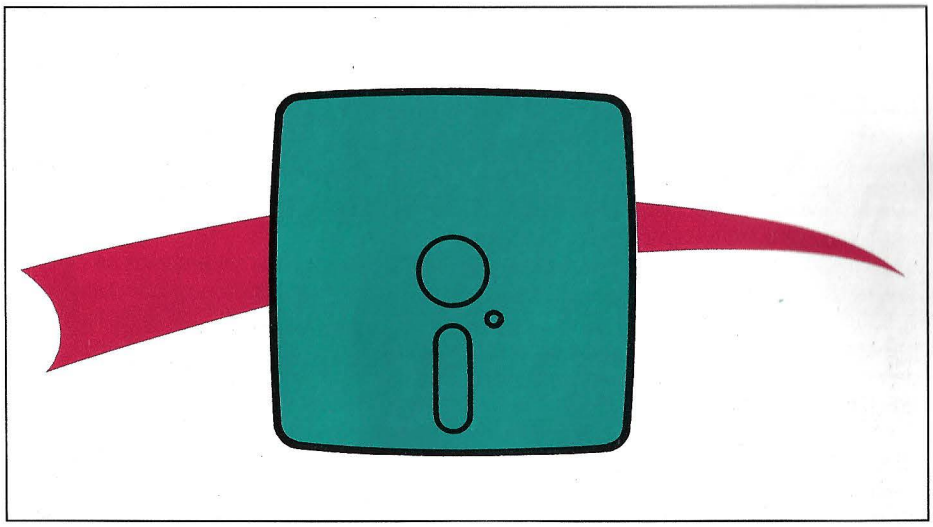


La orden PRINT# sirve de transporte para los comandos del disco, a través del canal número 15.

Una vez abierto el canal para la comunicación, hay que utilizar los comandos PRINT#<archivo> o INPUT#<archivo> según se quiera introducir datos en el fichero o leerlos del mismo para depositarlos en la memoria central del ordenador.
El funcionamiento de ambos comandos coincide con el tradicional de las órdenes PRINT e INPUT, hasta el punto de que se respeten los signos de puntuación que separan a las distintas variables que los acompañan. Así, un punto y coma (";") situará los datos uno a continuación del otro, mientras que si se separan por comas (",") se dejarán los espacios en blanco necesarios, entre variable y variable en el disquete, de igual forma que si se fueran a representar en la pantalla.
Como ejemplo ilustrativo del acceso a un archivo secuencial, el programa adjunto, denominado "ERROR DE DISCO", utiliza el comando INPUT# para obtener la información del canal de errores de la unidad de disco:
Otro comando utilizado para la lectura de datos de un archivo secuencial es el que adopta el formato:

GET#<archivo>

Su única diferencia con INPUT# consiste en que GET# lee los datos byte a byte (de carácter en carácter). Esta característica lo hace más útil cuando no se conoce con exactitud el contenido ni el formato de un archivo. Un ejemplo ilustrativo lo aporta el programa "LEER ARCHIVO" cuyo listado se adjunta y cuya funcionalidad es la de leer el contenido de cualquier archivo residente en disco.



PRINT# e INPUT# actúan para los archivos en disco de forma análoga a como lo hacen PRINT e INPUT para la pantalla y teclado respectivamente.

LOS ARCHIVOS ALEATORIOS

La unidad de disco de Commodore admite la operación con dos tipos de archivos aleatorios: los así llamados en la nomenclatura Commodore, cuya principal aplicación se encuentra en el lenguaje máquina, y los archivos relativos, más adecuados para manejar información de cualquier tipo. Como ya se ha mencionado, el disquete es dividido por el DOS en pistas y sectores, disponiendo cada sector de un total de 256 bytes libres para el almacenamiento de datos.
En el caso de los archivos aleatorios es posible acceder directamente a cada uno de los bloques del disco, mediante comandos similares a los ya descritos. Estos se envían a través del canal 15 de comandos, apoyándose en las órdenes "OPEN" y "PRINT#". Por lo demás, es necesario ahora abrir un segundo canal para el trasvase de datos, lo que se logra con la siguiente orden:

OPEN <archivo>, <dispositivo>,
  <canal>, "#"

Todos los parámetros son ya conocidos, excepto el referenciado por el símbolo "#" que indica el buffer de datos a utilizar para el archivo aleatorio. Este símbolo puede ir seguido por un número, por ejemplo el #2, lo que señala que se quiere utilizar el buffer número 2.
El comando que permite el trasvase de un determinado bloque del disco al canal de datos abierto es BLOCK-READ, el cual adopta el siguiente formato:

PRINT#15, "BLOCK-READ:"<canal>;
  <dispositivo>; <pista>; <bloque>

Junto a los parámetros ya conocidos, hay que especificar también la pista y el bloque exacto que se desea leer. Hecho esto, la información se puede obtener ya recurriendo a las órdenes "INPUT#" o "GET#", lo que es más habitual.
El comando opuesto al descrito es BLOCK-WRITE, cuyo formato es idéntico al anterior. Su ejecución debe ser poste-

rior al envío de los datos por el canal mediante la orden "PRINT#".
Otros dos comandos opuestos son BLOCK-ALLOCATE y BLOCK-FREE. El primero se encarga de reflejar en el BAM que el bloque indicado está ocupado por datos y no está disponible a partir de ese momento. El segundo realiza la operación contraria, es decir, libera un bloque cuyo contenido ya no se desea conservar. Por ejemplo:

PRINT#15, "B–A : "0; PISTA; BLOQUE

Si se intenta utilizar un bloque ya ocupado, se generará un mensaje de error que debe ser leído del canal de errores. Junto al mensaje de error, 65: NOBLOCK (bloque ocupado), se indicará al usuario el siguiente bloque libre del disco.
El inconveniente que presenta este tipo de archivos es que resulta muy difícil seguir la pista de los bloques que se van utilizando. Para evitar dicho problema se suele crear paralelamente un archivo de tipo secuencial cuyo contenido coincide con la lista de las pistas y bloques usados.
El buffer que se utiliza para el trasvase de datos posee un puntero que señala al último byte de datos escrito o al siguiente que se va a leer. Es posible hacer que este puntero señale a la deseada porción de datos de un bloque mediante el comando BUFFER-POINTER, cuyo formato es el siguiente:

PRINT#15, "BUFFER-POINTER:
"<canal>, <posición>

Su ejecución permite dividir cada bloque en registros independientes y éstos, a su vez, en campos que proporcionan estruc-

COMANDOS DEL DISCO

Comando	Formulación en el argumento de OPEN o PRINT#	Función
NEW	"NEW:<nombre>, <identificador>"	Formatea e inicializa un disco
COPY	"COPY:<nom. nuevo>=<nom. antiguo>"	Copia un archivo en disco
RENAME	"RENAME:<nom. nuevo>=<nom. antiguo>"	Cambia de nombre a un archivo
SCRATCH	"SCRATCH:<nombre>"	Borra un archivo del disco
INITIALIZE	"INITIALIZE"	Inicializa la unidad de disco
VALIDATE	"VALIDATE"	Reorganiza los archivos en disco
OPEN	"O:<nombre>, <tipo>, <dirección>"	Abre un fichero secuencial
OPEN	"#" o "#<número de buffer>"	Abre un canal de datos para acceso aleatorio
BLOCK-READ	"BLOCK-READ:" <canal>; <disp.>; <pista>; <bloque>	Traslada un bloque de datos del disco al canal
BLOCK-WRITE	"BLOCK-WRITE:"<canal>; <disp.>; <pista>; <bloque>	Opuesto al anterior
BLOCK-ALLOCATE	"BLOCK-ALLOCATE:"<disp.>; <pista>; <bloque>	Señala en el BAM el bloque utilizado
BLOCK-FREE	"BLOCK-FREE:" <disp.>; <pista>; <bloque>	Libera en el BAM el bloque desechado
BUFFER-POINTER	"BUFFER-POINTER:"<canal>; <posición>	Posiciona el puntero del buffer
OPEN	"<nombre>,L,"+CHR$(longitud registro)	Crea un archivo relativo
PRINT#	"P"CHR$(<canal>+96)CHR$(LSB) CHR$(MSB)CHR$(<posición>)	Posiciona el puntero de un archivo relativo

El puntero del BUFFER de un archivo aleatorio se puede situar en cualquier punto de éste mediante el comando "BUFFER-POINTER".

turas de almacenamiento más efectivas. Y todo ello sin más que asignar una determinada <posición> dentro del bloque a cada uno de estos campos y registros.

LOS ARCHIVOS RELATIVOS

Este nuevo tipo de archivos se caracteriza por permitir la estructuración de los datos en registros, y éstos a su vez en campos, lo que potencia su utilidad a la hora de manejar datos en régimen de acceso directo.
Para crear un archivo relativo se utiliza el siguiente formato:

```
OPEN <archivo>, <dispositivo>,
  <canal>, "<nombre$>, L, "+CHR$
  (<longitud del registro>)
```

El DOS establecerá una serie de punteros que señalarán a los sectores adyacentes a cada registro. De esta forma, cada sector puede apuntar hasta 720 registros, que pueden incluir hasta 254 caracteres por registro, lo que convertiría al disquete completo en un único archivo relativo.
Una vez creado un archivo relativo, su apertura para el acceso resulta ya más simple, puesto que no hay que especificar más que su nombre, junto a los restantes parámetros típicos del comando OPEN. Para acceder a cualquier registro del archivo abierto hay que empezar posicionando el puntero del archivo en el registro y posición deseada, lo que se hace con la siguiente orden:

```
PRINT#<archivo>, "P" CHR$
  (<canal1>+96) CHR$(<registro LSB>)
  CHR$(<registro MSB>)
  CHR$(<posición>)
```

En ella, <registro LSB> corresponde al byte menos significativo del número de registro al que se quiere acceder y <registro MSB> al byte más significativo. La necesidad de dos bytes para este dato deriva de que con un byte sólo se pueden representar los números del 0 al 255, y no hay que olvidar que un archivo en la unidad de disco 1541 de Commodore se puede contener más de 700 registros.
Para calcular el valor de estos dos bytes puede procederse como sigue. Por ejemplo, si el registro al que se quiere acceder es el número 520, el MSB será MSB=INT (520/256) y el LSB será entonces: LSB=520−MSB*256. Al evaluar ambas expresiones se obtienen los siguientes valores: MSB=2 y LSB=8.
El parámetro <posición> se refiere, naturalmente, a la posición dentro de cada registro. Su colaboración permite apuntar a un campo determinado dentro del registro.
El acceso al fichero relativo se hará, por lo tanto, en función del número del registro deseado. Si se quiere acceder por medio de una clave o código a cada registro, habrá que construir paralelamente un archivo secuencial que relacione cada clave con el número de registro correspondiente.

```
10 REM ERROR DE DISCO
20 OPEN15, 8, 15
30 INPUT#15, NUM, MENSA$, PISTA, SECTOR
40 PRINT CHR$(147)
45 IF NUM=0 THEN 60
50 PRINT "Error-"; NUM;
60 PRINT"   "; MENSA$
70 PRINT "En pista: "; PISTA; " sector: "; SECTOR
80 CLOSE15 : END
```

```
 10 REM LEER ARCHIVO
 20 INPUT "Archivo "; A$
 30 INPUT "Tipo "; T$
 40 OPEN15, 8, 15 : OPEN 2, 8, 2, "0 : "+A$+", "+T$+", R"
 50 INPUT#15, NUM, MENSA$, PISTA, SECTOR
 60 IF NUM>0 THEN PRINT NUM, MENSA$, PISTA, SECTOR : END
 70 GET#2, N$
 80 IF ST=0 THEN 100
 90 CLOSE15, 2 : END
100 PRINTASC(N$);
110 GOTO 50
```

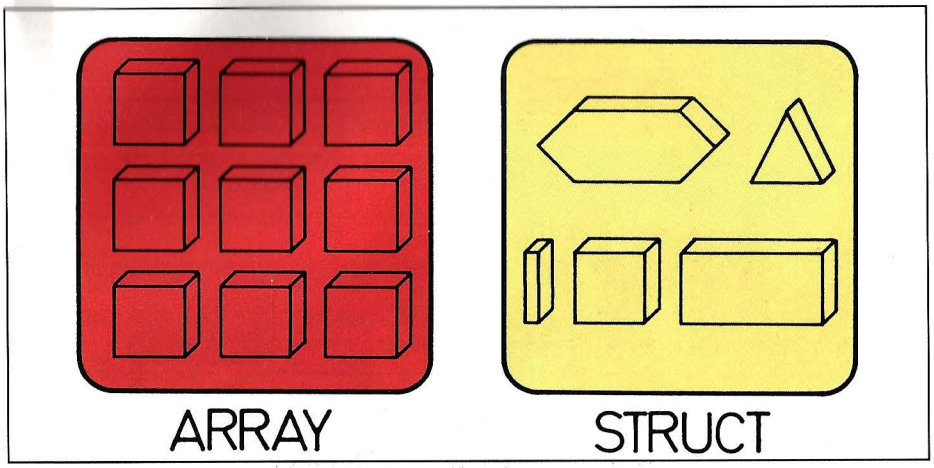
El lenguaje C (6)

Estructuras de datos: los registros

El tema central de este capítulo lo constituye la segunda y última estructura de datos en C. Ya se avanzó en el capítulo precedente que había pocas de ellas; realmente, no cabe afirmar que son una multitud.

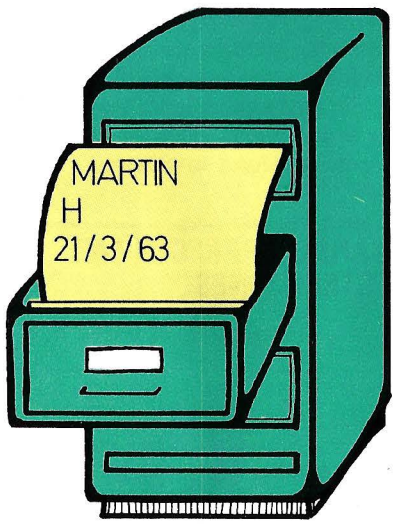
Para entrar en el tema puede repasarse mentalmente los RECORDS del Pascal. En el C, la palabra clave que introduce a un registro es "struct" y la forma de manipularlos es muy semejante en ambos lenguajes, aunque con limitaciones como veremos.

UTILIDAD Y MANEJO DE REGISTROS



La diferencia entre arrays y "struct" la establecen los tipos de elementos que uno y otro pueden contener. Mientras que el array sólo puede incluir datos de un mismo tipo, las "struct" carecen de esta restricción.

A menudo nos vemos en la necesidad de agrupar datos de diversa naturaleza bajo un nombre común; en estos casos la estructura array se nos queda pequeña, ya que con ella sólo es posible agrupar datos de iguales características (ver figura).




```
#define MAXLONG 20

struct fecha
      int dia;
      char mes[4];
      int año;
} ;

struct empleado {
      char nombre[MAXLONG];
      char sexo;
      struct fecha nacimiento;
} ;
```

Las "struct" son anidables tal como ilustra el ejemplo adjunto, cuyo contenido coincide con las primeras líneas de un programa para gestionar la nómina de una empresa.

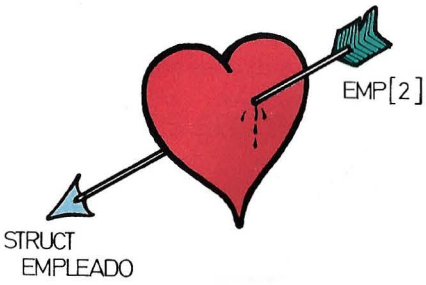
Pensemos en un programa que necesite seguir la pista de distintos hechos ocurridos en distintas fechas. Para almacenar las fechas puede crearse un "struct" (registro o estructura) de la siguiente forma:

```
struct fecha {
    int dia;
    char mes [4];
    int año;
} ;
```

El equivalente para los conocedores del Pascal sería:

```
TYPE fecha = RECORD
                dia: integer;
                mes: array [1..3] of
                  char;
                año: integer
             END;
```

y cuidado con la dimensión de "mes": recuerde el carácter nulo que se introduce al final de un array para saber cuándo termina. En el lenguaje C es necesario contar con él en todo momento.



Empleado	Sexo	Sueldo
Angel	h	20000
Pepita	m	20000

```
struct empleado {
        char nombre[20];
        char sexo;
        int sueldo;
} ;

main( )
{
        int i;
        struct empleado emp[2];

        emp[0].nombre="Angel";          emp[1].nombre="Pepita";
        emp[0].sexo=`h`;                emp[1].sexo=`m`;
        emp[0].sueldo=20000;            emp[1].sueldo=20000;
        printf(" Empleado          Sexo        Sueldo\n");
        for(i=0;i < =1;++i)
            printf(" %s                %c          %d\n",emp[i].nombre,
                                    emp[i].sexo,emp[i].sueldo);
        }
```

Para el tratamiento de la nómina pueden almacenarse datos sobre los empleados (suponemos que nuestra empresa tiene tan sólo dos) en un array, siendo cada elemento del mismo una "struct" con los datos de su ficha.

En el ejemplo se declara una estructura de nombre "fecha" y que contendrá tres elementos: dos de tipo entero y otro que es un array de caracteres.
Es importante reparar en que por el momento no se tiene aún ninguna variable de este tipo. Sólo se ha indicado al compilador que se prepare para lo que puede venir, al igual que ocurre en Pascal con la sección TYPE.
Las declaraciones de estructuras suelen ir al principio del programa, al igual que los "#define".
Para declarar una variable que sea una estructura del tipo "fecha" se hará lo siguiente:

```
struct fecha nacimiento;
```

Ahora sí disponemos ya de la variable "nacimiento" que tiene la estructura comentada.
La forma de acceder a los elementos de esta variable es análoga a como se hace en Pascal:

```
nacimiento.dia = 21;
nacimiento.mes = "mar";
nacimiento.año = 1963;
```

Ahora se tiene en "nacimiento" la información sobre una fecha determinada: el 21 de marzo de 1963.

Al igual que con los arrays, se pueden inicializar "struct" directamente, siempre que estas sean externas o estáticas, nunca automáticas. Suponiendo que estamos en alguno de estos supuestos, la variable "nacimiento" podrá ser inicializada de la siguiente forma:

```
struct fecha nacimiento=
  {21,"mar",1963};
```

Las "struct" se pueden anidar entre ellas. Suponga el clásico ejemplo de la nómina de empleados de una empresa. Las primeras líneas de un programa en C para gestionar esta nómina podrían tomar un aspecto semejante al que aparece en la figura adjunta. Si se define una variable "emp", que designará a un empleado concreto, como sigue:

```
struct empleado emp;
```

se accederá al mes de su nacimiento de la siguiente forma:

```
printf("%s  n",emp.nacimiento.mes);
```

Por otra parte, las variables "struct" pueden ser declaradas en el momento de hacer la propia declaración de la "struct":

```
struct fecha  {
    int dia;              /* Se declaran "nacimiento" */
    char mes [4];         /* y  "muerte" del tipo  */
    in año;               /* "fecha"               */
    }    nacimiento,muerte;
```

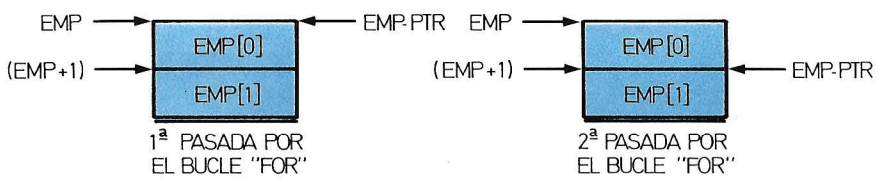
Debido a la existencia de esta segunda forma de declaración, es muy importante no olvidar el ";" que sigue al último "}" de la declaración de la "struct".
Son numerosos los casos en los que un "matrimonio" entre arrays y "struct" resulta muy conveniente. En la figura adjunta se tiene un ejemplo de este tipo de unión.
Hasta aquí todo habrá resultado sencillo para los que ya conozcan el lenguaje Pascal. Sin embargo, pronto aparecerán nuevas emociones... Demos de nuevo la bienvenida a los punteros.

CUANDO APARECEN LAS RESTRICCIONES

En las versiones actuales de los compiladores de C existen dos restricciones que

están en vías de ser eliminadas en las próximas versiones del lenguaje. La primera restricción es que no se pueden manejar las "struct" como variables normales, en el sentido de que, teniendo dos estructuras "a" y "b" del mismo tipo, no es posible hacer "a=b", sino que habría que asignarlas campo a campo. Además —y aquí está la segunda restricción— tampoco se puede pasar una estructura completa como argumento de una función, aunque sí se pueden pasar campos individuales.

Por el momento, nuestra mejor arma está en los punteros: de igual forma que podíamos acceder a los elementos de un arrray considerando que el propio nombre del array era un puntero al elemento cero del mismo, podemos obtener la dirección de




```
struct empleado {
      char nombre[20];
      char sexo;
      int sueldo;
   } ;

main( )
  {
      struct empleado emp[2];
      struct empleado *emp_ptr;

      emp[0].nombre="Angel";              emp[1].nombre="Pepita";
      emp[0].sexo='h';                    emp[1].sexo='m';
      emp[0].sueldo=20000;                emp[1].sueldo=20000;
      printf("  Empleado          Sexo          Sueldo\n");
      for(emp_ptr=emp;emp_ptr<=emp + 1;++emp_ptr)
         printf("  %s          %c          %d\n",
               emp_ptr->nombre,emp_ptr->sexo,emp_ptr->sueldo);
      }
```

Nueva versión del programa de nóminas.

comienzo de nuestra estructura, y pasar esta dirección como parámetro a las funciones que lleven a cabo la tarea requerida.

Para afianzar todo esto nos remitimos al programa "Nueva revisión del programa de nóminas" que, básicamente, es una reforma del programa precedente.

Ambos entregan la misma salida, aunque el modo de operar es distinto. Observe la declaración:

```
struct empleado *emp_ptr;
```

Con ella declaramos a "emp_ptr" como un puntero hacia la estructura "empleado", de igual forma que en otras ocasiones hemos declarado punteros a enteros, caracteres o números reales (float).

Preste atención ahora al bucle "for". Este empieza inicializando el puntero "emp_ptr" a "emp"; esto es, se hace que "emp_ptr" señale al elemento cero del array "emp". Con este estado de cosas se entra en el cuerpo del bucle, que escribirá los campos de la estructura "emp[0]", los cuales han sido referenciados como sigue:

```
emp_ptr -> nombre
emp_ptr -> sexo
emp_ptr -> sueldo
```

en lugar de serlo a través del típico " .". Ello se debe a que ahora estamos utilizando un puntero para acceder a una estructura en vez de emplear su nombre, en cuyo caso utilizaríamos el " .". El símbolo "->" está compuesto por los símbolos de resta y de "mayor que" y se trata como una unidad.

En otras palabras, como en la primera pasada, "emp_ptr" está apuntando a "emp[0], las expresiones:

```
emp_ptr -> nombre      y
emp[0].nombre
```

son equivalentes.

En la siguiente pasada, "emp_ptr" es incrementado en una unidad, es decir, lo suficiente como para que señale hacia

La unión hace la fuerza

En C existe un tipo de variable llamado "union" con el cual se pueden definir variables que, en distintos momentos, pueden contener por ejemplo, un entero, un carácter o un real, aun teniendo el mismo nombre. La "union" son muy parecidas a las "struct" en cuanto a sintaxis, manejo y restricciones. Por ejemplo:

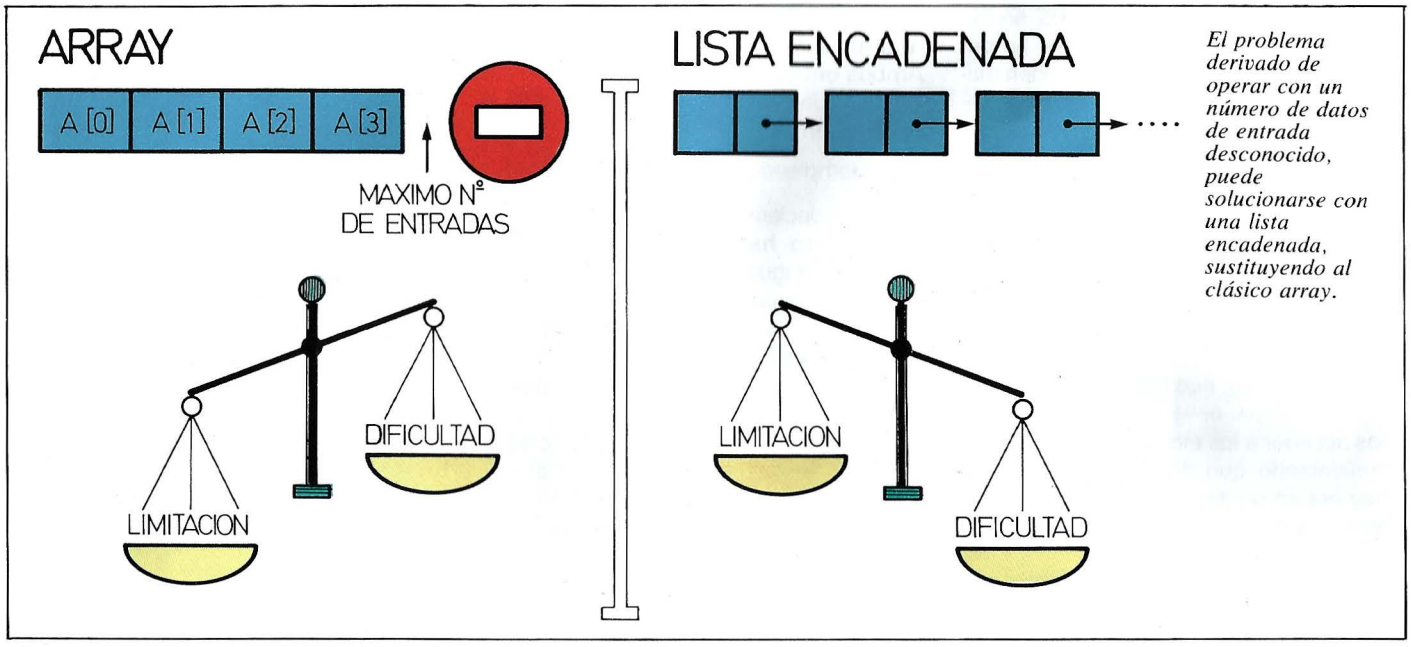
```
union ejemplo {
      int entero;
      float real;
} desc;
```

declara a "desc" como capaz de contener un real o un entero; ello equivale a tener en un mismo programa las declaraciones:

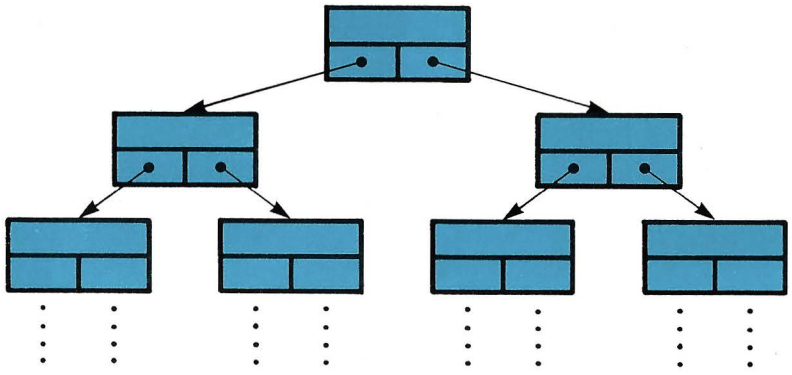
```
int desc;
float desc;
```

Cualquiera de estos dos tipos (que sabemos que ocupan distinta cantidad de memoria) podrá ser asignado y utilizado en expresiones con "desc".

Dedicar unas líneas tan sólo a la "union" en un curso introductorio está fuera de lugar, pero el objetivo de hacerlo no es otro que resaltar de nuevo lo cerca que estamos del hardware al programa en lenguaje C, aun disponiendo de estructuras de control avanzadas como "while", "while-do", "for", etc.



El problema derivado de operar con un número de datos de entrada desconocido, puede solucionarse con una lista encadenada, sustituyendo al clásico array.



Un árbol binario tiene importantes aplicaciones en algunos problemas de ordenación de elementos. También existen situaciones en las que una estructura de este tipo resulta ideal para representar los datos del problema.

"emp[1]" y el proceso se repite por última vez.

ESTRUCTURAS AUTORREFERENCIADAS

Este nombre tan poco expresivo se refiere a aquellas estructuras que sirven para crear y manipular las "listas encadenadas» y los "árboles binarios".
Estos dos "entes" informáticos (por darles un nombre) resuelven de manera elegante y sencilla problemas que de otra forma serían muy difíciles de tratar. Hay ocasiones en las que la cantidad de datos de entrada para un determinado programa no se conoce de antemano. Si no disponemos de facilidades para crear listas encadenadas tendremos que recurrir a dimensionar un array, el cual en unas ocasiones se nos quedará pequeño y en otras nos sobrará espacio, desperdiciando memoria. Con una lista encadenada se ocupará sólo el espacio necesario en cada momento, pagando el precio de una mayor complejidad en el tratamiento de la información (ver figura).
La gestión de listas encadenadas en C es bastante más compleja que en Pascal, por lo que no la trataremos aquí. Simplemente mencionaremos que una lista encadenada se construye en torno a un "struct" como el que sigue:

```
struct elemento {
        int valor;
        struct elemento *siguiente;
} ;
```

en donde "elemento.valor" será el dato almacenado y "elemento.siguiente" un puntero que servirá de enlace a la siguiente estructura.
Los árboles binarios se pueden construir en torno a una estructura como la siguiente:

```
struct nodo {
    char palabra[10];
    struct nodo *derecha;
    struct nodo *izquierda;
} ;
```

Como se observa en la figura, un árbol binario está compuesto por nodos como el de arriba, enlazados por punteros a otros nodos.

Apple Macintosh (y 4)

MacPaint o la sencillez del dibujo por ordenador

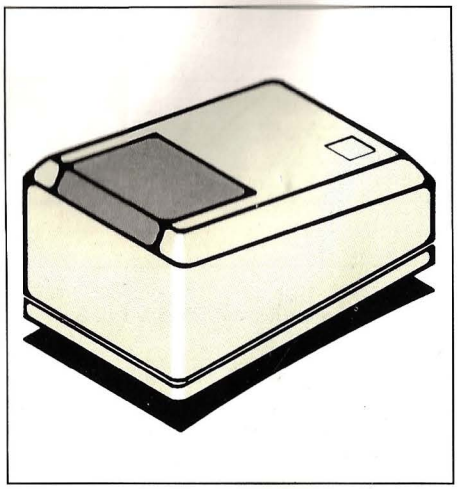
Desde los primeros días de la historia informática, en los cuales las habilidades gráficas del ordenador no pasaban más allá de los listados por impresora, obtenidos a golpe de caracteres, se ha recorrido un gran trecho en las representaciones gráficas por ordenador.

Las aplicaciones gráficas han tenido un desarrollo más lento que otras facetas del dominio informático (gestión, cálculo, etc.), debido a los propios requerimientos hardware (gran capacidad de almacenamiento y microprocesadores con alta velocidad de ejecución de instrucciones) no disponibles a unos costes razonables hasta hace bien pocos años. También ha influido en este retraso la orientación original de la informática hacia el campo de las ciencias matemáticas, factor que convirtió al ordenador en un "devorador" de números y ecuaciones.

El diseño asistido por el ordenador, CAD (Computer Aid Design), se ha puesto de moda en los últimos años, abarcando campos tan dispares como el cine, la automoción, arquitectura, aeronáutica... Descendiendo incluso hasta el campo de los microordenadores, dentro del cual no existe prácticamente ningún representante que no posea unas capacidades gráficas más que apreciables.

Dentro del citado campo destaca la aplicación gráfica MacPaint, destinada al ordenador Macintosh de la firma Apple Computer Inc, capaz de realizar gráficos espectaculares mediante una interface con el usuario clara y sencilla. Sus facultades permiten que el usuario no experto sea capaz de editar documentos gráficos, sin más que seleccionar iconos representativos de la acción a ejecutar o elegir comandos de los menús disponibles.

Como es lógico MacPaint aprovecha las utilidades del Macintosh, siendo su uso muy parecido al de otras aplicaciones desarrolladas para este microordenador. El ratón ejerce un papel muy relevante, minimizando la importancia del teclado cuya función deja de ser tan primordial como en otros equipos, para pasar a convertirse en un instrumento de ayuda para la introducción de textos y para la ejecución rápida de ciertos tipos de comandos.

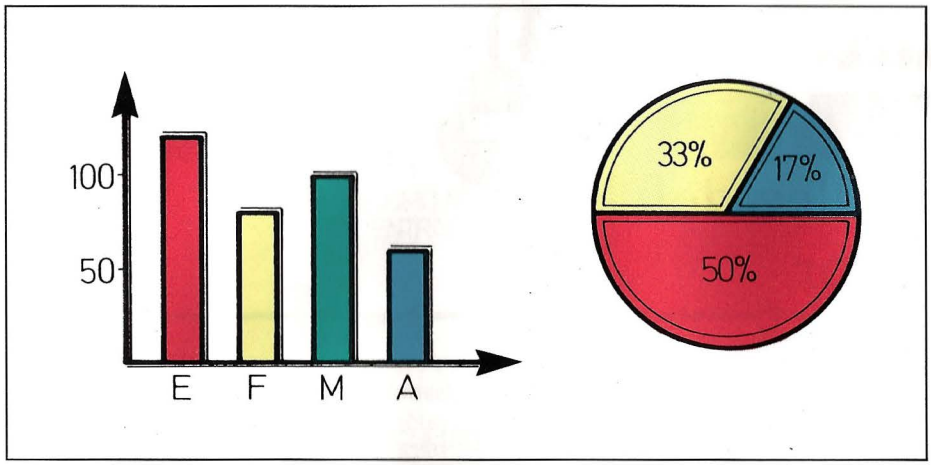


En la aplicación gráfica MacPaint, el ratón sustituye con ventaja al pincel en múltiples aspectos.

ACCESO AL MACPAINT

La activación del programa MacPaint se produce de la forma habitual en el Macin-

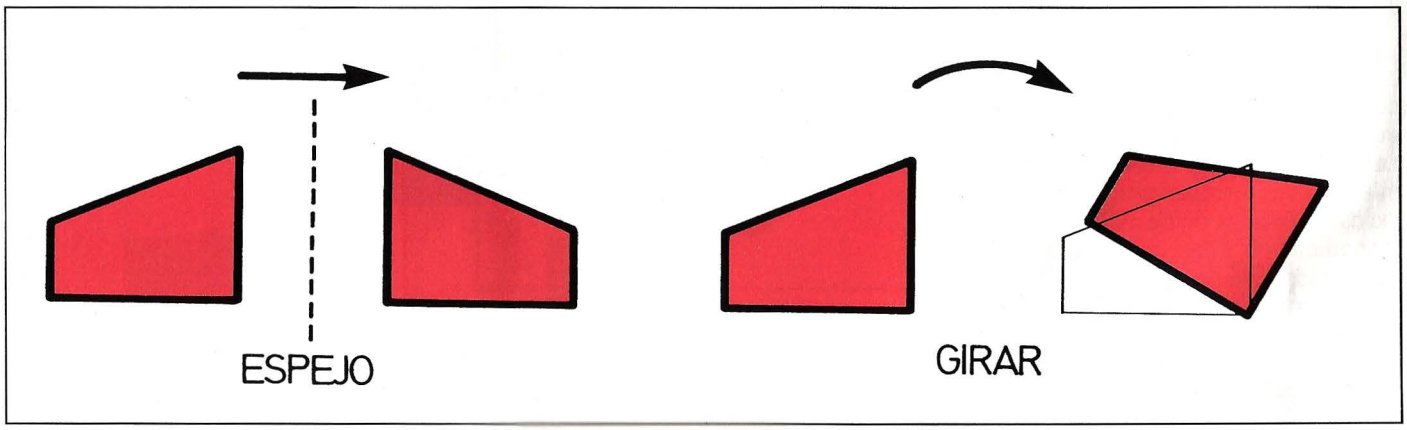
El Apple Macintosh, completado por las aplicaciones básicas MacWrite y PacPaint (entregadas por el fabricante con el equipo base), constituyen una herramienta de gran eficacia y facilidad de uso para el usuario final.



Los clásicos diagramas de gestión también son fácilmente generables con la ayuda de MacPaint. Es posible incluso crear documentos mezclando gráficos y texto en cualquier medida.

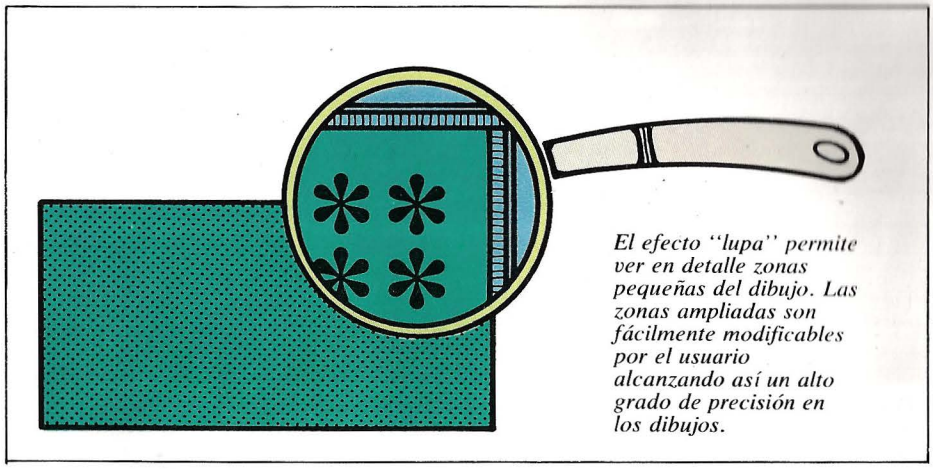
Paint, junto con la repetición del comando "Abrir", muestra la ventana de trabajo.
Con la ventana abierta se accede a la unidad de trabajo del MacPaint: el documento. Un documento equivale a una hoja de papel de 20 por 25 centímetros, pudiéndose acomodar dentro de estos límites zonas gráficas y zonas de texto sin ningún tipo de restricción en la mezcla de unas con otras.
La ventana está dividida en varios dominios:

— Lista de herramientas disponibles para dibujar, añadir texto, recorrer el documento y seleccionar elementos del documento. Esta ocupa una tabla de dos columnas localizada en el margen izquierdo de la pantalla.



La actuación sobre la zona deseada del dibujo se realiza una vez que ésta ha sido seleccionada. Al efecto, la aplicación cuenta con una gran variedad de utilidades para ayudar a la transformación de gráficos.

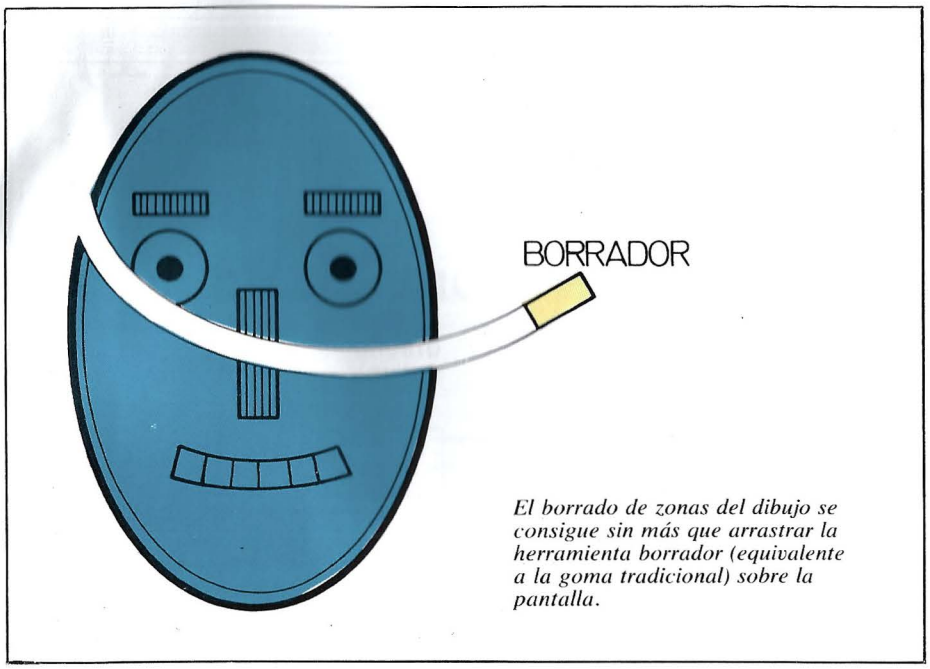
tosh: insertando en la unidad al efecto el disco que contiene el programa y eligiendo el comando "Abrir" del menú Archivo (o con una doble pulsación sobre el ratón). De esta forma se accede a todas las aplicaciones contenidas en el disquete. La selección, con la ayuda del ratón, del icono representativo del Mac-



El efecto "lupa" permite ver en detalle zonas pequeñas del dibujo. Las zonas ampliadas son fácilmente modificables por el usuario alcanzando así un alto grado de precisión en los dibujos.

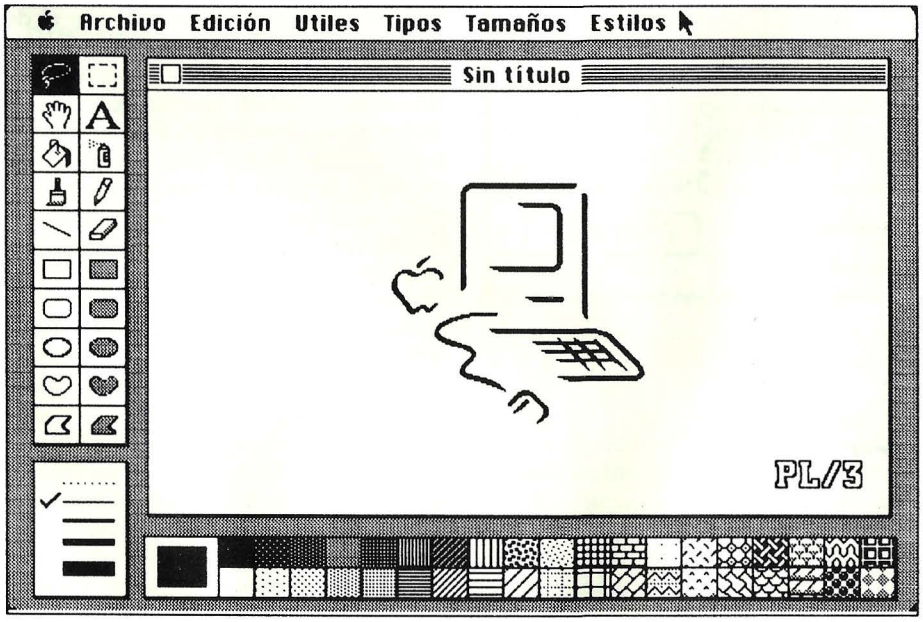
— Anchos de línea y borde disponible para la confección de dibujos, localizados en la esquina inferior izquierda de la ventana.
— La banda inferior de la ventana está ocupada por los diseños o tramas utilizables para rellenar contornos cerrados.
— En la franja superior de la pantalla se localizan los menús "desplegables" (pull-down) sobre los que se elegirán los comandos; bajo ellos está la barra de título y el cuadro de cierre del documento.
— Finalmente, ocupando la mayor parte de la pantalla está la ventana de dibujo, la cual muestra un tercio del documento.
Desde el momento en el que la ventana del dibujo está activa, la confección de un gráfico sobre el documento seleccionado se limita a la activación de las herramientas adecuadas, y al empleo de los comandos contenidos en los distintos menús.

LAS HERRAMIENTAS DE TRABAJO



El borrado de zonas del dibujo se consigue sin más que arrastrar la herramienta borrador (equivalente a la goma tradicional) sobre la pantalla.

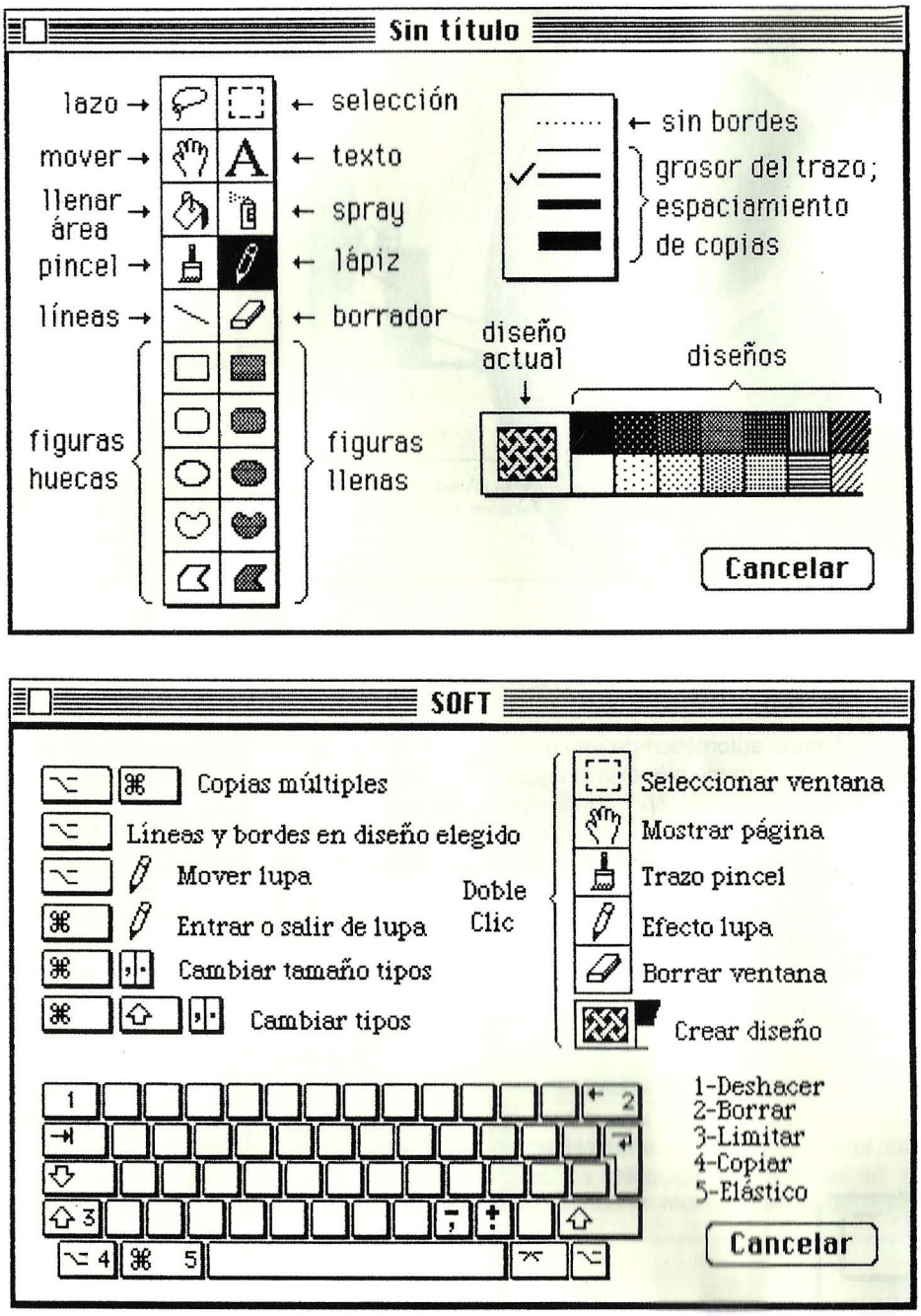
Al igual que un pintor usa su pincel, paleta, pinturas, etc., el usuario de MacPaint dispone de una colección de herramientas con las que plasmar su arte; ya sea en un dibujo artístico, un gráfico de gestión, o en las más variadas composiciones.
Las herramientas se seleccionan al llevar hasta ellas el puntero que gobierna el ratón, apretando y soltando rápidamente el botón de éste cuando el puntero señale la herramienta deseada. La activación de una herramienta queda reflejada al invertirse su color y el de fondo, resultando así fácil distinguir la herramienta en uso.
Los efectos de los diversos útiles disponibles son los siguientes:
— Selección de una zona del dibujo para actuar posteriormente sobre ella. De este modo se puede corregir o modificar parte del dibujo.
— Arrastre del dibujo fuera de la ventana para poder acceder a la totalidad del documento.
— Introducción de texto en cualquier zona del dibujo, disponiendo para ello de todas las posibilidades que ofrecen los menús de tipos, tamaños y estilos de letra.
— Rellenado automático de un contorno cerrado con la trama que elija el usuario.
— Pintado con "aerosol" (difuminado) con el patrón de trama seleccionado.
— Trazado de líneas a mano alzada con el tipo de trazo activo.
— Superposición sobre un fondo de una línea de color inverso a éste.
— Trazado de líneas rectas.
— Borrado parcial o total del contenido del documento.
— Dibujo de rectángulos, óvalos y polígonos huecos, o bien rellenos, con el diseño actual.
— Trazado de contornos sin forma geométrica definida, ya sean huecos o rellenos.
La utilización de esta surtida caja de herramientas posibilita la edición de documentos con un contenido gráfico que por otros métodos hubiera consumido una enorme cantidad de recursos, tanto humanos como de ordenador.



En la pantalla de trabajo del MacPaint se encuentran reunidas todas las herramientas, diseños de trama y menús de comandos necesarios para la creación de gráficos.

DEFINICION DEL ENTORNO

Las operaciones que llevan a cabo las herramientas se realizan normalmente sobre ciertos valores predefinidos del grosor del trazo de las líneas y bordes, así como del patrón de trama o tipo de relleno a aplicar a los contornos cerrados.
Los trazos de las líneas y bordes van desde un fino punteado hasta un grosor apreciable.
La utilización de distintas tramas permite crear elementos tan dispares como una tela floreada y una pared de ladrillos, sin



La propia ventana de dibujo de la aplicación MacPaint muestra la colección de herramientas y comandos que están a disposición del usuario.

más que rellenar la superficie deseada con el fondo elegido. Están disponibles un total de 38 diseños con los que es posible imitar la apariencia de casi cualquier tipo de superficie. En el caso improbable de no poseer el patrón idóneo, siempre cabe la posibilidad de editar un nuevo tipo de diseño en sustitución de uno de los existentes.

La selección de cualquier tipo de trazo o diseño de trama puede efectuarse en todo momento, escogiendo uno de los que aparecen en el recuadro inferior izquierdo o en la franja inferior de pantalla, respectivamente. Desde luego, cabe la posibilidad de mezclar en un mismo dibujo varios tipos de trazos y diseños de trama.

LOS MENUS DE LA APLICACION

No todas las acciones que se realizan sobre un documento son realizables con el surtido de herramientas, por lo que es necesario disponer de una serie de comandos agrupados en menús para llevar a cabo ciertas tareas. Los menús disponibles pueden clasificarse en cuatro grandes grupos:

— Menús que definen la forma y disposición del texto; entre éstos se encuentran los menús de tipos, tamaños y estilos, capaces de proporcionar a los textos una gran variedad de formas y tamaños.

— Menú de acceso a documentos. La apertura de documentos para su creación o modificación, así como su cierre y posterior salvaguarda en disco quedan cubiertos con el menú archivo.

— Menú de edición; sirve para modificar parte de un documento, disponiendo de las utilidades clásicas (cortar, copiar, pegar, borrar, etc.) así como de comandos destinados a girar y voltear las zonas seleccionadas previamente con las herramientas existentes al efecto.

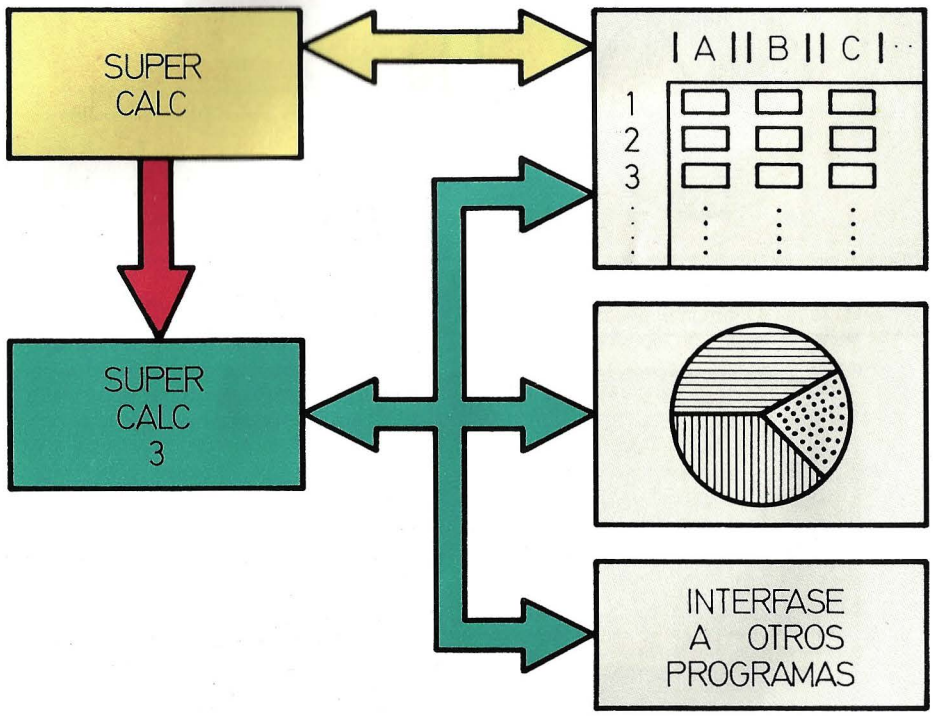
— Menú de útiles. Está compuesto por una serie de comandos destinados a facilitar el diseño y manejo de los gráficos y textos contenidos en un documento. Entre estos comandos cabe destacar el de acceso a una colección de posibles trazos para el pincel, más amplia que la comentada anteriormente; la obtención de imágenes especulares de un elemento de dibujo; efecto "Zoom" para ampliar zonas de dibujo; visualización de la página completa del documento; impresión, etc.

En resumen cabe mencionar que el sistema operativo del Apple Macintosh, al igual que sus aplicaciones, tiene una indudable ventaja: la disponibilidad de una interface con el usuario capaz de simplificar el proceso de comunicación del binomio hombre-máquina, con ello el usuario sólo debe concentrarse en hacer el trabajo, en lugar de en cómo hacerlo. Por otra parte, la propia filosofía del Macintosh hace que el microprocesador deba realizar un ingente trabajo para hacer más cómoda la actividad del usuario, resultando su operación algo más lenta que la característica de otros microordenadores basados en el mismo microprocesador.

SuperCalc 3 (1)

Algo más que una hoja electrónica

La empresa Californiana SORCIM Corporation ha desarrollado diversos paquetes de software horizontal, entre ellos cabe destacar la serie de hojas electrónicas denominadas SUPERCALC. Hasta principios de los años 80, SUPERCALC estuvo situada en puestos privilegiados dentro del "ranking competitivo" entre este tipo de programas. Pero en 1982 apareció el paquete LOTUS1-2-3 y se produjo un cambio en las reglas de competición entre hojas electrónicas: ya no sólo se les pedía que resolvieran problemas de lápiz, papel y calculadora, sino que debían ofrecer múltiples posibilidades adicionales: obtención de gráficos, elaboración de informes, etc. En contestación a este reto, SORCIM presentó poco después una nueva versión de su hoja electrónica, a la que denominó SUPELCALC 3.



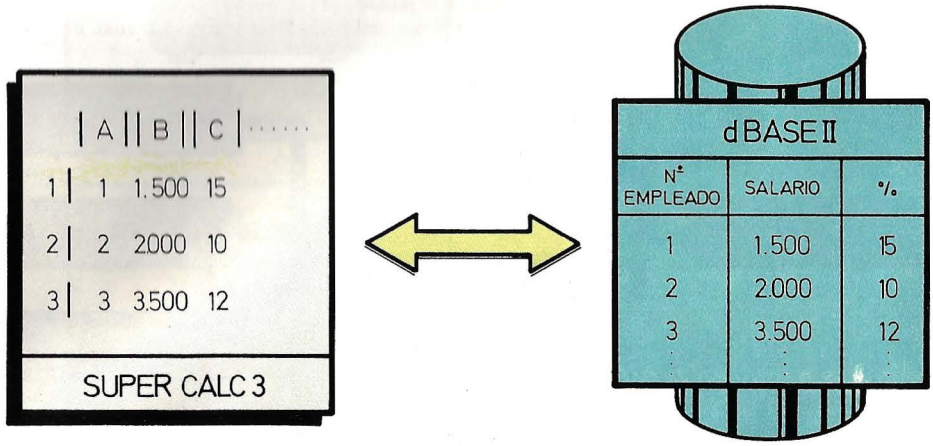
Supercalc es una hoja electrónica en el sentido tradicional; en cambio, Supercalc 3 llega a ofrecer opciones similares a las de algunos paquetes integrados.

¿QUE ES SUPERCALC 3?

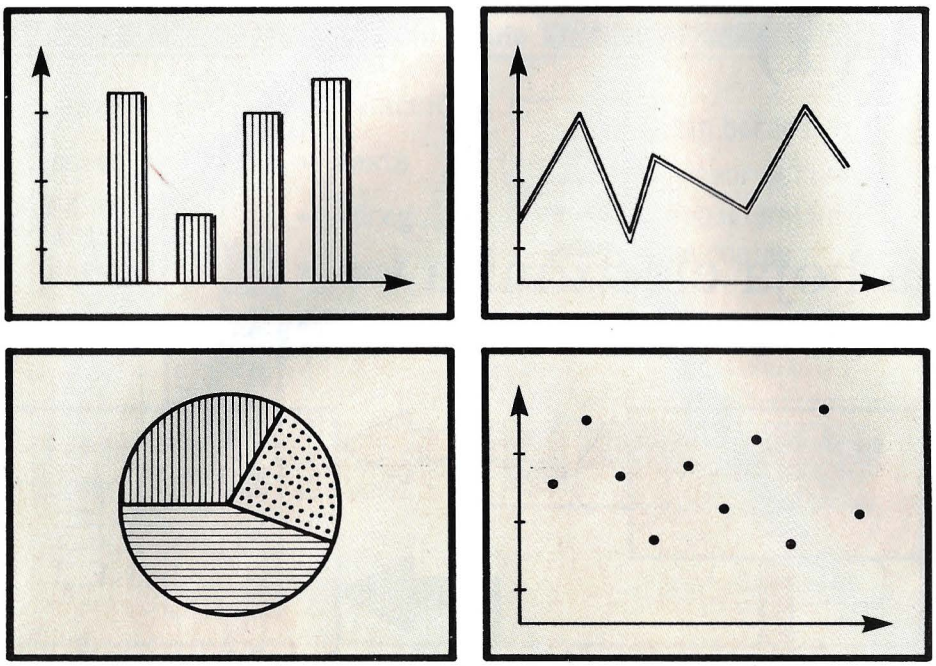
Aunque su fundamento esencial es la aplicación de hoja electrónica, SUPERCALC 3 puede ser definido como paquete de software integrado. En realidad está "a caballo" entre ambas categorías: desde luego es muy potente como hoja electrónica; en cambio, como paquete integrado se queda corto.

Independientemente de la categoría en la que se clasifique, SUPERCALC 3 tiene las siguientes cualidades:

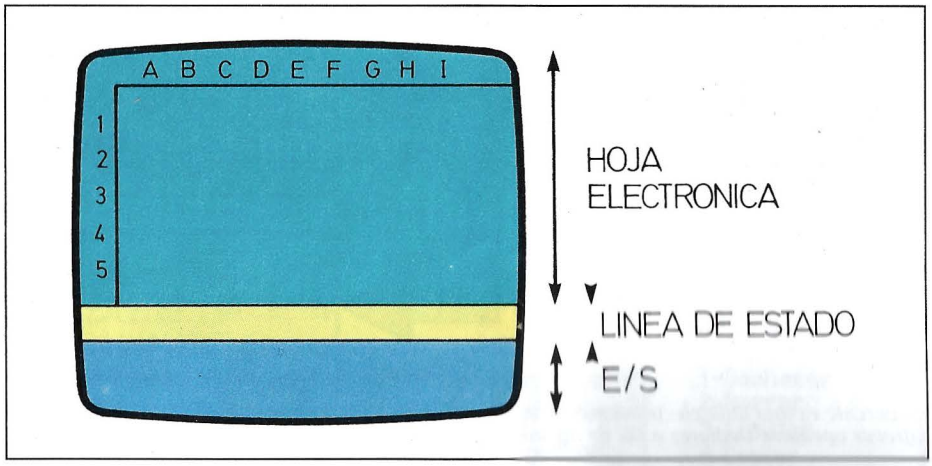
- Todas las funciones típicas de las hojas electrónicas: adaptación del formato de la matriz, funciones científicas y comerciales, de calendario y lógicas, capacidad de modelización, cálculo y recálculo, facilida-



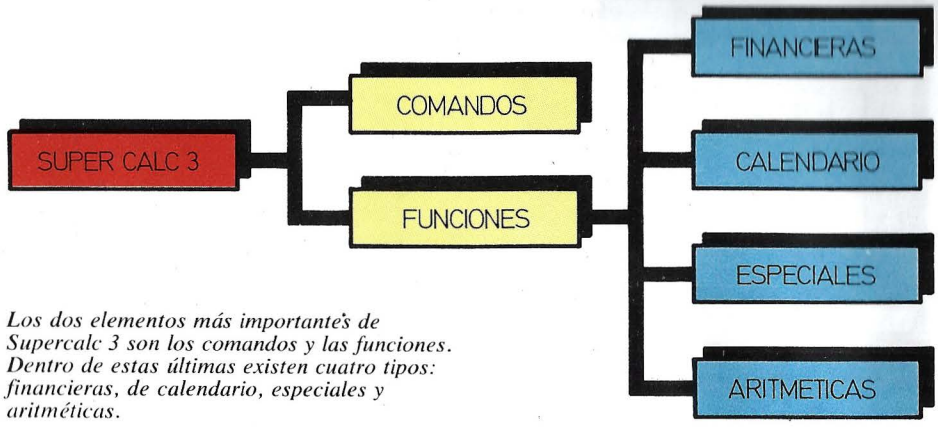
Una de las opciones más interesantes de Supercalc 3 es la posibilidad de relacionarse con otros programas, por ejemplo con la base de datos dBASE II.



Dentro de las opciones gráficas de Supercalc 3 se puede elegir entre varios diagramas distintos. Barras, líneas, tartas y puntos son los tipos de representaciones más utilizadas.



En una típica pantalla de trabajo con Supercalc 3 se pueden distinguir tres zonas perfectamente diferenciadas: la propia hoja electrónica, una línea de estado y una zona de entrada/salida.



Los dos elementos más importantes de Supercalc 3 son los comandos y las funciones. Dentro de estas últimas existen cuatro tipos: financieras, de calendario, especiales y aritméticas.

des para el archivo de la información en soportes externos, posibilidad de obtener resultados escritos, etc.

- Utilización de gráficos para la representación de los datos ubicados en la hoja electrónica. Al efecto se dispone de varios tipos de diagramas entre los que se puede elegir: de barras, de líneas, de tarta, de puntos...
- Posibilidad de incluir los informes producidos, tanto estándar como gráficos, en otros documentos.

Evidentemente, para que pudiera considerarse a SUPERCALC 3 como un auténtico paquete integrado, le faltaría aumentar un poco su capacidad de proceso de texto y, fundamentalmente, simular o incluir el funcionamiento de una base de datos. Aunque, para considerarlo como una simple hoja electrónica, le sobra su capacidad gráfica y su relativa aptitud para el proceso de textos.

FUNDAMENTOS DEL SUPERCALC 3

El sistema empleado por este programa para identificar a los distintos elementos de la matriz coincide con el tradicional: números (1, 2, 3, ..., 254) para las filas y letras (A, B, C, ..., BK) para las columnas.

Para "arrancar" el programa basta con teclear SC3 y pulsar RETURN, por supuesto, después de haber introducido el disquete que contenga al programa en la unidad correspondiente. De inmediato aparecerá una pantalla de presentación, ofreciendo la posibilidad de pedir ayuda al propio programa (HELP) o entrar directamente en la aplicación. Dentro de esta pantalla de presentación, SUPERCALC 3 también indica el significado de las teclas programadas.

Inicialmente, el elemento activo de la hoja electrónica es el A1; si bien, éste se puede desplazar mediante las teclas de movimiento o con el comando GO TO, cuya misión consiste en facilitar los desplazamientos dentro de la hoja electrónica.

En la pantalla del ordenador, estando activo SUPERCALC 3, se pueden apreciar las siguientes zonas:

1. Zona de hoja electrónica, que actuará como una ventana por la que se visualizará parte de la matriz.

2. Línea de estado en la que se visualizará el estado de la hoja electrónica, incluyendo, entre otras cosas, el elemento activo de la matriz.
3. Línea de mensajes que en general sirve para que SUPERCALC 3 se comunique con el usuario.
4. Línea de entrada, complementaria de la anterior ya que sirve para que el usuario se comunique con el programa.

CARACTERISTICAS TECNICAS DE SUPERCALC 3

Cada uno de los elementos de la hoja electrónica puede contener un literal, un dato numérico o una fórmula. Dado que el objeto de SUPERCALC 3 es la realización automática de cálculos, una característica muy importante reside en las distintas fórmulas que pueden ser utilizadas por el usuario. SUPERCALC 3 organiza sus funciones en cinco grupos distintos:

- FUNCIONES LOGICAS

Dispone de cuatro funciones distintas: IF,

Algunos usuarios de hojas electrónicas

Podemos afirmar, sin riesgo de exagerar, que prácticamente cualquier persona puede ser usuario de una hoja electrónica. Esta afirmación cabe deducirla sin más que considerar que una hoja electrónica es la alternativa moderna para la resolución de problemas de "lápiz, papel y calculadora". A continuación se detallan cinco colectivos en los que un programa de hoja electrónica puede constituir una eficaz herramienta cotidiana.

1. Ejecutivos

Cada vez es más usual que los miembros de la dirección de una empresa dispongan de un ordenador para su uso exclusivo. En este caso sin duda nunca faltará entre los programas que manejen una hoja electrónica.

Los problemas que el ejecutivo tratará personalmente son de tal índole que resultaría imposible diseñar programas tradicionales para su resolución. Por otro lado, los datos manejados suelen ser confidenciales y susceptibles de múltiples modificaciones, tanto en su estructura como en su contenido. Si unimos ambas premisas a la rapidez con la que el ejecutivo necesita las respuestas para la toma de decisiones, la solución lleva a pensar en la idoneidad de una hoja electrónica.

2. Economistas

Los estudios financieros tratados normalmente por los economistas suelen corresponderse, en la mayoría de los casos, con modelos teóricos que simulan el comportamiento de una empresa, del mercado... Mediante la utilización de una hoja electrónica se puede diseñar un modelo oportuno, modificándolo sencillamente cuando se considere necesario.

Normalmente, la fuente de datos que alimenta a estas aplicaciones suele ser la contabilidad financiera o analítica. En el caso de que éstas se encuentren mecanizadas, será posible realizar cómodamente el traspaso de información.

3. Matemáticos e Ingenieros

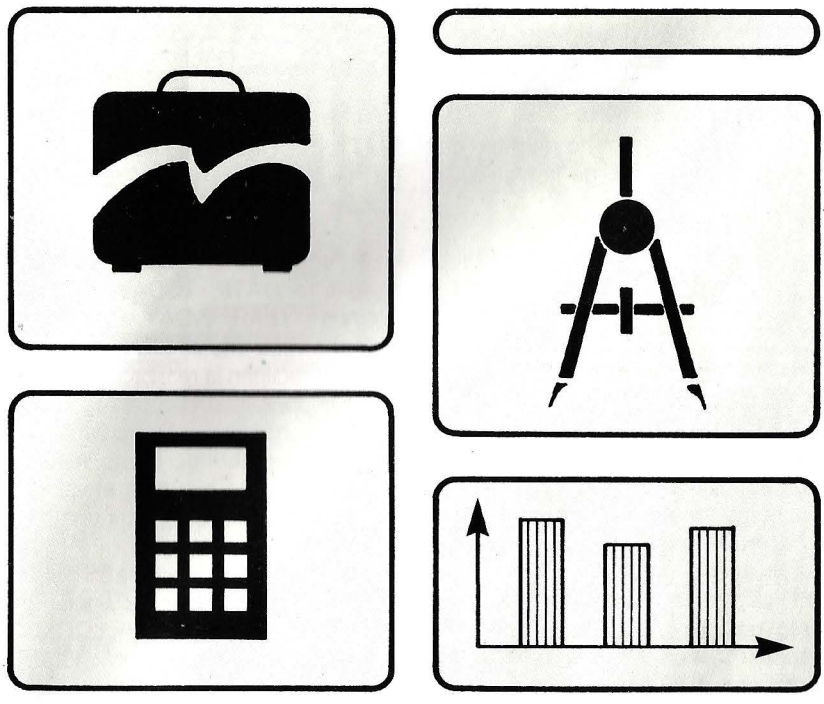
El elemento más comúnmente utilizado tanto por matemáticos como por ingenieros son fórmulas, y ya hemos visto repetidamente que una de las principales características de las hojas electrónicas consiste en que son capaces de recalcular algunos elementos (fórmulas) en función de otros. Por lo tanto, cuando la complejidad de los algoritmos no sea excesiva, una hoja electrónica puede ser una poderosa herramienta de trabajo para este tipo de usuarios.

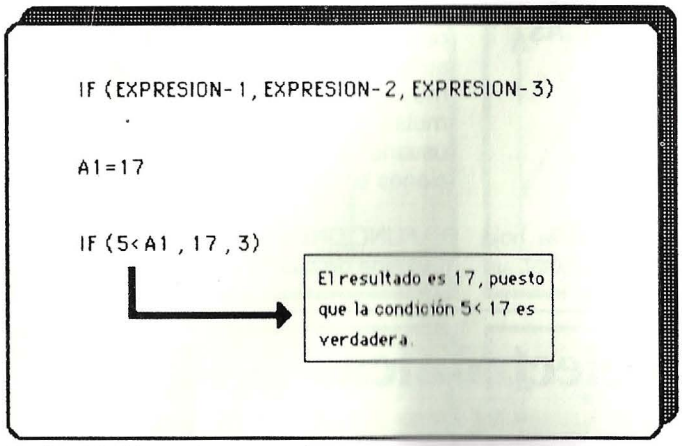
4. Profesionales liberales

Un médico, un abogado, un corredor de bolsa y, en definitiva, prácticamente cualquier profesional no ligado a las tradicionales estructuras de una empresa puede encontrar en una hoja electrónica el apoyo que le aportarían los distintos departamentos de servicio de una empresa.

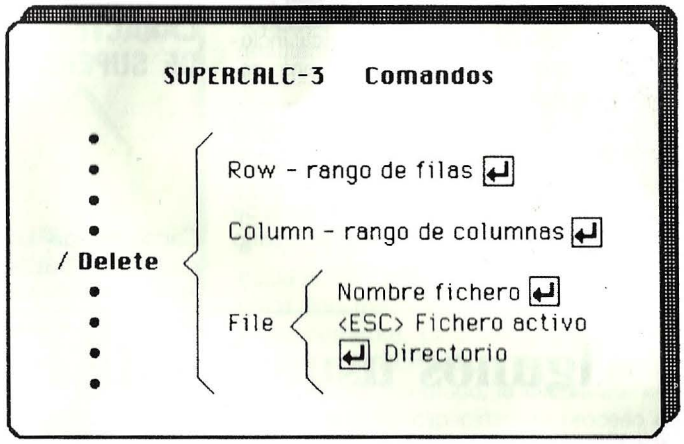
5. Usuarios domésticos

Por último, aunque numéricamente tal vez sean los primeros, no podemos olvidar al típico usuario casero que puede utilizar una hoja electrónica para las aplicaciones más diversas e insospechadas. Cabe recordar que una de las características más apreciadas de las hojas electrónicas es su versatilidad, que las hace candidatas a resolver problemas de muy diversa índole.





La función lógica IF permite utilizar dos fórmulas alternativas, según se verifique o no cierta condición.



La estructura de comandos es de tipo arborescente. En cualquier caso, Supercalc 3 detecta a los comandos por el hecho de que su primer carácter debe ser "/".

OR, AND y NOT. La primera permite utilizar dos expresiones alternativas en función del cumplimiento o no de otra expresión. Las funciones OR y AND ya son bien conocidas: OR producirá resultado VERDADERO si alguna de las expresiones utilizadas como parámetros es VERDADERA, y FALSO en caso contrario; a su vez, AND sólo producirá resultado VERDADERO si todos los parámetros son también VERDADEROS y FALSO en cuanto que uno sólo de ellos sea FALSO. Por último, NOT producirá como resultado FALSO si la expresión de entrada es VERDADERA, y viceversa.

● FUNCIONES FINANCIERAS

SUPERCALC 3 ofrece al usuario cinco funciones financieras: NPV, IRR, PMT, FV y PV.

1. NPV calcula el valor neto del flujo de caja; su valor se obtiene de la siguiente manera:

$$NPV = \sum_{j=1}^{k} A_j (1+r)^{-j}$$

donde: j=número de pe iodos; Aj=flujo de caja del periodo; r=coeficiente de interés.

2. IRR calcula el coeficiente interno de retorno; su fórmula de cálculo se basa en un proceso iterativo que en algunos casos puede dar error ante parámetros equivoc dos.

3. PMT calcula la cuota de devolución constante por periodo para un determinado capital PV, a un coeficiente i de interés y pagadero en n periodos, mediante la siguiente fórmula:

$$PMT = PV \cdot \frac{i}{1-(1+i)^{-n}}$$

4. FV calcula el valor futuro para un pago constante mediante la siguiente expresión:

$$FV = PMT \cdot \frac{(1+i)^n - 1}{i}, \text{ donde}$$

PMT es la cantidad constante, i el coeficiente de interés y n el número de periodos.

5. PV determina el valor actual para un pago constante mediante la siguiente expresión:

$$PV = PMT \cdot \frac{1-(1+i)^{-n}}{i}, \text{ donde}$$

PMT es la cantidad constante, i el coeficiente de interés y n el número de periodos.

● FUNCIONES DE CALENDARIO

Mediante DATE, TODAY, DVAL, DAY, MONTH, YEAR, WDAY y IDATE, el programa ofrece una amplia gama de funciones que facilitan la realización de cualquier tipo de operación con fechas. Por ejemplo, mediante las funciones MONTH, DAY y YEAR, se obtiene como resultado el mes, el día y el año para el valor de entrada que debe representar una fecha.

● FUNCIONES ESPECIALES

SUPERCALC 3 ofrece las siguientes funciones especiales: ERROR, LOOKUP, NA, I, DATE, ISTEXT, IS NUM, IS ERROR, ISNA, TRUE y FALSE. Todas ellas tienen como misión permitir al usuario evaluar las características del parámetro que se les pase. Así, por ejemplo ISDATE, ISTEXT e ISNUM, examinan si el parámetro es una fecha, un texto o un número, respectivamente.

● FUNCIONES ARITMETICAS

El último grupo de funciones aportadas por SUPERCALC 3 lo constituyen las denominadas funciones aritméticas. No vamos a enumerarlas todas por su excesivo número, simplemente citaremos que dispone de prácticamente todas las funciones matemáticas convencionales.

Las funciones son un elemento de vital importancia para el buen funcionamiento de una hoja electrónica; pero no hay que olvidar que para la gestión del programa el usuario necesita utilizar una serie de comandos, con los que decidirá las operaciones que desea realizar.

SUPERCALC 3 dispone de 22 comandos distintos, todos ellos se caracterizan por comenzar por el carácter "/" y a continuación el comando propiamente dicho. Para invocarlos basta con teclear la barra "/" y el primer carácter del nombre del comando.

Algunos de los comandos disponen de varias opciones adicionales que deben ser seleccionadas por el usuario mediante una segunda letra. De nuevo, esta segunda letra puede tener un conjunto de opciones secundarias entre las que el usuario debe elegir.

Sin duda, el sistema de comandos de SUPERCALC 3 resulta muy cómodo una vez que se elige cuál de ellos se desea ejecutar. En el próximo capítulo se presentarán los principales comandos del programa, haciendo hincapié en la capacidad gráfica de SUPERCALC 3.

Archivos en el Spectrum

Creación y tratamiento de archivos en Microdrive

A lo largo de esta sección de la obra se ha estudiado la naturaleza, características y tratamiento de los archivos secuenciales y de acceso directo. Si bien, estos temas fueron tratados bajo un enfoque general y resulta obvio que no todos los ordenadores utilizan los mismos comandos. Por este motivo es necesario abordar el estudio por separado de algunos de los ordenadores más importantes cuyo método para el tratamiento de archivos diverge del caso general. El presente capítulo se concreta en el análisis de este aspecto aplicado al ZX-Spectrum conectado, a través del Interface-1, a la unidad ZX-Microdrive.

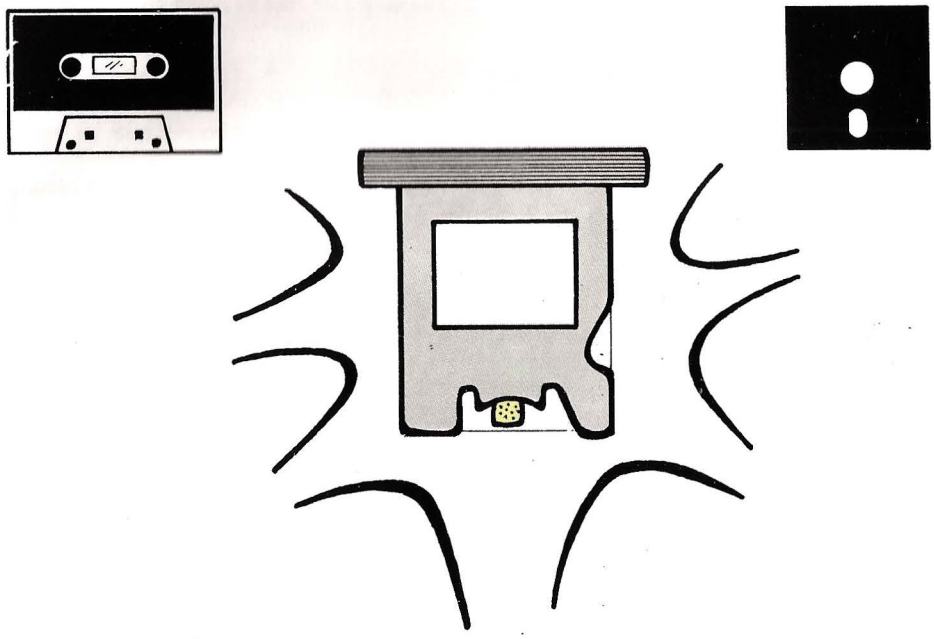
Los Microdrives son unidades de almacenamiento masivo, cuyo soporte de memoria son cartuchos de cinta magnética, enrollada de tal forma que no tiene fin. Estos dispositivos son medios de almacenamiento masivo que presentan ciertas ventajas respecto a los tradicionales magnetófonos a casete. No obstante, sus características distan mucho de acercarse a las que brindan las unidades de disco.

Las unidades de Microdrive se conectan al Spectrum a través del Interface-1. Este dispositivo, además de controlar a los Microdrives, potencia al Spectrum con la aportación de una red local y un interface RS/232.

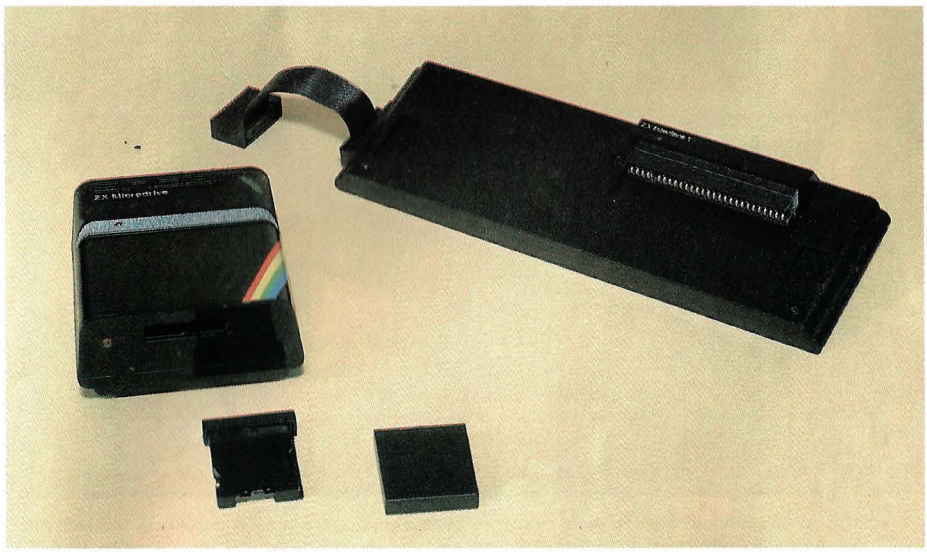
En todo caso, dado que el tema que nos ocupa es relativo al tratamiento de archivos, pasaremos por alto la forma de explotar estas dos últimas posibilidades.

El método de trabajo con los Microdrives y su soporte, los cartuchos, se parece en algunos aspectos al aplicado en el caso de las unidades de disco. Sin embargo, sus posibilidades son sustancialmente inferiores.

La primera operación a realizar, tanto con un disco como con un cartucho, es el formateado del soporte magnético. Esta es



Los Microdrives constituyen una alternativa adicional, entre el disco y el casete, para el almacenamiento masivo de información en el Spectrum.



Las unidades de Microdrive constituyen un original periférico de almacenamiento cuyo soporte son cintas magnéticas alojadas en un diminuto cartucho de material plástico.

una operación durante la cual se crea además el directorio.
El directorio corresponde al índice del contenido del referido volumen de información. En el caso de un cartucho, el directorio contiene en todo momento referencias a los archivos que se almacenan en el mismo. Para realizar esta operación es necesario ejecutar el comando FORMAT, el cual adopta la siguiente formulación general:

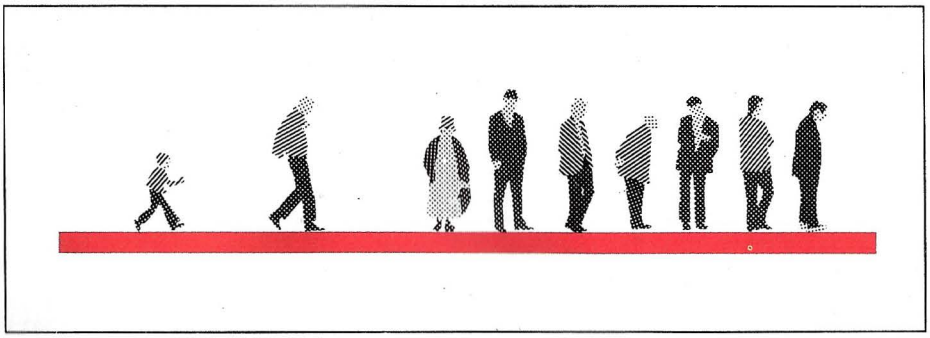
FORMAT*"m";n;<nombre>

en donde:

n: es el número de la unidad de Microdrive en el que se encuentra el cartucho que se desea formatear (puede haber hasta ocho unidades conectadas).

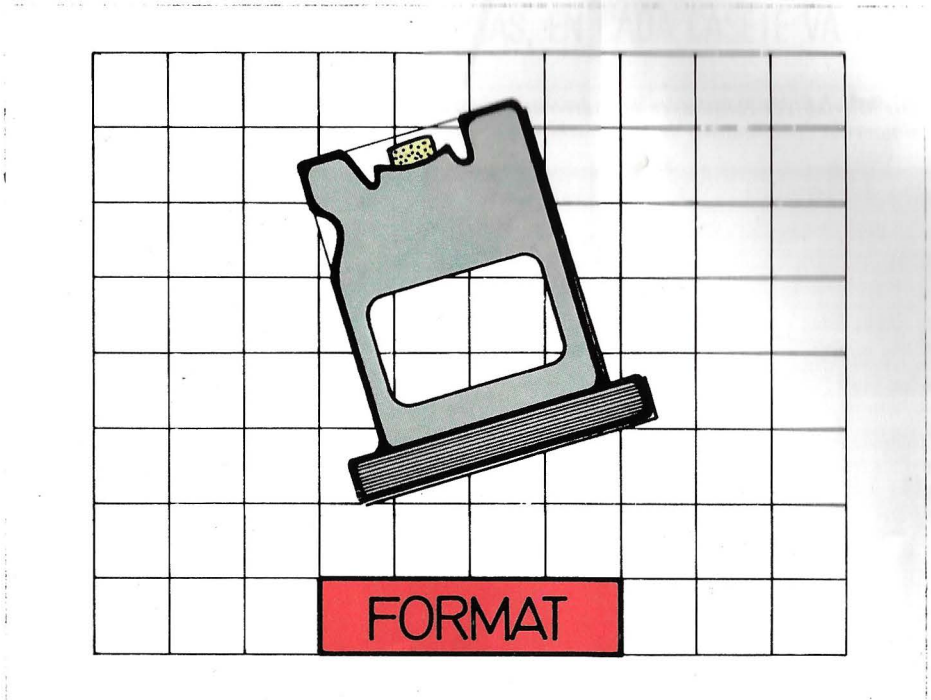
<nombre>: nombre que se desea otorgar al cartucho. Puede coincidir con una cadena de caracteres encerrados entre comillas, o con una variable de cadena.
Otra operación fundamental para el manejo de los cartuchos es la obtención de su catálogo o directorio. Esta operación se puede realizar mediante el comando CAT, cuya formulación es:

CAT n



Por su propia naturaleza, los cartuchos de Microdrive admiten tan sólo archivos de tipo secuencial. En ellos los datos se almacenan uno tras otro, en estricto orden, y para acceder a cualquiera de ellos es preciso pasar por todos los que lo preceden.

Siendo n el número de Microdrive en el que reside el cartucho del que se desea obtener el catálogo.
Resulta fundamental conocer en cualquier momento el catálogo o directorio, ya que no siempre se recuerda qué programas o archivos se encuentran grabados en un determinado cartucho. Dicho comando puede utilizarse también para comprobar si una determinada operación se ha realizado correctamente.
Los nombres de los archivos creados en Microdrive siguen aproximadamente las mismas reglas aplicadas a los archivos en casete. Esto es: el nombre ha de constar de no más de 10 caracteres. No obtante el nombre puede incluir cualquiera de los "tokens" propios del Spectrum, con lo que, aparentemente, el número de caracteres que componen el nombre se ve así incrementado.



Al igual que ocurre con los discos, los cartuchos de Microdrive deben ser previamente formateados para que sea posible realizar en ellos cualquier operación de lectura o escritura.

COMANDOS GENERALES PARA LA MANIPULACION DE ARCHIVOS

Los comandos dedicados a la manipulación de archivos deben permitir las siguientes operaciones básicas: grabar, leer, modificar, verificar, mezclar y borrar archivos.
En el caso del Microdrive, los comandos al efecto admiten las mismas extensiones que sus homólogos destinados al manejo del casete; esto es: CODE para bytes, SCREEN$ para el caso particular de la pantalla y LINE para precisar la autoejecución de un programa BASIC.
Para grabar un programa en un cartucho alojado en la unidad de Microdrive se utiliza el comando SAVE, aunque con una formulación distinta de la habitual:

SAVE*"m";n;<nombre>(<extensiones>)

en donde:

n: señala el número de unidad de Microdrive en la que se va a realizar la grabación.

<nombre>: corresponde al nombre que se desea dar al archivo sujeto a la operación ordenada.

<extensiones>: pueden ser, opcionalmente, LINE, CODE o SCREEN$.

Para leer un archivo de programas ha de utilizarse el comando LOAD, cuya formulación mantiene un gran paralelismo con la propia de SAVE:

LOAD *"m";n;<nombre>(<extensiones>)

en donde:

n: indica la unidad de Microdrive en uso.

<nombre>: nombre del archivo de programa que se desea leer.

<extensiones>: pueden ser, opcionalmente, CODE ó SCREEN$.

Una vez efectuada la grabación de un programa, es posible que el usuario quiera asegurarse de que ésta se ha efectuado correctamente. Para ello cuenta con el comando VERIFY. Este comprobará que el archivo se ha grabado correctamente, comparando la información en él contenida con la que se encuentra presente en memoria. Desde luego, su ejecución sólo será eficaz si permanece inalterable el contenido de la memoria interna que se desea cotejar. Este hecho justifica el escaso empleo del referido comando con la extensión SCREEN$. Su formulación general es la siguiente:

VERIFY *"m";n;<nombre>(<extensiones>)

siendo:

n: número de la unidad de Microdrive donde se encuentra el cartucho activo.

<nombre>: nombre del programa grabado en Microdrive con el que se desea comparar el contenido actual de la memoria.

<extensiones>: pueden ser CODE o SCREEN$, opcionalmente.

En ciertos casos es necesario utilizar dentro del programa en curso una rutina que se confeccionará aparte o bien que se encuentra inmersa en otro programa. En tal caso, hay que recurrir al comando MERGE, el cual permite leer un programa y "mezclarlo" con el que se encuentra en memoria. Dicha "mezcla" se realiza sin

FORMAT *

Permite formatear un cartucho de Microdrive.

Formato: FORMAT * "m"; n; <nombre>

Ejemplo: FORMAT * "m"; 1; "NOMBRE"

CAT

Lee el directorio de un cartucho.

Formato: CAT n

Ejemplo: CAT 1

SAVE *

Almacena un programa en Microdrive.

Formato: SAVE * "m"; n; <nombre>[<extensiones>]

Ejemplos: SAVE * "m"; 1; "PROG-1"
SAVE * "m"; 2; "ROG" LINE 10

Aun cuando la opción básica para el almacenamiento externo de información en el ZX-Spectrum la constituyen los magnetófonos a casete, la mayor velocidad de acceso del Microdrive, sumada a su moderada economía, han convertido a este periférico en una frecuente alternativa.

LOAD *

Lee un programa residente en cartucho.

Formato: LOAD * "m"; n; <nombre>[<extensiones>]

Ejemplo: LOAD * "m"; 1; "ROL"

OPEN#

Abre un archivo de datos.

Formato: OPEN#C; "m"; n; <nombre>

Ejemplo: OPEN#4; "m"; 1; "DATOS-1"

CLOSE#

Cierra un archivo de datos.

Formato: CLOSE#n

Ejemplo: CLOSE# 5

problemas siempre que los números de línea de ambos programas no coincidan.
No obstante, si los números coinciden se producirá una situación anómala, derivada del hecho de que no pueden coexistir en memoria dos líneas con el mismo número. En tal caso, de efectuarse la operación MERGE, desaparecerán las líneas coincidentes del programa que previamente se encontraba en memoria.
La formulación del comando MERGE es la que sigue:

MERGE *"m";n;<nombre>

en donde:

n: es el número del drive que se desea emplear.
<nombre>: corresponde al nombre del archivo cuyo contenido se desea mezclar con el programa que se encuentra en memoria.

Por último, hay que hacer mención a un comando cuyo empleo resultará novedoso para los que sólo hayan empleado unidades de casete. Se trata del comando ERASE, cuya misión es borrar un archivo. Su utilidad es manifiesta en los Microdrives ya que, a diferencia de lo que ocurre con los casetes, en los cartuchos no es posible situar la cabeza lecto-grabadora al principio del programa no deseado y grabar sobre el mismo la nueva información.
La sintaxis de una instrucción de esta índole es:

ERASE "m";n;<nombre>

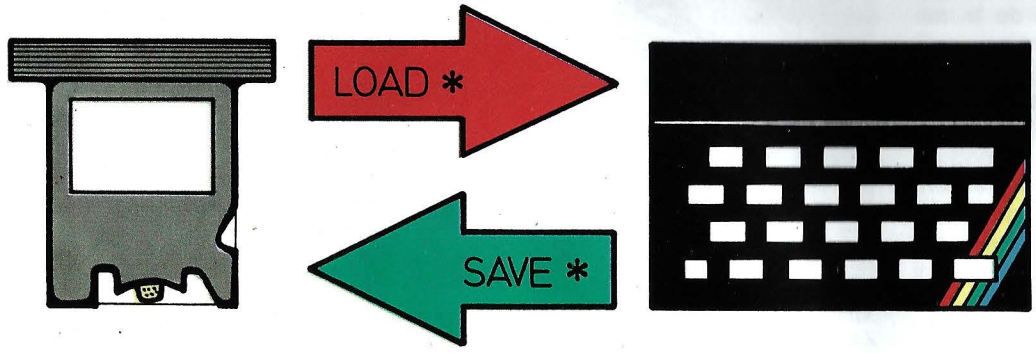
en donde:

n: es el número de la unidad donde se encuentra el cartucho.

<nombre>: es el nombre del archivo que se desea eliminar.

COMANDOS PARA LA MANIPULACION DE ARCHIVOS DE DATOS

Debido a la propia naturaleza de los Microdrives y a la estructura del soporte utilizado (cinta magnética continua), esta unidad de almacenamiento sólo permite trabajar con archivos de tipo secuencial. Esta es la principal limitación de estas unidades.
En los capítulos precedentes dedicados al estudio de los archivos secuenciales, se apuntaron las cuatro operaciones básicas a realizar en un archivo de este tipo: apertura, cierre, escritura y lectura.
La apertura del archivo corre a cargo del



Los tradicionales comandos SAVE y LOAD (seguidos por un asterisco y por los correspondientes parámetros) permiten el trasvase de información entre el Spectrum y el Microdrive.

comando OPEN, cuya formulación adopta en este caso el siguiente aspecto:

OPEN #c;"m";n;<nombre>

en donde:

c: corresponde al número de canal seleccionado (puede haber varios canales abiertos siempre y cuando no coincidan los números).

<nombre>: es el nombre del archivo que se desea abrir.

n: número de la unidad de Microdrive utilizada.

La última operación a realizar tras cualquier acceso a uno de estos archivos es la de cierre. Esta operación es imprescindible; de no efectuarse los archivos quedarán abiertos causando irremediables problemas. El comando a utilizar al efecto es CLOSE:

CLOSE #n

siendo:

n: el número del canal asociado al archivo que se desea cerrar.

Para introducir datos en el archivo en uso se utiliza el recurrido comando PRINT, aunque con una sintaxis distinta de la habitual:

PRINT #n;<datos>

en donde:

n: número de canal asociado al archivo con el que se está trabajando.

<datos>: lista de datos que se desea grabar en un registro del archivo secuencial.

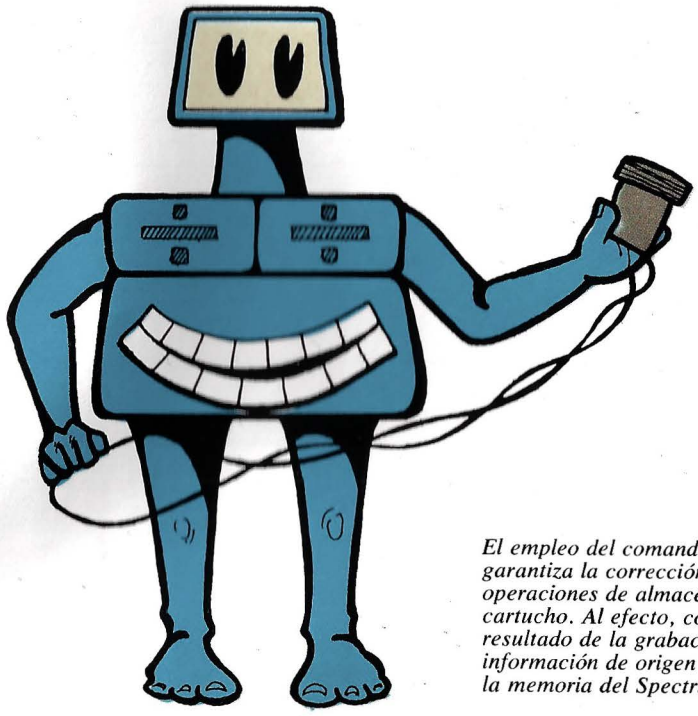
Por último, para leer los datos grabados en el archivo se emplea el comando INPUT:

INPUT #n;<lista de variables>

en donde:

n: corresponde al número de canal asociado con el archivo.

<lista de variables>: lista de variables a



El empleo del comando VERIFY garantiza la corrección de las operaciones de almacenamiento en cartucho. Al efecto, compara el resultado de la grabación con la información de origen residente en la memoria del Spectrum.

PRINT#

Escribe en un archivo de datos.

Formato: PRINT#n <datos>

Ejemplo: PRINT#3; A$, "**"

INPUT#

Lee de un archivo de datos.

Formato: INPUT#n <datos>

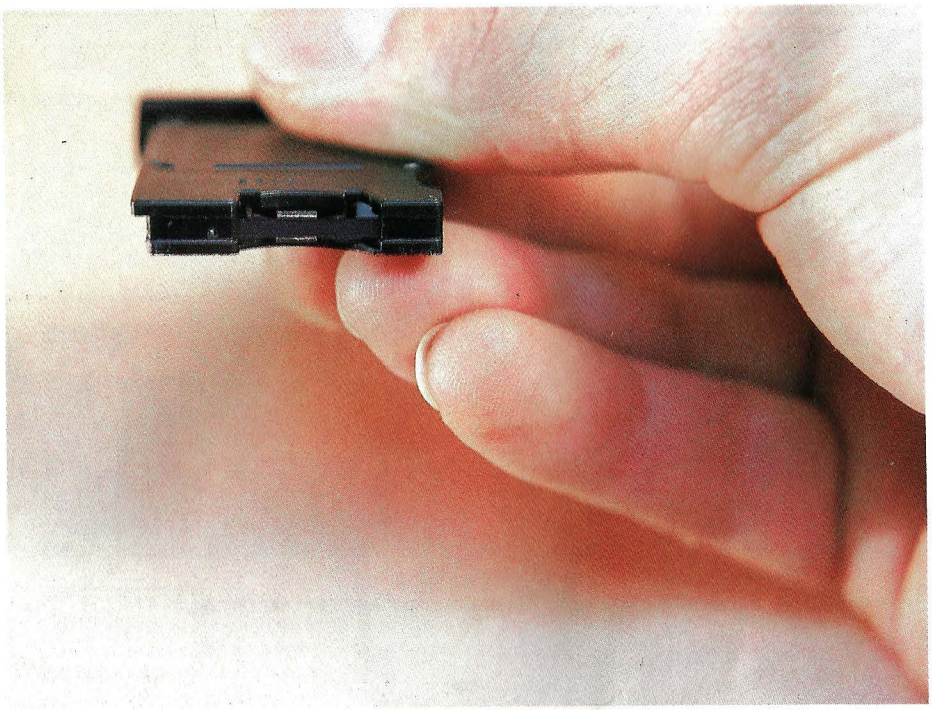
Ejemplo: INPUT#n; A$

INKEY$#

Lee un carácter de un archivo.

Formato: INKEY$#n

Ejemplo: INKEY$#4

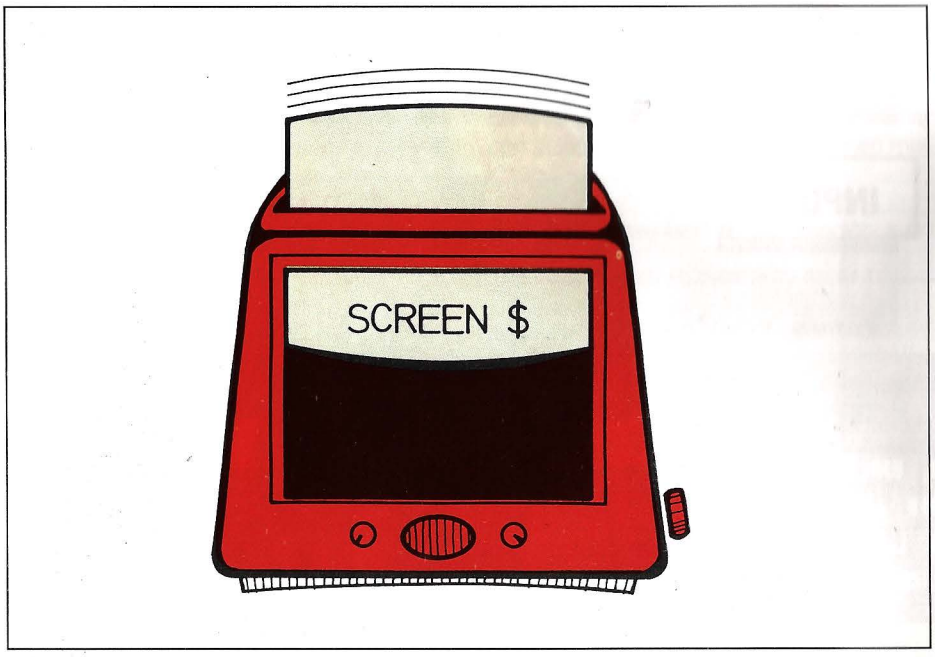


El soporte de almacenamiento utilizado por las unidades de Microdrive son cartuchos de cinta magnética enrollada a modo de "cinta sin fin" (con los extremos unidos). Dada su naturaleza, se trata de soportes de tipo exclusivamente secuencial, semejantes a las tradicionales cintas en casete.

las que se otorgarán los datos que se leen del archivo.

Existe otra función adicional orientada a la lectura de un archivo. Se trata de INKEY$#, cuyo formato es el que sigue:

INKEY$ #n



La opción SCREEN$ aplicada a los comandos LOAD, SAVE* o VERIFY*, permite manipular archivos cuyo contenido es información de pantalla.*

Este comando permite la letura de los caracteres grabados en un archivo, de uno en uno.

EJEMPLOS PRACTICOS

Creación de un archivo secuencial

El primer ejemplo es un programa muy sencillo que permite la creación de un archivo secuencial. Los datos se solicitan a través de instrucciones INPUT y el programa se encarga de irlos introduciendo en el archivo externo abierto al efecto.
El final del archivo se marca por medio de un elemento cuyo contenido es fijo y que consiste en tres asteriscos. Si no se marcara el final del archivo y se intentara leer su contenido con posterioridad, se produciría un error que detendría la ejecución del programa.
Cualquier operación con un archivo empieza por su apertura. Tal acción lleva implícita además la propia creación del archivo puesto que éste no existe:

```
20 INPUT "INTRODUZCA EL DATO";LINE A$
```

A continuación, hay que establecer el medio de captación de los datos a grabar; por ejemplo, mediante una instrucción INPUT:

```
30 IF A$="FIN" THEN GOTO 60
```

Para indicar al programa que no se desea introducir más datos en el archivo, es suficiente con responder al INPUT con la palabra FIN. Al detectar esta palabra, el programa pasará a ejecutar la rutina de cierre del archivo con el que se está trabajando:

```
40 PRINT#5;A$
```

La instrucción de la línea 40 es la encargada de grabar en el archivo el dato introducido. Obsérvese que el número de canal asociado ha de coincidir con el especificado en el momento de abrir el archivo en cuestión:

```
10 OPEN #5;"m";1;"DATOS"
```

Para proseguir con la introducción de da-

tos es necesario que la secuencia del programa regrese a la instrucción de la línea 20, lo cual se logra por medio de un simple GOTO:

```
50 GOTO 20
```

Finalmente, hay que añadir las instrucciones adecuadas para cerrar el archivo. En primera instancia habrá que escribir en él los caracteres *** que servirán para detectar el final del archivo. Tras ello puede ya ejecutarse la oportuna orden CLOSE:

```
60 PRINT#5;"***"
70 CLOSE#5
80 STOP
```

Rutina de lectura

El siguiente ejemplo es una rutina adecuada para leer los datos que fueron introducidos en el archivo por medio del anterior programa.
La rutina en cuestión está preparada para detectar el fin del fichero (caracteres ***) y evitar de esta forma la interrupción del programa por efecto dè un error.
La primera operación, como siempre, consiste en la apertura del archivo:

```
10 OPEN #5;"m";1;"DATOS"
```

Tras ello, se leerá un elemento del archivo por medio de la siguiente instrucción:

```
20 INPUT#5;A$
```

En la línea 30 se detectará si el dato leído coincide con el indicador de fin de archivo:

```
30 IF A$="***" THEN GOTO 60
```

Para mostrar por pantalla el elemento extraído por efecto de la operación de lectura, basta con imprimir la variable A$ por medio de la instrucción siguiente:

```
40 PRINT A$
```

En este punto, es preciso que la secuencia del programa regrese a la línea 20 para que sea leído un nuevo dato del archivo:

```
50 GOTO 20
```

Y, por último, hay que proceder al cierre del archivo y, cómo no, detener la ejecución del programa:

VERIFY *

Permite verificar si una grabación se ha efectuado correctamente.

Formato: VERIFY * "m"; n; <nombre>[<extensiones>]

Ejemplo: VERIFY * "m"; 1; "DATO"

MERGE *

Mezcla dos programas.

Formato: MERGE * "m"; n; <nombre>

Ejemplo: MERGE * "m"; 1; "ALIEN"

ERASE

Borra un archivo del disco.

Formato: ERASE "m"; n; <nombre>

Ejemplo: ERASE "m"; 1; "GOLF"

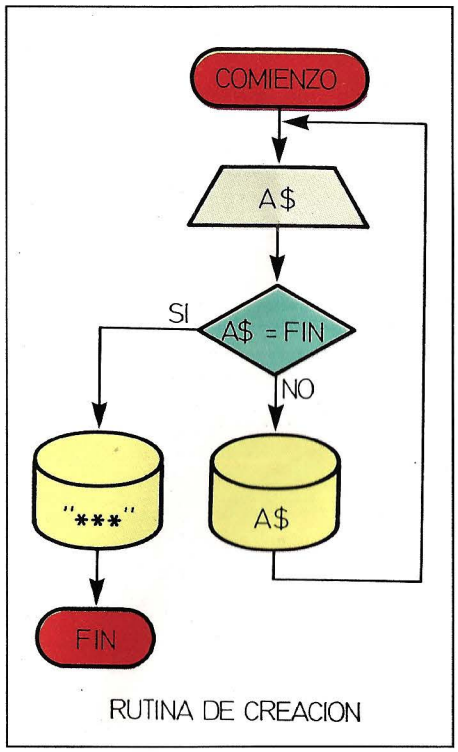


Diagrama de flujo asociado a la rutina de creación de un archivo secuencial descrita en el texto.

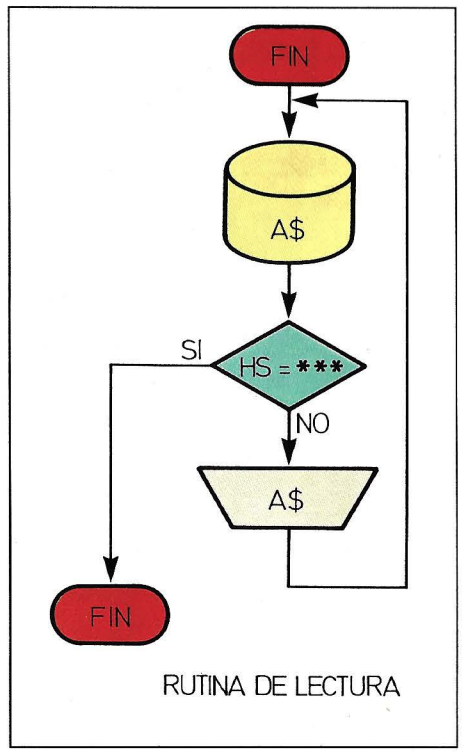


Diagrama de flujo correspondiente a la rutina de lectura de un archivo secuencial.

Rutina 1: Creación de un archivo secuencial

```
 5 REM RUTINA DE CREACION
10 OPEN #5; "m"; 1; "DATOS"
20 INPUT "INTRODUZCA EL DATO"; LINE A$
30 IF A$="FIN" THEN GO TO 60
40 PRINT #5; A$
50 GO TO 20
60 PRINT #5; "***"
70 CLOSE #5
80 STOP
```

Rutina 2: Lectura de un archivo secuencial

```
 5 REM RUTINA DE LECTURA
10 OPEN #5; "m"; 1; "DATOS"
20 INPUT #5; A$
30 IF A$="***" THEN GO TO 60
40 PRINT A$
50 GO TO 20
60 CLOSE #5
70 STOP
```

Rutina 3: Actualización de un archivo secuencial

```
 5 REM RUTINA DE ACTUALIZACION
10 OPEN #5; "m"; 1; "DATOS"
20 OPEN #4; "m"; 1; "DATOS.1"
30 INPUT #5; A$
40 IF A$="***" THEN GO TO 70
50 PRINT #4; A$
60 GO TO 30
70 INPUT "INTRODUZCA EL DATO"; LINE A$
80 IF A$="FIN" THEN GO TO 110
90 PRINT #4; A$
100 GO TO 70
110 PRINT #4; "***"
120 CLOSE #5: CLOSE #4
130 STOP
```

```
60 CLOSE#5
70 STOP
```

Rutina de actualización

La tarea más compleja a realizar con un archivo secuencial es la de actualización de su contenido. Evidentemente, ésta no puede efectuarse de una forma directa, de ahí que haya de recurrirse a otros artificios. La solución reside en actualizar el archivo actual copiando los elementos ya existentes en un nuevo archivo, y grabando a continuación los nuevos elementos.
Como de costumbre, se empezará abriendo los archivos implicados en la operación; dos en el caso que nos ocupa:

```
10 OPEN #5;"m";1;"DATOS"
20 OPEN #4;"m";1;"DATOS.1"
```

Acto seguido, hay que instrumentar los medios necesarios para leer los datos del archivo de origen, uno a uno, y grabarlos en el archivo de destino. Por supuesto, cuando en el archivo de origen se detecte la marca de fin de archivo (***), habrá que saltar el bloque de instrucciones que gestionarán la introducción en el archivo de destino de los nuevos datos implicados en el proceso de actualización. Las acciones descritas quedan en manos de las siguientes instrucciones:

```
30 INPUT#5;A$
40 IF A$="***" THEN GOTO 70
50 PRINT#4;A$
60 GOTO 30
```

La zona encargada de la introducción de los nuevos datos, que ampliarán los ya existentes, y de colocar la marca de fin del archivo en su nuevo emplazamiento coincide con la que sigue:

```
70 INPUT "INTRODUZCA EL DATO";
   LINE A$
80 IF A$="FIN" THEN GOTO 110
90 PRINT#4;A$
100 GOTO 70
110 PRINT#4;"***"
```

Por último, sólo queda cerrar los archivos utilizados y detener la ejecución del programa:

```
120 CLOSE#5:CLOSE#4
130 STOP
```

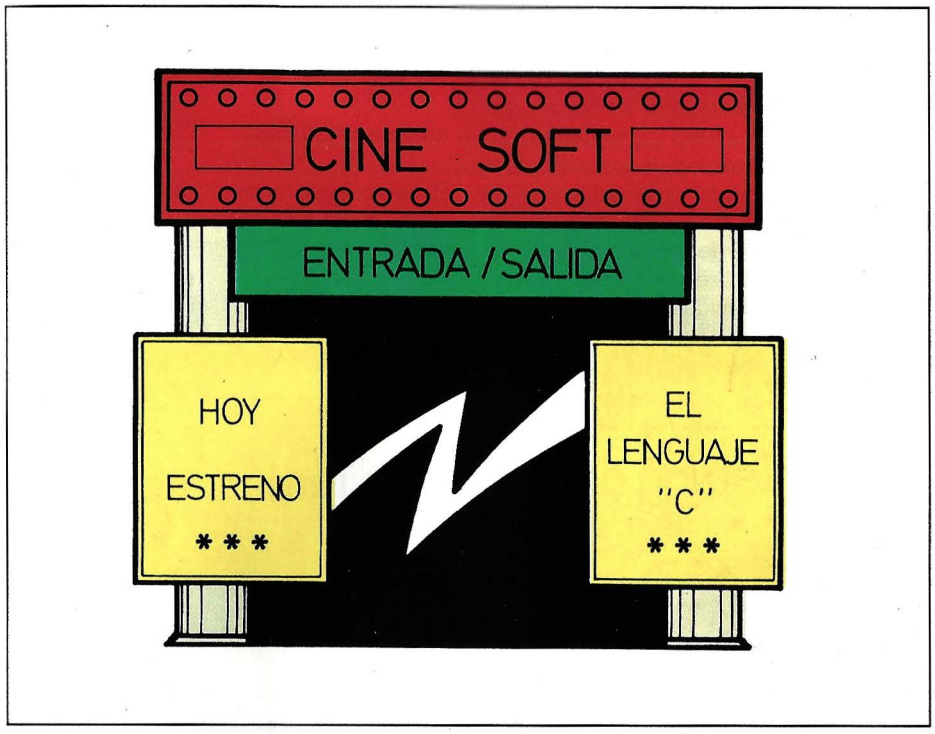
El lenguaje C (y 7)

El punto final

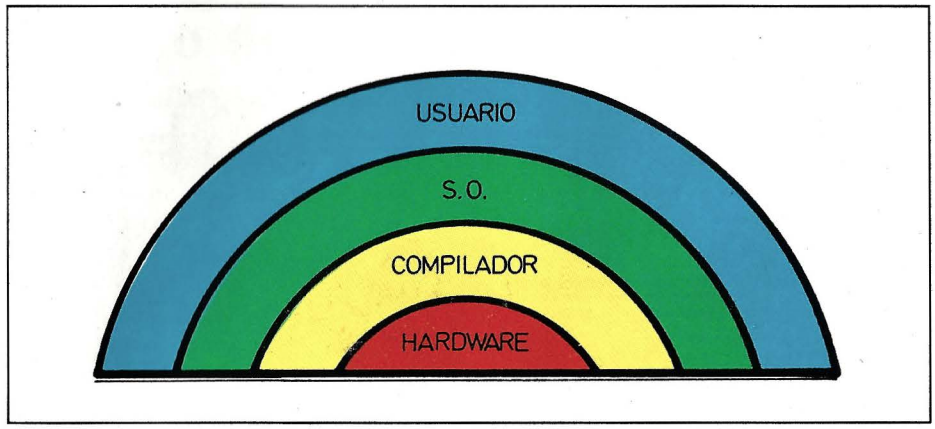
Aquí concluye la zona de la obra dedicada al lenguaje C. Aunque el "cuerpo doctrinal" ya ha sido sentado en los anteriores capítulos, todavía quedan algunos pequeños detalles por tratar.
Los programas propuestos hasta el momento tenían todos una cosa en común: nunca realizaban entradas por el teclado. El porqué, se verá en el siguiente párrafo.

LA BIBLIOTECA DE ENTRADA/SALIDA

Un buen número de las obras dedicadas a tratar un determinado lenguaje de programación comienzan explicando cómo realizar una salida por la pantalla a través de instrucciones del tipo PRINT, WRITELN o, en nuestro caso, "printf". Nosotros optamos por ese camino, pero pronto nos detuvimos dado que la entrada/salida en C es algo particular.
Las funciones que invocan la salida como las anteriormente citadas, o la entrada como son INPUT o READLN, constituyen una de las partes más complicadas de la implementación de un lenguaje, y están fuertemente condicionadas por el entorno del compilador: el sistema operativo.
Ya se ha comentado que el lenguaje C se diseñó para su uso en el entorno del sistema operativo Unix, y, por lo tanto, la entrada/salida se efectúa apoyándose en las facilidades que proporciona éste. Previendo la posible utilización del C en otros entornos, las rutinas de entrada/salida —entre las que está "printf"— se encuentran en un fichero separado del resto del programa, el cual es gestionado por el "linker" para producir el código ejecuta-



Para facilitar la actuación del "C" en cualquier entorno, las operaciones de entrada salida son gobernadas por rutinas (como "printf") independientes del programa. Estas son manipulables por el usuario para adaptarlas a cada entorno de trabajo específico.



En realidad no estamos tan cerca del ordenador como parece. Unas partes descansan sobre otras, siendo el hardware el último eslabón de la cadena.

ble. De esta forma, basta con manipular el fichero de las rutinas de entrada/salida adaptándolo a nuestro entorno concreto. Los nombres de estas rutinas no suelen cambiar, aunque sí su contenido.
La rutina que de forma general permite realizar las entradas por teclado se denomina "scanf". En la figura se reproduce el aspecto de esta función, el cual es muy parecido al de "printf", si bien el "string de formato" tiene un significado distinto. Ahora especifica el tipo de entrada que se va a teclear.
En el ejemplo ilustrado, el formato de "scanf" está indicando que la entrada estará compuesta por un carácter, un número indefinido de espacios en blanco (esto lo señala el espacio en blanco que hay entre "%c" y "%d") y un entero. La única forma de que una vez comenzada la ejecución de "scanf" nos salgamos de ella, es introducir los elementos especificados en el string de formato. Por ello, si se accionara la tecla de retorno de carro, o ENTER, a continuación del carácter "a", el programa seguiría ejecutando "scanf" esperando la segunda entrada.

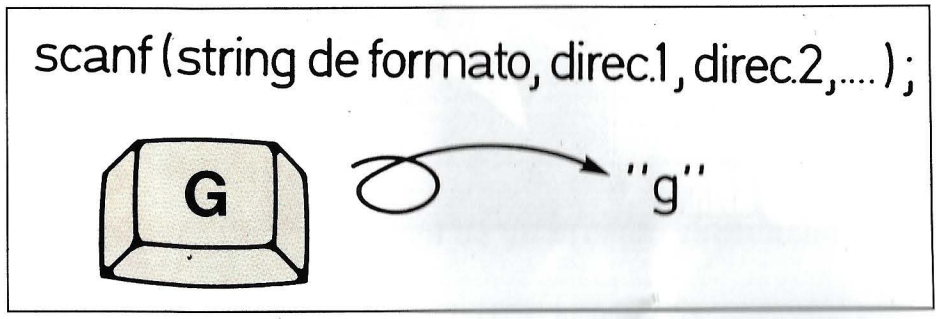
Las variables hay que referenciarlas a través de su dirección, no de su contenido, por lo que es imprescindible el uso del operador "&" que da la dirección de la variable que lo sigue.
Normalmente, encontraremos a "scanf" demasiado torpe y difícil de manejar para nuestros propósitos concretos, por lo que lo mejor será redactar nuestras propias funciones de entrada/salida. En este sentido será útil la función "getchar", la cual detiene la ejecución de un programa hasta que se pulsa una tecla, devolviendo en ese momento el carácter correspondiente. Esta función no necesita ningún argumento, por lo que una llamada a la misma tiene el siguiente aspecto:

```
carácter=getchar();
```

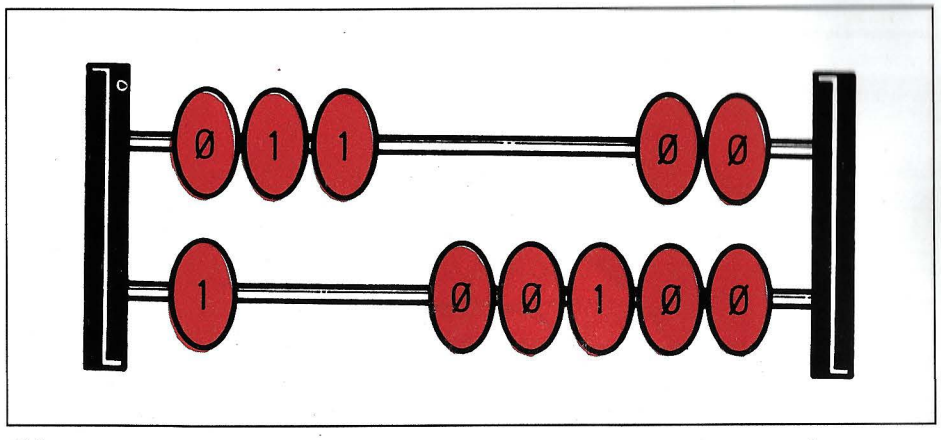
El fiel acompañante de "getchar" es "putchar" que, cómo no, escribe en pantalla el carácter especificado en su argumento. Por ejemplo:

```
putchar (`a`);
```

escribe el carácter `a` en la pantalla. En la figura adjunta aparece un ejemplo de utilización de ambas funciones.
Todas estas funciones son tan sólo una parte de las que debe contener la biblioteca de entrada/salida del compilador que se esté manejando. Hay otras, si bien las principales son las mencionadas.



Aspecto de la función "scanf".



El lenguaje "C" también incorpora un conjunto de operadores adecuados para realizar operaciones lógicas a nivel de bit.

LAS OPERACIONES CON BITS

El lenguaje C incorpora una serie de operadores que permiten realizar operaciones lógicas con los bits de un entero o de un carácter, pero nunca con una variable de tipo "float" o "double". Estos operadores son los siguientes:

&	and lógico
|	or lógico
^	or exclusivo lógico (xor)
«	desplazamiento a la izquierda
»	desplazamiento a la derecha
~	complemento a uno

Así, por ejemplo:

```
c=n & 0X7F;
```

hace que se ponga a cero el bit más significativo de "n". Se observa que ha sido necesario anteponer al número hexadecimal 7F los caracteres "0X", precisamente para indicar al compilador que se trata de este tipo de constante.
Una expresión del tipo:

```
x « 2;
```

hace que los bits de "x" sean desplazados dos posiciones a la izquierda, rellenando los espacios que van quedando a la derecha con ceros. El resultado es análogo al que se hubiera obtenido con la expresión:

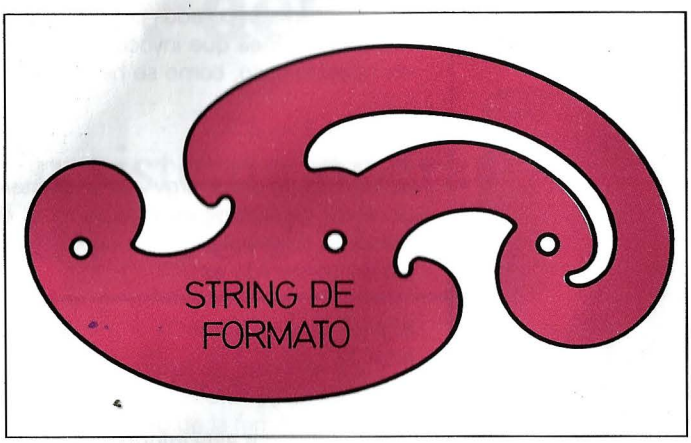
```
x=x*4;
```

Pero la rapidez de ejecución del primer método es muy superior a la del segundo.

El operador "~" se aplica de la siguiente forma:

```
~x;
```

y, simplemente, cambia cada cero de "x" por un uno y viceversa.
Hay que poner especial cuidado en no confundir los operadores "&&" y "&" ni "||" y "|". Un carácter de más o de menos tiene en C gran importancia.




```
main()
{
        char un_car;
        int un_ent;

        scanf("%c %d",&un_car,&un_ent);
        printf("%c %d  n",un_car,un_ent);
}
```

El string de formato es la plantilla según la cual se analizará la entrada realizada.

EL POCO PRIVILEGIO DE SER "main"

Se ha comentado que "main" era como cualquier otra función del programa, y que sólo se diferenciaba en que siempre era la primera en ser ejecutada. Sin embargo, al contrario que la mayoría de las funciones, no lleva ningún parámetro... ¿Seguro? Vamos a ver que ello es falso: "main" sí puede llevar parámetros.

Para ejecutar un programa una vez compilado, basta teclear el nombre del fichero generado por el compilador. Pensando en un programa para sumar dos números, se puede tener la situación del primer ejemplo de la figura (caso a); en él, el programa acabará llegando a algún punto en el que pida los valores de los números a sumar para, posteriormente, continuar la ejecución. En el lenguaje C cabe la posibilidad de pasar datos al programa en el momento de llamarlo (segundo ejemplo de la misma figura, caso b), sin necesidad de esperar a que éste los pida.

En el segundo caso, "main" tiene dos argumentos. Su cabecera tomaría el siguiente aspecto:

```
main(argc,argv)
int argc;
char *argv[];
{
        .........
        .........
}
```


```
main()
{
        for(;;)
            putchar(getchar());
}
```

Además de ilustrar el funcionamiento de "putchar" y "getchar", el ejemplo muestra la forma de realizar un bucle infinito en "C". Los caracteres más realzados corresponden a la entrada realizada por el teclado, los otros constituyen la salida por pantalla.

Cómo decir mucho en pocas palabras

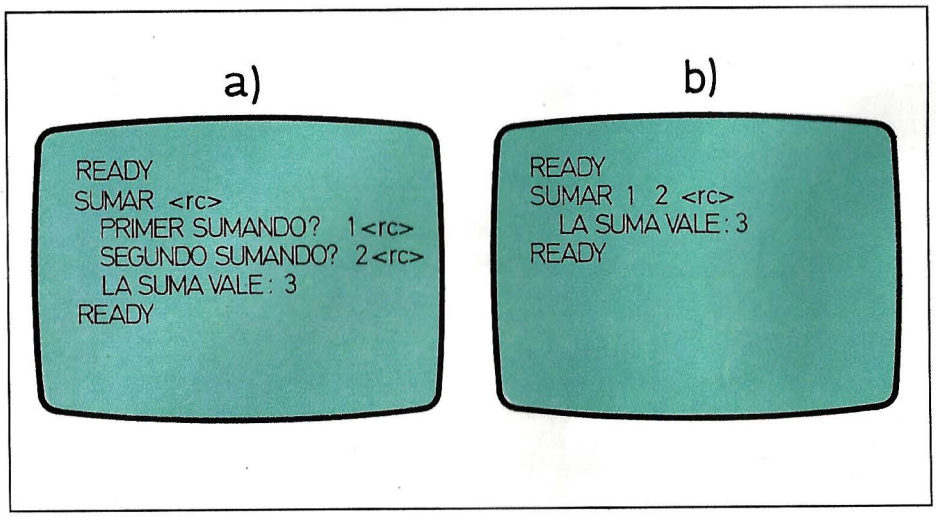
En varias ocasiones a lo largo de esta obra se ha mencionado que uno de los objetivos que se planteó el lenguaje C fue la concisión; es decir, expresar un cálculo en el menor número posible de líneas de listado. En la práctica resultará que para aquel que no conozca un poco el lenguaje, los programas en C se le antojarán escritos en lengua de otra galaxia.

Si todavía no está convencido de que el C es un lenguaje en el que la concisión es una característica primordial, observe la expresión siguiente:

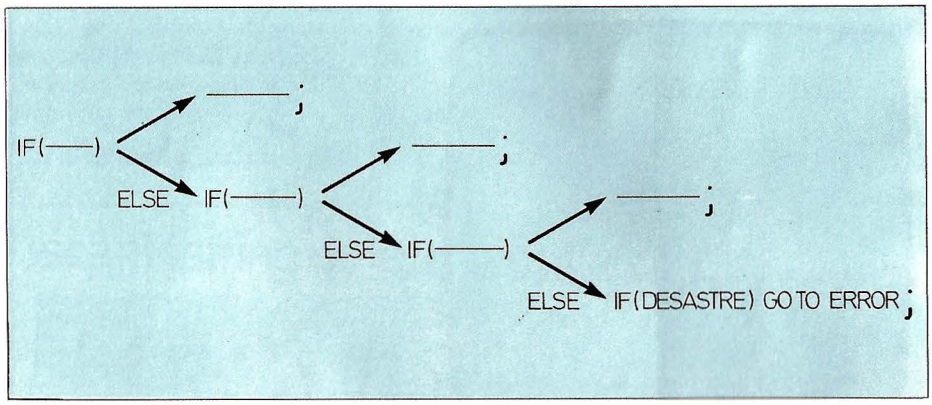
```
x=1 ? y=2 : y=3 ;
```

Para el que no haya visto este tipo de notación con anterioridad, le resultará muy difícil deducir que es completamente equivalente a la siguiente estructura:

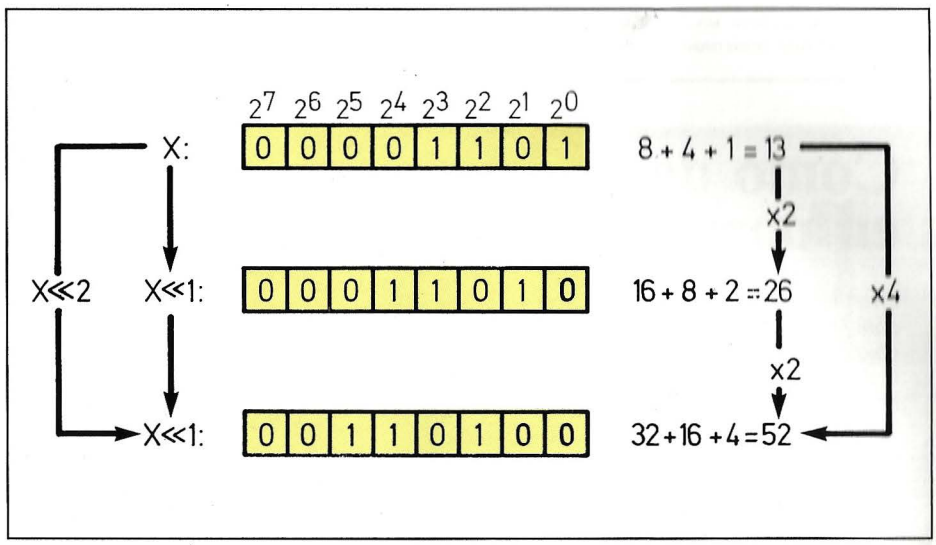
```
if (x=1)
    y=2;
else
    y=3;
```



En ambos ejemplos de ejecución, "sumar" es el nombre de fichero que contiene al programa en "C"; programa que, previamente, habrá sido compilado.



La mejor victoria es una retirada a tiempo... aunque sea esgrimiendo la orden GOTO.



Cómo multiplicar de una forma rápida a base de ordenar el desplazamiento de los bits. Distintos desplazamientos dan distintos productos de potencias de dos. Por el extremo derecho se van introduciendo bits "cero" a medida que se desplazan los bits del número multiplicado.

"argc" es el número de palabras (grupos de caracteres separados por blancos) que había en la línea que invocó al programa; en nuestro caso, como se ha tecleado:

sumar 1 2 <r.c.>

su valor sería tres. "argv" es un puntero a un array de strings de caracteres el cual contiene tales palabras. En nuestro ejemplo su contenido es:

argv[0] .... "sumar"
argv[1] .... "1"
argv[2] .... "2"

Puede utilizarse esta información para calcular los valores a sumar sin necesidad de tener que incluir en el programa las rutinas necesarias para realizar la entrada por teclado.
Realmente, no tiene tanta importancia llamarse "main".

LA SENTENCIA "goto"

Los saltos incondicionales que representan las sentencias "goto" son innecesarios en un lenguaje estructurado como el C, aunque hay ocasiones en las que llevar al límite este "estructuralismo" sólo puede acarrear dolores de cabeza: una sentencia "goto" en el momento y lugar adecuados puede ahorrar mucho trabajo.
Un ejemplo interesante sobre el uso de esta sentencia lo constituye la necesidad de "salir" de una estructura muy anidada, como la representada en la figura adjunta. Si al llegar a un determinado nivel nos encontramos con un error que obligue a abandonar todo el proceso, lo mejor es atajar por lo fácil con un "goto".
La sentencia "goto" tiene el siguiente aspecto:

goto <etiqueta>;

donde <etiqueta> es un nombre como los utilizados para las variables, seguido del signo ":". Al ejecutarse el "goto", el control bifurcará a las sentencias que siguen a la etiqueta.
Cabe recordar que el Pascal, el lenguaje estructurado por excelencia, incluye la sentencia "GOTO", lo cual es una buena excusa para utilizar esta orden en otros lenguajes no tan estructurados.

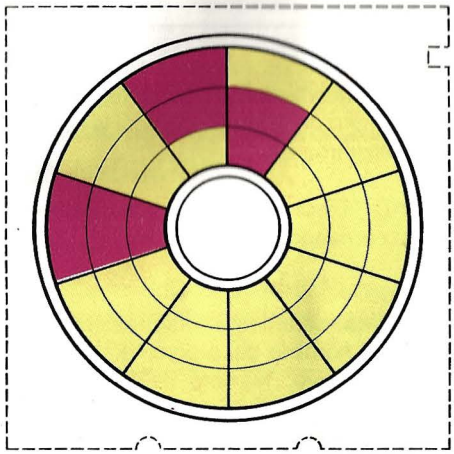
S.O. del CBM-1541

Un sistema operativo de disco para VIC-20 y Commodore 64

La unidad de disquete 1541 de Commodore está integrada dentro de la familia de periféricos para almacenamiento de información constituida por modelos como los 2040, 2031LP, 4040 y SFD1001; este último con una capacidad de 1 Megabyte. Todas estas unidades constituyen una mejora, tanto cuantitativa como cualitativa, respecto a la popular unidad de cinta DATASSETTE, al disfrutar de una mayor capacidad de almacenamiento, así como de una mayor rapidez y flexibilidad en el manejo de información.

CARACTERISTICAS GENERALES

Los microordenadores VIC-20 y Commodore-64 son los destinatarios de la unidad de disco flexible 1541. Esta consta de una fuente de alimentación, el dispositivo mecánico de lectura/escritura y la circuitería de control. El conjunto de estos elementos está alojado en un único mueble, dando lugar a un sistema compacto. La presencia de un microprocesador del tipo 6502, además de 16K de ROM y un chip de 2K de memoria RAM, hace que la unidad de disco 1541 sea un dispositivo inteligente. En ella descansa la responsabilidad de gestionar todo el trasiego de información entre el disquete y el microordenador, además de atender a todos los procesos propios de lectura/escritura en el disquete. Esto significa que la incorporación de dicha unidad de disquete al VIC-20 ó al Commodore-64 no merma la cantidad de memoria disponible ni la velocidad de proceso del microordenador. Los disquetes utilizados son de 5 y 1/4 de pulgadas, de simple cara y simple densidad, y capaces de almacenar hasta 170K a repartir entre un máximo de 144 ficheros.



La tabla de disponibilidad de bloques, BAM, lleva la contabilidad de los bloques ocupados y disponibles en el disquete.

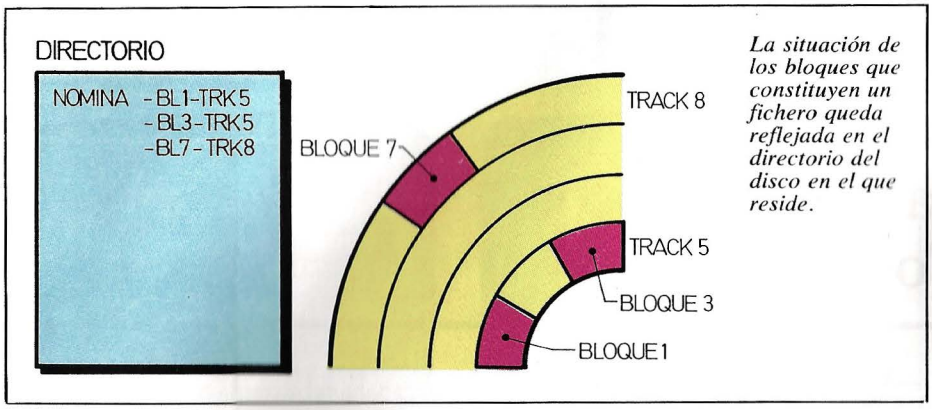
La incorporación de otros ficheros se ve facilitada por la existencia de un doble conector de tipo serie en la parte trasera del mueble. A través del mismo es posible encadenar el sistema hasta un máximo de cinco unidades de disquete y una impresora.

Cabe destacar también la compatibilidad de la unidad 1541 con otros miembros de su familia (4040, 2031, 2040), posibilitando así la migración de software de un sistema a otro sin necesidad de introducir cambio alguno.

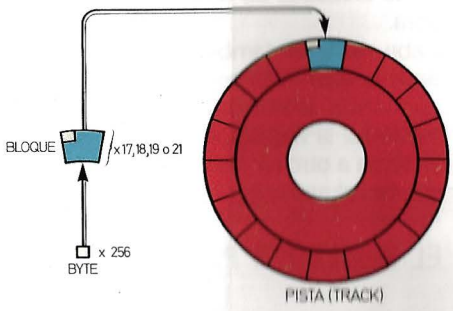
EL ACCESO A PROGRAMAS

La utilización de programas almacenados en cualquier medio (disquete, cinta o cartucho), es con mucho la actividad más veces realizada por el usuario; de ahí que sea conveniente que su explotación resulte independiente del dispositivo que almacena el programa. Así pues, la carga y almacenamiento de programas se efectúa con las mismas sentencias del lenguaje BASIC, ya conocidas por su empleo en el manejo del casete. No obstante, existen

Con la colaboración de una unidad de disco —por ejemplo, la Commodore 1541— y su correspondiente sistema operativo, un sencillo microordenador como el Commodore 64 puede franquear el umbral de acceso a la explotación de programas evolucionados que deban acceder a amplios volúmenes de información.



La situación de los bloques que constituyen un fichero queda reflejada en el directorio del disco en el que reside.



La información contenida en el disquete está agrupada en bytes, bloques y pistas (tracks) de tamaño variable, según estén más o menos distanciados del centro del disco.

algunos matices aplicables tan sólo a programas residentes en disquete.
El formato de la sentencia para la carga de un programa es:

LOAD N nombre $,<no. dispositivo>, <no.comando>

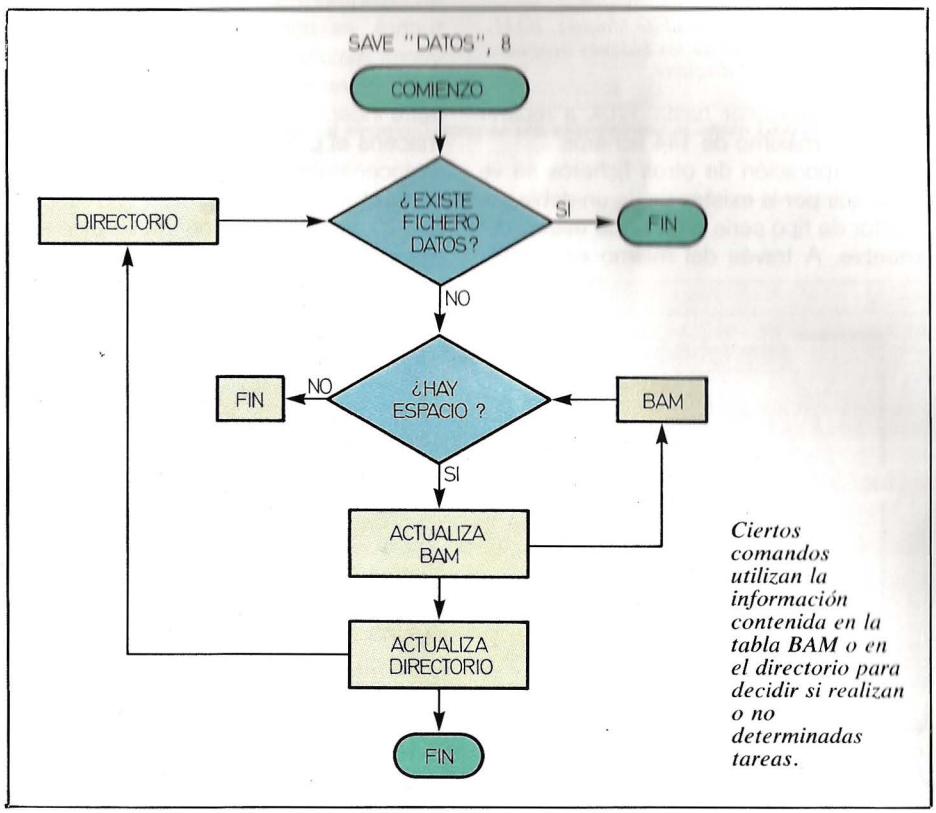
Siendo:
nombre$: cadena de caracteres que contiene el nombre del programa.
no.dispositivo: número de la unidad de disquete en uso.
no.comando: para programas BASIC tiene el valor cero, y toma el valor uno para código máquina, caracteres definibles y otras funciones que dependen de su situación en memoria.
La carga de programas se ajusta al formato:

SAVE nombre$,<no. dispositivo>, <no.comando>
SAVE`@0:'nombre$,<no.dispositivo>, <no.comando>

según se desee grabar un programa sólo si no existía previamente en el soporte de memoria, o en cualquier circunstancia.
La comprobación de que un programa ha sido almacenado correctamente se lleva a cabo con la sentencia:

VERIFY nombre$,<no.dispositivo>

Al ejecutar esta orden se compara la versión existente en memoria con la que se acaba de guardar en disco, avisando de cualquier discrepancia.
Como prioridad específica del acceso a programas en disco cabe destacar la existencia de caracteres "comodín" o "wildcards" que permiten referenciar a múltiples ficheros a la vez. Esta es una facultad muy de agradecer cuando no se conoce perfectamente el nombre del programa al que se desea acceder.

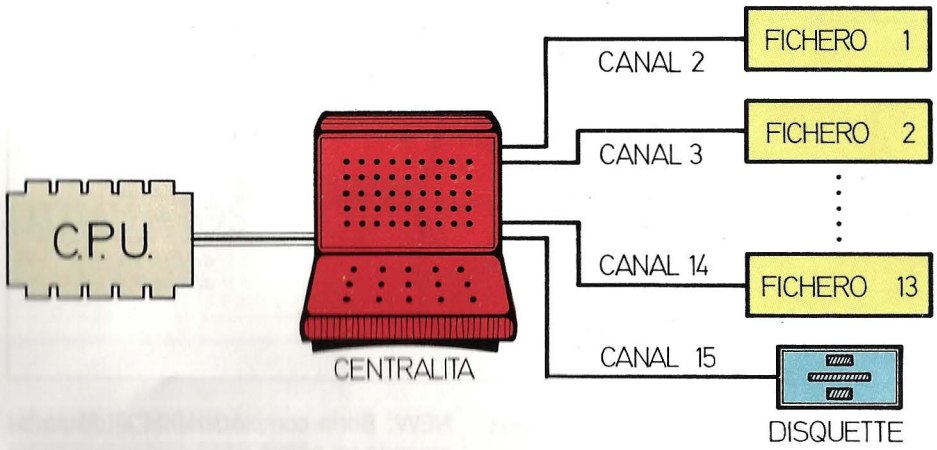


Ciertos comandos utilizan la información contenida en la tabla BAM o en el directorio para decidir si realizan o no determinadas tareas.

EL SISTEMA OPERATIVO

El sistema operativo de la unidad de disquetes, con independencia de las propiedades aportadas por el software de base del microodenador, reconoce una colección de comandos que permiten gestionar y mantener la información contenida en el disquete.
La ejecución de estos comandos exige un conocimiento exhaustivo por parte del sistema operativo de la situación de la información en el disco. Para mayor facilidad, el propio sistema operativo organiza la información en bloques de datos de 256 bytes cada uno.
El simple agrupamiento de la información en bloques no da al S.O. una idea cabal de cuáles son los bloques libres y cuáles los ocupados. De ahí que sea necesario contar con una tabla en la que se vaya reflejando el estado de los bloques; ésta se

denomina tabla de disponibilidad de bloques o BAM (Block Availability Map).
Aunque con la ayuda de la tabla de disponibilidad se sabe ya dónde colocar la nueva información, queda aún un detalle muy importante: ¿Cómo recuperarla? Los ficheros son conjuntos de bloques localizados en distintas posiciones del disquete, luego hay que saber qué bloques son los correctos y en qué orden deben estar colocados para formar un conjunto de información significativa, en lugar de constituir un simple montón de ceros y unos sin significado alguno.
La resolución de este problema pasa por la creación de una tabla (directorio) que contenga el nombre de cada fichero residente en el disco y los bloques que lo componen ordenados correctamente.
Mediante las tablas descritas (BAM y directorio) el sistema operativo es capaz de gestionar la información contenida en el disquete, manteniéndola actualizada con la colección de comandos que se describen a continuación.



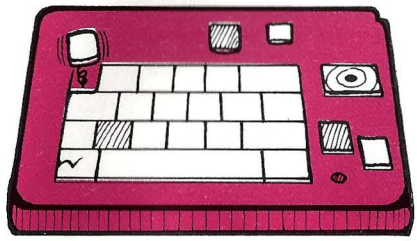
El flujo de información desde la CPU a los ficheros, y viceversa, se realiza a través de canales.

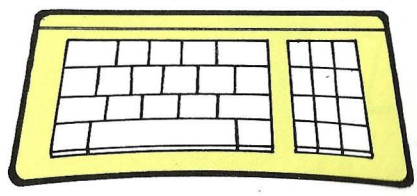
COMANDOS DEL S.O.

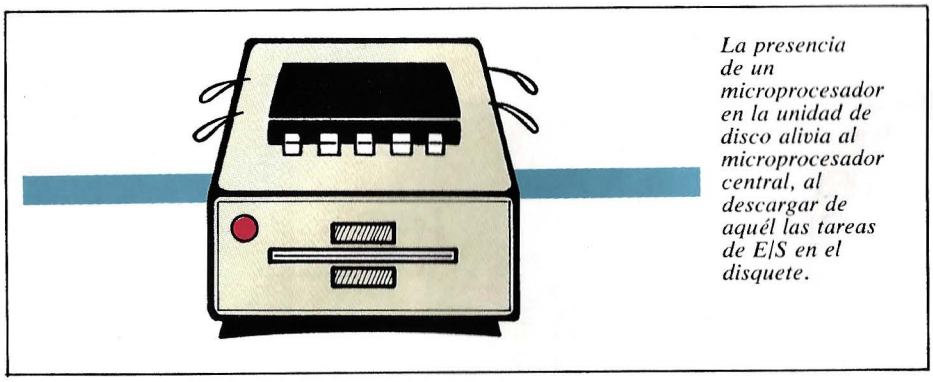
La comunicación con la unidad de disco se fundamenta en las sentencias OPEN y PRINT# del lenguaje BASIC, utilizadas normalmente para la creación y escritura de ficheros de cinta. Con estos dos comandos también se puede abrir la comunicación con la unidad de disco (OPEN) y enviar comandos PRINT# hacia la unidad de disco desde un programa BASIC; para

La sopa de teclas

El teclado es, junto con la pantalla, el periférico más conocido. La importancia del mismo está en consonancia con su popularidad, al ser el dispositivo de entrada de datos más importante para la práctica totalidad de los microordenadores. Ello es razón más que suficiente para que los fabricantes pongan cada vez más énfasis en lograr un diseño de teclado que facilite el trabajo del usuario. A simple vista, un teclado de microordenador se parece enormemente a los clásicos teclados QWERTY de las máquinas de escribir; aunque también es fácil reparar en la existencia de otras teclas sin parangón en las máquinas de escribir. Las últimas tendencias en el diseño de los teclados tienden a agrupar las teclas en bloques de acuerdo a su funcionalidad.
El mayor bloque lo forman las teclas que se encuentran en cualquier máquina de escribir, junto con ciertas teclas especiales propias de cada sistema (CTRL, TAB, ESC, etc.). Las teclas de movimiento de cursor suelen agruparse en una zona independiente para facilitar su manejo, llegando a ser muy llamativas (tal es el caso de su disposición y tamaño en los equipos MSX). Un tercer grupo está constituido por las teclas de función programables, definibles por el usuario.
Puede existir incluso otro grupo integrado por cifras decimales y símbolos aritméticos; este es el "Keypad" destinado a la introducción masiva de datos numéricos.
Por lo que respecta a la naturaleza de las teclas, la variedad alcanza desde los teclados mecánicos, en franco desuso debido a su gran cantidad de fallos y averías, pasando por los teclados efecto capacitivo o núcleo de ferrita, hasta los más modestos de membrana, bastidor mecánico y "gomas de borrar", categoría ésta popularizada por el ZX-Spectrum.
La similitud de un teclado de ordenador con el de una máquina de escribir, se hace trizas cuando se compara el modo de funcionamiento de uno y otro.
Mientras que en la máquina de escribir su efecto es el de la impresión mecánica de un signo sobre un papel, el teclado de un microordenador ha de hacer un tratamiento complejo de la señal que se genera cuando se presiona una tecla; siendo necesario en muchos casos un microprocesador que se ocupe de dicha tarea y se encarga de establecer la comunicación con el procesador central.
El tanto por ciento del volumen de datos introducidos por el teclado, ha ido decreciendo a medida que aparecían otros periféricos complementarios (ratón, lápiz óptico, tableta digitalizadora, etc.). Con ello se ha reducido ligeramente su protagonismo inicial, y es posible que tal vez se llegue a su sustitución cuando los sistemas de reconocimiento de voz sean plenamente operativos. No obstante, hoy por hoy, el teclado es todavía un elemento insustituible.







La presencia de un microprocesador en la unidad de disco alivia al microprocesador central, al descargar de aquél las tareas de E/S en el disquete.

tal fin hay que servirse de unas líneas especiales para la transmisión de datos: los canales.
La formulación correcta de las referidas instrucciones es la siguiente:

OPEN<no.fichero>,<no.dispositivo>,<no.canal>,<comandos>
PRINT#<no.canal>,<comando$>

siendo:
no.fichero: número de identificación del fichero (de 1 a 255)
no.dispositivo: número de la unidad de disquete.
no.canal: número del canal por el que se va a efectuar la transmisión (0 a 15)
comando$: cadena de caracteres que contiene un comando.
Como puede observarse, la unidad de disco reacciona como si se tratara de un fichero; es capaz de recoger y ejecutar los comandos que el microordenador le transmite a través de un canal determinado (el 15).
Los comandos disponibles, una vez abierto el canal de comunicación son:

NEW: Borra completamente el disquete, creando un nuevo BAM y un nuevo directorio.
Formato: PRINT# 15, "NEW:nombre-disquete""

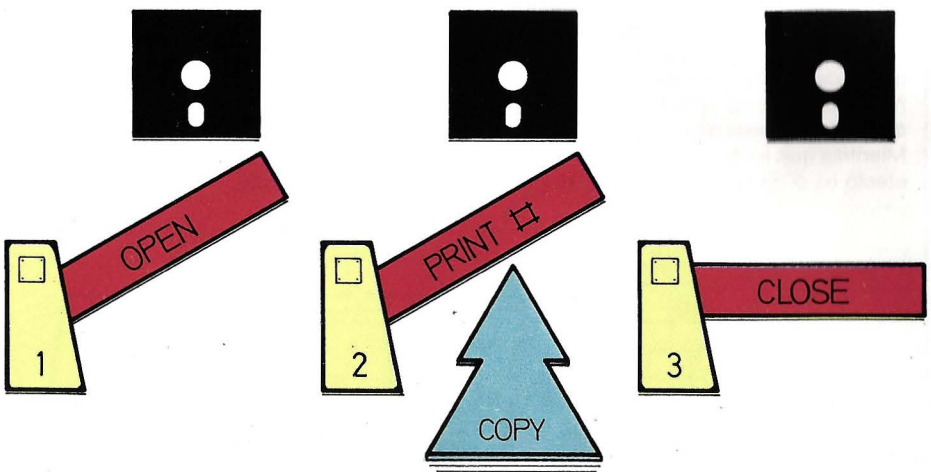
COPY: Copia un fichero o programa ya existente en otro nuevo.
Formato: PRINT# 15, "COPY: fichero-nuevo=fichero-viejo"

RENAME: Cambia el nombre de un fichero presente en el directorio.
Formato: PRINT# 15, "RENAME:nombre-nuevo=nombre-antiguo"

SCRATCH: Borra un fichero del directorio, dejando bloques libres.
Formato: PRINT# 15, "SCRATCH:nombre-fichero"

INITIALIZE: Deja la unidad de disco tal como se encontraba en el momento de conectarla.
Formato: PRINT# 15, "INITIALIZE"

VALIDATE: Elimina el desorden introducido en el directorio después de muchas creaciones y borrados de ficheros.
Formato: PRINT# 15, "VALIDATE"



El envío de comandos desde un programa BASIC al sistema operativo de disco, se produce con órdenes semejantes a las utilizadas para el tratamiento de ficheros.

CLOSE: Cierra el canal de comunicación con la unidad de disco.
Formato: CLOSE# 15.

TIPOS DE FICHEROS

Los distintos métodos de organización de datos en los ficheros nos lleva a disponer de tres posibles tipos de ficheros:

- Secuenciales: se caracterizan por almacenar los elementos de información uno a continuación de otro y separados por marcas. El acceso a un elemento de información sólo es posible si se ha pasado previamente por todos aquellos que lo preceden.
- De acceso directo: en este caso la nomenclatura es engañosa, pues estos ficheros caen fuera de la definición clásica de ficheros de acceso directo, ya que el único acceso aleatorio que permiten es a nivel de bloque, por lo que son de poca utilidad en programas que manejan información en volúmenes de menos de un bloque.
- Relativos: equivalen a los ficheros de acceso directo clásico; estos ficheros son estructurables en registros y éstos en campos. El S.O. lleva el control de los bloques utilizados, olvidándose el usuario de éstos para centrarse únicamente en los registros y campos.

OBSERVACIONES

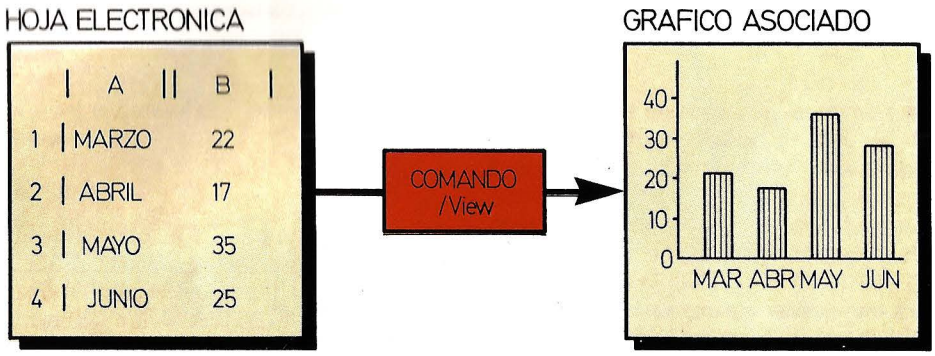
La unidad de disco flexible 1541, junto con el sistema operativo de disco incluido en ella, permite gracias a sus conectores en serie el encadenamiento de hasta 5 unidades de disquete y una impresora. Ello proporciona al sistema capacidad para enfrentarse con problemas de cierta envergadura, aunque la carencia de protección en los ficheros y de una estructura con directorios jerárquicos hace que pueda presentarse algún inconveniente en aplicaciones de gestión.

SuperCalc 3 (y 2)

Descripción de comandos y sesión práctica

El capítulo precedente inició el estudio de la hoja electrónica SUPERCALC 3. En él se vieron las características generales del programa y los distintos grupos de funciones que se encuentran a disposición del usuario para efectuar los cálculos necesarios.

Ahora se completa el estudio describiendo los principales comandos que SUPERCALC 3 ofrece para la gestión de la hoja electrónica. Con objeto de no convertir estas páginas en una simple receta, la descripción de los comandos se hará de forma general, sin entrar en detalles específicos.



La base informativa para la producción de gráficos se almacena en la hoja electrónica. El comando VIEW se limita a presentar el gráfico asociado.

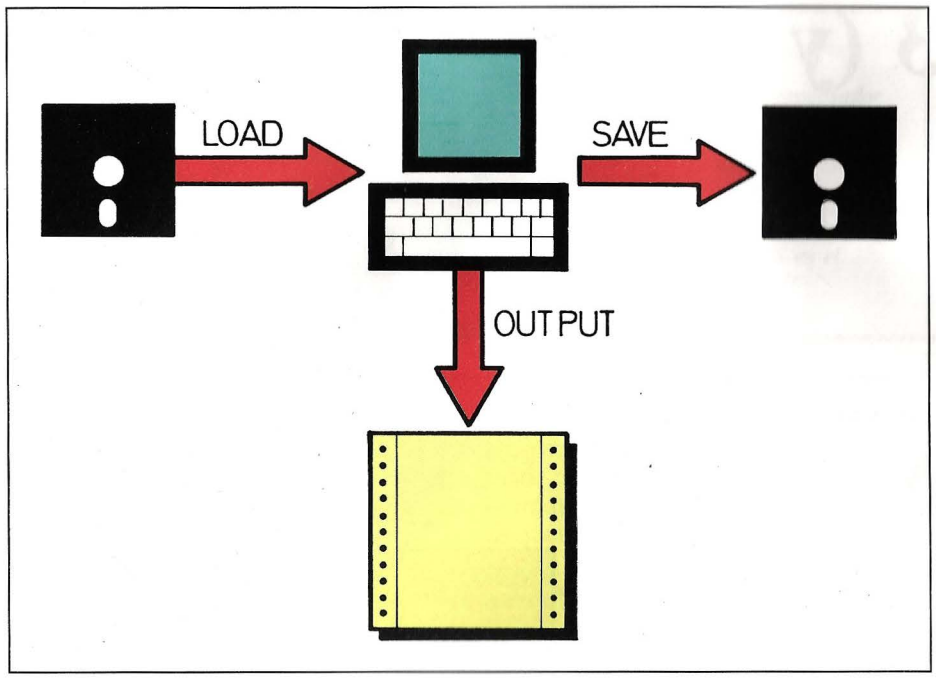
COMANDOS DEL SUPERCALC 3

El número de comandos que incorpora el programa es notorio y, en algunos casos, las funciones que desempeñan son similares a las que ya hemos descrito para otras hojas electrónicas. Tal vez la mayor novedad estribe en los comandos dedicados a la producción de gráficos. Dichos comandos los estudiaremos con rigor, mientras que de los restantes nos limitaremos a esbozar los más importantes.

GRAFICOS CON SUPERCALC 3

La producción de gráficos con SUPERCALC 3 se basa en el contenido de la hoja

SUPERCALC es un programa de hoja electrónica del que existen distintas versiones, algunas tan evolucionadas como la denominada SUPERCALC 3 analizada en estas páginas.



Los tres comandos para la comunicación no interactiva de Supercalc 3 permiten obtener informes escritos, leer y escribir en soportes externos de memoria.

electrónica; esto es: mediante la utilización del comando /VIEW se puede conseguir que los datos almacenados en la matriz se representen en forma de diagrama. Cuando el usuario ejecuta el comando VIEW, el programa le mostrará un mensaje en el que aparecerán nueve opciones distintas:

1. (≠) Numeración de gráficos.
La primera opción sirve para que el usuario asigne un número al gráfico producido; de esta forma se pueden tener numerados hasta nueve gráficos simultáneamente.

2. (?) Descripción.
Mediante la segunda opción del comando de gráficos se puede visualizar la información que describe las características de los gráficos definidos.

3. (Data) Descripción de datos.
La información de la hoja electrónica está contenida en una gran matriz, con parte de la cual se debe obtener una representación gráfica. Mediante la opción DATA del comando VIEW, el usuario decide el subconjunto de elementos de la hoja electrónica que se tendrán en cuenta para construir el diagrama elegido.

4. (Graph-Type) Tipo de gráfico.
Una vez definidos los elementos de la hoja electrónica que contienen la información a representar gráficamente, el usuario debe indicar el tipo de diagrama que desea producir. Existen siete posibilidades distintas:

— Diagrama de tarta.
— Diagrama de barras.
— Diagrama de barras superpuestas.
— Diagrama de líneas.
— Diagrama HI-10.
— Diagrama de puntos.
— Diagrama de superficie.

Mediante esta opción el usuario podrá elegir cómodamente el tipo de representación gráfica deseada, e incluso cambiarla en el caso de que al visualizarlo varíe su opinión en cuanto al tipo de gráfico más apropiado. Estas operaciones se realizarán automáticamente, sin necesidad de reintroducir o asignar de nuevo los subconjuntos de la hoja electrónica de donde se tomarán los valores.

5. (Time-Labels) Etiquetas de marca.
Además de los datos que contiene la información que dará lugar al gráfico elegido, el usuario puede marcar otro subconjunto de elementos de la hoja electrónica donde estarán almacenados los valores numéricos o alfabéticos que se utilizarán como etiquetas para el eje X del diagrama; por ejemplo: los nombres de las barras, los valores utilizados como marcas en un diagrama de líneas, etc.

6. (Variable-Labels) Etiquetas de variables.
Análogamente, en otro subconjunto de elementos de la hoja electrónica se pueden asignar las etiquetas que identificarán a las distintas variables representadas; por ejemplo, para cada una de las porciones de una tarta.

7. (Point-Labels) Etiquetas de punto.
Al igual que las dos opciones anteriores, en este caso se pueden asignar subconjuntos de la hoja electrónica como etiquetas de puntos: por ejemplo, para cada par X-Y de un diagrama de puntos.

8. (Headings) Cabeceras
Cada diagrama puede tener asignado un conjunto de caracteres que aparecerán en la representación gráfica; mediante esta opción se puede especificar, además de la cabecera propiamente dicha, su ubicación dentro del diagrama.

9. (Options) Opciones.
La última posibilidad dentro del comando VIEW, consiste en la definición de una serie de opciones de carácter general para el gráfico a producir, como por ejemplo: formatos, escalas, etc.

OTROS COMANDOS

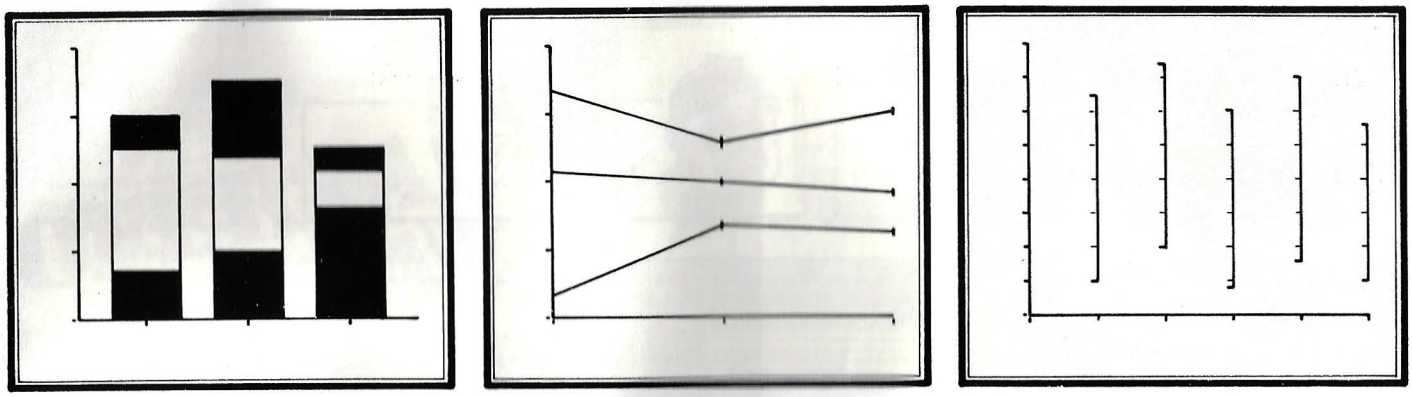
Además del comando VIEW para la obtención de gráficos, SUPERCALC 3 dispone de numerosos comandos para gestionar la hoja electrónica. Vamos a citar algunos de ellos.

- BLANK

Permite eliminar la información contenida en un elemento o en varios de la hoja electrónica.

- COPY

En determinados casos resulta intere-



Además de los típicos diagramas de barras, líneas, tartas y puntos (ya mostrados en el capítulo anterior), Supercalc 3 puede producir las representaciones que se muestran en esta figura: barras superpuestas, superficies y Hi-Lo.

sante duplicar el contenido de una parte de la hoja electrónica en otra zona de la matriz. Para ello SUPERCALC 3 ofrece el comando COPY.

- DELETE

Como su propio nombre indica (DELETE en inglés significa borrar) la misión de este comando consiste en borrar el contenido de un elemento o conjunto de elementos.

- FORMAT

Si se desea que la información contenida en una parte de la hoja electrónica adopte un formato especial, se debe utilizar el comando FORMAT el cual permite las siguientes opciones:

— Formato entero.
— Formato general.
— Formato exponencial.
— Formato dólar.
— Formato con ajuste a la derecha.
— Formato con ajuste a la izquierda.
— Formato con representación lineal.

- LOAD

Se encarga de leer una hoja electrónica contenida en un archivo externo y la carga sobre la matriz de la hoja electrónica ubicada en la memoria principal.

- OUTPUT

Permite obtener copias impresas de todos o parte de los elementos de la hoja electrónica activa.

- QUIT

Sirve para finalizar la sesión de trabajo con SUPERCALC 3 y, en consecuencia, devuelve el control del ordenador al sistema operativo.

- SAVE

El comando SAVE, complementario de LOAD, permite almacenar en un archivo sobre soporte externo el contenido de la hoja electrónica activa.

- WINDOW

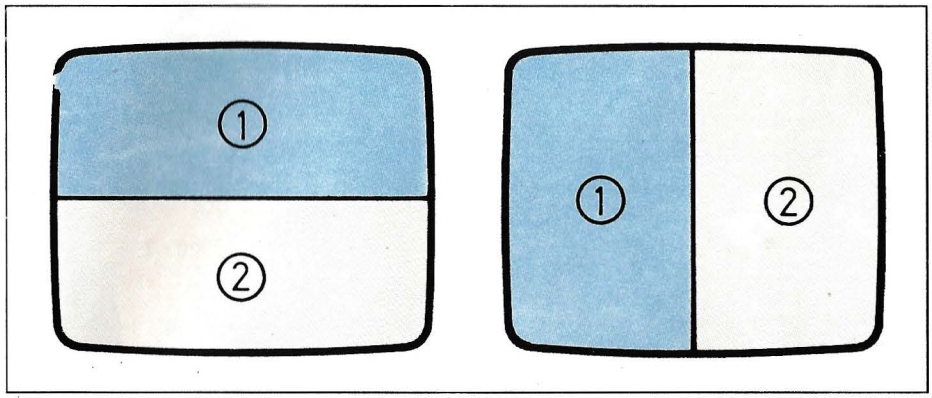
Este comando, casi tradicional en los modernos paquetes de software horizontal, permite definir ventanas y simultanear el trabajo con dos hojas electrónicas.

SESION PRACTICA DE TRABAJO

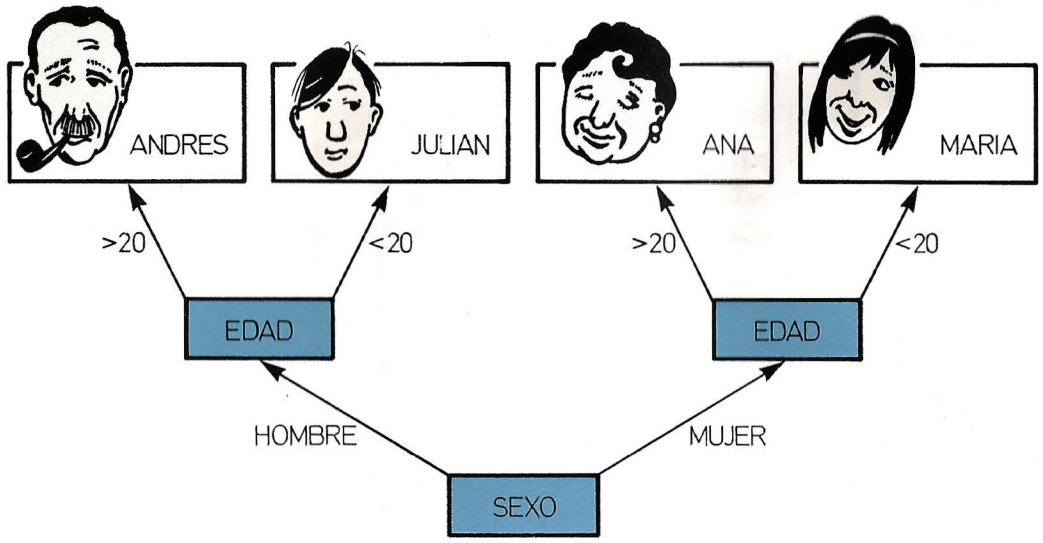
Para finalizar vamos a describir una sesión práctica de trabajo con SUPERCALC 3. Nuestro objetivo consiste en diseñar un modelo capaz de "adivinar" el nombre de un personaje en función de algunas de sus características. Suponga, para simplificar el problema, que tan sólo existen cuatro personas entre las que hay que elegir: Julián de 17 años, María de 19, Andrés de 45 y Josefa de 42. Antes de analizar la estructura de la hoja electrónica, estudiemos cómo se resolvería el problema manualmente. En primer lugar podríamos preguntar si se trata de un hombre o de una mujer y, a cotinuación, si tiene más de 20 años. Evidentemente, con una contestación para estas dos preguntas tendremos suficiente información para deducir cuál de los personajes es el buscado.

Para plasmar este razonamiento tan sólo son necesarios dos elementos de la hoja electrónica que podemos fijar en cualquier posición; por ejemplo, en D 7 introduciremos 1 si se trata de un hombre y 2 en el caso de que sea una mujer, y en la posición D 9 teclearemos la edad del personaje a adivinar.

En las casillas D 15, D 16, D 17 y D 18,



Mediante el comando WINDOW, el usuario de Supercalc 3 puede dividir la pantalla en dos ventanas lógicas, dispuestas horizontal o verticalmente.



La representación arborescente resulta ideal para diseñar modelos lógicos: los nodos del árbol "son" las preguntas a realizar, las ramas "son" las posibles respuestas y las hojas "son" las conclusiones.

```
-----------------------------------------
-                                       -
-    DECISIONES          LOGICAS        -
-                                       -
-----------------------------------------
-                                       -
- SEXO DEL PERSONAJE . . .    ?(07)     - 1 --> HOMBRE, 2 --> MUJER
-                                       -
- EDAD DEL PERSONAJE . . .    ?(09)     -
-                                       -
-----------------------------------------
-            RESPUESTA                  -
-----------------------------------------
-                                       -
-      JULIAN . . . . . AND (D7=1,D9<20) -
-      ANDRES . . . . . AND (D7=1,D9>20) -
-      MARIA . . . . . AND (D7=2,D9<20)  -
-      JOSEFA . . . . . AND (D7=2,D9>20) -
-                                       -
-----------------------------------------
```

La hoja electrónica diseñada como ejemplo en este artículo se basa en dos elementos incógnita D7 y D9. En función de ellos y mediante las fórmulas lógicas indicadas, se calculan las respuestas a producir.

```
-----------------------------------------
-                                       -
-    DECISIONES          LOGICAS        -
-                                       -
-----------------------------------------
-                                       -
- SEXO DEL PERSONAJE . . .       1      - 1 --> HOMBRE, 2 --> MUJER
-                                       -
- EDAD DEL PERSONAJE . . .      30      -
-                                       -
-----------------------------------------
-            RESPUESTA                  -
-----------------------------------------
-                                       -
-      JULIAN . . . . . .     FALSE      -
-      ANDRES . . . . . .     TRUE       -
-      MARIA . . . . . .      FALSE      -
-      JOSEFA . . . . . .     FALSE      -
-                                       -
-----------------------------------------
```

En la figura se puede apreciar cómo el ordenador adivina el personaje ANDRES, sin más que evaluar las funciones AND incluidas en el diseño anterior.

podemos situar cuatro fórmulas lógicas que, en función de los dos parámetros de entrada, valgan VERDADERO (TRUE) o FALSO (FALSE). Si el modelo funciona correctamente, tan sólo una de ellas deberá tomar el valor TRUE, precisamente en la línea correspondiente al personaje adivinado.

Suponga que asociamos D 15 a Julián, D 16 a Andrés, D 17 a María y D 18 a Josefa. Ya sólo nos falta elegir la fórmula apropiada en cada caso; para ello optaremos por la función ADN y como parámetros suyos incluiremos las siguientes condiciones:

D 15: SEXO = HOMBRE (D 7 = 1) y EDAD < 20 (D 9 < 20)

D 16: SEXO = HOMBRE (D 7 = 1) y EDAD > 20 (D 9 > 20)

D 17: SEXO = MUJER (D 7 = 1) y EDAD < 20 (D 9 < 20)

D 18: SEXO = MUJER (D 7 = 1) y EDAD > 20 (D 9 > 20)

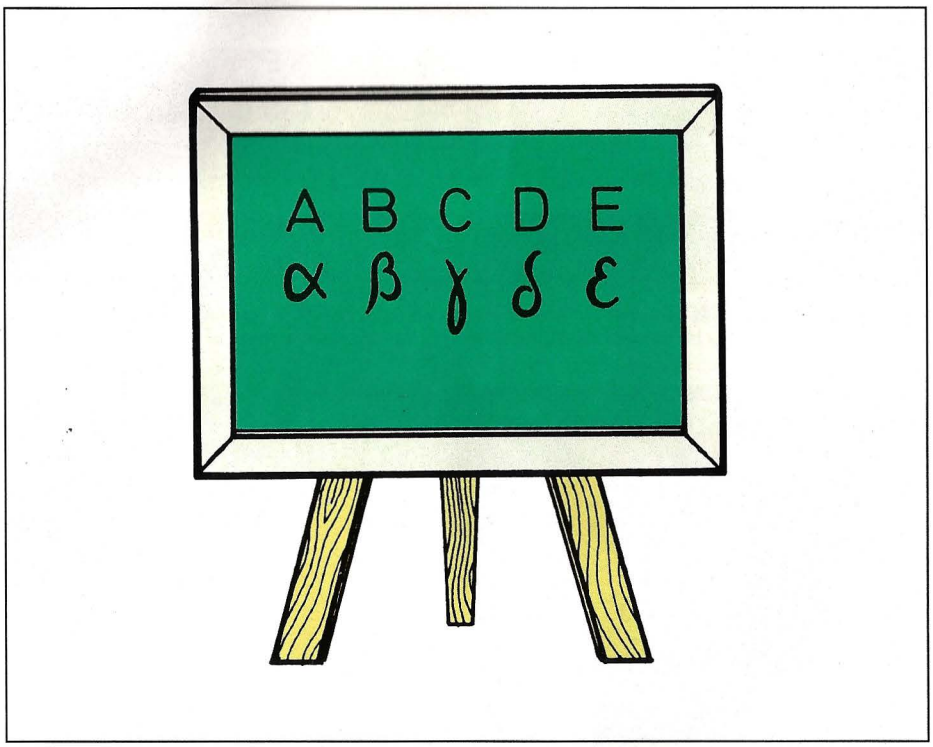
Para terminar el modelo del diseño sólo queda adornar la hoja electrónica con cabeceras y literales que identifiquen las casillas que contendrán las preguntas y las que mostrarán las respuestas. Para finalizar la sesión de trabajo, grabaremos el modelo en su soporte externo, de forma que se pueda recuperar en otras sesiones y, tras ellos, buscaremos a algún espectador para que se maraville ante la facilidad con la que toma decisiones lógicas nuestro ordenador.

Al margen del carácter lúdico del ejemplo, se pueden utilizar modelos similares capaces de ayudarnos a tomar decisiones más importantes y complicadas. Para ello es suficiente con sustituir las preguntas y respuestas y con modificar la lógica inherente a las funciones.

Otras herramientas gráficas

Caracteres programables y «Sprites»

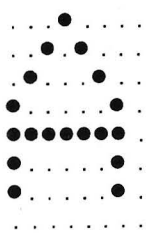
Entre las posibilidades gráficas comentadas hasta el momento se encuentran el trazado de rectas y curvas y el manejo de los colores. Todo ello se realiza por medio de los comandos específicos de que está dotado el ordenador. Otro método para crear dibujos reside en el uso de los denominados caracteres semigráficos. Estos caracteres especiales permiten plasmar figuras predefinidas a través de un comando PRINT. Los caracteres semigráficos suelen incluir pequeños redondeles y cuadrados del tamaño de un carácter. En el presente capítulo se aborda la redefinición de caracteres y otras herramientas que permiten un uso más completo de las posibilidades de la pantalla

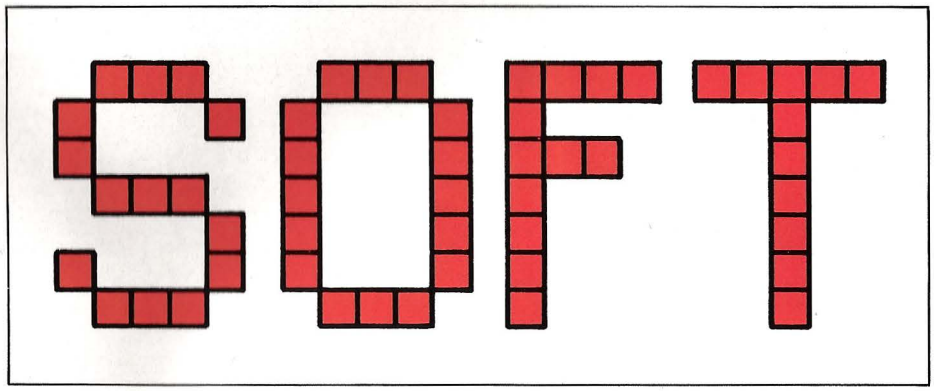


Un nuevo método para confeccionar dibujos desde el control del BASIC consiste en alterar la forma de los caracteres que integran el repertorio del ordenador.

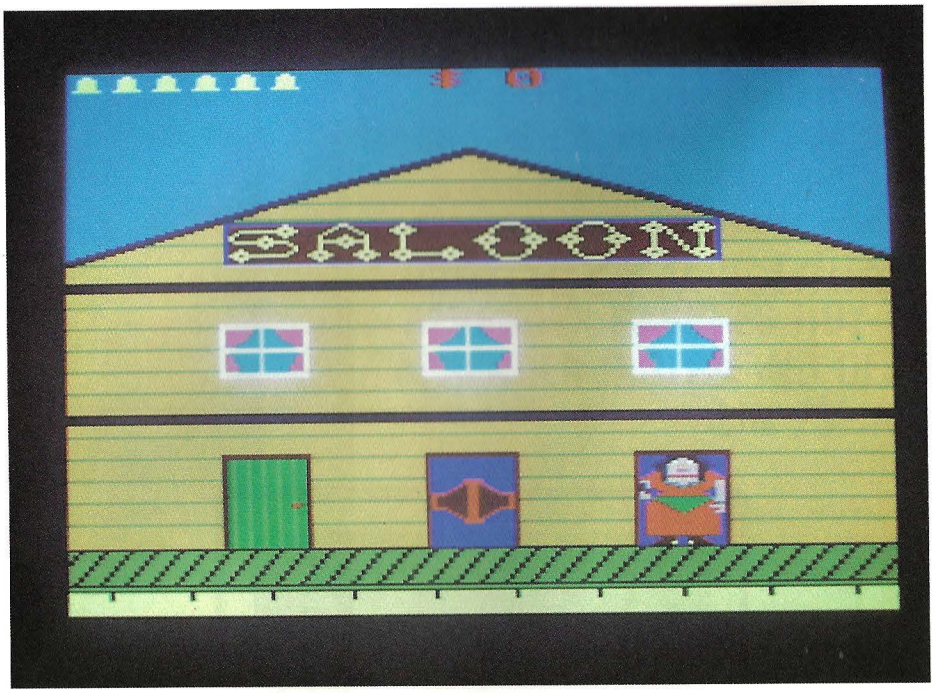
LOS CARACTERES

Los caracteres que presenta el ordenador en la pantalla están formados por puntos. La disposición de estos puntos da lugar a la "matriz" del carácter. Por regla general, la matriz consiste en un cuadrado de ocho por ocho puntos. Cada punto puede estar encendido o apagado, dando lugar así al carácter a representar. Por ejemplo, el carácter "A" puede representarse mediante la siguiente matriz:





Los caracteres que muestra la pantalla del ordenador están formados por una matriz de puntos luminosos.



Aspecto de una pantalla de juego del microordenador AMSTRAD. El correspondiente programa hace uso de los sprites para representar y dar animación a los personajes.

Un plato fuerte de la carta BASIC es el que reúne a las herramientas gráficas adecuadas para la programación de caracteres y sprites.

En donde el símbolo "." indica un punto apagado y el símbolo "●" encendido. Es norma usual dejar libres la última fila y columna para que se cree una separación entre caracteres contiguos. Como se puede apreciar, cada línea tiene ocho puntos. Esto quiere decir que cada una de las filas que forman el carácter puede almacenarse en un solo byte. Por lo tanto, para almacenar la imagen de un carácter se necesitan ocho bytes.

El ordenador no conoce de por sí la imagen de cada carácter, y por ello ha de tener una zona de memoria dedicada al almacenamiento de dichas imágenes. Esta zona suele estar localizada en la memoria ROM (de sólo lectura) y se denomina "banco de caracteres". Cuando es necesario representar un carácter en pantalla, el ordenador busca la imagen en el banco de caracteres y la proyecta en la posición adecuada de la pantalla.

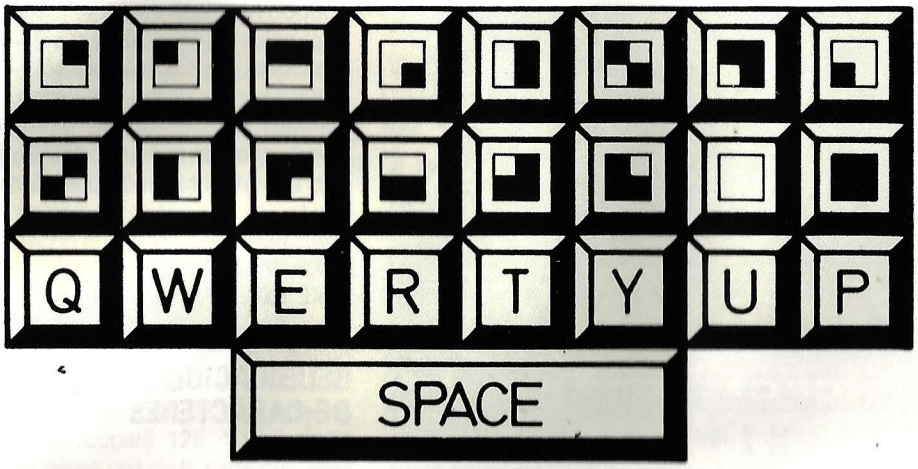
De lo señalado se deduce que para crear un nuevo juego de caracteres bastaría con variar el contenido de los bytes del banco de caracteres. Desafortunadamente esto no es posible, ya que dicho banco se encuentra almacenado en ROM (la zona de memoria cuyo contenido no es posible alterar). De todas formas existen "trucos" para solventar ese problema.

PROGRAMANDO CARACTERES

Supongamos por un momento que el banco de caracteres se encontrará en memoria RAM (memoria cuyo contenido sí puede ser alterado). En tal caso, para cambiar un carácter bastaría con "encender" o "apagar" los bits oportunos de las posiciones de memoria en las que se encuentra almacenado el carácter en cuestión.

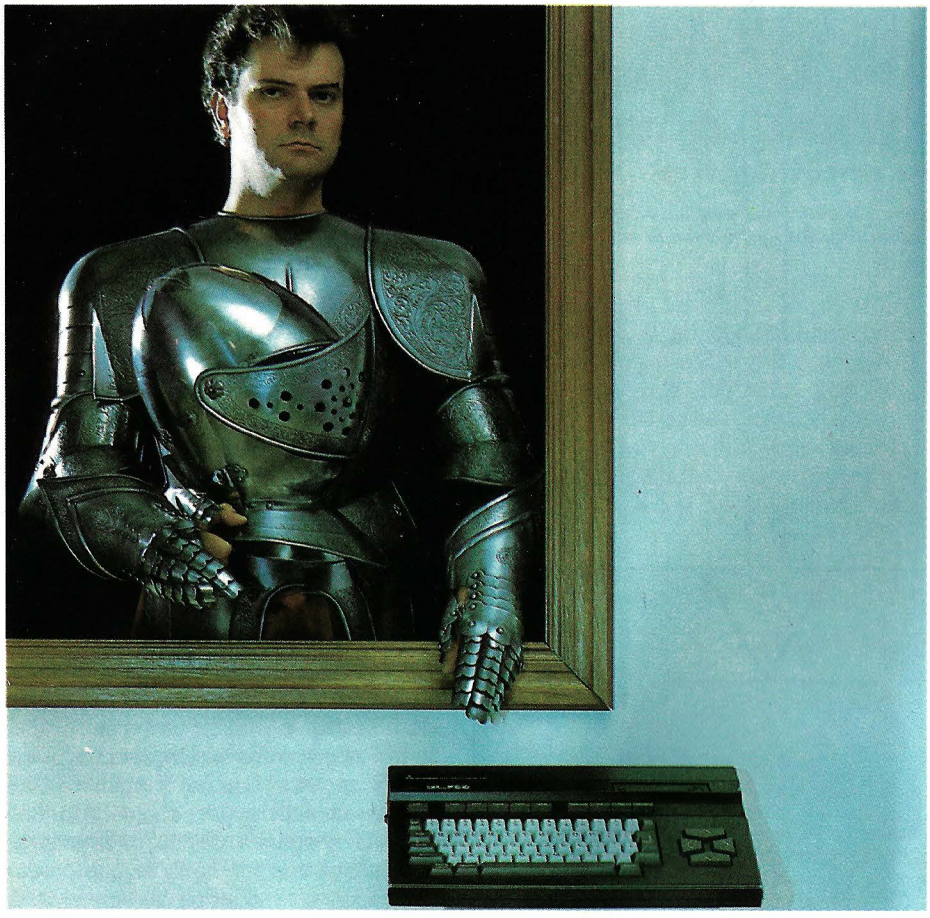
El comando BASIC que permite el acceso a posiciones individuales de memoria es POKE. Como se vio en su momento, este comando permite colocar un entero (de 0 a 255) en una determinada posición de memoria. El uso de POKE no facilita el acceso a cada bit individualmente, sino al byte en conjunto. Ello obliga al cambio de una fila completa de la matriz del carácter.

El siguiente paso consiste en identificar el efecto que produce el almacenamiento de un entero en una de estas posiciones de



Algunos ordenadores disponen de un juego de caracteres semigráficos que permiten realizar dibujos sencillos.

memoria. La cosa no puede ser más sencilla: la sucesión de ceros y unos (bits apagados o encendidos) de cada fila se corresponde con el entero introducido, pero escrito en el sistema binario. Por ejemplo, el almacenamiento del número 9 en una de esas posiciones produciría una línea de este tipo:



Las técnicas para la definición y tratamiento de sprites permiten crear formas gráficas y controlar su "animación" sobre la pantalla conectada al ordenador.

. . . .●. .●

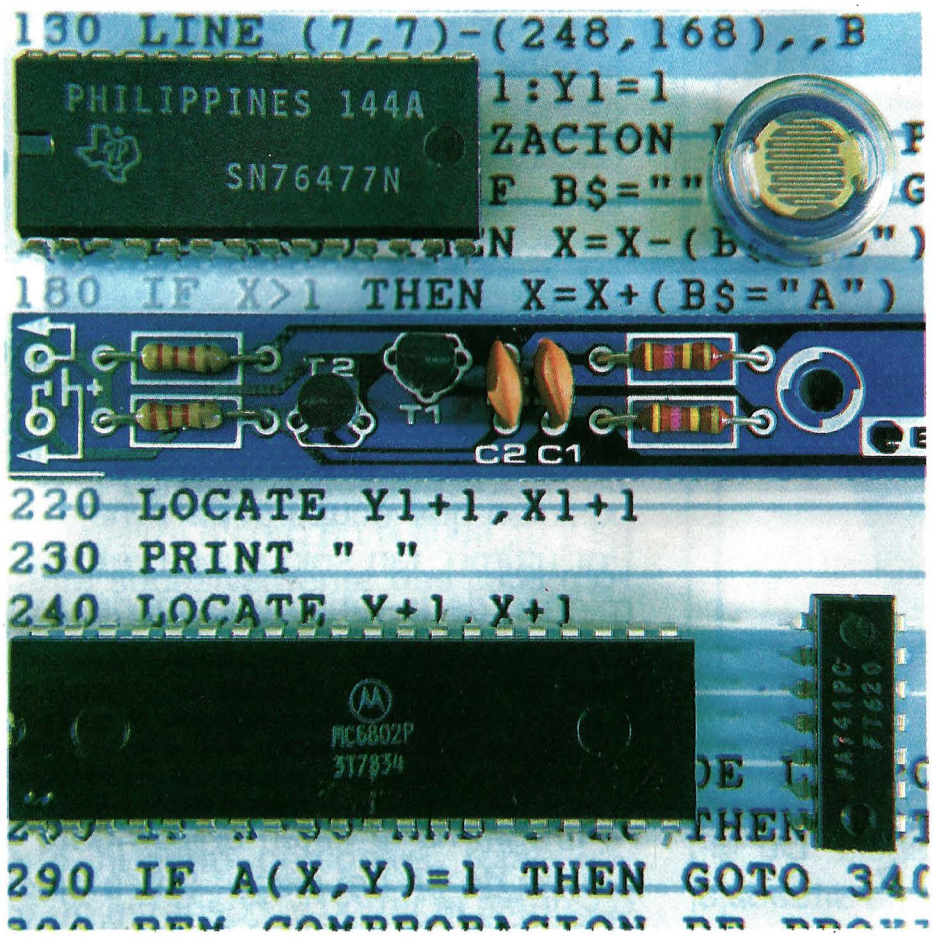
Ello se debe a que la configuración binaria 00001001 (o más sencillamente 1001) corresponde al número decimal 9. Así pues, conociendo el paso de binario a decimal se puede alterar la imagen de un carácter sin más que ejecutar los adecuados POKES.

Para aclarar algo más las ideas se va a mostrar un ejemplo de lo anteriormente mencionado. Se parte de la base de que los caracteres se encuentran en RAM. El carácter a alterar es el "A", que por comodidad supondremos situado en la posición número 1000. Las posiciones de memoria que interesan son las siguientes:

1000 . . .●. . . .
1001 . .●.●. . .
1002 .●. . .●. .
1003 ●. . . . .●.
1004 ●●●●●●●.
1005 ●. . . . .●.
1006 ●. . . . .●.
1007 . . . . . .

Tras el cambio, se desea que el carácter adquiere el siguiente aspecto:

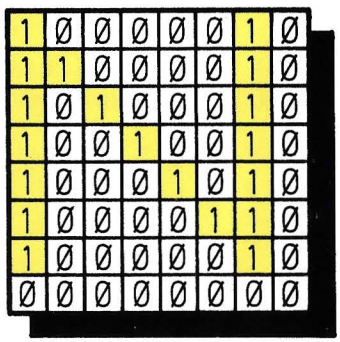
1000 .●●●●●. .
1001 ●. . . . .●.
1002 ●. . . . .●.
1003 ●. . . . .●.
1004 ●●●●●●●.
1005 ●. . . . .●.
1006 ●. . . . .●.
1007 . . . . . .



El fundamento técnico que otorga a cualquier equipo doméstico sus amplias y versátiles facultades gráficas, se encuentra en los "chips" especializados que apoyan la actividad del microprocesador.

Es decir, sólo se alterarán las tres primeras filas. El paso siguiente consiste en calcular los números decimales que corresponden a los binarios 01111100 y 10000010. Para ello emplearemos una sencilla regla: a cada "uno" que aparece en el número binario se le asigna una cantidad dependiendo de la posición en la que se encuentra; luego, basta con sumar esas cantidades para obtener el valor decimal. Las cantidades claves coinciden con las ocho primeras potencias de 2. Contando de derecha a izquierda, los respectivos unos valen 1, 2, 4, 8, 16, 32, 64 y 128. De esta forma, el número 01111100 vale 4 + 8 + 16 + 32 + 64 = 124 y el 10000010 vale 2 + 128 = 130: los valores decimales 124 y 130, respectivamente.
Tras este sencillo cálculo sólo queda por almacenar los datos calculados en las posiciones adecuadas. Esto último se realiza por medio del comando POKE. Estas serían las instrucciones oportunas:

```
POKE 1000,124
POKE 1001,130
POKE 1002,130
```



La forma de cada carácter está en correspondencia con un determinado conjunto de ceros y unos almacenados en memoria.

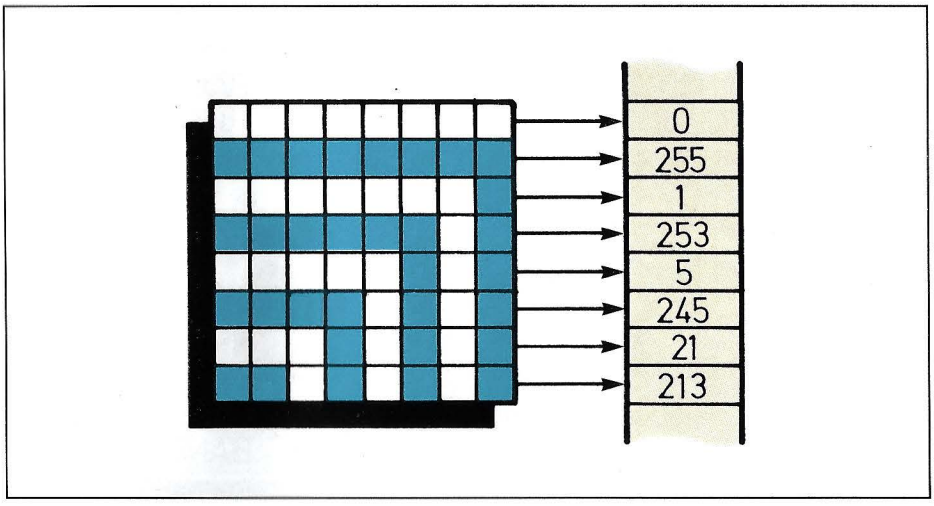
REUBICACION DEL BANCO DE CARACTERES

Anteriormente se habló de la imposibilidad de variar la imagen de los caracteres contenidos en ROM. No obstante, puede realizarse un "truco" que permite llevar a cabo la alteración de los caracteres. Para ello es necesario conocer algo más acerca del interior de la máquina.
El banco de caracteres se encuentra situado a partir de una determinada posición de memoria. El ordenador accede a la imagen de cada carácter sumando al número que indica esa posición inicial la distancia que separa a la imagen de esa primera posición.
Por ejemplo, sea N la primera posición del banco de caracteres. Para acceder al primer carácter se suma N + 0 dando lugar a la posición primera de ese carácter. Para el segundo, se ha de calcular N + 8 (ya que las 8 primeras posiciones pertenecen al primero). En general, para determinar la posición del carácter número X se ha de calcular N + (8 * X).
El número que indica el inicio del banco de caracteres (el denominado N) suele estar almacenado en una de las variables del sistema. Estas se encuentran en RAM y, por lo tanto, es posible alterarlas. Una vez identificada la variable del sistema que contiene esa información ésta se puede modificar por el uso del comando POKE. Si se altera esa variable, el ordenador buscará las imágenes de los caracteres en otro lugar de la memoria, lo que proporcionará unas imágenes completamente aleatorias de los mismos. Ello puede remediarse, claro está; e incluso aprovechar

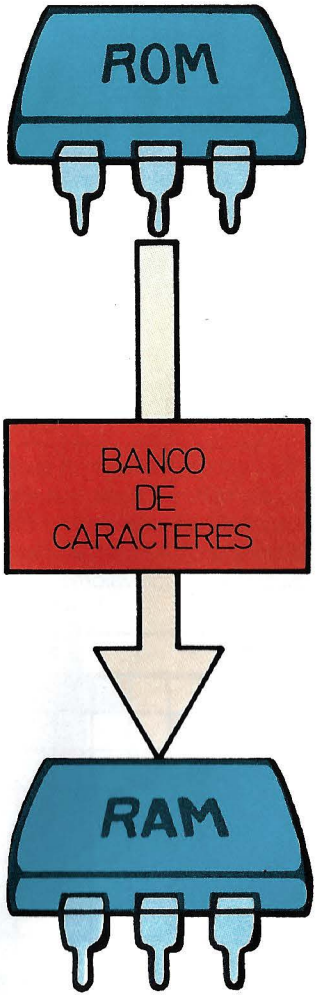
esta posibilidad para nuestro objetivo, cual es modificar el repertorio de caracteres del ordenador.
Para hacer un uso provechoso de la referida variable del sistema, es conveniente que a partir de la dirección introducida en ella exista una copia del banco de caracteres. Los pasos a seguir son los siguientes: identificar una posición de memoria que sea propicia, copiar el banco de caracteres a partir de dicha posición y alterar al contenido de la variable del sistema para que apunte a la nueva dirección.
El primer paso revela que no vale cualquier dirección. Ello se debe a la longitud del banco de caracteres. Si el ordenador dispone de un repertorio de 125 caracteres, el banco ocupará 125 * 8 = 1024 bytes (ocho bytes por cada carácter). Para copiar el banco se necesitará, pues, una zona de 1024 bytes libres. Esto significa que la nueva posición de memoria debe estar al comienzo de una zona RAM de 1024 bytes, y que la alteración de dichos bytes no debe producir ningún efecto desastroso (desde luego, dicha zona no debe coincidir con la destinada al almacenamiento del programa, a las variables del sistema, etc.).
Una vez determinada la posición de memoria idónea, habrá que situar a partir de ella las imágenes de los caracteres. Esto último se puede realizar por medio de un programa BASIC. En realidad, lo único que ha de hacer el programa es copiar el contenido de unas posiciones de memoria en otras. A continuación se muestra una posible rutina adecuada para copiar del banco de caracteres.

```
10 REM REUBICADOR DE CARACTERES
20 INPUT "POSICION ORIGINAL";PO
30 INPUT "NUEVA POSICION";NP
40 FOR I=0 TO 1023
50 POKE NP+I,PEEK(PO+I)
60 NEXT I
```

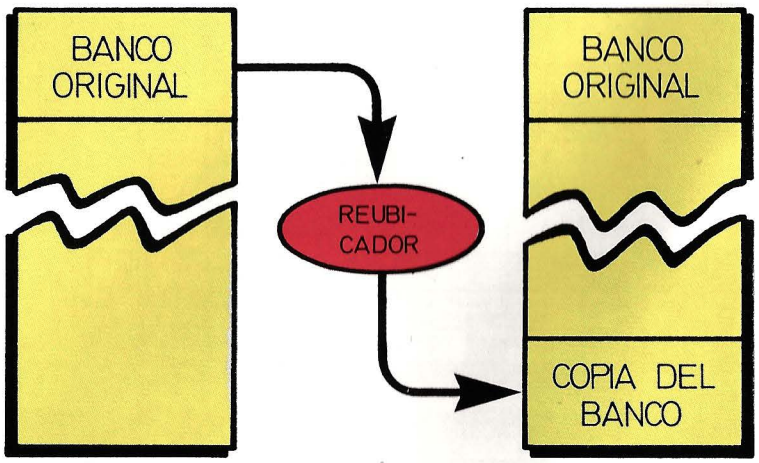
Tras la ejecución de esta rutina se dispondrá de un juego de caracteres alternativo a partir de la posición de memoria introducida como respuesta al comando INPUT de la línea 30. El bucle FOR hace variar a I de 0 a 1023. Con ello, la expresión PO + I recorre los 1024 bytes del banco de caracteres. La función PEEK proporciona el contenido de cada una de esas posiciones de memoria, valor éste que se deposita en la correspondiente nueva posición



Cada fila de un carácter ocupa una posición de memoria. En consecuencia, un carácter completo necesita 8 bytes de memoria para su almacenamiento.



Para que sea posible alterar el repertorio de caracteres, éste ha de ser previamente trasladado de memoria ROM a RAM.



El programa reubicador descrito en el texto permite efectuar una copia del banco de caracteres en otra zona de memoria.

NP + I mediante el comando POKE de la línea 50.
Inmediatamente después de la ejecución de la rutina reubicadora se puede cambiar el contenido de la variable del sistema por el valor de la nueva posición. Ello permite conmutar el banco antiguo por el nuevo. La reubicación y posterior conmutación de bancos da como resultado un nuevo banco de caracteres en RAM. O lo que es lo mismo: un banco de caracteres alterables. La programación de dichos caracteres puede ya efectuarse siguiendo las normas comentadas en los anteriores párrafos.

USO DE LOS CARACTERES PROGRAMADOS

En algunos ordenadores resulta ocioso el método de la reubicación del banco de caracteres. Ello se debe a que en algunos aparatos (como es el caso del Spectrum de Sinclair) una parte de los caracteres se almacena en memoria RAM. Con esto se permite su fácil programación por parte del usuario. El método a seguir para alterar esos caracteres es idéntico al que se utiliza tras el proceso habitual de la reubicación.
Los caracteres programados son fácilmente utilizables, sin más que introducirlos en el argumento de un comando PRINT. Esto permite posicionarlos en cualquier lugar de la pantalla. También es posible su movimiento, por el método de ir imprimiéndolos en posiciones sucesivas de la pantalla.
Por regla general, los caracteres programados se agrupan para formar una figura mayor. En ese caso, cada carácter contendrá sólo una parte de la figura final. A la hora de su representación, las distintas partes se sitúan contiguas en el argumento de PRINT. Si las cuatro cuartas partes del dibujo de una cara se almacenan en los caracteres A, B, C y D, podrían utilizarse las dos líneas PRINT que siguen para visualizar correctamente el resultado:

```
10 PRINT "AB"
20 PRINT "CD"
```

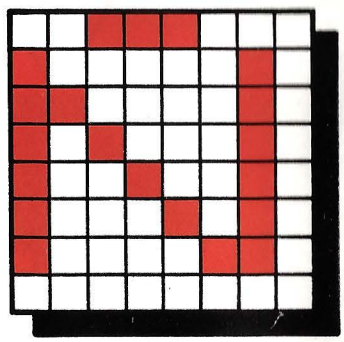
Para situar el dibujo en cualquier punto de la pantalla se hace uso de la opción AT asociada al comando PRINT (o del comando LOCATE en su caso).
Si se tienen los valores que indican la fila y la columna en las variables Y y X, respectivamente, puede emplearse la siguiente rutina posicionadora:

```
100 PRINT AT(X,Y); "AB"
110 PRINT AT(X,Y+1); "CB"
```

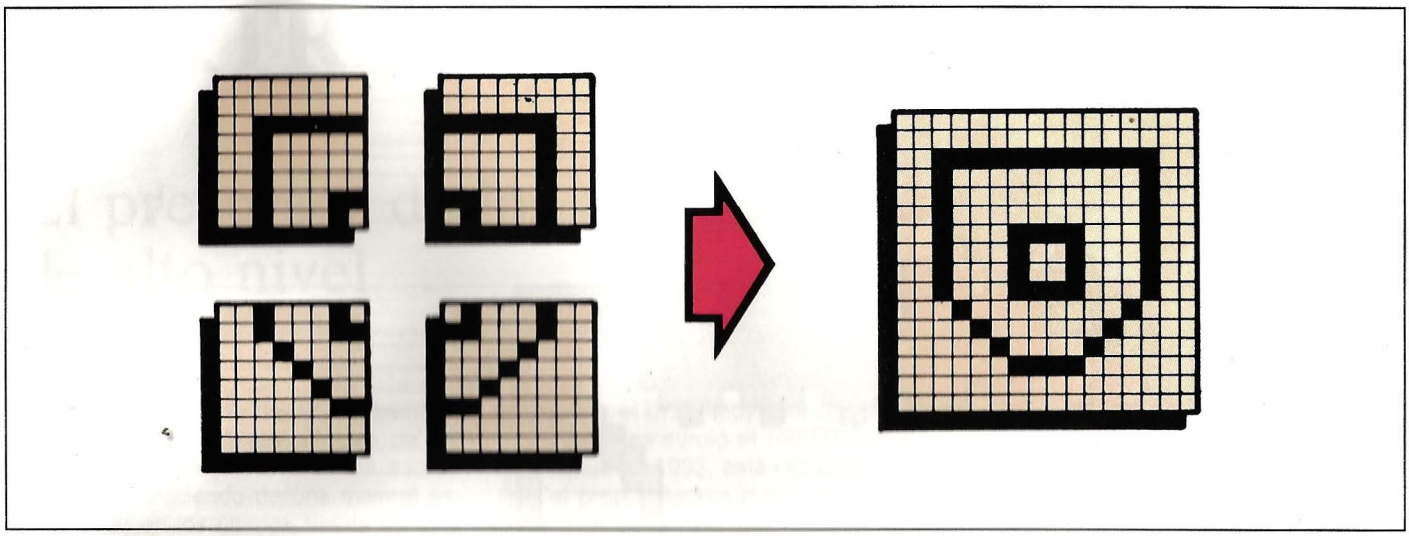
Una rutina semejante puede utilizarse con figuras de otro tamaño. Con una rutina de posicionamiento de este tipo es posible simular el movimiento de la figura variando los valores de X e Y.

SPRITES

La redefinición o programación de caracteres permite crear y utilizar con facilidad figuras móviles. Sin embargo, esta técnica



Una posible aplicación de los caracteres definidos es la creación de caracteres especiales no existentes en el repertorio original del ordenador; por ejemplo, la letra Ñ del castellano.



La agrupación de varios caracteres puede utilizarse para construir dibujos de mayores proporciones.

tiene sus limitaciones. Una de ellas es el hecho de que un carácter sólo puede situarse en la intersección de una fila y una columna de texto. Esto hace que el movimiento más suave posible se realice en saltos de ocho pixels (anchura de cada carácter).

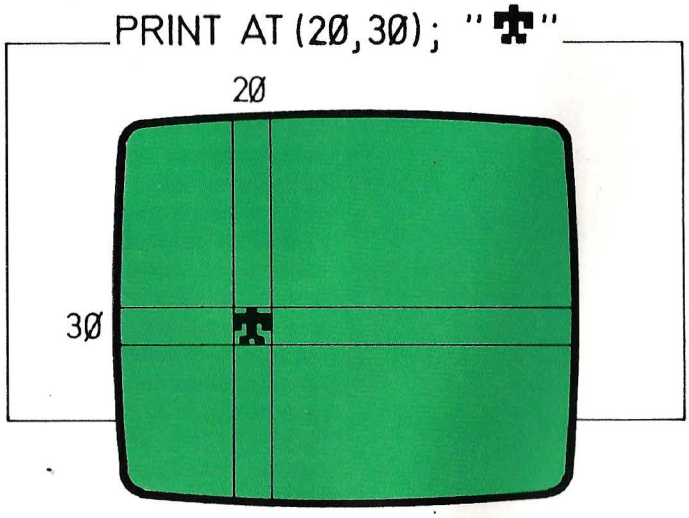
Para solventar este y otros problemas, en los más modernos ordenadores se hace uso de una nueva posibilidad: los denominados "sprites".

Un sprite no es más que un carácter programable mejorado. Entre esas mejoras cabe destacar la posibilidad de movimiento pixel a pixel y la detección de colisiones.

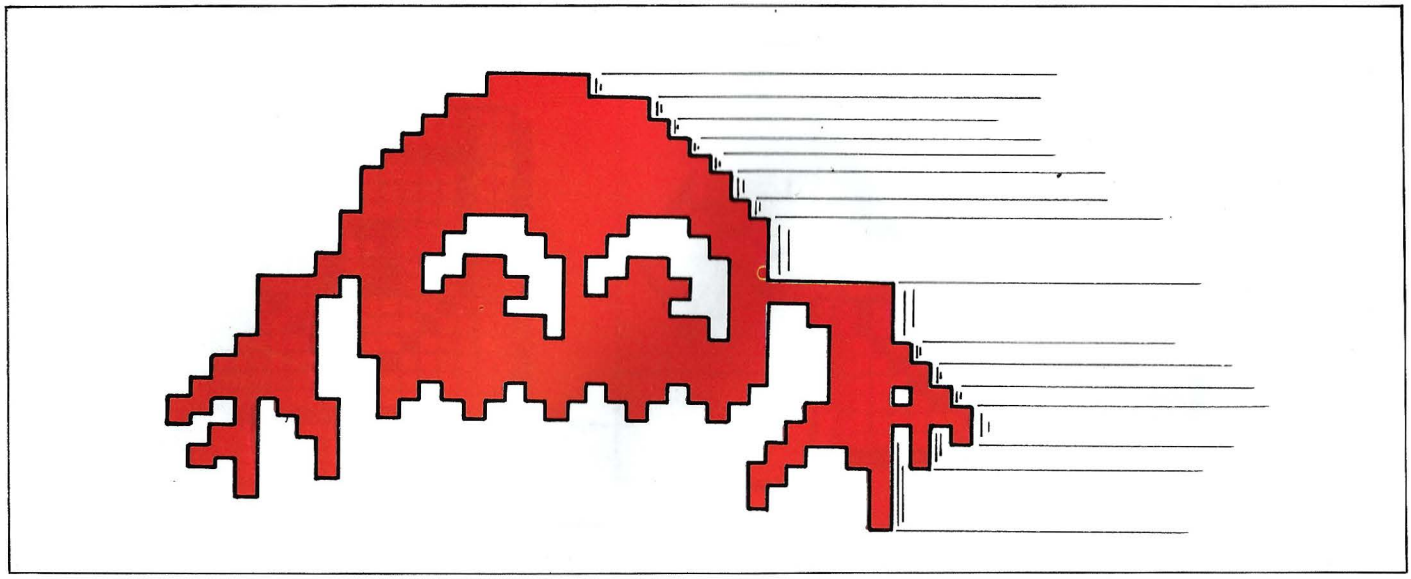
El dialecto BASIC que mejor emplea esta característica es el de Microsoft, el cual equipa a los ordenadores de tipo MSX. Ese será pues el que se comente brevemente en estas páginas.

La definición de la forma de un sprite es idéntica a la definición de un carácter en lo que se refiere a la transición pixels-bits. Sin embargo, en el BASIC de Microsoft esos bytes no se han de almacenar directamente en la posición de memoria, sino en una variable alfanumérica. Dicha variable alfanumérica se corresponde con uno de los elementos del array SPRITE$( ). Cada elemento, desde SPRITE$(0) hasta SPRITE$(255), almacena una imagen. El siguiente es un ejemplo válido de definición de un sprite:

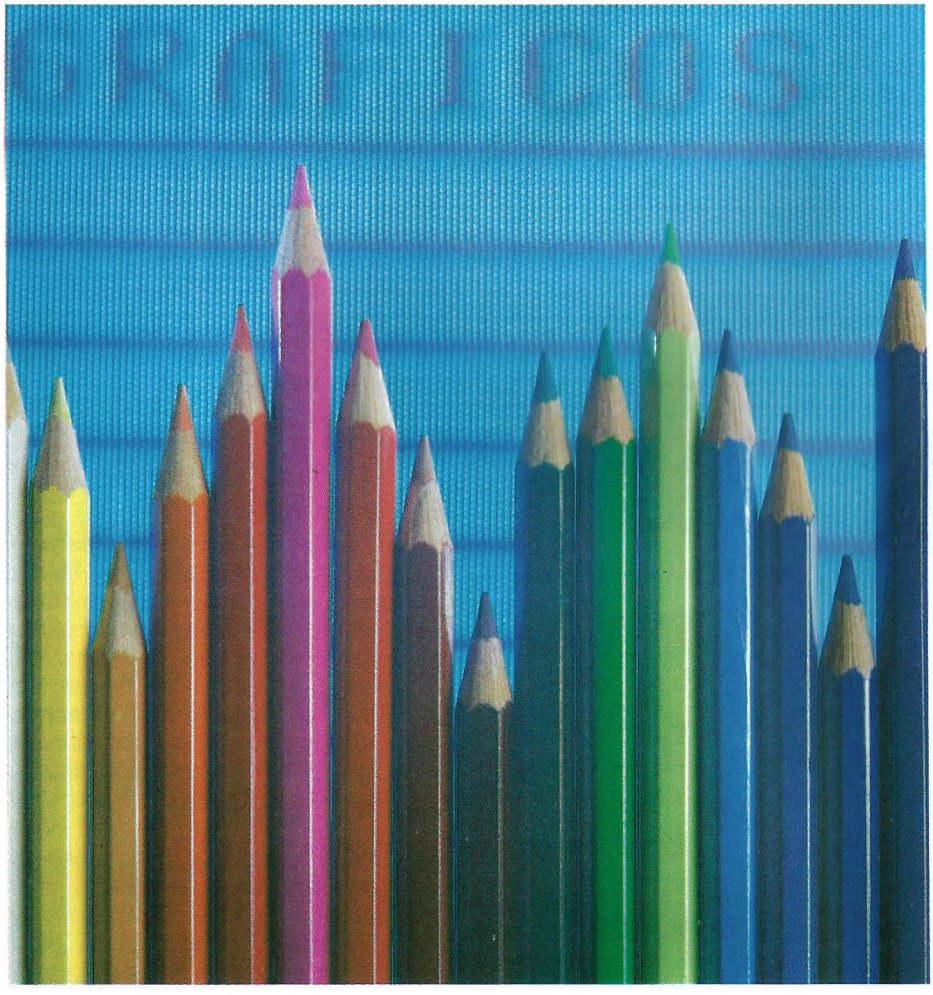
```
 10 FOR I=0 TO 7
 20 READ A$
 30 B$=B$+CHR$(VAL("&B"+A$))
 40 NEXT I
 50 SPRITE$(0)=B$
1000 DATA 00111100
1010 DATA 01111110
1020 DATA 11011011
1030 DATA 11011011
1040 DATA 11111111
1050 DATA 00111100
1060 DATA 01000010
1070 DATA 10000001
```



Para utilizar en la pantalla los caracteres definidos basta con hacer uso del comando PRINT.



La técnica de trabajo con sprites permite crear formas gráficas y darles animación en pantalla. Su empleo resulta muy adecuado para la programación de juegos.



Este método tan exótico es el que se utiliza para la creación de sprites. La forma se define a base de un bloque de instrucciones DATA. Estos datos se convierten al formato adecuado por medio de la serie de funciones CHR$(VAL ("&B" +... aplicadas a la ristra de unos y ceros que definen las filas del sprite. La información de las diferentes filas se concatena (+) en la variable B$ y ésta se guarda posteriormente en SPRITE$, creándose así el sprite.

Una vez definido, el sprite puede ser ubicado en cualquier parte de la pantalla mediante el comando PUT SPRITE. Este comando admite en su argumento el número del sprite a representar (en el ejemplo es el sprite número cero) acompañado de las coordenadas (en alta resolución) que señalan su posición. Otro dato aportable es el color del sprite.

El empleo de PUT SPRITE permite (como ya se indicó más arriba) el movimiento suave del sprite. Ello se debe a la posibilidad de situarlo a partir de cualquier pixel de la pantalla.

Otra ventaja del uso de los sprites es la detección de colisiones. Estas se manejan exactamente igual que los errores. En lugar de ON ERROR se emplea ON SPRITE GOSUB. Esta última característica permite detectar colisiones proyectil-marciano en juegos, mandando la ejecución a una rutina adecuada para el tratamiento de la colisión.

FORTRAN (1)

El precursor de los lenguajes de alto nivel

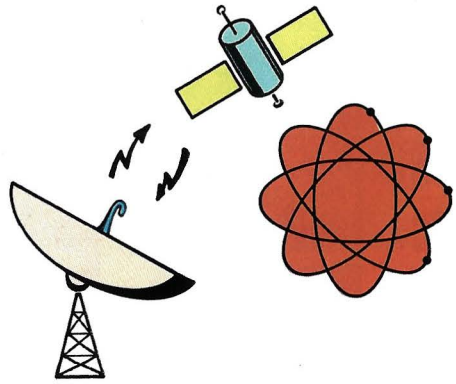
Si hay que buscar la «semilla» del árbol genealógico de los lenguajes, el cual ha estado creciendo de una manera espectacular en los últimos treinta años, la encontraremos en FORTRAN. Esta palabra está formada por las sílabas iniciales de FORmula TRANslation, es decir, «traducción de fórmula». Y esto fue lo que hizo que el FORTRAN ganara adeptos rápidamente: para un científico enfrascado en complicados cálculos matemáticos, el pasar de escribir una expresión en términos cabalísticos a hacerlo tal y como la veía escrita suponía una ventaja incuestionable.

UN POCO DE HISTORIA

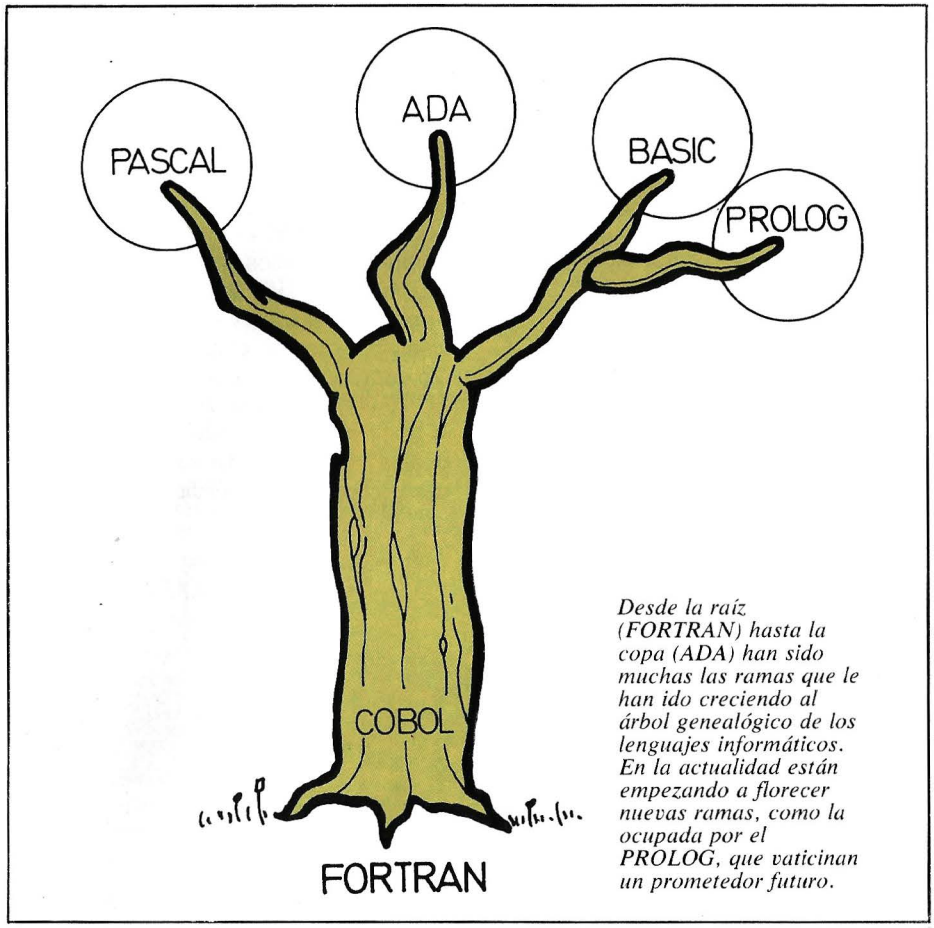
FORTRAN es fruto de la evolución natural de la historia de la informática (ver figura). Después de la utilización del código máquina y del lenguaje ensamblador, el siguiente paso a dar tenía que consistir en un lenguaje de alto nivel, con el que la tarea de la programación pasara a ser un oficio, en vez de un arte como venía siendo hasta entonces.

Aunque posteriormente se empezó a considerar a los ordenadores como potentes herramientas para gestionar grandes volúmenes de información, en un primer momento se les enfocó hacia actividades puramente científicas y «devoradoras de números». La consecuencia de este punto de vista fue un lenguaje fuertemente orientado hacia estos aspectos.

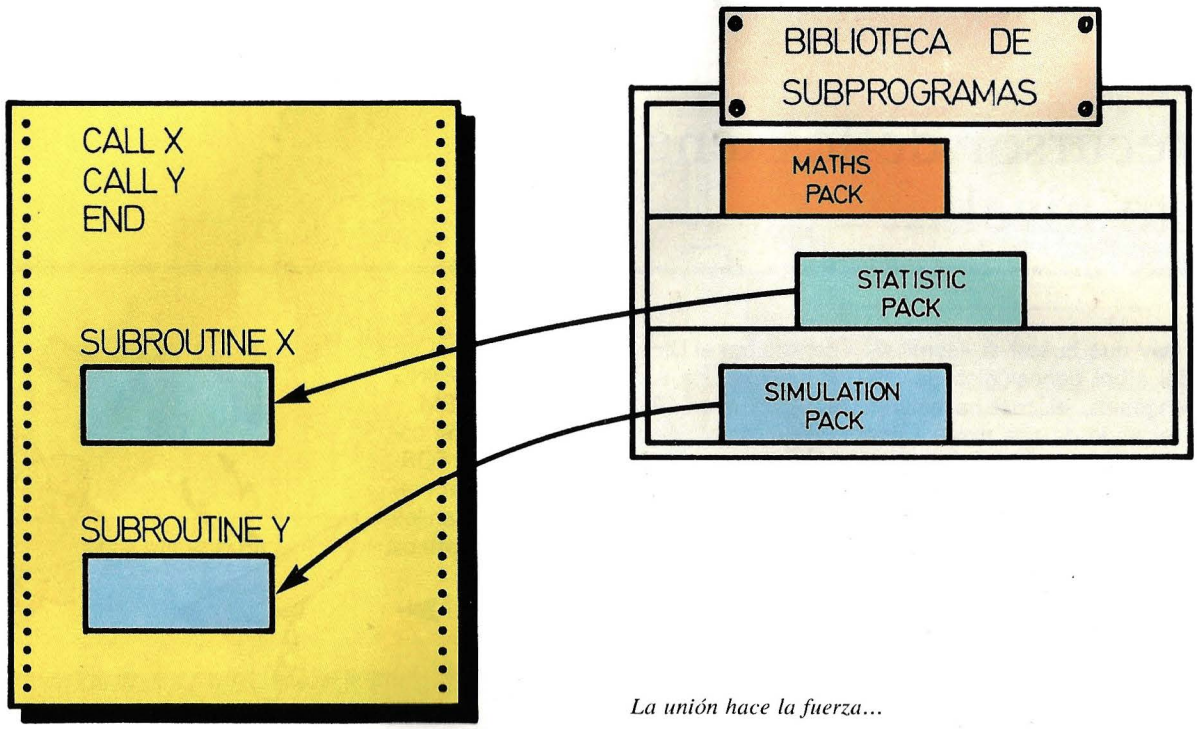
La idea de que era necesario algo parecido al FORTRAN estaba ya presente en el año 1949, cuando apareció el SHORT-CODE, creado por el Dr. Mandy para Univac. Con la misma idea surgió el SPEED-CODING de Backus en 1953, esta vez para IBM. Fue el propio Backus quien en 1954 comenzó el desarrollo del lenguaje FORTRAN (también en IBM) cuyo primer manual vio la luz en 1956. En los años en los que se centra nuestra historia no estaba



*El lenguaje **FORTRAN** es fruto de la evolución tecnológica y, en definitiva, del propio desarrollo de la ciencia informática.*



*Desde la raíz (**FORTRAN**) hasta la copa (ADA) han sido muchas las ramas que le han ido creciendo al árbol genealógico de los lenguajes informáticos. En la actualidad están empezando a florecer nuevas ramas, como la ocupada por el **PROLOG**, que vaticinan un prometedor futuro.*



La unión hace la fuerza...

desarrollada aún la llamada "Teoría formal de lenguajes" (ver cuadro), que es uno de los mejores ejemplos de la repercusión en el terreno práctico de divagaciones teóricas. Con esta teoría, un reducido grupo de personas pueden desarrollar un compilador libre de errores en un 99% en unas pocas semanas.

Cabe observar al respecto que Backus y su buen número de colaboradores, tardaron dos años en realizar el primer compilador de FORTRAN; el cual, por cierto, tenía un gran número de errores.

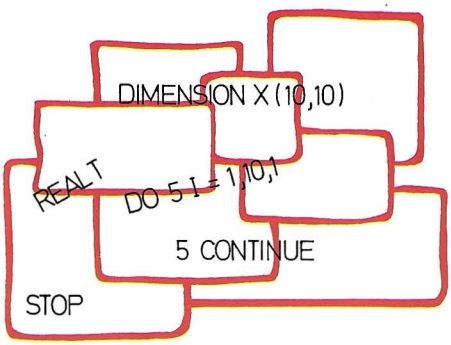
A partir de este momento, la forma de trabajar consistía en esperar a que los usuarios del compilador detectaran los errores y los comunicaran a IBM. Aquí, si el error no era «monstruoso», se le daba a la compañía que lo detectó un "parche" para salir del aprieto. Todos los errores eran catalogados y una vez que se tenían un puñado de ellos se lanzaba al mercado una nueva versión del compilador con los errores corregidos y con nuevas sentencias y mejoras en el lenguaje.

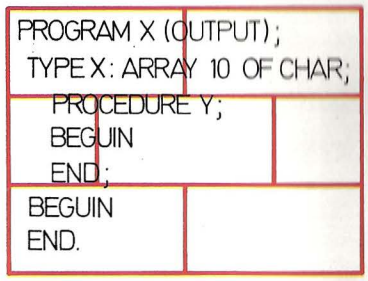
De esta forma se llegó en el año 1962 al FORTRAN IV (cuarta versión del original). A partir de aquí la "Teoría formal de lenguajes" empezó a dar sus frutos y la experiencia ganada en el diseño de compiladores hicieron que se abandonara esta forma caótica de trabajar.

En la actualidad podemos encontrar las versiones FORTRAN-77 o FORTRAN-80 para ordenadores personales. Sus principales atractivos residen en la facilidad para manejar expresiones matemáticas complejas y en su velocidad de ejecución; ambas propiedades lo hacen conveniente para su uso en aplicaciones científicas, en las que el volumen de cálculo es la principal característica.

El FORTRAN se ve hoy en día superado por lenguajes como el Pascal o por algunas versiones avanzadas del Basic. Sin embargo, tiene todavía importancia por la gran cantidad de subrutinas escritas en FORTRAN que hay en las bibliotecas de subprogramas de los grandes ordenadores. De esta forma, si se conoce un poco de FORTRAN se pueden hacer programas interesantes tan sólo a base de acceder a dicha biblioteca.

Veremos que las similitudes entre FORTRAN y Basic son más que numerosas. De hecho, el segundo nació como una revisión del primero, y se comenzó a utilizar como lenguaje científico en algunas universidades norteamericanas, acabando en lo que hoy todos conocemos.





Para los conocedores de los lenguajes de programación estructurados, la apariencia de un programa FORTRAN les resultará cuando menos caótica.

ESTRUCTURA DE UN PROGRAMA

En realidad es difícil hablar de tal estructura en un programa FORTRAN. La única forma de modularizarlo es a través de FUNCTION, y de SUBROUTINEs, que son algo así como las funciones y procedimientos en Pascal (ver figura).
En FORTRAN no existe una zona de declaraciones propiamente dicha, ya que algunas variables tienen tipos implícitos asociados en función de la primera letra del nombre (hablaremos más despacio de ellos), aunque esta declaración puede sustituirse por una explícita realizada por el programador.
Los tipos al estilo Pascal son inexistentes en FORTRAN, al igual que lo son estructuras como WHILE-DO y REPEAT-UNTIL.
Como se indicó en el párrafo anterior, para enfrentarse al FORTRAN hay que tener en la mente al Basic, lo cual no nos resultará excesivamente complicado.

Actualmente existen versiones de FORTRAN —FORTRAN 70, FORTRAN 80...— desarrolladas especialmente para ordenadores personales.

VARIABLES FORTRAN

Una variable en FORTRAN es una cadena de uno a seis caracteres alfanuméricos, de los que el primero será obligatoriamente una letra.
La declaración de variables enteras y reales se puede hacer implícita o explícitamente, esta última a través de las llamadas "sentencias de declaración explícita".
Una variable que no ha sido declarada explícitamente es considerada en FORTRAN como:
ENTERA: si empieza por I, J, K, L, M o N
REAL: si tal variable empieza por una letra distinta de la anterior (A . . .H o O . . .Z)
Los tipos de variables que pueden manejarse son:

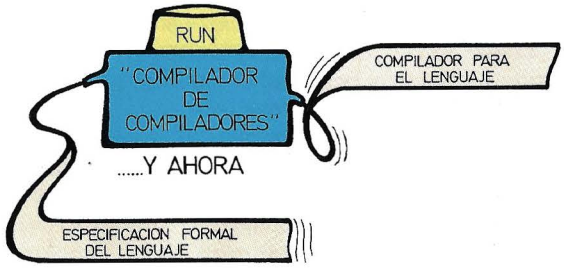
- INTEGER: equivalentes a sus homónimas del Pascal. En un programa donde no haya sentencias de declaración explícita, variables como:

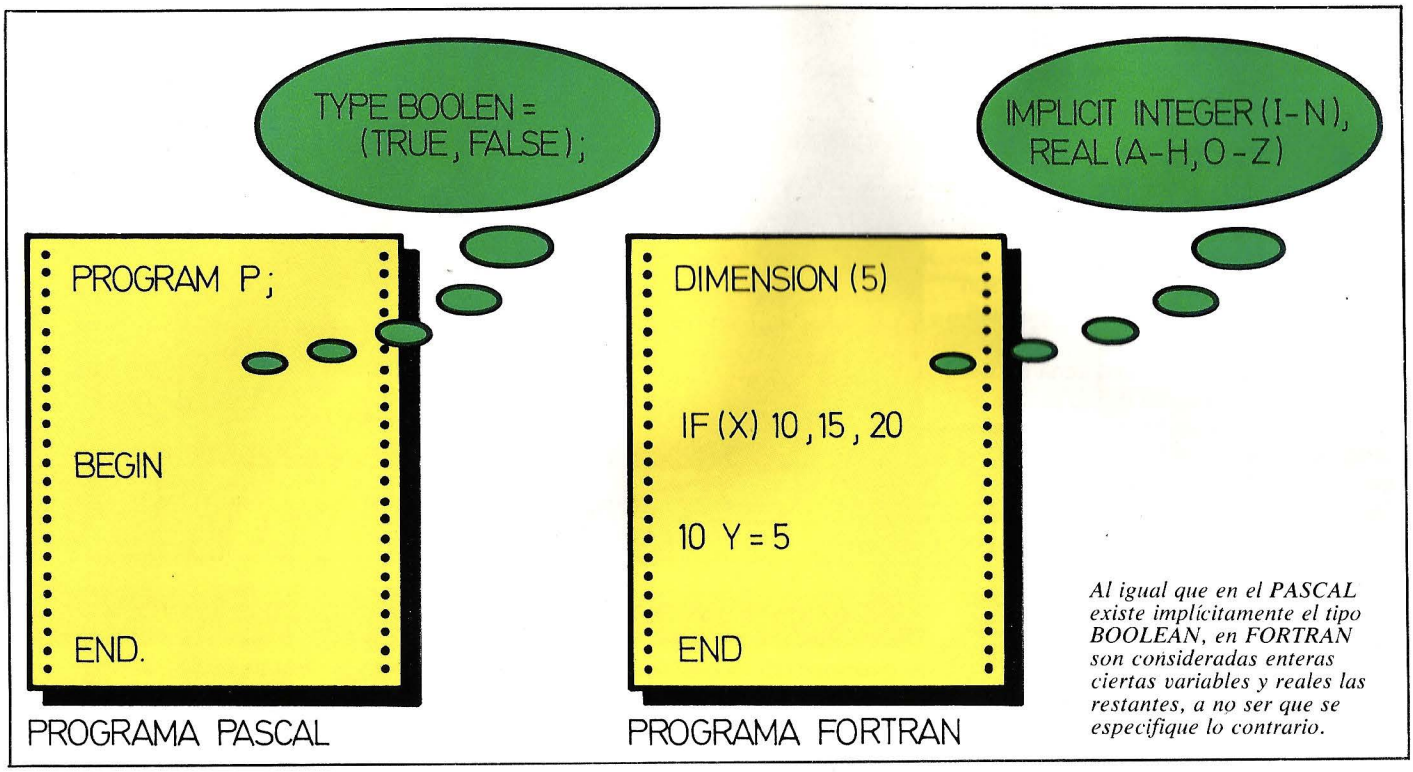
I
N
NUMERO

La teoría formal de lenguajes

Los primeros compiladores para lenguajes de alto nivel constituían un buen ejemplo de labor puramente artesanal. Al no existir un fundamento teórico sobre la construcción de tales programas, su confección se basaba en recopilar las experiencias anteriores y en pruebas de ensayo y de error.
Por las fechas en las que el FORTRAN IV vio la luz del día, una serie de lingüistas, fundamentalmente Noam Chomsky y Ferdinand de Saussure habían desarrollado unas teorías sobre el lenguaje natural que constituyeron la "gramática estructuralista". Sus principales pilares son la división de las oraciones en estructura profunda y superficial, y la división de las mismas en sintagmas.
Estos lingüistas, en colaboración con expertos en ordenadores, aplicaron los resultados de la gramática estructuralista al campo de los lenguajes de programación, sentando las bases de la Teoría formal de lenguajes.
La consideración de esta teoría a la hora de diseñar un nuevo lenguaje supone disponer de un compilador para el mismo en un tiempo record, frente a lo que se tardaría en el caso de no hacerlo así.
La herramienta más sorprendente, fruto de la aplicación de esta teoría, es un programa llamado "compilador de compiladores"; el cual, a partir de una especificación de un lenguaje, es capaz de realizar gran parte del esfuerzo necesario para diseñar un compilador del mismo.

ANTES.....





Al igual que en el PASCAL existe implícitamente el tipo BOOLEAN, en FORTRAN son consideradas enteras ciertas variables y reales las restantes, a no ser que se especifique lo contrario.

serán consideradas enteras. Obsérvese que las variables "I" y "N" son las que se utilizan habitualmente como subíndices de expresiones matemáticas, los cuales son siempre números enteros.

- REAL: representan valores en punto (o coma) flotante. Variables como:

ENTERA
DISCRIMINANTE

serán reconocidas como reales.

- DOUBLE PRECISION: son variables que —teóricamente— tienen el doble de precisión que las reales. Son equivalentes al tipo "double" del lenguaje C. Su declaración se realizará obligatoriamente a través de una sentencia de declaración explícita.

Mientras que una variable REAL se puede representar en notación de punto flotante o exponencial, las de DOUBLE PRECISION sólo admiten la notación exponencial; si bien, la "E" que precede al exponente es sustituida por una "D", como por ejemplo: 23.4D-08.

- LOGICAL: equivalentes al tipo "boolean" de los lenguajes que conocemos. Podrán tomar los valores .TRUE. o .FALSE. (¡Atención a los puntos!) y su declaración se hace igual que en el tipo anterior.

- COMPLEX: están destinadas a contener números complejos. Tanto la parte real como imaginaria se consideran números reales. Un ejemplo de constante de este tipo podría ser:

(3.5,4.3)

que representa un complejo de parte real 3.5 e imaginaria 4.3.

Una sentencia de declaración explícita tiene el siguiente aspecto:

<tipo> <lista de variables>

Donde "tipo" es una de las cinco palabras clave vistas más arriba. Ejemplos de declaraciones explícitas podrían ser:

```
INTEGER  R1,R2,VALOR
REAL     I1,ABC
LOGICAL  ESTADO,FIN
```

Veamos que "I1" ha sido declarada como REAL. Una declaración explícita como la que se ha realizado para esta variable rompe con la norma implícita que la consideraría como INTEGER por empezar por "I". Sin embargo, dicha norma sigue siendo válida para otras variables que empiecen por tal letra.

Este tipo de sentencias han de ir al principio del programa y se ha de verificar que el mismo nombre de variable no aparezca en más de una sentencia de declaración.

La sentencia IMPLICIT tiene por objeto declarar de un tipo determinado todas aquellas variables que empiecen por un carácter alfabético dado; por ejemplo:

```
IMPLICIT DOUBLE PRECISION (A,D-X)
```

hará que todas las variables que empiecen por A o por D,E ... V, W, X sean de doble precisión.

De igual manera podríamos decir:

```
IMPLICIT INTEGER(I-N), REAL(A-H, 0-Z)
```

Aunque ello no es necesario, ya que el ordenador considerará normalmente de este tipo a las variables especificadas.

En FORTRAN también pueden utilizarse arrays, con la única limitación de que los índices con los que se referencian los elementos han de ser mayores que cero. Para declarar una matriz de 10×10 elementos reales bastaría con ejecutar:

```
DIMENSION MATRIZ(10,10)
REAL MATRIZ
```

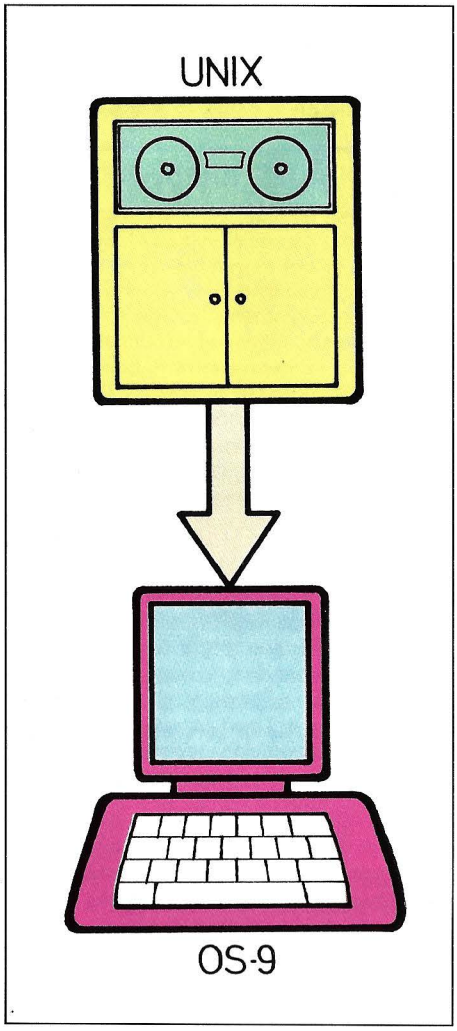
o bien:

```
REAL MATRIZ(10,10)
```

OS-9 (1)

Un potente sistema operativo para pequeños equipos

A través del estudio de los diferentes sistemas operativos abordados en esta obra, ha habido ocasión de comprobar las diversas formas de atacar el problema de la gestión de los recursos informáticos del ordenador: tanto por lo que se refiere a periféricos, como a programas de aplicación, así como a la interface ordenador-usuario. Cada uno de los sistemas operativos tratados resolvía el problema con ciertas peculiaridades. Uno de ellos llegó a ser señalado como el sistema operativo tal vez con más posibilidades de futuro entre los destinados al ámbito de los microordenadores de tipo profesional. Se trataba del sistema operativo UNIX.



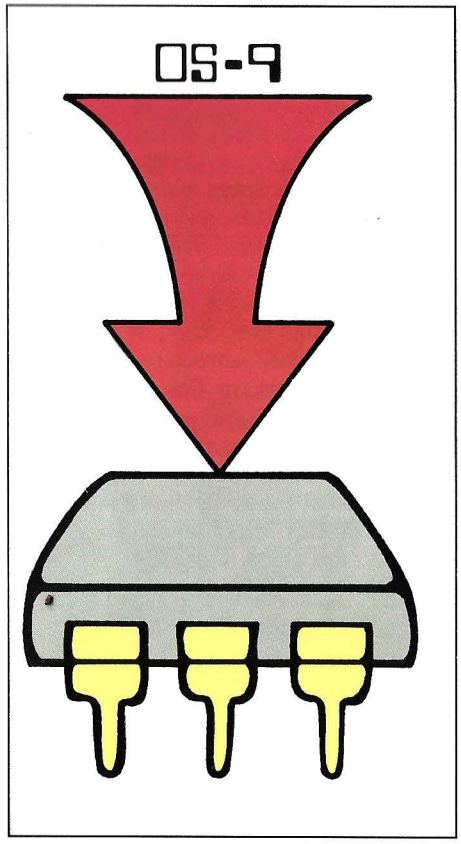
El sistema operativo OS-9 es un descendiente directo del popular y potente UNIX.

Una de las grandes ventajas del UNIX residía en su transportabilidad y en su alta capacidad para adaptarse a otros ordenadores, debido al hecho de que está escrito en un lenguaje de alto nivel: el "C". Una muestra de esta posibilidad la aporta el sistema operativo 0S-9: un descendiente directo del UNIX aunque adaptado a microordenadores destinados al mercado doméstico y educativo, como es el caso del popular DRAGON.

INTRODUCCION AL OS-9

El sistema OS-9 es un sistema operativo multitarea y multiusuario; es decir, permite la existencia de varios usuarios trabajando simultáneamente con el ordenador, o bien la ejecución de varios programas funcionando concurrentemente. Como soporte del mismo se emplea un equipo basado en un microprocesador Motorola 6809; precisamente, el empleado en el DRAGON.

El origen de los requerimientos de este sistema operativo se encuentran en el desarrollo por parte de Microware de un nuevo dialecto del lenguaje BASIC con la colaboración de Motorola durante los últimos años de la década de los setenta. El desarrollo de este lenguaje presuponía la existencia de un potente sistema operativo que soportará las peticiones que le pudieran ser impuestas por el lenguaje.



El OS-9 gestiona las tareas de los distintos usuarios fundamentalmente a través de la memoria interna del equipo.

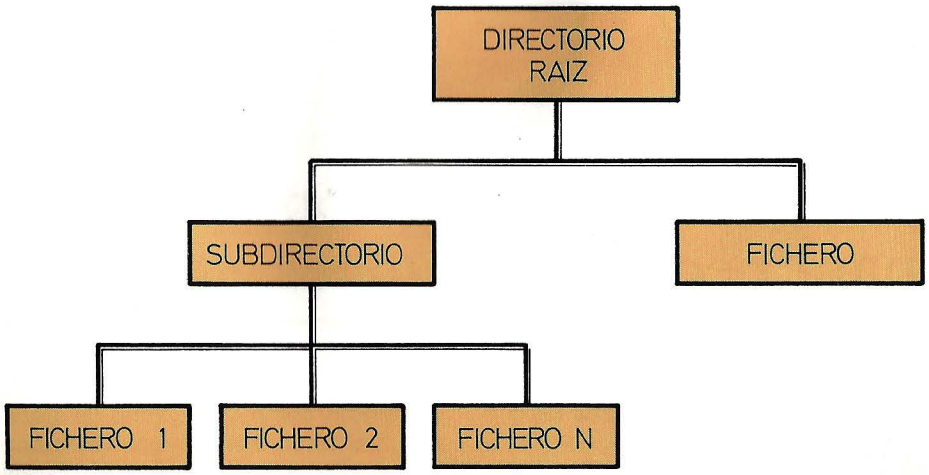
El OS-9 se inspiró plenamente en el sistema operativo UNIX de Bell Laboratories. La copia no fue directa, toda vez que las necesidades de un pequeño microordenador, así como sus disponibilidades hardware, no son las mismas que las propias de los ordenadores sobre los cuales el UNIX se desarrolló inicialmente. Las dife-

rencias fundamentales del OS-9 con respecto al UNIX son las que se detallan a continuación:

- El OS-9 ha sido diseñado para un entorno de memoria en el que conviven zonas de RAM y ROM, soportando código reentrante de forma más efectiva.
- El lenguaje en el que está escrito el OS-9 es el ensamblador correspondiente al microprocesador 6809, en lugar del lenguaje de alto nivel C del UNIX, con lo cual se consigue mejorar el rendimiento del producto resultante.
- Durante la operación en régimen multitarea/multiusuario, el OS-9 no gestiona el almacenamiento dinámico en disco de los programas en ejecución, sino que los conserva en memoria en todo momento.

Todas estas diferencias son consecuencia lógica de la diferencia de entornos en los que se explotan estos sistemas operativos. El UNIX fue diseñado para ordenadores como por ejemplo PDP-11 de Digital, equipos con una potente CPU, abundancia de memoria central y de sistemas de almacenamiento secundario de gran capacidad y reducidos tiempos de acceso. En estas condiciones, resulta lógico que el OS-9 mantenga continuamente en memoria los programas, toda vez que los tiempos de acceso a los disquetes que constituyen el medio de almacenamiento principal de los microordenadores, y sobre los que está grabado el OS-9, son muy largos en comparación con el tiempo medio de acceso a los discos de los ordenadores sobre los cuales reside normalmente el UNIX. Esta diferencia hace ilógico cualquier intento de paginación, dado que produciría en los tiempos de respuesta del equipo un retraso inaceptable para el usuario.

Por razones similares, cabe señalar como lógica la utilización de un lenguaje ensamblador para el montaje del sistema operativo, con preferencia a un lenguaje de alto nivel. Con la primera opción se consigue mejorar la respuesta a las peticiones del usuario, al disminuir los requisitos de ciclos de máquina exigidos a la CPU.



La estructura de información propia del OS-9 es arborescente, al igual que en el caso del UNIX.

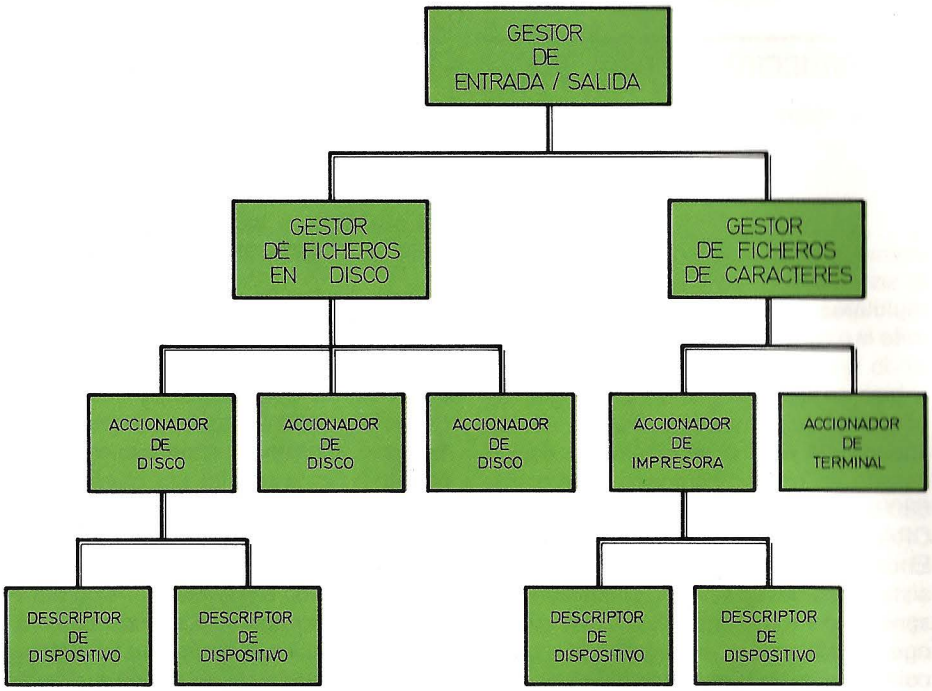
NECESIDADES HARDWARE

Como requisitos mínimos para dotar a un equipo del sistema operativo OS-9, se necesitan 4 Kbytes de memoria ROM y 2 Kbytes de memoria RAM. Hay que tener en cuenta que el OS-9 está compuesto por una serie de módulos que se organizan y enlazan automáticamente en el momento de arrancar el sistema. Esta estructura permite la reconfiguración del sistema sin más que cargar en memoria los módulos necesarios.

A efectos de desarrollo, la configuración típica mínima es la señalada a continuación:

- 64 Kbytes de memoria RAM, dividida entre la necesaria para la programación en lenguaje de alto nivel y la necesaria para la programación en lenguaje ensamblador.
- 4 Kbytes de memoria en ROM, de los cuales 2 Kbytes deben estar direccionados en $F800-$FFFF, mientras que los restantes son independientes en su posición.

A efectos operativos y de ampliación de la capacidad de memoria del ordenador, el



Estructura del sistema de gestión de entrada/salida del OS-9.

controlador de entrada/salida del ordenador conviene que esté localizado en una posición de memoria los más alta posible; de esta forma será posible incrementar la capacidad de RAM del equipo con menores problemas para el usuario.

LA ESTRUCTURA INTERNA DEL OS-9

La estructura de este sistema operativo consta de un conjunto de módulos, cada uno de los cuales está destinado a cumplir una serie de funciones específicas. Los

El OS-9 es un sistema operativo concebido para pequeños microordenadores basados en el microprocesador 6809 de Motorola.

El ordenador «rentable»

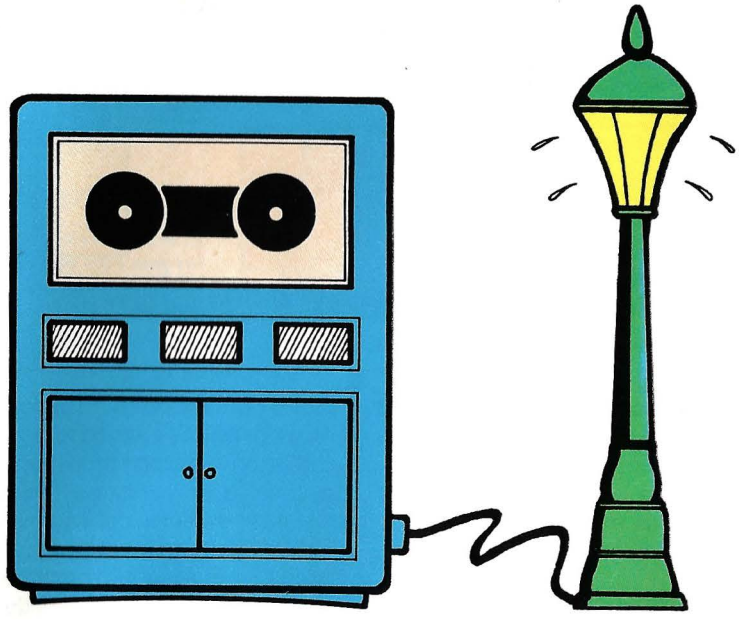
Una afirmación plenamente extendida en nuestros días es que el ordenador es una máquina maravillosa capaz de efectuar casi cualquier tarea que le sea encomendada. Mucho de verdad hay en ello ya que, efectivamente, si un ordenador se programa adecuadamente puede llevar a cabo multitud de tareas con *mayor* o *menor* efectividad. Y ahí se encuentra la clave. Un ordenador puede gestionar igualmente la nómina de una empresa con diez mil empleados, como la de otra empresa con tan sólo tres empleados. No obstante, en este último caso el coste del equipo así como el de las operaciones necesarias para que se ejecute la referida actividad pueden hacer mucho más rentable para el usuario adoptar un proceso manual.

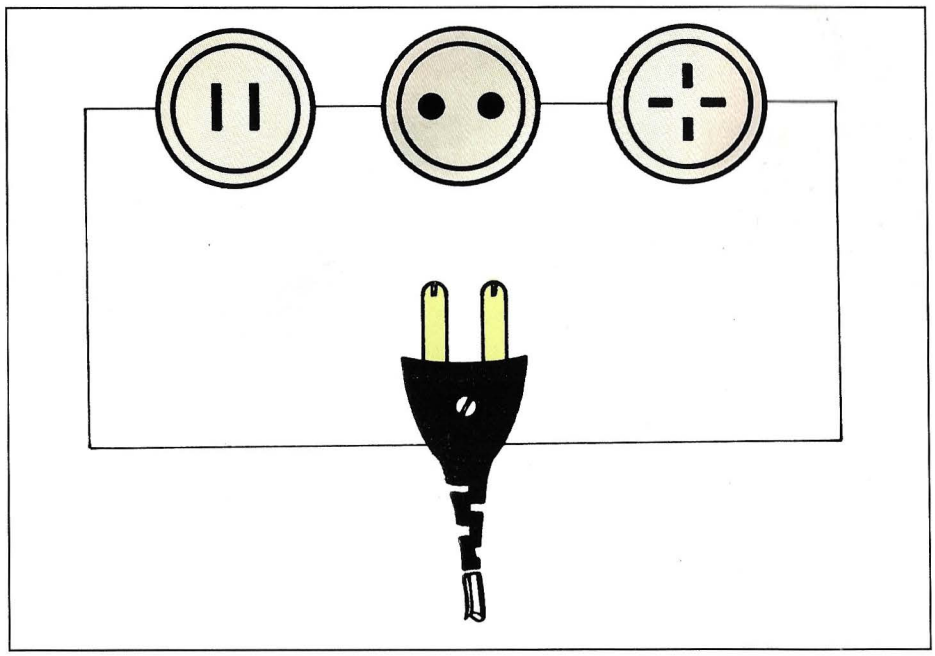
El ordenador es también altamente rentable en el control de procesos industriales: puede controlar en tiempo real la evolución de un gran número de señales, y tomar decisiones para la mejor operación y rendimiento del conjunto global. Sin embargo, el ordenador no será rentable si su actividad se reduce a controlar el encendido y apagado de las luces exteriores de una casa.

En términos generales, las tareas que un ordenador puede llevar a cabo con mayor rendimiento son las que comparten alguna de las siguientes características:

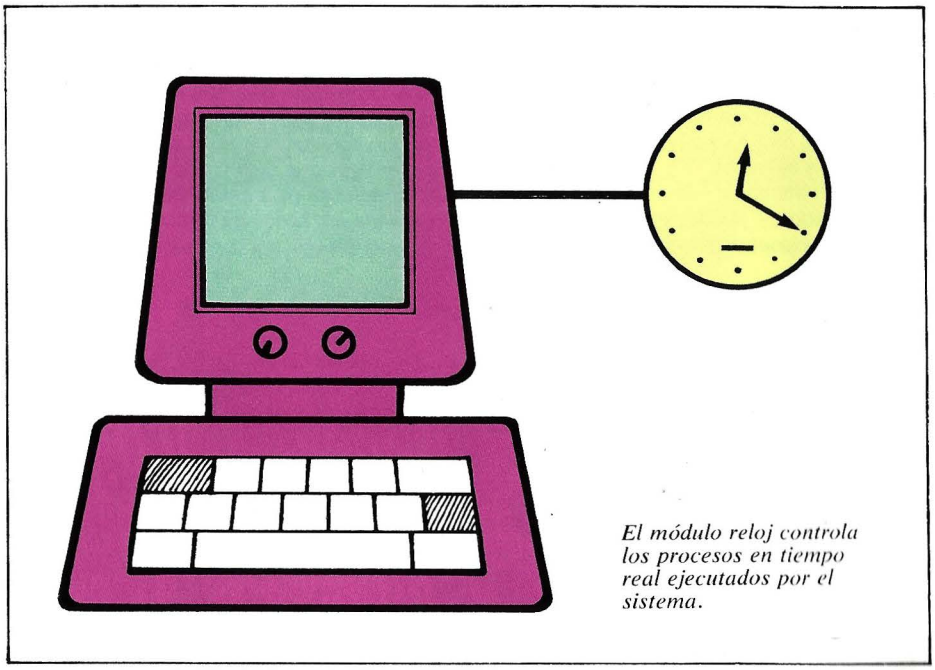
- Hay que manipular y procesar grandes volúmenes de datos residentes en ficheros.
- Deben controlarse procesos repetitivos que pueden tomar ventaja de la velocidad de proceso del ordenador.
- Procesos de gestión en los que hay que realizar un gran número de transacciones.
- Se trata de ejecutar procesos de gestión altamente definidos.

Dada su naturaleza —una mera máquina—, el ordenador no puede mostrar ninguna de las restricciones emocionales propias del ser humano: ira, fatiga, frustración... y tampoco puede hacer juicios de valor, a menos que sea programado específicamente para ello y los datos le sean suministrados de forma adecuada. De ahí que corresponda al usuario considerar los factores que determinan si el empleo del ordenador para determinada actividad se ajusta a criterios de rentabilidad, eficacia, precisión y comodidad.





El descriptor de dispositivos controla la conexión física de los periféricos al microordenador.



El módulo reloj controla los procesos en tiempo real ejecutados por el sistema.

módulos son seleccionados en función de las necesidades del equipo sobre el que va a ser implementado, de manera que si un ordenador no tiene unidades de disco no necesitará los módulos del OS-9 relacionados con el control de dichas unidades.

La base del sistema son los módulos NUCLEO, RELOJ e INIC. El primero, como su propio nombre indica, proporciona los servicios principales de control del sistema, como son la gestión de la memoria, de los programas en curso y el enlace con los restantes módulos del sistema. El módulo RELOJ gestiona el reloj interno del sistema para la adecuada operación del software en tiempo real. El módulo INIC es una tabla que contiene los parámetros de inicialización del sistema, y es utilizada por el núcleo durante el proceso de arranque, asignándose a través de la misma valores tales como los nombres de los dispositivos del sistema.

En un nivel inmediatamente inferior al módulo INIC se encuentra el gestor de entrada/salida.

En el siguiente nivel aparece el gestor de ficheros, encargado de gobernar las peticiones de entrada/salida a través de periféricos. Este se divide en dos zonas. Por un lado se encuentra el denominado "gestor de ficheros para bloques aleatorios", el cual procesa todas las operaciones relacionadas con unidades de disco, y por otro lado aparece el gestor de ficheros de caracteres secuenciales, el cual trabaja sobre dispositivos de entrada/salida de naturaleza secuencial, tales como impresoras y terminales.

El hecho de que se hable de gestores de ficheros se debe a que, al igual que ocurre con el UNIX, el OS-9 trata a los dispositivos de entrada/salida como si de ficheros se tratase.

En un nivel inferior y siguiendo una jerarquía descendente por lo que se refiere al control de periféricos, se encuentran las denominadas unidades de accionamiento de periféricos. Su cometido reside en el control de actividades físicas básicas de entrada/salida sobre el hardware específico. A través de estas unidades de accionamiento es posible preparar gestores específicos para controlar elementos hardware distintos de los instalados inicialmente en el ordenador. El nivel final en la gestión de entrada/salida corresponde al descriptor de dispositivo. Estos descriptores están relacionados directamente con la conexión física y son pequeños ficheros en los que se almacenan los datos correspondientes a las puertas de entrada/salida. Almacenan, por ejemplo, los nombres lógicos y la dirección física de las puertas, así como otros datos de inicialización necesarios.

Se observa, pues, que la estructura del sistema de comunicaciones de entrada/salida es jerárquico en su alcance, reduciéndose desde una óptica amplia a otra más específica. Este tipo de estructura hace que resulte muy fácil controlar una amplia gama de periféricos. En el momento que el usuario lo considere oportuno pueden añadirse al sistema los módulos necesarios para aumentar el número o variar las características de los periféricos ya instalados.

Open Access (1)

Un paquete integrado con seis entornos de trabajo

El paquete integrado OPEN ACCESS está formado por seis módulos independientes, aunque coordinados, cuyo copyright pertenece a la firma S.P.I. (Software Products International). Si bien la mayoría de los paquetes integrados tienen como módulo central a la hoja electrónica, en este caso, OPEN ACCESS se basa en el programa encargado de almacenar y recuperar la información, es decir, en la base de datos. Como "satélites" suyos existen otros cinco módulos: un programa de comunicaciones, un procesador de textos, un sistema de gráficos, una hoja electrónica y una agenda electrónica.

Se dedican tres capítulos de la obra a profundizar en este potente paquete integrado, concretamente en su versión española.

MODULOS DEL OPEN ACCESS

Dado que OPEN ACCESS está basado en módulos integrados, la misma información puede ser tratada de muchas formas diferentes; con ello se obtiene un rendimiento muy alto al poder utilizar en múltiples entornos la misma información, sin necesidad de introducirla más que una vez.

El gestor de base de datos realiza las tareas de tratamiento de información. Los datos almacenados en la base de datos pueden ser utilizados para operaciones de cálculo mediante la hoja electrónica, representados gráficamente en una pantalla apropiada, ubicados en informes escritos mediante el procesador de textos, etc.

A continuación, y como primera toma de contacto con OPEN ACCESS, se van a describir las características generales de los seis módulos integrantes del paquete de aplicación.

- *Gestor de base de datos*

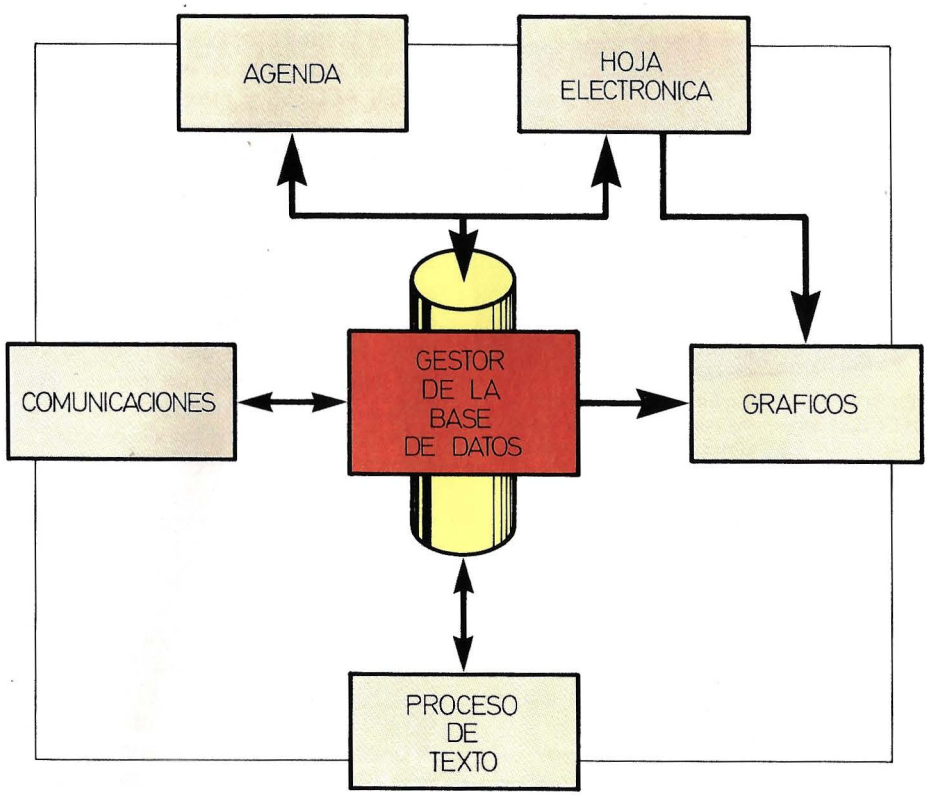
Se basa en un modelo relacional para estructurar la información y permite almacenar, obtener y manipular datos cómoda y eficazmente. Para ello ofrece la posibilidad de definir formatos de pantalla que, junto con un sencillo lenguaje de manipulación, permite explotar la información, tanto interactivamente como mediante la producción de informes escritos.

- *Hoja electrónica*

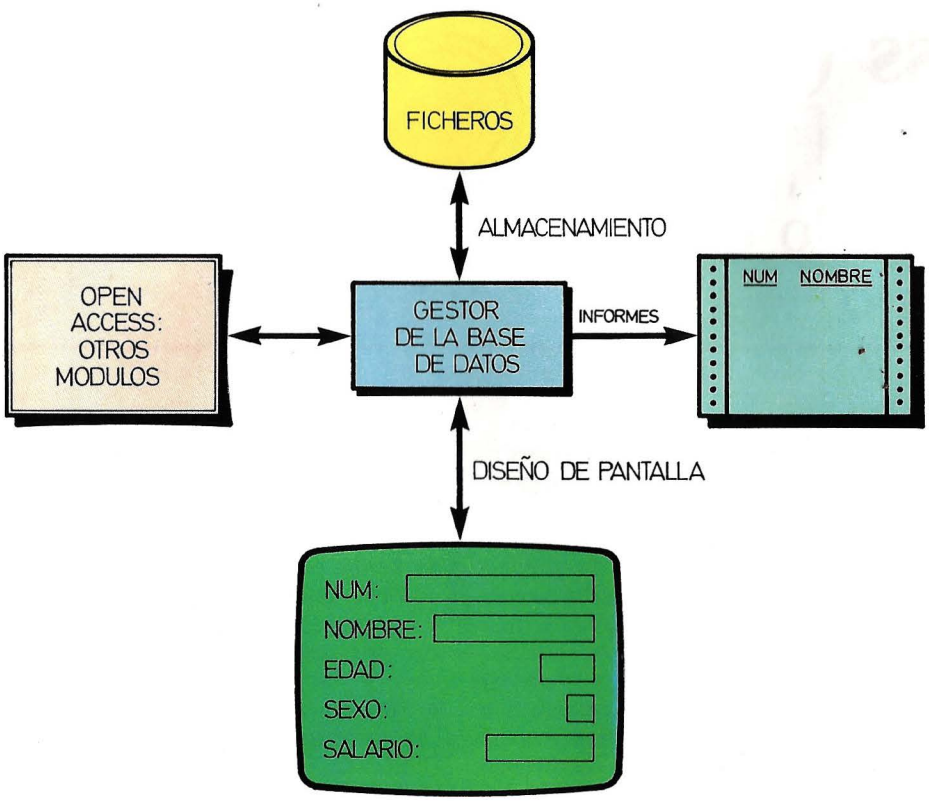
El carácter evidentemente matemático de este módulo lo hace ideal para la realización de cálculos de todo tipo. Además de estar "conectado" con la base de datos, de la que puede leer y en la que puede escribir, también permite que los datos tratados mediante la hoja electrónica puedan ser reproducidos en forma gráfica.

- *Proceso de textos*

Las características de este módulo de OPEN ACCESS no difiere en absoluto de los restantes programas para el tratamiento de textos ya descritos en la obra.



OPEN ACCESS es un sistema integrado compuesto por seis módulos o entornos de trabajo entre los que es posible mantener un flujo de intercambio de datos.



El gesto de base de datos constituye el módulo central del paquete integrado. Por lo demás, dicho módulo está capacitado para realizar las labores tradicionales propias de cualquier base de datos.

	HORAS	PTS.	TOTAL
JUAN	10.00	5	50
LUIS	12.00	7	84
PEDRO	15.00	10	150

La hoja electrónica integrada en OPEN ACCESS incorpora comandos y funciones adecuados para resolver cómodamente los problemas de lápiz, papel y calculadora.

Así, es posible insertar, borrar y cambiar textos, mover textos de un lugar a otro, utilizar diferentes formatos de párrafos, fijar normas para la separación automática de palabras, realizar distintos tipos de justificaciones, etc.

- *Comunicaciones*

Mediante la utilización de los correspondientes modems, el módulo de comunicaciones permite establecer un "diálogo" entre el ordenador en el que se explota el OPEN ACCESS y otros ordenadores, situados en la misma ciudad, en el mismo país o, en definitiva, en cualquier lugar del mundo.

- *Gráficos*

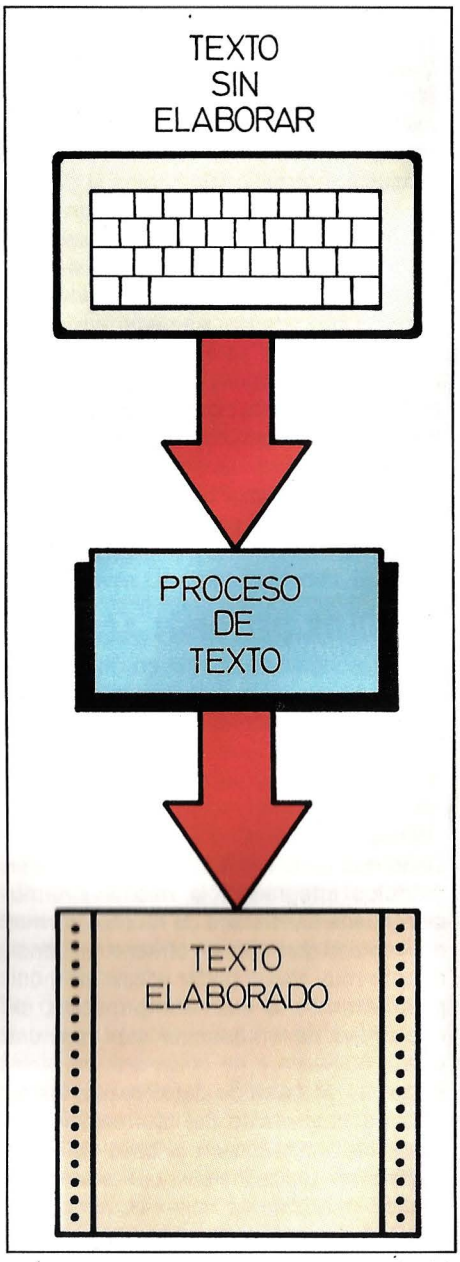
A partir de datos numéricos, el módulo gráfico de OPEN ACCESS puede producir diseños en color. Dentro de los distintos tipos de diagramas disponibles se incluyen los tradicionales de barras, líneas, tarta... Tal vez la novedad más notoria consiste en la posibilidad de producir diagramas de barras tridimensionales. Los datos necesarios para los gráficos pueden ser introducidos directamente en este módulo, u obtenidos de la base de datos o de la hoja electrónica.

- *Agenda*

El sexto y último módulo de OPEN ACCESS tiene como misión controlar el tiempo y facilitar la organización del usuario. Para ello incluye un calendario electrónico, un espacio para la realización de anotaciones diarias y un sistema de direcciones y teléfonos. Además incorpora la posibilidad de producir listados de citas, cuadernillos de notas, apuntes, mensajes, etc.

CARACTERISTICAS GENERALES DE OPEN ACCESS

Tras encender el ordenador y activar el disco inicial de la aplicación OPEN AC-



El módulo de OPEN ACCESS encargado del proceso de textos facilita la confección de documentos elaborados e imprimibles.

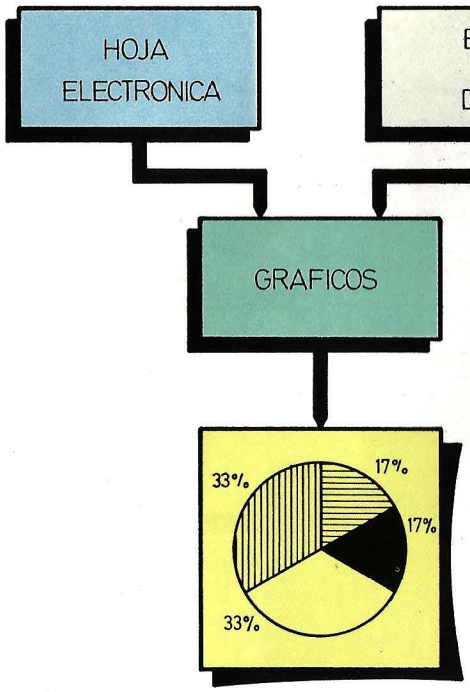
CESS, hay que teclear OA y pulsar <RETURN>. Inmediatamente, aparecerá en la pantalla del ordenador un menú de opciones básicas o entornos de trabajo. El programa solicitará la fecha del día y, en el resto de la sesión, "recordará" dicha fecha cuando el usuario lo solicite.
Para seleccionar alguno de los módulos del menú, basta con desplazar el cursor mediante las teclas de desplazamiento, hasta que éste se sitúe sobre el módulo deseado. Otra posibilidad para elegir un módulo consiste, simplemente, en introducir la primera o dos primeras letras de su denominación:

— GE para entrar en el gestor de la base de datos.
— H para activar el módulo hoja electrónica.
— P para entrar en el procesador de textos.
— GR con lo que se entrará en el sistema gráfico.
— A si se desea activar la agenda.
— C para activar el módulo de comunicaciones.

OPEN ACCESS es un completo paquete de software integrado cuyos entornos pueden competir en capacidad y potencia operativa con aplicaciones independientes del mismo tipo.

Inteligencia humana versus inteligencia artificial

La inteligencia es una de las características más preciadas y a la vez más desconocida de los seres humanos. Hace algunas décadas, nació una nueva ciencia dedicada a estudiar la posibilidad de reproducir, mediante programas de ordenador, el funcionamiento de la mente humana. Los resultados obtenidos en este campo han sido espectaculares, hasta tal punto que en la actualidad hay desarrollados programas inteligentes que sirven para muy diversas misiones: sistemas expertos, simuladores, "cerebros" de robot, etc.
En muchos círculos intelectuales se ha discutido apasionadamente sobre la posibilidad de que los programas denominados inteligentes realmente lo sean. En efecto, estos programas pueden catalogarse como auténticamente inteligentes, aunque hasta ahora su nivel es muy bajo comparado con la inteligencia humana; es más, sin lugar a dudas, un hipotético termómetro capaz de medir la inteligencia daría valores más altos para un mono que para el más brillante de los robots.
Para ratificar la anterior afirmación, contaremos una anécdota que ocurrió en un laboratorio de Inteligencia Artificial. Después de haber desarrollado un robot inteligente, especializado en localizar un racimo de plátanos situado en el techo de una habitación, y colocar convenientemente algunos objetos para alcanzar los plátanos, decidieron comparar el comportamiento del robot con el de un mono. Para ello, situaron el racimo de plátanos en el techo, "desparramaron" aleatoriamente los objetos que teóricamente el mono debería superponer para alcanzar los plátanos y, justo cuando el cuidador pasaba bajo los plátanos, el mono brincó sobre sus hombros, atrapó los plátanos y se los comió inmediatamente.
Evidentemente, con las técnicas actuales, los programas inteligentes carecen de la capacidad intuitiva que demostró el mono. Tal vez con el tiempo...

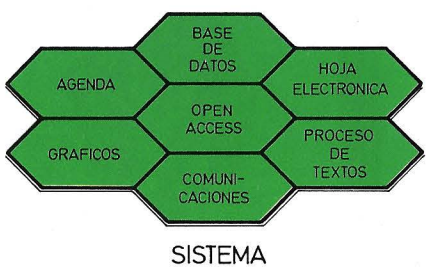


Para producir gráficos con OPEN ACCESS se utilizan los datos almacenados en los entornos de hoja electrónica y/o base de datos. El módulo gráfico se limita a reproducir la información en forma de diagrama.

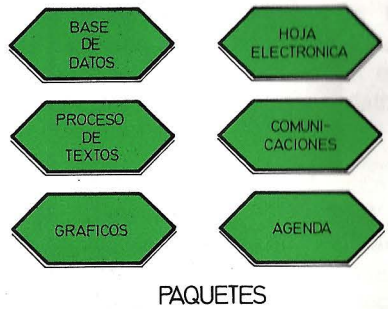
— U para solicitar la ejecución de una utilidad.

— S para finalizar la sesión de trabajo.

Una vez realizada esta operación, aparecerá un nuevo menú ofreciendo determinadas posibilidades. Si el usuario desea finalizar el trabajo con este módulo y volver al menú principal, debe seleccionar la línea OPCIONES. También desde el menú principal puede elegirse la opción S, con lo que se dará por terminado la sesión de trabajo con OPEN ACCESS y, en consecuencia, se devolverá el control del ordenador al sistema operativo.
En cualquier momento y dentro de cualquier módulo, el usuario puede solicitar AYUDA, con lo que se visualizarán cortas descripciones de las funciones disponibles. Otra posibilidad adicional de este programa consiste en comportarse como una máquina de calcular, para ello vale con pulsar la tecla <CALC> y, a continuación, introducir una expresión numérica; para obtener el resultado de la expresión introducida hay que pulsar la tecla "igual" (=).
Como última característica general de OPEN ACCESS podemos citar la posibilidad de preparar procedimientos ejecutables, es decir cadenas de comandos que se ejecutarán automáticamente. Cada uno de los procedimientos diseñados por el usuario debe almacenarse en un archivo y podrá ser ejecutado sin más que recordar el nombre de dicho archivo.



SISTEMA INTEGRADO



PAQUETES SIN INTEGRAR

Mediante un sistema integrado se obtiene una ventaja fundamental sobre el software sin integrar: la común utilización de los datos por parte de las diversas aplicaciones.

UTILIDADES

En el próximo número se iniciará una descripción formal de los módulos integrantes de OPEN ACCESS; no obstante, antes de finalizar el presente capítulo vamos a describir las principales utilidades que el programa ofrece al usuario.
Una vez que el usuario ha seleccionado la opción U del menú principal, aparece un nuevo menú con las siguientes opciones:

- *Configurar*

Sirve para definir los parámetros del Sistema: entre otras cosas permite:
1. Que el usuario defina valores por defecto para diversos parámetros de las unidades de salida, formatos, etc.
2. Que el usuario defina, mediante cadenas de caracteres, las funciones que se ejecutarán sin más que pulsar una tecla de función.
3. Que el usuario defina los parámetros que caracterizan a la impresora conectada al equipo.

- *Restaurar fichero*

Sirve para examinar y corregir posibles problemas de estructura en los ficheros utilizados por la base de datos y por la agenda. Para ello ofrece dos opciones:
1. Comprobar si una base de datos está dañada.
2. Recuperar la información que contuviera la base de datos antes de sufrir el daño.

- *SIF Intercambio*

Permite cambiar las estructuras de ficheros con distintos tipos de formato.

- *Editor de Macros*

Esta utilidad tiene como misión facilitar la edición de macros, de forma que el usuario pueda visualizar y en su caso modificar los procedimientos desarrollados por él mismo.

- *Reservado*

Como última opción dentro del menú de utilidades, el usuario puede ubicar los procedimientos que él desee.
En resumen, la opción Utilidades del menú principal de OPEN ACCESS no puede considerarse como un módulo adicional del paquete integrado, sino más bien como un conjunto de opciones que facilitan la explotación del programa.

BASIC avanzado

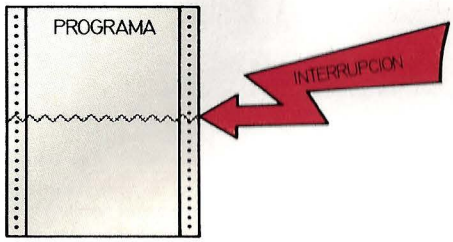
Gestión de interrupciones y teclas de función

La ejecución secuencial de las líneas de un programa escrito en BASIC, no permite el control del mismo en consonancia con el tiempo real. Así, normalmente, no será posible llevar a cabo una determinada acción en el mismo momento en el que se produce la situación que da lugar a ella; sino que será necesario esperar a que el programa llegue a la línea en la que se ha codificado la detección de dicho evento, con lo que siempre se producirá un cierto retraso.

Un ejemplo puede ser el siguiente. Suponga que se tiene un ordenador conectado, mediante la interface adecuada, a un detector de la presión de una caldera de vapor. El ordenador debe controlar mediante un programa BASIC que la presión de la caldera no supere un cierto valor máximo, de peligro, y explote en consecuencia. Si se alcanza ese valor máximo, el ordenador debe dar una señal de aviso, o bien actuar sobre la caldera mediante otra interface apropiada, para bajar la presión.

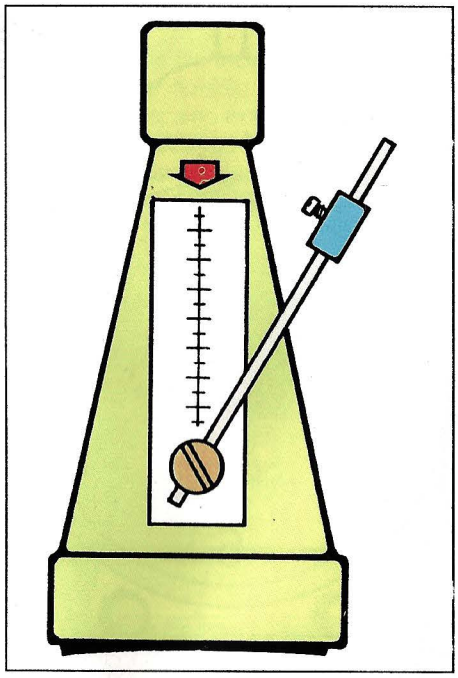
La programación de este objetivo puede ser tan simple como ordenar una lectura continua del vapor de la presión de la caldera, y comprobar si ha superado el valor máximo. Aunque bien es cierto que utilizar a todo un señor ordenador tan sólo para este cometido puede ser un gran lujo. Lo habitual es que, a la vez, se puedan ejecutar otro tipo de programas y, cada cierto intervalo de tiempo, detener la ejecución de estos para dedicar unos breves instantes al que controla la caldera.

Como es fácil intuir, puede suceder que uno de los otros programas demore demasiado en su ejecución y cuando se intente examinar la presión de la caldera, sea ya demasiado tarde. El resultado es fácil de imaginar. Puesto que no se puede prever el momento preciso en el que se alcanza esa situación, se necesita alguna técnica que permita "interrumpir" al ordenador en el preciso momento en el que se produzca el hecho que debe ser atendido con prioridad.



Las interrupciones permiten detener la ejecución de un programa, para así atender a una rutina de tratamiento del suceso que ha provocado la interrupción.

Esta técnica no es otra que la del uso de las interrupciones hardware del microprocesador. Su explicación detallada exigiría entrar en el campo del código máquina, lo que se sale fuera del objetivo de esta obra. No obstante, sin llegar a las profundidades del hardware ni a situaciones tan drásticas como las del ejemplo anterior, también es posible desde el BASIC la técnica de las interrupciones.



Con INTERVAL es posible generar una interrupción del programa principal, cada vez que transcurra un cierto intervalo de tiempo.

INTERRUPCIONES EN BASIC

Existen múltiples situaciones que pueden acaecer en un momento impredecible, y mediante las cuales se puede ejecutar una interrupción BASIC. Así, por ejemplo, se habló en un capítulo anterior de la colisión entre dos SPRITES y de la situación derivada de accionar el botón de disparo de un JOYSTICK. Ambas circunstancias dan pie para interrumpir el desarrollo normal del programa y acceder a una rutina para el tratamiento de estos eventos.

Existen otras tres situaciones habituales que permiten "interrumpir" la ejecución de un programa BASIC. Estas son las siguientes: la finalización de un determinado intervalo de tiempo, que se relaciona con la palabra clave INTERVAL; la acción sobre determinadas teclas de función especiales, anexas al teclado normal, y controladas por la palabra clave KEY, y la pulsación de la tecla STOP (normalmente junto con la tecla CONTROL) cuya palabra de control es STOP.

Cualquiera de estos eventos puede generar una interrupción instantánea del programa BASIC en curso. Cada uno de ellos necesitará entonces una rutina adecuada para el tratamiento de esa interrupción, rutina que se selecciona con el siguiente comando:

ON <evento> GOSUB

Bifurca a una subrutina al producirse una interrupción provocada por el <evento> indicado.
<eventos>: STRIG, SPRITE, STOP, KEY, INTERVAL

Formato: On <evento> {=<núm.>} GOSUB <nl.>[,...,<nl.>]

Ejemplos: 10 ON INTERVAL=100 GOSUB 1000
30 ON KEY GOSUB 100, 200, 300

Nota: <num.>=número de cincuenta-avos de sg. para INTERVAL.

<evento> ON OFF/STOP

...e o desactiva momentáneamente la interrupción asociada al <evento> indicado. Tiene relación con el comando ON <evento> GOSUB.
<eventos>: STRIG, SPRITE, STOP, KEY, INTERVAL

Formato: <evento> [ON/OFF/STOP]

Ejemplos: 10 KEY(3) OFF
20 INTERVAL STOP
30 STOP ON

<nl.> ON <evento> GOSUB
<nl.>[,<nl.>,...]

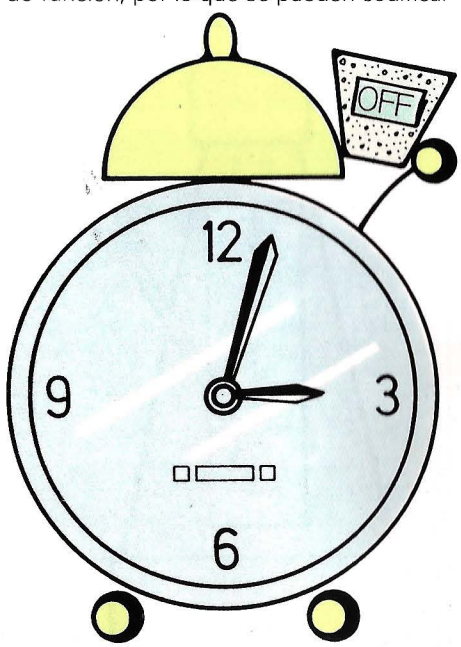
En ella, el <evento> puede ser cualquiera de las siguientes palabras clave: INTERVAL, KEY, STOP o las ya conocidas: STRIG y SPRITE.
En el caso de INTERVAL se debe especificar además el número de cincuenta-avos de segundo que se desea que transcurran antes de producir la interrupción. Por ejemplo, la línea:

```
10 ON INTERVAL=200 GOSUB 1000
```

producirá la interrupción del programa BASIC que se esté ejecutando a intervalos de 4 segundos, haciendo que se ejecute la subrutina de tratamiento situada en la línea 1000.
Si se desea interrumpir el programa cuando se pulse alguna de las teclas especiales de función, se empleará la siguiente formulación, incluyendo un número de línea asociado al tratamiento de cada una de las teclas de función que existan. Así, si son 5 las teclas de función, la siguiente línea BASIC:

```
10 ON KEY GOSUB 100,200,300, ,500
```

hará que se ejecute la subrutina de la línea 100 al pulsar la primera tecla, la de la línea 200 si se pulsa la segunda, y así sucesivamente. La cuarta tecla de función no producirá interrupción alguna, pues no se ha indicado el número de línea asociado a su tratamiento. No obstante, es obligatorio respetar su posición, por lo que se ha representado la "coma" correspondiente. Normalmente, existirán hasta diez teclas de función, por lo que se pueden codificar otras tantas rutinas de tratamiento. Tal posibilidad resultará muy útil para facilitar el control directo de un programa por medio de ellas.



Mediante el comando <evento> OFF, se pueden desactivar las interrupciones hasta el instante en el que se ejecute la orden <evento> ON.

La instrucción <evento>: STOP no necesita parámetros suplementarios y será útil para interrumpir la ejecución de un programa mediante la pulsación de la tecla STOP.
En cualquiera de los casos señalados, la interrupción se produce una vez que se termina de ejecutar la sentencia BASIC en curso. Acto seguido salta la ejecución a la subrutina correspondiente y, tras ello, se retorna a la instrucción siguiente a la que se estaba ejecutando cuando se produjo la interrupción.
Una vez programadas todas estas interrupciones, es posible también programar su aceptación o no en un momento determinado. Ello se debe realizar por medio de la siguiente instrucción BASIC:

<nl.> <evento> [ON] [OFF] [STOP]

En la que <evento> puede ser uno cualquiera de los ya estudiados (INTERVAL, KEY, STOP, STRIG, SPRITE), e irá seguido por alguna de las palabras alternativas indicadas en el formato (ON, OFF, STOP).
Previa a la ejecución de esta sentencia, debe haberse ejecutado la correspondiente intrucción ON <evento> GOSUB.
Si el <evento> es KEY será preciso especificar dentrás de él, y entre paréntesis, el número de tecla de función correspondiente.
La opción ON activa y permite la interrupción, de modo que ésta tendrá lugar si se produce a partir de entonces su <evento> particular. Para desactivar una interrupción, se debe elegir la opción OFF, con lo que el ordenador ignorará los eventos producidos a partir de ese momento.
La opción STOP permite desinhibir las interrupciones, pero éstas serán anotadas internamente para generarlas en cuanto se ejecute una nueva opción ON, para ese mismo <evento>.
Estos son algunos ejemplos que muestran la posible sintaxis de este tipo de instrucción:

```
10 KEY(2) ON
100 INTERVAL STOP
50 STOP OFF
```

La línea 10 permite poner en estado de autorización las interrupciones generadas al pulsar la tecla de función 2. La línea 100 impedirá momentáneamente las interrupciones periódicas generadas cada inter-

valo de tiempo indicado; éstas se producirán cuando se ejecute posteriormente una sentencia INTERVAL ON. Por último, la línea 50 hará que sean ignoradas las interrupciones generadas por medio de la tecla STOP.

LAS TECLAS DE FUNCION

En párrafos anteriores se han hecho varias referencias a las teclas de función. Este es el momento de entrar más a fondo en su significado y utilidades.

Cuando se está en modo edición y se pulsa, por ejemplo, la tecla correspondiente a la letra A, el ordenador ejecutará una rutina interna que derivará en la impresión en pantalla de dicha letra. Tal respuesta se fundamenta en una serie de acciones elementales. En primer lugar, el ordenador advierte el hecho de que ha sido pulsada una tecla, mediante una interrupción interna generada por el controlador del teclado. Después se identifica la tecla que ha sido accionada y se decodifica su valor para obtener el código correspondiente. Por último, y según sea este código, se imprime en pantalla el patrón de bits correspondiente a ese carácter, o bien se ejecuta la función específica asociada a la tecla pulsada (RETURN, función de edición...).

Cada tecla tiene asignada por lo tanto, una función específica predefinida, que sólo podrá ser combinada mediante ciertos trucos de programación en BASIC, fundamentados en el uso de los comandos INKEY$ o GET. Estos obtienen el código de la tecla pulsada; códigos a partir de los que será posible tomar acciones distintas de la de imprimir en pantalla el carácter asociado a esa tecla. He aquí un sencillo ejemplo:

```
...
100 C$=INKEY$ : IF C$="" THEN GOTO 100
110 IF C$="F" THEN PRINT "FIN" : END
120 GOTO 10
```

En él se asigna artificiosamente a la tecla "F" la misión de imprimir el mensaje "FIN" y acabar el programa con el comando END. Se puede decir, en cierto modo, que esas teclas son programables artificialmente.

Sin embargo, lo ideal sería que fuesen totalmente programables, en el sentido de que fuera posible modificar el código interno que entregan. Pues bien, esto es posible en muchos ordenadores presentes en el mercado, gracias a la existencia de las denominadas "teclas de función". Teclas que suelen estar físicamente separadas de las convencionales alfanuméricas y que pueden llegar al número de diez.

A cada una de las teclas de función se le puede asignar no sólo el carácter que desee el usuario, sino incluso una cadena de hasta 16 caracteres en muchos casos, además de caracteres de control.

El comando BASIC que permite asignar una determinada cadena a una de estas teclas es el que sigue:

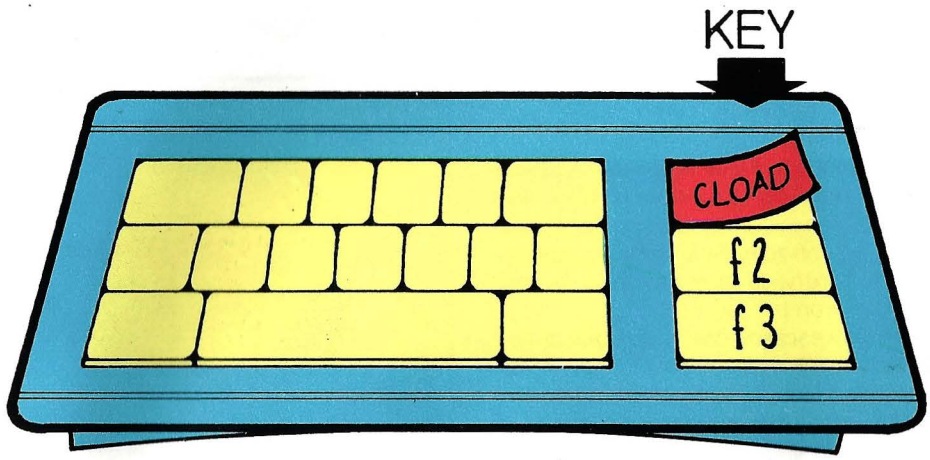
KEY <num. de tecla>, <cadena>

Donde el <núm. de tecla> es el que normalmente tiene asignado la tecla de función cuyo cometido se quiere modificar, y <cadena> es el nuevo texto a asignarle. Así, por ejemplo, si se desea que al pulsar la tecla de función 1 se ejecute el comando RUN<CR>, es suficiente con indicar a la máquina:

KEY 1, "RUN"+CHR$(13)

En dicha orden se asume que CHR$(13) es el carácter correspondiente a la acción

La presencia de teclas de función programables por el usuario es un detalle frecuente en los modernos microordenadores, tanto de tipo doméstico como profesional. En la fotografía se observan las ocho teclas de función (de color azul) del microordenador ENTERPRISE.



El comando KEY permite programar las teclas de función, modificando su contenido.

KEY

Redefine la cadena de caracteres que se generará al pulsar las teclas de función.

Formato: KEY <núm. de tecla>, <cadena>

Ejemplos: 10 KEY 1, "RUN"+CHR$(13)
20 KEY 2, " "
30 KEY N, Z$

KEY [ON/OFF]

Borra o imprime en pantalla la línea con los contenidos de las diferentes teclas de función.

Formato: KEY [ON/OFF]

Ejemplos: 10 KEY OFF
50 KEY ON

del retorno de carro (acción habitual de la tecla RETURN o ENTER).
Por este sencillo método se habrá asignado a la tecla de función 1 la misión de ejecutar un programa BASIC, sin que haya necesidad de escribir letra por letra el comando RUN y pulsar después la tecla RETURN. Ahora, con sólo pulsar la tecla de función 1, se ejecutará directamente el programa que resida en memoria.
Para hacer que una de las teclas de función no ejecute nada, es decir que quede inactiva, no habrá más que asignarle la cadena nula (""), como se indica en el siguiente ejemplo:

```
KEY 10,""
```

Las posibles aplicaciones de las teclas de función son inagotables. Son útiles, por ejemplo, para imprimir textos que se repitan con cierta frecuencia, o datos como el nombre del usuario, o los comandos de uso más frecuente, como AUTO, LIST, LOAD, SAVE...
Los ordenadores que poseen estas teclas de control suelen asignarles normalmente unas tareas iniciales, éstas suelen visualizarse en una zona de la pantalla, especialmente reservada para ello, ya que suele coincidir con la línea inferior.
Debido al escaso espacio que ofrece esta línea, y a que cada tecla de función puede tener asociado un texto de hasta 16 caracteres, este texto no se visualizará completo en la línea de pantalla reservada al efecto. Para ver el contenido completo de cada tecla de función se dispone del comando KEY LIST, el cual proporciona un listado por pantalla de los contenidos íntegros de cada tecla.
Una posible ejecución de este comando daría lugar a la siguiente pantalla:

```
KEY LIST<CR>

RUN
LIST
SAVE
AUTO
SAVE "D:
LOAD "D:
CLOAD
JUAN PEREZ
SI
NO
■
```

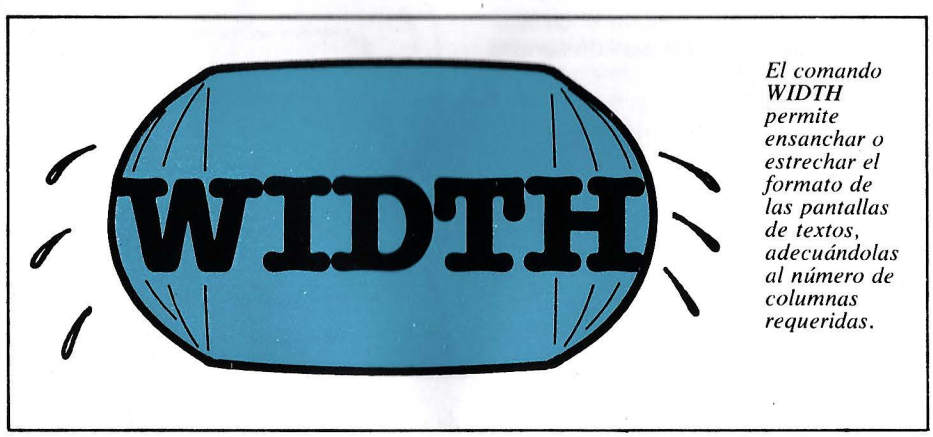
La existencia de la línea inferior de la pantalla destinada a recordar el cometido de las teclas de función, disminuye el espacio de visualización utilizable para otras finalidades. Para eliminar estas líneas y así disponer de una pantalla limpia de mensajes se dispone del siguiente comando BASIC:

```
KEY [ON] [OFF]
```

Mediante la opción OFF se hace desaparecer de la pantalla la línea de significados de las teclas de función, mientras que para hacerla aparecer de nuevo se utilizará la opción ON.

VARIANDO LA ANCHURA DE LA PANTALLA

El número de columnas visualizables en una pantalla de texto varía según el modo de presentación elegido y de acuerdo a su resolución máxima. Una resolución baja no permitirá más que 20 caracteres por línea, o menos en algunos casos; mientras que una pantalla de texto de alta resolución puede llegar a contener hasta 80 caracteres por línea.
Con independencia de la resolución, que será prefijada por medio de comandos tales como SCREEN, MODE o GRAPHICS, interesa en muchos casos seleccionar a voluntad la anchura eficaz de la pantalla. De esta forma, será posible, por ejemplo, reducir su anchura y así poder disponer del espacio de pantalla restante para otros menesteres. Al efecto, algunos dialectos



El comando WIDTH permite ensanchar o estrechar el formato de las pantallas de textos, adecuándolas al número de columnas requeridas.

del BASIC incluyen un comando específico.

WIDTH <número de columnas>

Su utilidad estriba en seleccionar el número de columnas que se desean visualizar para un determinado modo de pantalla de texto. El número de columnas seleccionado debe estar comprendido entre 1 y el número máximo de columnas admisible para esa resolución en modo texto.
Suponga que se trabaja con una pantalla cuya máxima resolución en modo texto es de 20 columnas por línea. Si se elige este modo y se visualiza el texto PRIMERTITULO, mediante la correspondiente orden (PRINT "PRIMERTITULO"), se obtendrá la siguiente pantalla:

```
PRIMERTITULO
■
```

Mientras que si se ejecuta previamente a la misma orden PRINT, la instrucción SCREEN 1 : WIDTH 6 el resultado será:

```
PRIMER
TITULO
■
```

Lo primero que se observa es que la anchura de la pantalla de textos ha sido reducida a 6 columnas, por lo que cada seis caracteres visualizados se produce un cambio a la línea siguiente. Además, se observa que el texto así representado queda centrado, debido a que el comando WIDTH modifica el número de columnas alternativamente a derecha e izquierda.
Una vez seleccionada la anchura, ésta quedará normalmente memorizada y en activo cada vez que se vuelva a utilizar ese modo de textos.

WIDTH

Modifica el número de columnas que se visualizarán en una pantalla de texto o en una impresora. El nuevo formato de pantalla queda centrado.

Formato: WIDTH <núm. de columnas>

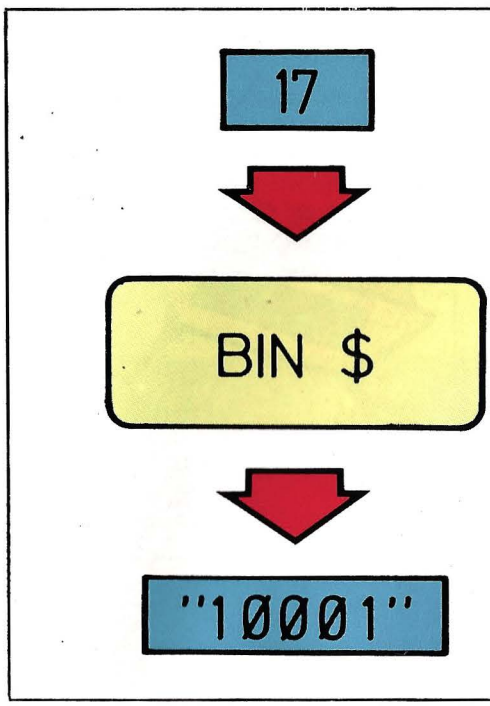
Ejemplos: 10 SCREEN 1 : WIDTH 30
30 SCREEN 2 : WIDTH 10

LOS SISTEMAS DE NUMERACION

El sistema de numeración decimal es el habitualmente utilizado en las relaciones humanas, pero no es el único, ni mucho menos. Sin ir más lejos, los ordenadores utilizan internamente el sistema binario cuyos números están compuestos por cadenas de "ceros" y "unos".
Algunos intérpretes de BASIC permiten obtener los equivalentes binarios de números decimales. Como herramienta de ayuda para el manejo de números binarios, existe una función BASIC cuyo cometido es obtener una cadena de caracteres, compuesta por ceros y unos, equivalente a la expresión binaria del número decimal dado en su argumento. Esta función tiene el siguiente formato:

X$=BIN$ (<número entero>)

Como ya se sabe, las funciones producen un resultado que debe ser tratado por un comando o, en su defecto, almacenado en alguna variable para su manipulación posterior. En este caso, dicha misión se ha encomendado a la variable alfanumérica X$. No hay que perder de vista que el resultado entregado por BIN$ es una cadena de caracteres.
El argumento de BIN$ debe ser un número entero, lo que constituye en cierto



El BASIC dispone de la función BIN$ para transformar un número entero en su equivalente en notación binaria o base 2.

TABLA DE CONVERSION

ORDENADOR	Interrupciones		Teclas de función		Conversión de código			WIDTH	Carga y almacenamiento binario	
	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento> [ON/ OFF/ STOP]	KEY <núm.>, <var$>,	KEY [LIST/ ON/ OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH <núm.>	BSAVE <arch.>, <dir1>, <dir2>, [<ejec.>]	BLOAD <arch.> [,R] [,<off.>]
APPLE II (APPLESOFT)	—	—	—	—	—	—	—	—	BSAVE <arch.>, <dir1>, <lon>	BLOAD <arch.>
APRICOT (M-BASIC)	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH [LPRINT] <núm.> (1)	BSAVE <arch.>, <dir2>, <lon.>	BLOAD <arch.>, <dir2>
ATARI	—	—	—	—	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—	—	—	—	—
DRAGON	—	—	—	—	HEX$ (<núm.>)	—	—	—	—	—
EQUIPOS MSX	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento>[ON/ OFF/STOP]	KEY <núm.>, <var$>	KEY[LIST/ ON/OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH <núm.>	BSAVE <arch.>, <dir1>, <dir2>, [<ejec.>]	BLOAD <arch.> [,R] [,<off>]
HP-150	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH [LPRINT] <núm.> (1)	BSAVE <arch.>, <off.>,<lon.>	BLOAD <arch.> [,<off.>]
IBM PC	ON <evento> GOSUB <nl.> (1)	<evento> [ON/OFF/STOP] (1)	KEY <núm.>, <var$>	KEY [LIST/ ON/OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH <disp.>, <núm.>	BSAVE <arch.> <dir1>,<dir2> [<ejec.>]	BLOAD <arch.>[,R] [,<off.>]

modo una pequeña limitación. El rango de este entero debe estar comprendido entre los valores −32768 y +65535. Los márgenes positivo y negativo deben respetarse para evitar un error de "overflow" (desbordamiento). Téngase en cuenta que los números negativos son tratados de la misma forma que se manejan internamente, es decir, en complemento a dos. Así, BIN$(−1) producirá como resultado la siguiente cadena: "1111111111111111"; mientras que BIN$(−32768), dará como resultado: "1000000000000000". Como se puede observar, la longitud máxima de la cadena obtenida es de 16 caracteres.

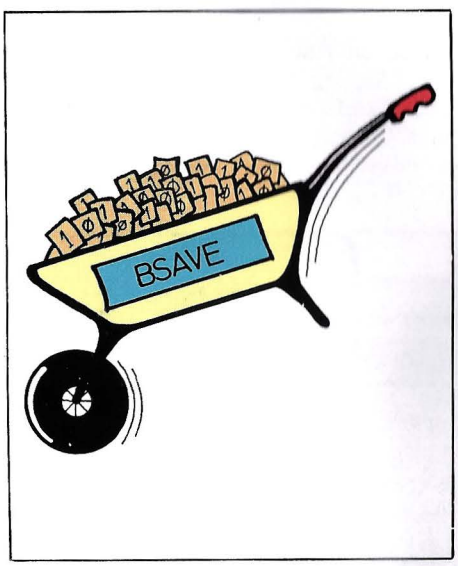
La conversión opuesta, es decir, el paso de binario a decimal, se puede realizar en muchos casos con la ya conocida función VAL, de la siguiente forma:

```
NUM=VAL("&B"+X$)
```

Siendo X$ la cadena de unos y ceros que representa el número en binario. Los signos "&B" se utilizan para indicar que se trata de un número en el sistema binario. Otro sistema de numeración, muy utilizado sobre todo por los programadores en código máquina, es el hexadecimal. Este sistema utiliza 16 dígitos diferentes para representar los números. Los dígitos se obtienen al añadir a las 10 cifras decimales (del 0 al 9), las letras de la A a la F. Existe, por tanto, otra función especial que obtiene la cadena de caracteres hexadecimales representativa del número decimal dado. Esta función tiene el siguiente formato:

```
X$=HEX$ (<número entero>)
```

En este caso, el argumento queda limitado también a números enteros y con el mismo rango que para la función BIN$. Así, pues, HEX$(−1) dará como resultado "FFFF", mientras que HEX$(−32768) entregará el resultado "8000".



BSAVE traslada una zona de RAM a un archivo en formato binario residente en cinta magnética o disco.

TABLA DE CONVERSION

ORDENADOR	Interrupciones		Teclas de función		Conversión de código			WIDTH	Carga y almacenamiento binario	
	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento> [ON/ OFF/ STOP]	KEY <núm.>, <var$>,	KEY [LIST/ ON/ OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH <núm.>	BSAVE <arch.>, <dir1>, <dir2>, [<ejec.>]	BLOAD <arch.> [,R] [,<off.>]
MPF	—	—	—	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH [LPRINT] <núm.>	—	—
NEW BRAIN	—	—	—	—	—	—	—	—	—	—
ORIC	—	—	—	—	HEX$ (<núm.>)	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	DEF KEY (<núm.>)= <var$>	KEY LIST	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—	WIDTH <disp.>, <núm.>	SBYTES <arch.>, <dir1>, <lon>	SBYTES <arch.>, <dir1>
SPECTRA-VIDEO	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento> [ON/OFF/ STOP]	KEY <núm.>, <var$>	KEY [LIST/ ON/OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH <núm.>	BSAVE <arch.>, <dir1>, <dir2> <ejec.>	BLOAD <arch>, <dir1>
ZX-SPECTRUM	—	—	—	—	—	—	BIN	—	—	—

<evento>: cualquiera de los siguientes, STRIG, STOP, KEY, INTERVAL, SPRITE. <nl.>: número de línea. <núm.>: número entero. <var.$>: variable alfanumérica. (1) El IBM-PC admite los siguientes <eventos>: KEY(n), STRIG, PEN (lápiz óptico) y COM (comunicaciones).

<núm.>: número entero. <arch.>: nombre del archivo. <dir.1>: dirección inicial. <dir.2>: dirección final. <ejec.>: dirección de ejecución. <off.>: dirección offset para cargar en cualquier zona RAM. <lon.>: longitud en direcciones. <disp.>: dispositivo o número de archivo. (2) Para impresora se usa la opción LPRINT.

Análogamente al caso binario, para la conversión inversa se utilizará la función VAL, aunque especificando ahora que se trata de un número en notación hexadecimal, mediante los signos "&H" colocados delante del número en cuestión:

```
NUM=VAL("&H"+X$)
```

El sistema de numeración de base 8 u OCTAL, es también frecuente en el mundo informático. Para él existe también una función análoga a las BIN$ y HEX$, con las mismas características. Se trata de OCT$ cuyo formato es idéntico a los anteriores.
En el argumento de cualquiera de las tres funciones presentadas son admisibles números expresados en cualquier sistema de numeración; ello se indicará en cada caso con el signo "&", seguido por la inicial del sistema de numeración. Así, para la hexadecimal se emplea "&H", para el octal "&O" y para el binario "&B".
Por ejemplo:

HEX$, BIN$, OCT$

Funciones que dan, a partir de un número entero, la cadena de caracteres equivalente en los sistemas de numeración hexadecimal, binario u octal, respectivamente.

Formato: <Var$>=[HEX$/BIN$/OCT$] (<núm. entero>)

Ejemplos:

```
10 A$=HEX$(100)
20 A$=BIN$(&HFFO)
40 PRINT OCT$(&B110101)
```

```
PRINT OCT$(&HFFFE)
177776
■
```

BSAVE

Almacena en un archivo binario la zona de memoria especificada, permitiendo indicar una dirección de autoejecución para cuando sea cargado de nuevo.

Formato: BSAVE <archivo>, <dir. inicial>, <dir. final> [,<autoejec.>]

Ejemplos: 100 BSAVE "C:",&H3000,&H30FF
200 BSAVE "D: AUTORUN",&HF000,&HF100,&HF000

BLOAD

Carga en memoria RAM un archivo de tipo binario; éste puede autoejecutarse una vez finalizada la carga. Mediante una dirección "offset" es posible realizar la carga en cualquier zona de la memoria RAM.

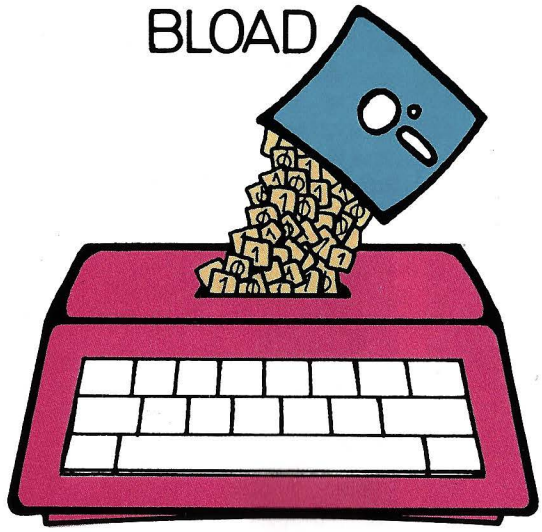
Formato: BLOAD <archivo>[,R] [,<offset>]

Ejemplos: 100 BLOAD "C:"
200 BLOAD "D: AUTORUN", R, 1024

La cadena que entregan estas funciones como resultado será de distinta longitud según sea el número a transformar. Así, por ejemplo, BIN$(7) producirá como resultado una cadena de tres caracteres de longitud: "111"; mientras que con BIN$(128) la cadena resultante tendrá 8 caracteres "10000000".
El problema se presenta al querer visualizar en la pantalla varios de estos números, todos con el mismo formato. Para que ello se produzca de forma correcta, será necesario añadir delante de la cadena resultante el número de ceros necesario. El que sigue es un artificio plenamente eficaz al respecto:

```
X$=RIGHT$("00000000"+HEX$(X$), 8)
```

Ahora, todos los números aparecerán en formato de ocho caracteres.



BLOAD devuelve un archivo binario a las mismas posiciones de RAM que ocupaba en el instante de ser guardado en cinta o disco. Sumando un offset, es posible cargarlo en otra zona cualquiera de RAM.

MANIPULANDO CODIGOS BINARIOS

Aunque ya se han estudiado dentro de la obra los comandos principales para programar en código máquina (POKE, PEEK, URS(), CALL, ...) quedan todavía en el tintero algunos comandos de indudable utilidad. Comandos que permiten, por ejemplo, transferir los códigos binarios de una zona de memoria a un archivo en disco o casete, y viceversa. Estos códigos binarios pueden pertenecer a un programa escrito en lenguaje máquina, o bien a un bloque de datos de pantalla.
El comando adecuado para transferir el contenido de una zona de memoria a un archivo es BSAVE, el cual adopta el siguiente formato:

BSAVE <archivo>, <comienzo>, <final>[,<ejecución>]

En donde <archivo> es una cadena de caracteres precedida por el código de dispositivo, representativa del nombre del archivo; <comienzo> y <final> son las direcciones inicial y final, respectivamente, que delimitan la zona de memoria que se desea transferir al archivo. Y la opción <ejecución> refleja la dirección de memoria a partir de la que se comenzará a ejecutar el código binario, una vez cargado éste y si ha sido indicada esta opción en la instrucción de carga. Por ejemplo:

```
BSAVE "C:",&hF000,&hF100
```

almacenará en un archivo en casete la zona de memoria comprendida entre las direcciones en hexadecimal F000 y F100, y no se ejecutará el código cargado.
El comando especializado en la carga de estos archivos binarios es el que sigue:

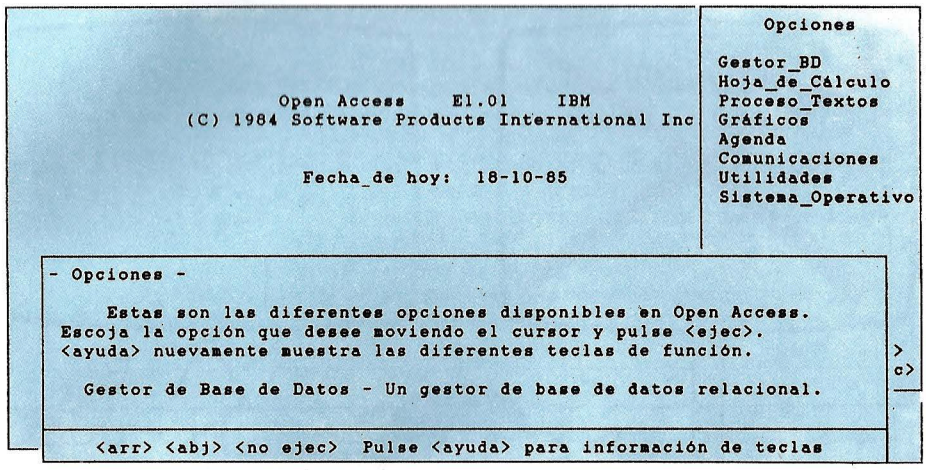
BLOAD <archivo>[,R] [,<offset>]

En donde <archivo> tiene el mismo significado que en el caso de BSAVE. La opción R indica que se desea autoejecutar el código cargado a partir de la dirección que se hubiese indicado en el comando BSAVE. La dirección <offset> se sumará a las direcciones de comienzo, final y ejecución que se indicasen con el comando BSAVE, lo que permitirá cargar este fichero en cualquier zona de la memoria del ordenador.

FORTRAN (y 2)

Estructuras de control y entrada/salida

En este segundo capítulo dedicado al FORTRAN se estudiarán las sentencias de control, los subprogramas y los mecanismos de entrada/salida. Estos últimos constituyen uno de los puntos más criticables del FORTRAN por su complejidad y falta de flexibilidad. Como atenuante hay que recordar que nuestro protagonista está orientado a aplicaciones de cálculo numérico, las cuales consisten generalmente en un gran volumen de cálculo generado a partir de unos pocos datos de entrada. Por esta razón, tales mecanismos pueden mantenerse a dicho nivel.



Las sentencias de control son las que dirigen la secuencia de ejecución del programa.

ESTRUCTURAS DE CONTROL

La tabla adjunta contiene un resumen de tales estructuras las cuales se describen a continuación.

- GO TO incondicional

Análogo al GOTO del Basic. Su forma es:

GO TO n

en donde "n" es un número entero que corresponderá a uno de los números de sentencia que aparecen en el programa. En FORTRAN no hay que preceder a cada sentencia con un número de línea como sucede en Basic. Sólo unas pocas sentencias pueden ir precedidas por tal número, el cual servirá como indicación del lugar al que hay que realizar saltos o para señalar el punto final de algunas estructuras que se describirán más adelante.

- GO TO calculado

Su forma es la que sigue:

GO TO $(n_1,n_2,...,n_k),i$

en donde "n_i" son números de sentencia e "i" una variable entera que no sea elemento de un array. Al llegar a este punto, se producirá el salto a la sentencia n_j si "i" vale j. Si "i" es menor que uno o mayor que k, la sentencia ejecutada será la que esté a continuación del GO TO.

- GO TO asignado

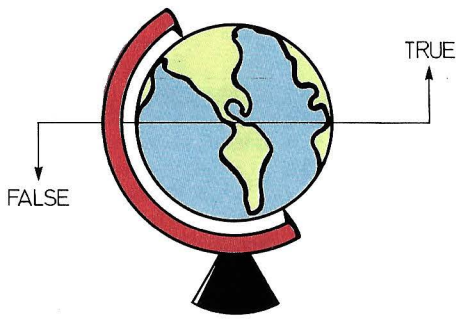
Su formulación es:

GO TO i, $(n_1,n_2,...,n_k)$

El tipo y significado de "n_i" son análogos al caso anterior. La diferencia está en que a variable "i" tomará los valores $n_1,n_2,...,n_k$ en vez de 1,2,...,k. Además, el valor concreto que en cada instante tome "i" le será asignado a través de una sentencia especial, ASSIGN, de esta forma:

ASSIGN j TO i

en donde "i" es la variable del GO TO y "j" una constante entera cuyo valor es el que ha de tomar dicha variable.



Una expresión lógica divide a todo el universo de posibles valores en dos alternativas: verdadero o falso.

- IF aritmético

Su aspecto es:

IF (a) n_1,n_2,n_3

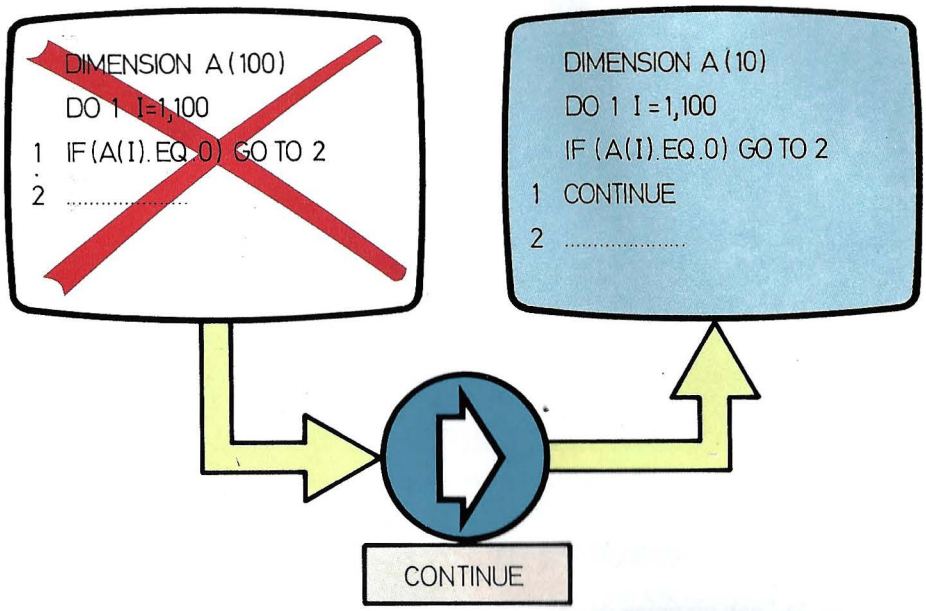
donde "a" es una expresión aritmética no compleja y n_1,n_2 y n_3 son números de sentencia. La bifurcación se realizará hacia una de estas dependiendo de que el valor resultante de evaluar "a" sea un número negativo, nulo, o positivo, respectivamente.

● IF lógico
Adopta la siguiente formulación:

IF (a) b

donde "a" es una expresión lógica cualquiera y "b" una sentencia ejecutable (esto es, no de tipo declarativo como IMPLICIT, DIMENSION, etc.) excepto una DO u otra IF de tipo lógico. La tabla adjunta revela los aspectos que pueden tomar las expresiones lógicas.

● DO
Esta es la forma básica de realizar bucles. Su aspecto general es:



Cómo solucionar el problema del DO...

DO n i=m_1,m_2,m_3

en donde "n" es el número de una sentencia ejecutable que aparecerá en el listado con posterioridad a la propia DO; puede ser cualquiera excepto una DO,GO TO, IF, STOP o RETURN. "i" es una variable entera no perteneciente a un array y m_1,m_2 y m_3 son constantes o variables enteras; si se trata de variables, tampoco pueden pertenecer a un array.
En operación, el rango de sentencias desde la propia DO hasta la "n" se ejecuta para valores de "i" desde m_1 hasta m_2, con incrementos de valor m_3. Este último parámetro es opcional y su valor por defecto es uno.
Si la última sentencia del bucle ha de ser una de las prohibidas, se soluciona la situación con la sentencia CONTINUE, cuyo efecto es el de volver a la cabecera del DO para iniciar una nueva iteración (ver figura).

● PAUSE
Al aparecer esta sentencia se detiene momentáneamente el programa y el usuario puede reinicializarlo o detenerlo definitivamente.

● STOP
Al llegar aquí el programa deja de ejecutarse, sin posibilidad de continuar la ejecución.

● END
Indica el final "físico" del programa principal de los subprogramas.

LA ENTRADA/SALIDA

Las sentencias de entrada/salida en FORTRAN toman el siguiente aspecto:

<código de operación> (<número de fichero>,<formato>) <lista>

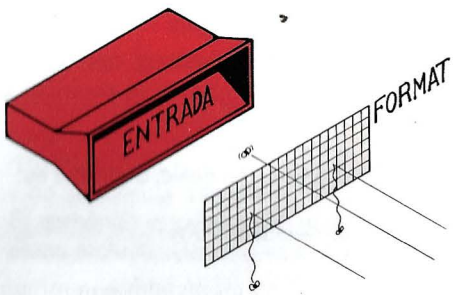
El código de operación puede ser una de las palabras clave READ o WRITE y la "lista" son las variables cuyos contenidos se van a leer o escribir. El "número de fichero" especifica el dispositivo de E/S a utilizar: pantalla, teclado, impresora, lectora de tarjetas perforadas (...en los viejos tiempos), etc. Este número depende de la configuración del ordenador utilizado y varía de unos equipos a otros. En el campo "formato" se incluye un número entero que será el de una sentencia FORMAT, en la cual se especifica la forma en que se ha de realizar la entrada/salida, de manera análoga a como lo hace el "string de formato" de la función "printf" o "scanf" del lenguaje C. Por ejemplo, el siguiente programa:

```
      INTEGER ENTERO
      ENTERO=1
      REAL=3.14
      WRITE (5,500) ENTERO, REAL
500   FORMAT (I5,F10.2)
      END
```

hará que se escriban las variables ENTERO y REAL a través del dispositivo que tiene asociado el número 5 (la pantalla, por ejemplo), con el formato que se especifica en la sentencia FORMAT cuyo número es 500. El formato "I5" reserva un campo de 5 posiciones que albergarán a un entero, mientras que "F10.2" reserva diez posiciones en total para un número real, dos de las cuales estarán dedicadas a la parte decimal. El resultado de la ejecución de nuestro programa sería, por tanto:

bbbb1bbbbb10.2b

en donde los caracteres "b" significan espacios en blanco.
El mismo comentario que se hizo sobre "scanf" en el lenguaje C es aplicable a la sentencia READ: el formato de la misma actúa como un filtro de la entrada que se está tecleando, admitiendo tan sólo los datos que estén de acuerdo con el mismo.
En la sentencia FORMAT, al igual que en el string de formato del C, se pueden incluir caracteres de control; la mayor parte de ellos aparecen en la correspondiente tabla, junto a los principales especificadores de campo.



La sentencia FORMAT establece el tamiz a través del cual serán tratados los datos de entrada o de salida.

```
      INTEGER FUNCTION FACT (*, X)
      INTEGER X
      FACT=1
      IF (X-1) 1, 4, 2
    1 RETURN 1
    2 DO 3 I=2, X
    3 FACT=FACT*I
    4 RETURN
      END
```

Ejemplo de llamada:

```
I=FACT($10,5)
```

PROGRAMA 1: Función FORTRAN para calcular el factorial de un número.

```
      SOBROUTINE FACT(*, X, XFACT)
      INTEGER X, XFACT
      XFACT=1
      IF (X-1) 1, 4, 2
    1 RETURN 1
    2 DO 3 I=2, X
    3 XFACT=XFACT*I
    4 RETURN
      END
```

Ejemplo de llamada:

```
CALL FACT ($10,VALOR,RESUL)
```

PROGRAMA 2: Subrutina para el cálculo del factorial de un número. En el ejemplo de llamada, "valor" y "resul" han sido declaradas previamente como variables enteras.

LOS SUBPROGRAMAS: FUNCTIONS Y SUBROUTINES

La declaración de una FUNCTION se realiza de la siguiente forma:

tipo FUNCTION f (a_1,...,a_n)

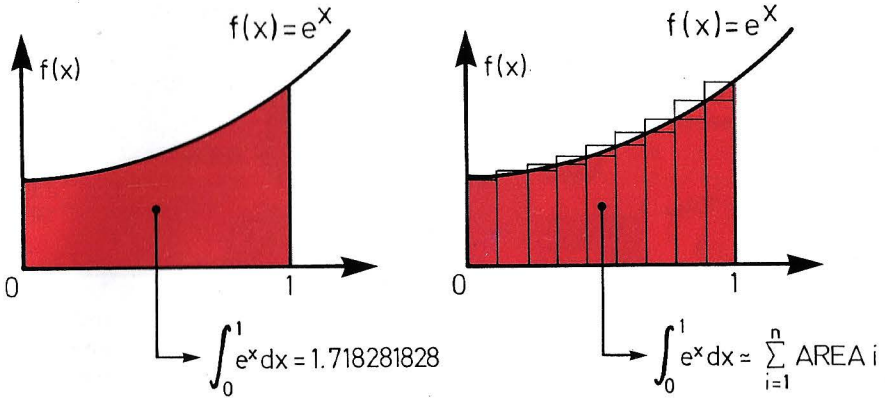
Especificadores de campo y caracteres de control

/	Salto de línea.
Wx	Deja "x" espacios en blanco a partir de la posición en que estaba.
Tx	Coloca la cabeza en la posición "x".
Ix	Formato para campo entero de "x" posiciones.
Fx.y	Formato para campo real en punto flotante de "x" posiciones, estando "y" de ellas reservadas a la parte decimal.
Ex.y	Formato para campo real en notación exponencial de "x" posiciones, estando "y" de ellas reservadas al exponente.
Dx.y	Como Ex.y aunque para variables de doble precisión.
Lx	Especifica un campo de "x" posiciones para una variable lógica.
' '	Los caracteres que estén escritos entre comillas simples serán escritos directamente.

¿Qué es el cálculo numérico?

Cuando las necesidades de cálculo sobrepasan la simple resolución de funciones conocidas como sumas, productos, cálculos trigonométricos, etc., y lo que interesa es resolver integrales, ecuaciones en derivadas parciales o sistemas de ecuaciones diferenciales, es preciso recurrir a técnicas de cálculo numérico. Estas, a base de sencillas operaciones elementales como las citadas en primer lugar, permiten resolver cálculos complejos. El precio a pagar será una cierta inexactitud en el resultado, lo cual se verá compensado por la facilidad con que se llegará al mismo.

Por ejemplo, si se desea calcular el área encerrada bajo una curva, habría que resolver la integral correspondiente y calcular su valor entre las abscisas de interés. La alternativa que ofrece el cálculo numérico consiste en calcular el resultado a partir de sumas de áreas elementales, como se observa en la figura. La precisión del resultado dependerá de la "elementalidad" de dichas áreas.



donde "f" es el nombre de la función, el cual, en el caso de excluir el campo "tipo", será el que lo determine. Los parámetros a_i pueden ser variables de cualquier tipo o asteriscos, en cuyo caso se utilizarán para posibilitar retornos múltiples.
El uso de las funciones es análogo al propio del Pascal; esto es: incluyendo el nombre de la función con sus argumentos entre paréntesis en una expresión.
Para marcar el final de la ejecución de una FUNCTION o SUBROUTINE se introduce una sentencia RETURN o RETURN n en el lugar apropiado. Si se utiliza la segunda forma, "n" debe ser la posición en la lista de argumentos de uno de ellos de tipo asterisco. Junto al texto aparece una función FORTRAN adecuada para calcular el factorial de un número. Como se observa, la palabra END marca el final físico de la subrutina. Una llamada a la función así declarada puede ser la que sigue:

I=FACT($10,5)

Comprobamos que el asterisco de la declaración corresponde en la llamada con "$10", en donde 10 será el número de alguna sentencia ejecutable. En nuestro caso, si como consecuencia de la ejecución del IF aritmético se llega a RETURN 1, se retornará al punto de llamada y de ahí se saltaría a la sentencia de número diez. Si el final de la función lo marca un sencillo RETURN, el control es devuelto sin más complicaciones, tomando la variable FACT el valor 120.
La otra forma de realizar un subprograma es a través de una SUBROUTINE. Su declaración toma el siguiente aspecto:

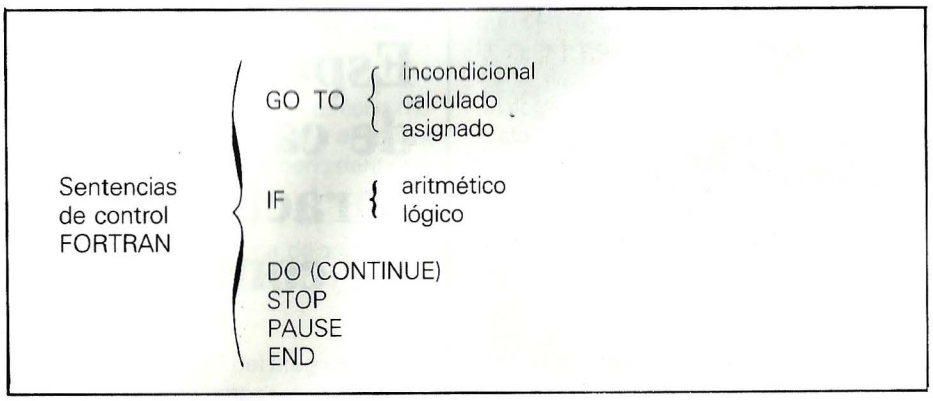
SUBROUTINE s(a_1,...,a_n)

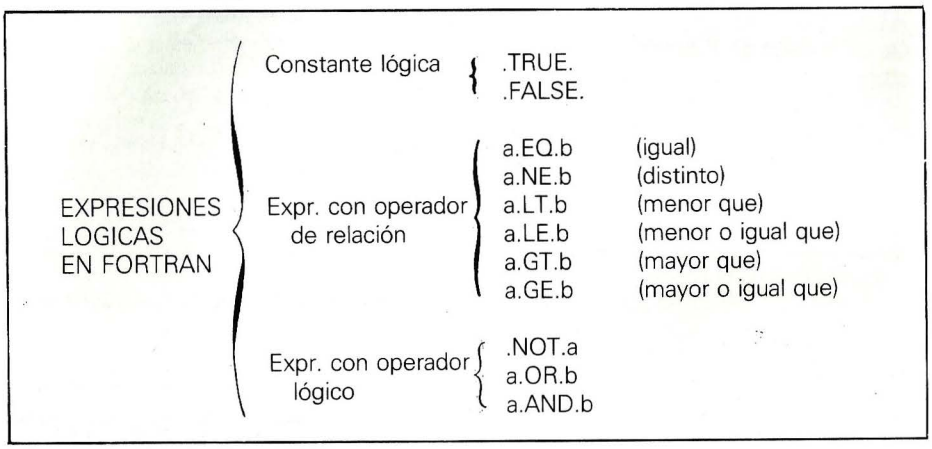
donde "s" es el nombre de la subrutina y a_1...a_n los argumentos, los cuales son siempre pasados por referencia, esto es: son todos análogos al tipo VAR en Pascal.
Una llamada a una subrutina se realiza a través de una sentencia CALL como sigue:

CALL s

Cabe observar que el nombre de la subrutina ("s") sirve tan sólo como referencia a la misma y que, por lo tanto, no lleva asociado tipo alguno ya que no albergará ningún valor. En la figura adjunta se ha resuelto el problema del cálculo del factorial a través de la subrutina. Las mismas consideraciones sobre los argumentos de tipo asterisco son aplicables al caso de las SUBROUTINEs.

Por último, hay que citar que se puede elegir el punto de entrada a una subrutina a través de la sentencia ENTRY, aumentando así la flexibilidad de las mismas.





PALABRAS CLAVES DEL FORTRAN

ASSIGN TO	ENTRY	REAL
BACKSPACE	EQUIVALENCE	RETURN
BLOCK DATA	EXTERNAL	REWIND
CALL	FORMAT	SAVE
CHARACTER	FUNCTION	STOP
CLOSE	GO TO	SUBROUTINE
COMMON	IF THEN	WRITE
COMPLEX	IMPLICIT	
CONTINUE	INQUIRE	
DATA	INTEGER	
DIMENSION	INTRINSIC	
DO	LOGICAL	
DOUBLE PRECISION	OPEN	
ELSE	PARAMETER	
ELSE IF THEN	PAUSE	
END	PRINT	
END IF	PROGRAM	
ENDFILE	READ	

OS-9 (y 2)

Memoria y multiprogramación

Debido a su directa descendencia del UNIX, el sistema operativo OS-9 está perfectamente capacitado para trabajar en el régimen denominado de multiprogramación. En esta modalidad de operación, también conocida como tiempo compartido, el OS-9 puede mantener más de un programa funcionando en el ordenador.

El procedimiento seguido normalmente por los ordenadores para operar en esta modalidad, es asignar un tiempo de CPU determinado a cada uno de los programas que están ejecutándose en su interior. Cuando llega el momento de que un determinado programa pase a ocupar los recursos de la CPU, éste es cargado en la memoria principal desde un área de almacenamiento especial, localizada en uno de los periféricos de almacenamiento masivo asociados al ordenador; mientras, el programa anteriormente en curso, es "descargado" de la memoria central del ordenador y depositado en la referida área de almacenamiento masivo.

Por medio de esta técnica, es posible mantener varios programas ejecutándose simultáneamente sobre el mismo equipo.

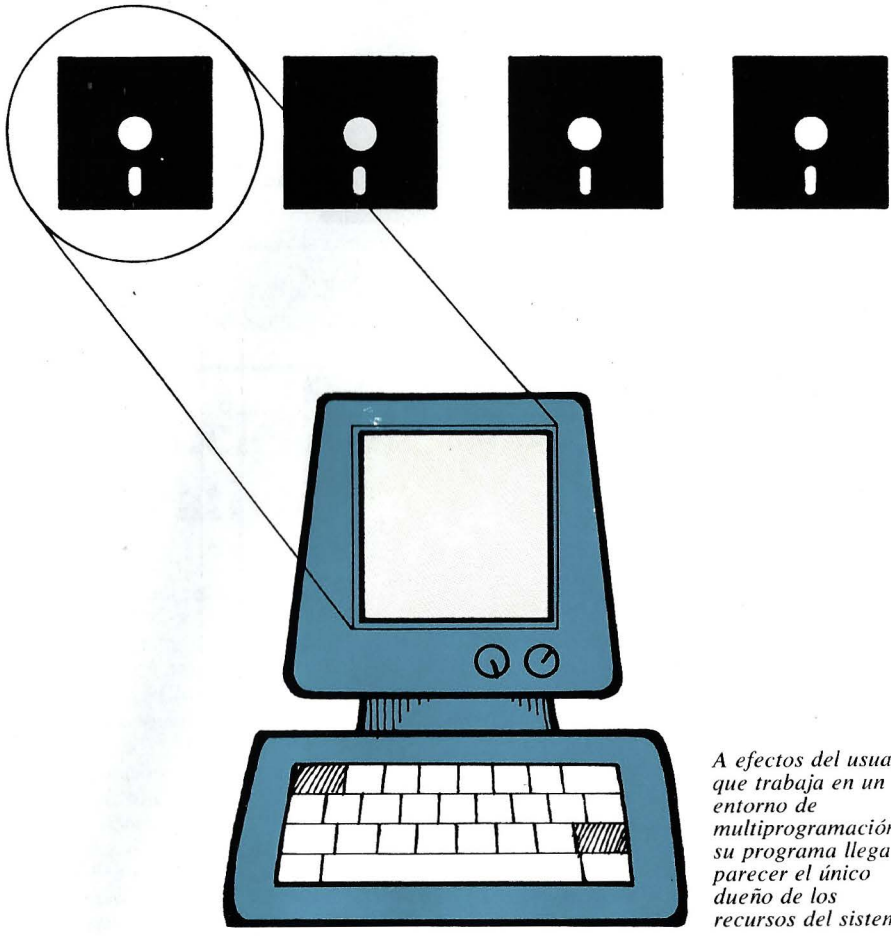
En el caso del OS-9 nos encontramos con una situación diferente, toda vez que el entorno para el cual ha sido creado, el de los microordenadores, posee unidades de almacenamiento masivo relativamente lentas y de escasa capacidad para permitir su empleo en esta modalidad de funcionamiento. Es por ello que el OS-9 se ve obligado a mantener en la memoria principal del ordenador a todos los programas en curso, y gestiona internamente su asignación a los recursos de la CPU.

Para cumplir con estas funciones, el sistema operativo tiene que controlar los siguientes factores:

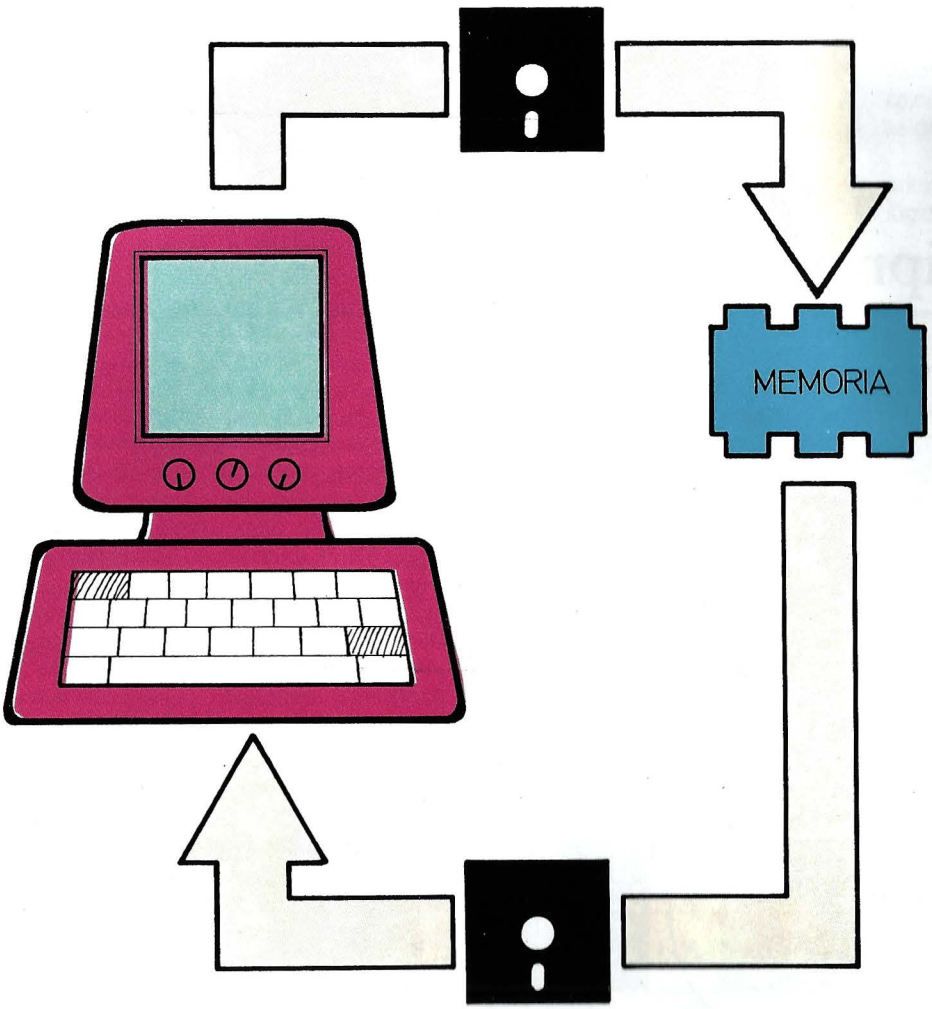
— Tiempo de CPU.
— Mapa de memoria y recursos disponibles de la misma.
— Sistema de Entrada/Salida

CONTROL DE LA CPU EN MULTIPROGRAMACION

Los grandes problemas derivados de un inadecuado control de tiempo de CPU, tienen su origen en el hecho de olvidar una regla fundamental del trabajo con ordenadores: que la operación de la CPU es muchísimo más rápida que la del ser humano o de los dispositivos de entrada/salida a través de los que comunica. Así, es frecuente que se pierda un tiempo precioso esperando que un programa gestione una operación de entrada/salida de información, en la que la CPU juega un papel muy restringido, puesto que la mayor parte del trabajo lo realiza el controlador del periférico con el que comunica.



A efectos del usuario que trabaja en un entorno de multiprogramación, su programa llega a parecer el único dueño de los recursos del sistema.

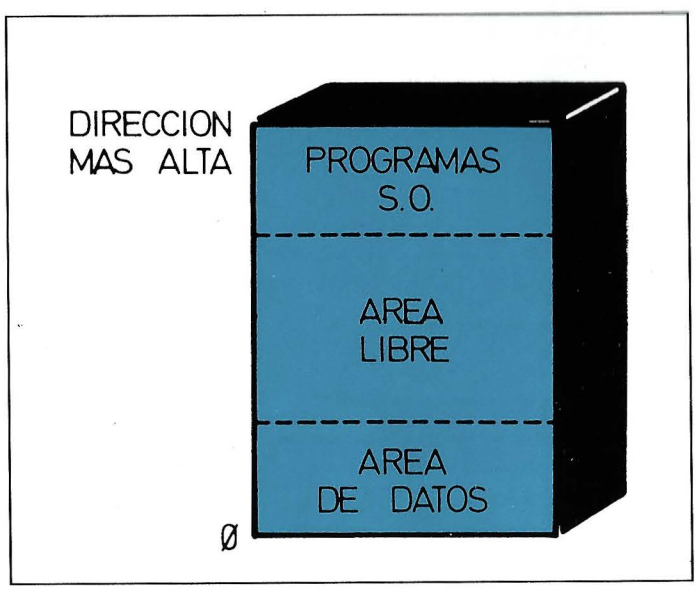


En el ámbito del OS-9, los programas en operación multitarea se mantienen continuamente en la memoria central del ordenador.

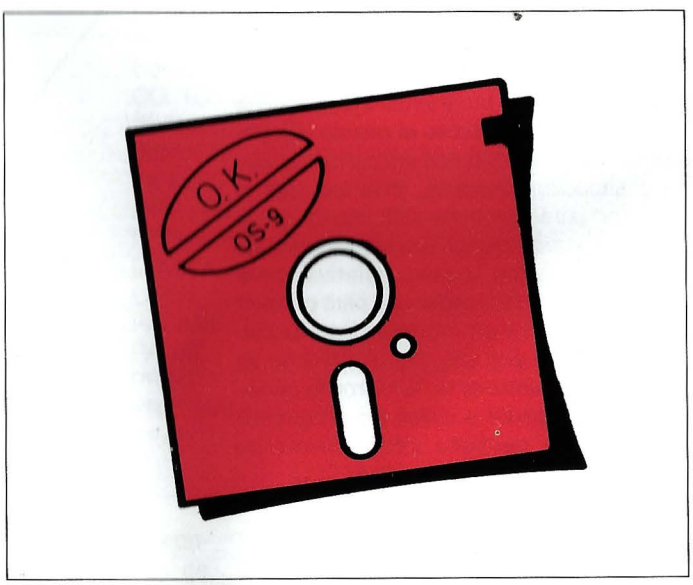
Como ejemplo típico de lo que acabamos de señalar cabe citar a los programas de tipo interactivo, en los cuales mientras que el usuario no introduzca la información necesaria para activar el proceso no puede llevarse a cabo ninguna tarea, con el consiguiente desperdicio de recursos.
El sistema operativo OS-9 hace uso de una técnica denominada "time slicing", por la cual divide el tiempo de CPU tan sólo entre aquellos programas que están en condiciones de hacer uso de sus recursos. Este proceso se lleva a cabo mediante el empleo de útiles especiales de hardware y software, cadenciados a partir del reloj de tiempo real del equipo. Las divisiones que se hacen del tiempo de CPU dependen de la prioridad relativa de los programas en ejecución, ya que, lógicamente, a mayor prioridad tanto mayor será el tiempo de CPU que se asigne al proceso en cuestión.
En términos generales, el tanto por ciento de recursos otorgados a un programa viene definido por la expresión:

Cuota de CPU=Prioridad del programa/ suma de prioridades

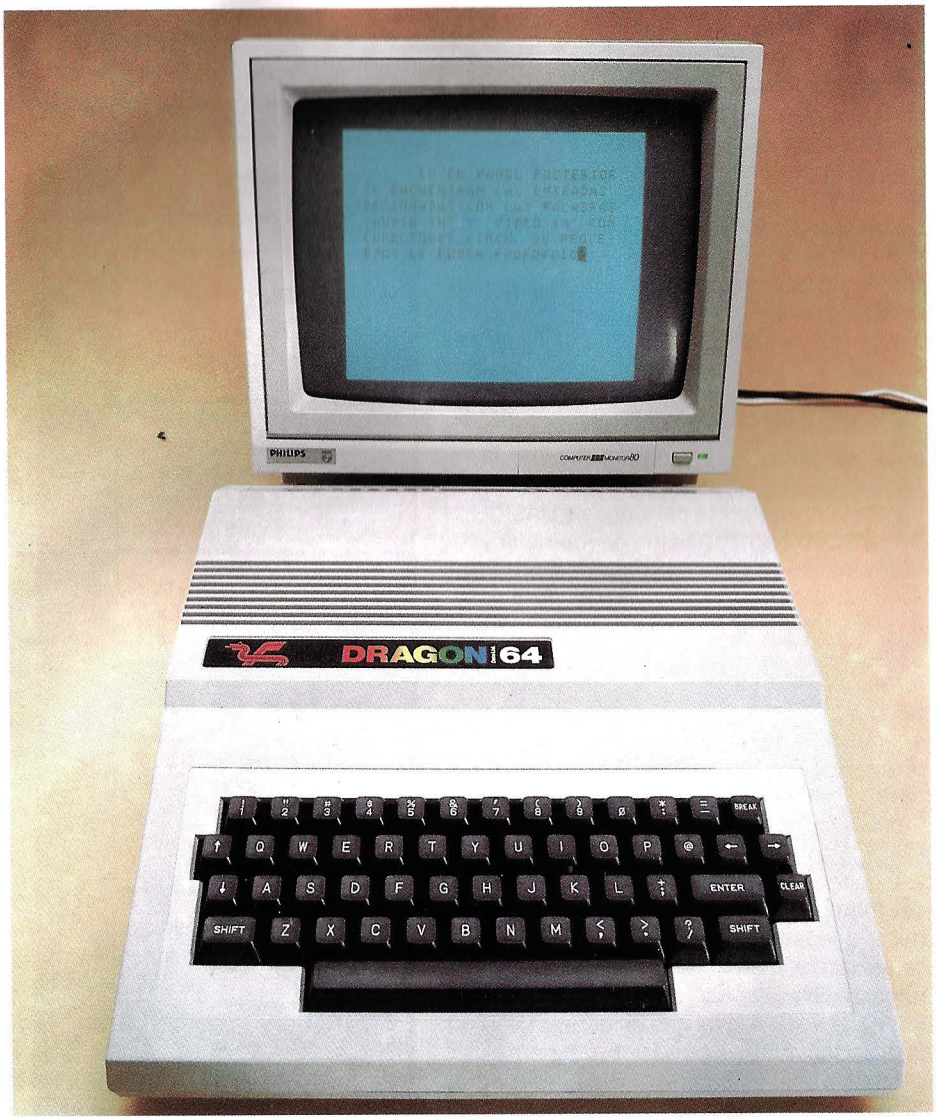
Este procedimiento permite que los programas funcionen de manera casi simultánea, dando la impresión de que cada programa es el único poseedor de los recursos del sistema. Lógicamente hay una penalización de tiempo puesto que a fin de cuentas son varios los programas que se reparten la atención de la CPU.



Estructura genérica de la memoria principal de un equipo gobernado por el sistema operativo OS-9.



El sistema operativo OS-9 sólo asigna recursos a los programas en disposición de utilizarlos.



Los microordenadores de la gama DRAGON cuentan con una versión del sistema operativo OS-9.

◄

El núcleo del sistema operativo OS-9

El núcleo del sistema operativo OS-9 actúa como gestor y supervisor de los recursos informáticos globales puestos a disposición del usuario. Reside en memoria ROM ocupando, junto con el módulo de inicialización de disco, cerca de 4 Kbytes.
Las funciones más destacables del núcleo son:
— Inicialización del sistema tras el arranque, por medio del módulo INIC.
— Gestión de multiproceso e interrupciones.
— Gestión de la memoria.
— Control de peticiones de servicio.
La inicialización del sistema se realiza cargando en memoria volátil los datos contenidos en ROM. Una vez evaluada la cantidad de memoria disponible, se cargará en la memoria principal el módulo necesario.
Tras estos pasos, se inicializan los tamaños de las tablas y los nombres de los dispositivos del sistema para, de inmediato, dar comienzo las operaciones de explotación.

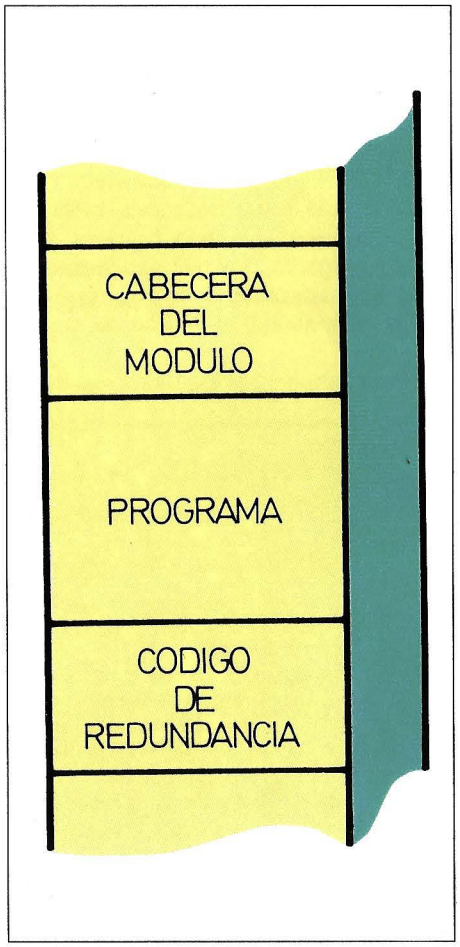
PROCESOS EN MULTIPROGRAMACION

A efectos de operación, el tiempo de CPU se asigna a los programas en funciónque se encuentren en un estado activo; esto es: en disposición de ejecutar funciones
Los diferentes estados de proceso son:
● Activo: este estado es el correspondiente a aquellos procesos que están llevando a cabo operaciones efectivas, siendo los únicos a los que se les asignan recursos de CPU.
● De espera: estado que corresponde a aquellos procesos cuya ejecución se encuentra suspendida, pendiente de que finalice otro programa.
● Durmiente: corresponde a aquellos procesos que han sido suspendidos a requerimiento propio, en espera de alguna señal predeterminada que los haga volver al estado activo.
Para poner en funcionamiento un nuevo proceso y asignar inicialmente los recursos que ha de consumir, es necesario efectuar una serie de operaciones. En el caso del sistema operativo OS-9, éstas se realizarán de manera automática apelando a la función FORK. Esta controla el proceso, y si no puede darle curso, transfiere un código de error al proceso a partir del cual se originó, de manera que éste pueda decidir las acciones posteriores a llevar a cabo. Cuando un determinado proceso tiene que generar nuevos procesos encadenados al mismo, ha de comunicar al sistema operativo las siguientes informaciones:
— Módulo primario: es también el nombre con el cual se conocerá al programa que ejecutará el nuevo proceso. Este puede cargarse en memoria desde un medio de almacenamiento externo, o bien puede encontrarse ya residiendo en memoria.
— Parámetros: son los datos que nece-

sita el nuevo proceso para su correcta operación.
— Número de usuario: en él se fundamenta la técnica empleada por el sistema de seguridad para distinguir y diferenciar los procesos pertenecientes a un usuario específico.
— Prioridad del proceso: permitirá determinar la proporción de tiempo de CPU que se otorgará al proceso lanzado. Normalmente, la prioridad será la misma que la del proceso desde el que se lanzó.

GESTION DE MEMORIA

Cuando se arranca el sistema operativo, la primera operación que se lleva a efecto es la asignación por parte del sistema de los recursos de memoria. En la parte más alta de la misma se cargan los distintos módulos del sistema operativo, la zona inferior de la memoria se destina a datos y la zona intermedia queda vacía para su posterior uso en la gestión de nuevos programas y procesos.



Disposición de los módulos en la memoria principal.

La memoria central del equipo es gestionada por el OS-9 de manera que se opere con los programas de forma reentrante, es decir, un mismo módulo de programa puede ser empleado por varios usuarios, si bien, y dado que cada uno de ellos puede utilizarlo de distinta forma, se destinan áreas de memoria independientes para las respectivas variables durante el tiempo de ejecución. Un ejemplo de ello lo aporta el intérprete del lenguaje BASIC-09; éste precisa 22 Kbytes de memoria y permite que varios usuarios accedan a él sin necesidad de nuevas copias, lo cual agotaría rápidamente los recursos de memoria del sistema. La gestión de los diferentes procesos que utiliza cada módulo de programa corre a cargo del sistema operativo, el cual eliminará la zona de memoria asignada una vez que los procesos hayan terminado. La asignación de memoria a los módulos de programa se realiza mediante la cabecera de dichos módulos, en la cual se indica cuál es la cantidad de memoria, tanto estática como dinámica, que ha de asignarse al proceso. La carga de programas en memoria puede hacerse por medio de la instrucción LOAD, la cual, al igual que en el caso de la instrucción FORK, carga los programas únicamente si éstos no se encuentran ya en memoria. En este último caso, cargará los datos de programa ya cargados. Si el programa se encuentra ya en memoria se seguirá un proceso análogo; si bien, se incrementará un contador, denominado de encadenamiento, que en cada momento controla los procesos que están haciendo uso del módulo en cuestión.
Un factor problemático lo constituye el hecho de que los programas reubicables pueden, en efecto, ser cargados en cualquier posición de memoria, pero no pueden ser reubicados dinámicamente después. En definitiva, cuando un programa se carga lo hace en el primer espacio libre adecuado a su tamaño. En el caso de que haya varios programas funcionando puede ocurrir que alguno de ellos finalice sus operaciones y libere memoria; no obstante, los diferentes espacios así liberados, aun siendo grande su capacidad conjunta, pueden estar desperdigados de manera que sólo liberen pequeños huecos insuficientes para cargar otros programas incluso de moderada longitud. Este fenómeno se conoce como fragmentación de memoria y, normalmente, sólo se detecta a través del comando MFREE el cual señala la posición de cada área de memoria sin utilizar, así como el tamaño de la misma. Para obviar tal inconveniente, es preciso volver a cargar los módulos dispersos para la memoria, asegurándose previamente de que no están en uso o no van a ser utilizados con inmediatez.

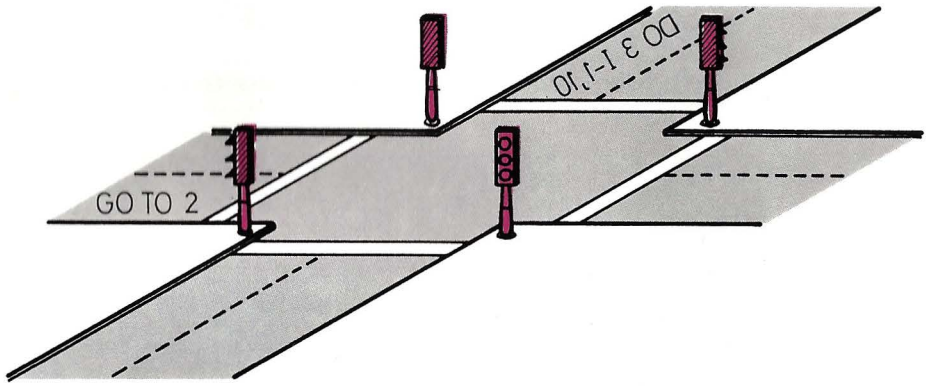
Mapa de memoria típico de un microordenador equipado con el sistema operativo OS-9.

Open Access (2)

Los entornos horizontales del Open Access

Avanzando en el estudio del paquete integrado OPEN ACCESS, se detallan en este capítulo las posibilidades que brindan al usuario las seis aplicaciones horizontales integradas en el mismo.

Suponga que el programa se encuentra almacenado en un disco rígido y que el operador teclea las iniciales del nombre del programa, OA, y a continuación pulsa RETURN. De inmediato, el programa solicitará la introducción del primero de los seis disquetes en los que se suministra el programa completo al ser adquirido; de esta forma, el propietario se asegura la no proliferación de copias "piratas". Después de verificar la originalidad del disquete, OPEN ACCESS presentará en pantalla un menú en el que se ofrecen los seis entornos disponibles.



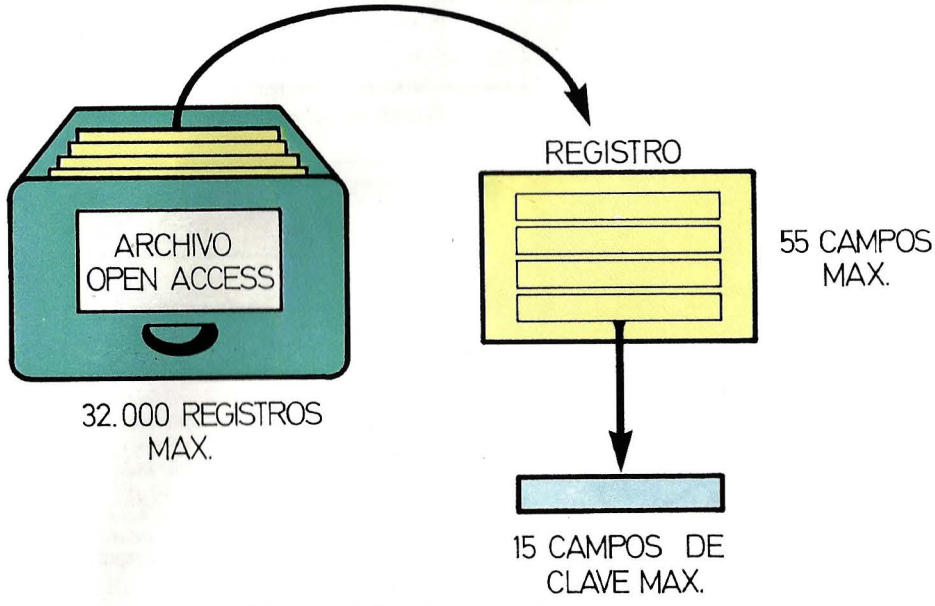
Inmediatamente después de entrar en OPEN ACCESS, el sistema presenta un menú con la solicitud de la fecha en curso. En este punto, al igual que en cualquier otro menú del OPEN ACCESS, existe la posibilidad de abrir ventanas de ayuda.

GESTOR DE BASE DE DATOS

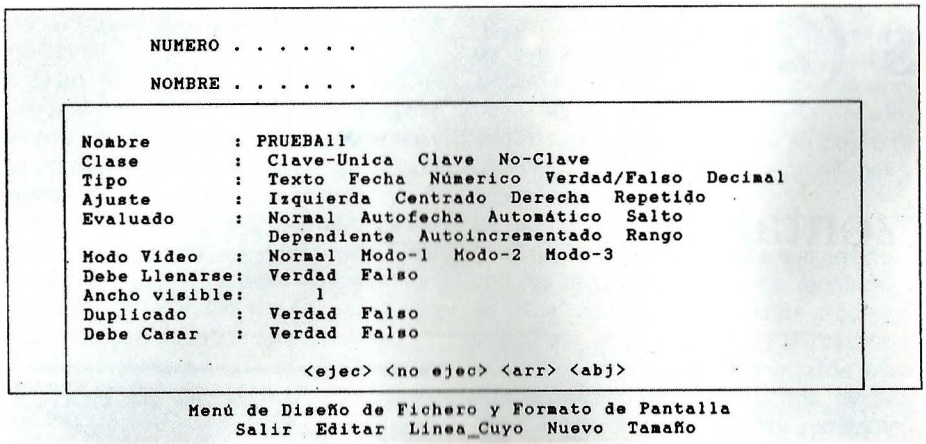
El objetivo de este módulo consiste en aportar un sistema de archivo mecanizado. Para ello se basa en un modelo relacional de forma que cada archivo de información estará constituido por hasta un máximo de 32.000 registros, cada uno de los cuales debe contener un número fijo de campos, sin exceder nunca de 55. Algunos de estos campos (15 como máximo) reciben el nombre de "clave" y sirven para identificar unívocamente a un registro; en consecuencia, el conjunto de campos que forman la clave debe ser distinto en cada uno de los registros de un fichero.

La información almacenada en cada campo puede ser de cinco tipos diferentes, atendiendo a la naturaleza de su contenido:

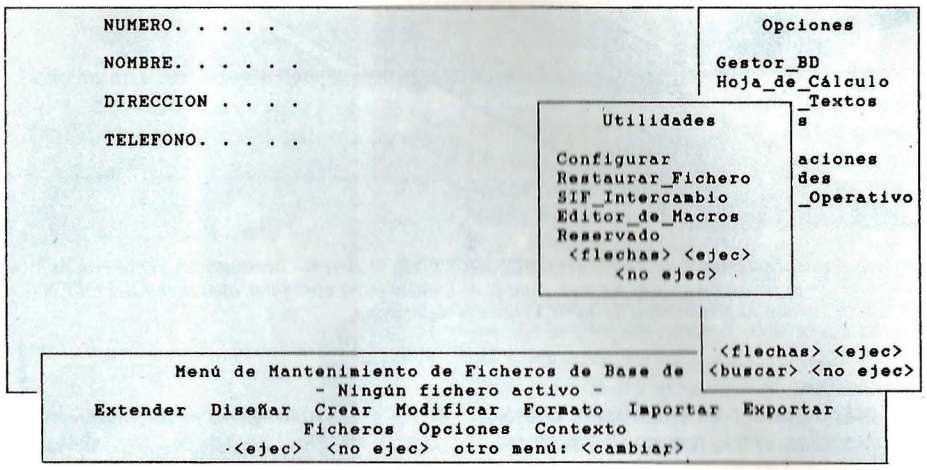
- Alfanumérico (texto): cadenas de letras, números y caracteres especiales.
- Decimal: números positivos o negati-



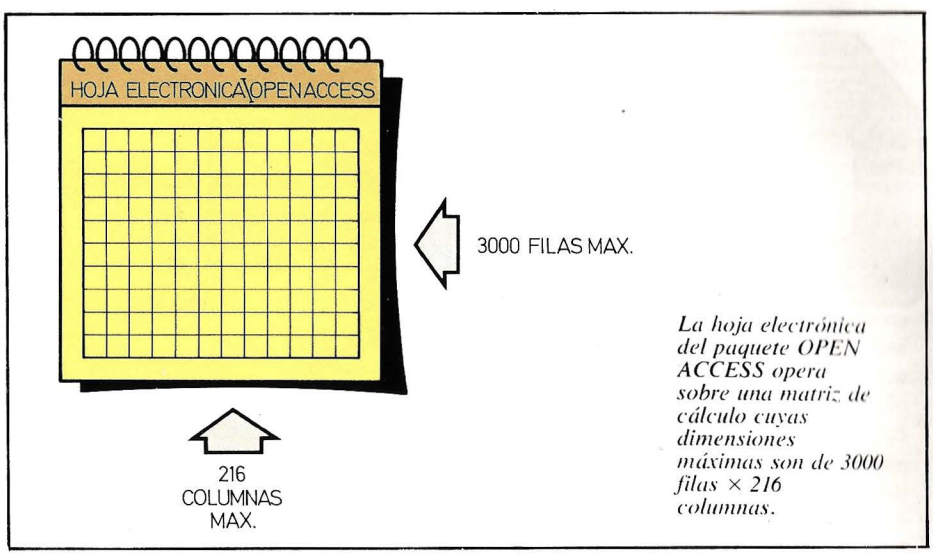
Características básicas del gestor de base de datos incluido en el paquete OPEN ACCESS.



En el diseño de un fichero hay que especificar los atributos que caracterizan a cada uno de los campos de un registro. En la figura se reproduce la ventana que presenta el programa para introducir dichos atributos.



Aspecto de la pantalla del ordenador después de los siguientes pasos. 1: visualizar desde el menú inicial la estructura de la base de datos, 2: seleccionar el comando "opciones", con lo que aparece en la pantalla (parte superior izquierda) el menú principal, 3: seleccionar dentro de este menú el comando "utilidades", cuyo efecto se traduce en la visualización de su respectivo menú.



La hoja electrónica del paquete OPEN ACCESS opera sobre una matriz de cálculo cuyas dimensiones máximas son de 3000 filas × 216 columnas.

vos, con o sin coma decimal (o punto decimal en notación inglesa).
- Fechas: expresiones representativas de una fecha de calendario.
- Lógico: pueden tomar tan sólo valores verdadero o falso.
- Numérico: números positivos o negativos, pero siempre sin decimales.

Cada fichero constituido con las características antes descritas debe tener asociado un nombre propio que lo identifique, y puede ser invocado en cualquier momento para realizar una operación de explotación sobre el fichero o una operación de mantenimiento.

Principales operaciones de explotación

Un primer grupo de operaciones de este tipo permite introducir nuevos registros, actualizar registros ya existentes y clasificar estos según un determinado orden. En resumidas cuentas, cabe afirmar que con estos tres comandos el usuario puede depositar información en el fichero.
Por supuesto, también existe otro grupo de comandos destinados a extraer información, ya sea mostrándola de forma interactiva por pantalla, o bien produciendo informes impresos en forma de listado, cartas, etiquetas, etc.
En cualquier caso, el lenguaje de "interrogación" a la base de datos es una derivación del lenguaje SQL de IBM. Este consiste en cuatro palabras básicas, llamadas cláusulas, y una serie de atributos que las complementan. Una descripción de dichas cláusulas es la que sigue:

— DE: permite especificar el fichero o ficheros *de* donde se obtendrá la información.
— ELIGE: especifica los campos que el usuario *elige* para operar.
— CUYO: establece las condiciones *cuyo* contenido servirá para recuperar los registros que las satisfagan.
— ORDEN: especifica los campos que servirán para dar un *orden* de clasificación al resultado.

Principales operaciones de mantenimiento

En este grupo de operaciones están incluidos los comandos necesarios para diseñar, crear y modificar la estructura de los ficheros que integrarán la base de datos. También existen otros comandos, denominados IMPORTAR y EXPORTAR, que facilitan la carga de datos en el fichero

y la migración de los mismos a otros ficheros, respectivamente.
La ejecución de operaciones de explotación y mantenimiento se realizará apoyándose en dos menús de ayuda, uno indica las opciones disponibles en cada comando y el otro las funciones asociadas a las teclas de función.

HOJA ELECTRONICA

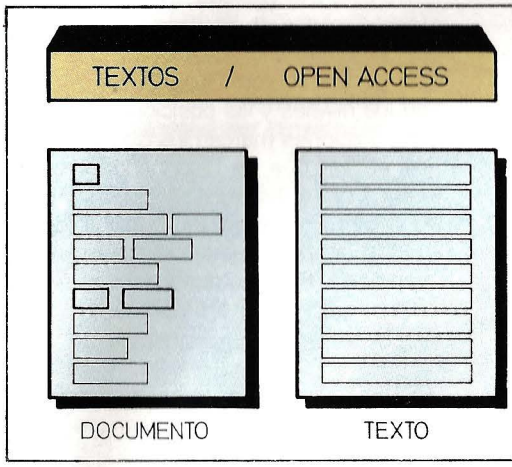
OPEN ACCESS incluye un módulo de hoja electrónica que permite trabajar con matrices de hasta 3000 filas y 216 columnas. La mecánica de funcionamiento de este módulo es muy similar a la de cualquier programa de este tipo; esto es: permite introducir literales, números y fórmulas en las celdas, de forma que se modelice la resolución de un problema de cálculo.
No vamos a detallar todas las funciones ofrecidas por el programa ni todos los comandos que facilitan su gestión. En todo caso, sí queremos destacar una posibilidad peculiar de esta hoja electrónica: la persecución de objetivos. Esta opción avanzada permite especificar valores "objetivo" para una o más variables relacionadas con otras variables que permitan alcanzar dicho valor. En definitiva, el programa resuelve automáticamente un modelo de "programación lineal", donde el objetivo se considera como variable dependiente; OPEN ACCESS resuelve el problema encontrando los valores mediante una función específica independiente. De esta forma, por ejemplo, a partir de información sobre los beneficios brutos de una empresa en los últimos años y de los distintos factores que han permitido alcanzar dichos beneficios, facturación, gastos, etc., se puede marcar como objetivo la consecución de un determinado beneficio para el presente año. El programa calculará automáticamente los valores que deben conseguirse en cada uno de los factores para alcanzar el objetivo.
Como último comentario sobre las posibilidades aportadas por la hoja electrónica de OPEN ACCESS, cabe apuntar la posibilidad de trabajar simultáneamente con cuatro modelos distintos en pantalla, lo que permite al usuario, en la mayoría de los casos, conjugar toda la información necesaria para resolver diversos problemas relacionados entre sí.

```
V1 A          |B              |C              |D              |E              |F      |
 1                              Microprocesador SPI32    AÑO 1983
 2
 3   INGRESOS      PORCENTAJE    BENEFICIOS    NUMERO DE       PRECIO
 4    BRUTOS       IMPUESTOS       NETOS       UNIDADES       UNITARIO        MES
 5 ================================================================================
 6  275900.00 R       10%        248310.00 R        310        890.00 R     Ene
 7  359560.00 R       12%        316412.80 R        404        890.00 R     Feb
 8  191350.00 R       12%        168388.00 R        215        890.00 R     Mar
 9  285690.00 R       12%        251407.20 R        321        890.00 R     Abr
10  178000.00 R       12%        156640.00 R        200        890.00 R     May
11  360450.00 R       12%        317196.00 R        405        890.00 R     Jun
12  285690.00 R       12%        251407.20 R        321        890.00 R     Jul
13  307050.00 R       12%        270204.00 R        345        890.00 R     Ago
14  208260.00 R       12%        183268.80 R        234        890.00 R     Sep
15  218050.00 R       12%        191884.00 R        245        890.00 R     Oct
16  405840.00 R       12%        357139.20 R        456        890.00 R     Nov
17   79210.00 R       12%         69704.80 R         89        890.00 R     Dic
18 S O F T W A R E   P R O D U C T S   I N T E R N A T I O N A L   I N C.

Mod.: B:RECALC                  70,9%  Puntero: A1     Actual: A1      +AA  V:1 #0

      entrada:
```

Las características de la hoja electrónica de OPEN ACCESS son muy similares a las de cualquier otro programa de este tipo. En la figura se observa un ejemplo reproducido directamente de la pantalla.

PROCESO DE TEXTOS

La primera decisión que debe tomar el usuario al activar el módulo de proceso de textos del OPEN ACCESS consiste en elegir entre dos opciones que aparecerán en la pantalla automáticamente: leer un fichero antiguo o crear uno nuevo. La elección entre ambas posibilidades se realizará, obviamente, según el objetivo de la sesión de trabajo.
En el caso de seleccionar la creación de un nuevo fichero, el programa solicitará un nombre para identificarlo y presentará dos nuevas opciones, entre las que el usuario debe elegir: documento o texto. Las diferencias entre ambos tipos de ficheros residen en las facilidades ofrecidas por cada uno al usuario. El tipo documento es mucho más versátil, ya que permite utilizar facilidades especiales para el proceso de textos, tales como: ajuste de márgenes, caracteres en negrilla, itálica o subrayado, distintos formatos en cada párrafo del documento, etc. En cambio, el tipo texto no incluye las facilidades anteriores ya que su estructura será estándar; la principal utilidad de este tipo de ficheros estriba en la posibilidad de utilizarlos perfectamente desde otros módulos de OPEN ACCESS, o incluso desde programas ajenos. Por ejemplo, un fichero de tipo texto puede utilizarse para preparar un programa en BASIC que, posteriormente, se ejecutará desde el sistema operativo. En este caso, puede afirmarse que OPEN ACCESS se



El procesador de textos de OPEN ACCESS permite la edición de dos tipos de archivos: texto o documento. En el segundo de ellos la aplicación vuelca todas sus posibilidades de selección de formato, tipo de letra...

```
Nombre       B:PROYECTO.CHT
Tipo           Superpuesto   Ventanas   Tres-D   Sencillo
Total Niveles      <1..30> 5
Total Posiciones   <1..30> 3
Número Nivel Actual        1
Nombre Nivel Actual    80
Tipo Nivel Actual       Barras   Líneas   Pastel
                Títulos en el Gráfico                  Color        Fondo
Arriba     Prevision Ventas Microprocesador SPI16          1            0
Lado       Graficos de negocios                            1            0
Abajo      Open Access por Software Products Intl.         1            0
Máximo deseado     8000000                        Dato Máximo     0
Mínimo deseado                                    Dato Mínimo     0
División de Ejes   <1..10>  4

Paleta de color    <0..2>   1
Color de Pantalla  <0..15>  1


 <ejec> <no ejec> <flechas> <tecl mov> <menú> <pintar> <ret> <ayuda> <calc>
```

Pantalla del ordenador en la que se deben especificar las características del diagrama a producir mediante el módulo gráfico de OPEN ACCESS.

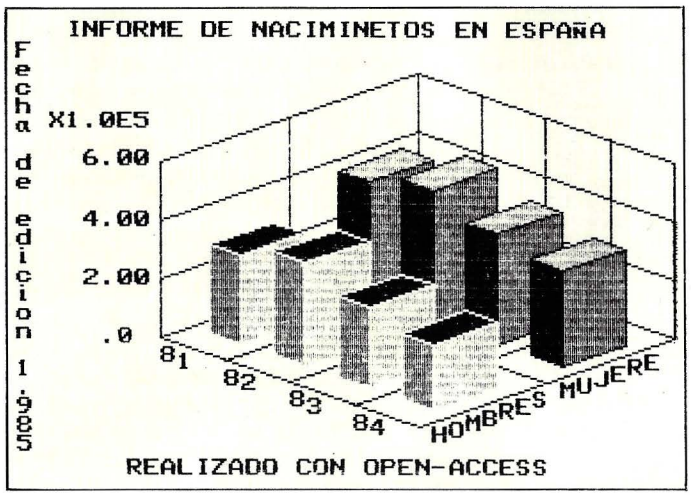
máximo y mínimo valor representable, división de los ejes, etc.
Antes de obtener el diagrama, el usuario debe seleccionar el tipo de representación deseada; para ello dispone de cuatro opciones distintas:

- SUPERPUESTO

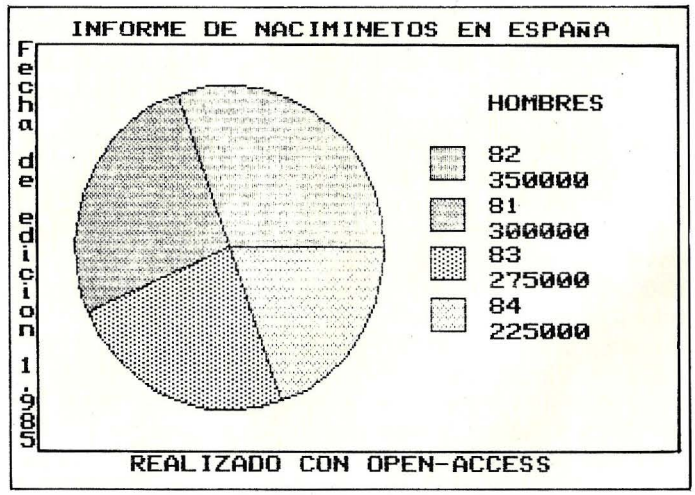
Permite simultanear en una única representación varios diagramas del mismo o de distinto tipo.

- VENTAS

También permite visualizar varios diagramas, pero cada uno de ellos por separado, en una ventana autónoma.



Sin duda, los gráficos en tres dimensiones son los más espectaculares que se pueden obtener con OPEN ACCESS. En la figura se muestra un ejemplo autoexplicativo realizado a partir de datos ficticios.



Dentro de los diagramas bidimensionales, OPEN ACCESS es capaz de producir gráficos de alta calidad, en blanco y negro o en color.

comporta como un potente editor de pantalla completa.
En cuanto a las características técnicas de dicho módulo, éstas no difieren demasiado de otros programas de proceso de textos ya descritos en esta obra.

GRAFICOS

Junto con el módulo de base de datos, el de gráficos es la auténtica estrella de OPEN ACCESS. Destaca su gran facilidad de manejo, que hará que el usuario domine este módulo con mucha rapidez y la notable calidad de los diagramas. Los dos conceptos fundamentales para la elaboración de un diagrama con OPEN ACCESS son los niveles y las posiciones.
Las posiciones marcan los distintos tiempos en los que se dispone de medidas (eje X), mientras que los niveles indican los distintos fenómenos medidos (eje Z); aunque también pueden intercambiar sus "papeles". Por supuesto, otro elemento fundamental para la representación lo constituyen los valores de cada componente en cada nivel (eje Y). Además, el usuario debe indicar otras características auxiliares, como por ejemplo: títulos superior, inferior y lateral del diagrama, má-

- TRES-D

Sin duda la más espectacular de las representaciones es la de tres dimensiones. Estéticamente resulta espectacular, aunque presenta ciertas dificultades para observar pequeñas diferencias en los valores.

- SENCILLO

La única posibilidad se limita a producir un diagrama de tipo plano, de carácter convencional.
En combinación con todas las características ya definidas, los diagramas pueden adoptar tres formatos bien conocidos: barras, líneas y pastel.

Tablas de decisión

Una técnica para dar eficacia a las tareas de programación

En el presente capítulo se pretende introducir al lector en una técnica de programación que le permitirá afrontar con mayor garantía y eficacia la confección de programas. Se trata de las tablas de decisión. Estas tablas empezaron a utilizarse en programación una vez que se comprobó que los tradicionales ordinogramas no eran suficientes en ciertos casos para resolver todas las situaciones que se presentaban.

La primera aplicación en la que se utilizaron en gran escala las tablas de decisión fue en el desarrollo de los lenguajes COBOL y FORTRAN, alrededor de 1958. La razón de su empleo residía en la presencia de una gran cantidad de decisiones frente a las que era necesario elegir un tratamiento.

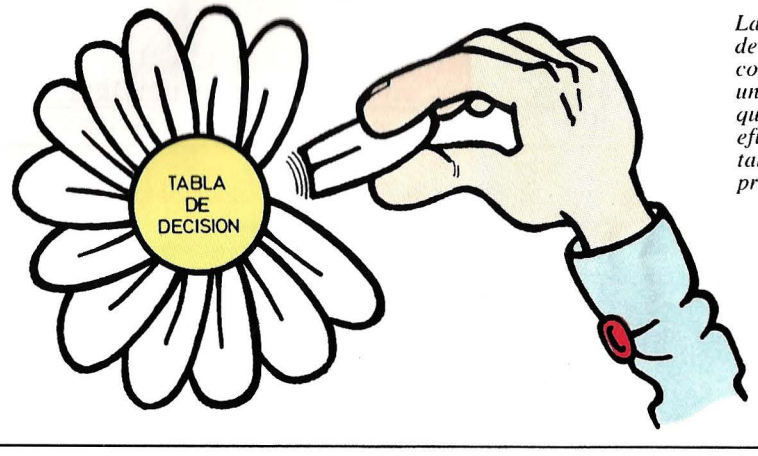
Las principales insuficiencias o limitaciones de los ordinogramas son las siguientes:

— Pueden presentarse varias soluciones para un mismo problema. Y todas ellas pueden ser perfectamente válidas.

— Son difíciles de modificar. Y ésta es una necesidad frecuente, puesto que habrá que modificar el ordinograma cada vez que se altere el tratamiento a otorgar a las entradas que se presenten.

— Pierden eficacia en los lenguajes evolucionados.

— No son sistemáticos.



Las tablas de decisión constituyen una técnica que da eficacia a las tareas de programación.

QUE ES UNA TABLA DE DECISION

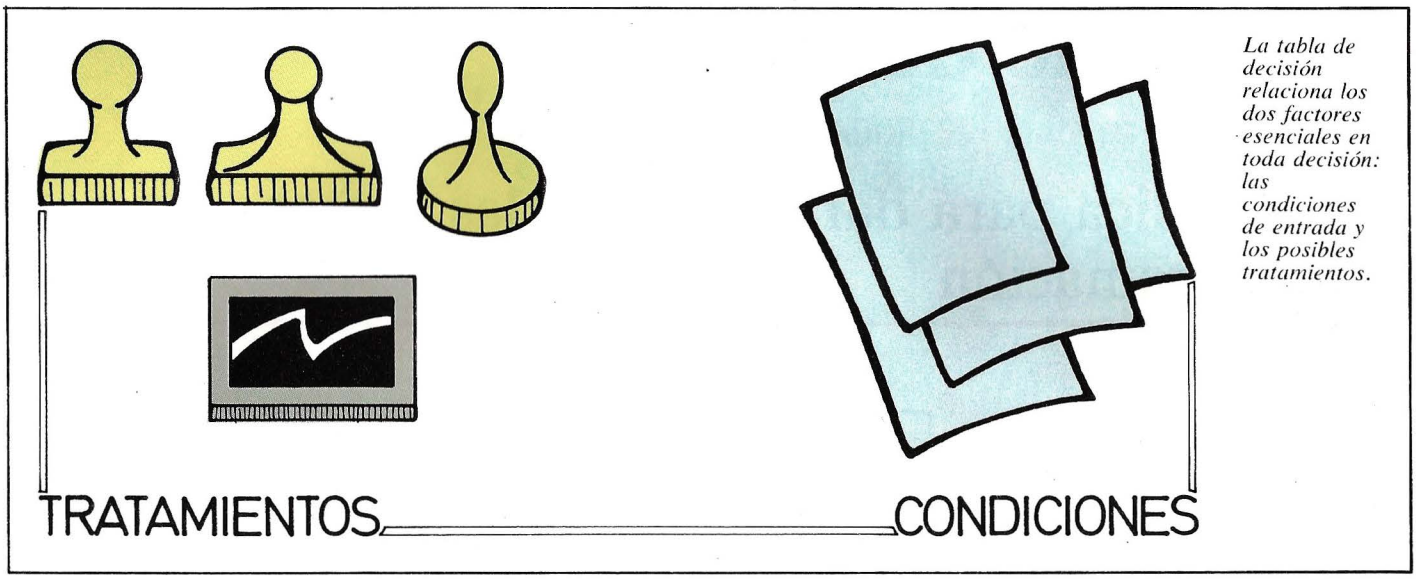
En la figura adjunta puede observarse la estructura típica de una tabla de decisión. Se observa que está dividida en cuatro

TABLA DE DECISION

C1	S	N	·····	S	S
C2	N	S	·····	S	N
(1)	:	:	(3)	:	:
CN	S	S	·····	N	N
T1	X				
T2		X			
T3				X	
(2)			(4)		
TM					X

(1) CONDICIONES
(2) TRATAMIENTOS
(3) ENTRADA DE CONDICIONES
(4) ENTRADA DE TRATAMIENTOS

Estructura de una tabla de decisión.



La tabla de decisión relaciona los dos factores esenciales en toda decisión: las condiciones de entrada y los posibles tratamientos.

zonas, representando cada una de ellas los siguientes conceptos.

1) *Condiciones que se pueden plantear*
Es un inventario o recopilación de todas las condiciones que intervienen en el proceso.

2) *Acciones o tratamientos*
En esta zona se han de incluir todos los posibles tratamientos o procesos que se deban llevar a cabo a partir de las condiciones impuestas.

3) *Entrada de condiciones*
Se incluyen todas las combinaciones posibles. En conjunto representa a todos los posibles casos que pueden surgir, incluyendo aquellos que no se debieran presentar en estricta lógica.

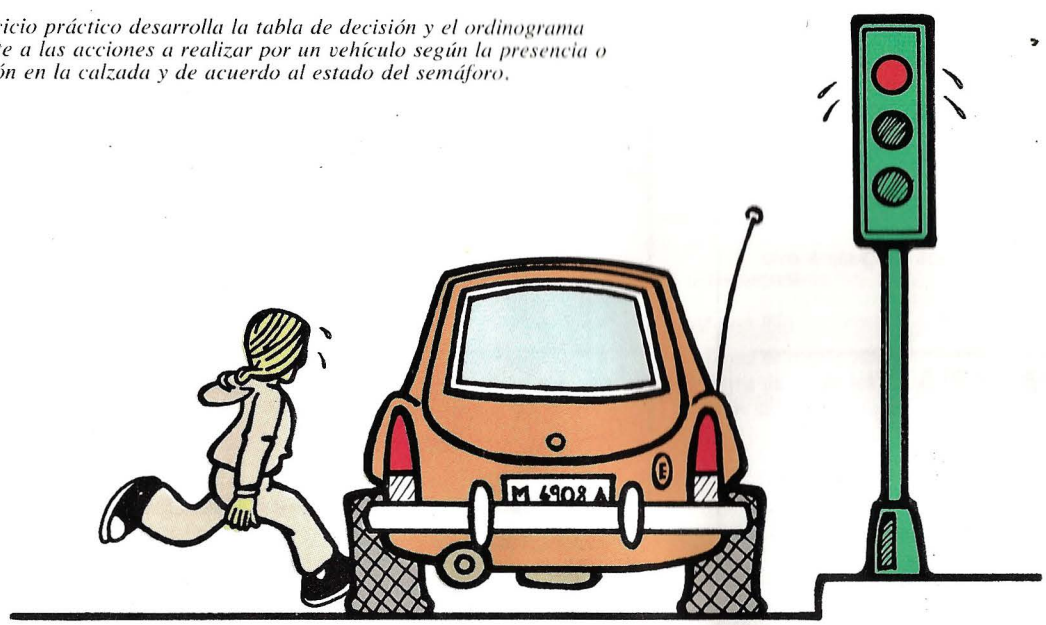
4) *Entrada de tratamientos*
Aquí se deben indicar todos los procesos o tratamientos a realizar. La selección de los tratamientos depende, evidentemente, de las condiciones que se presenten.

Antes de proseguir, es conveniente definir dos conceptos que forman parte de la terminología asociada a las tablas de decisión.

SITUACION: designa a cada una de las posibles combinaciones de valores que se pueden presentar en las entradas de condiciones.

REGLA DE DECISION: es el conjunto formado por la tabla de decisión y el tratamiento inherente a la misma.

El primer ejercicio práctico desarrolla la tabla de decisión y el ordinograma correspondiente a las acciones a realizar por un vehículo según la presencia o no de un peatón en la calzada y de acuerdo al estado del semáforo.



UN EJEMPLO PRACTICO

Suponga que pretendemos informar a un conductor sobre la acción que debe tomar al llegar a un cruce. La decisión se ha de basar en dos consideraciones: si el semáforo está verde y si se encuentra algún peatón en la calzada. Y las acciones a realizar son: detenerse en el caso de que el semáforo no esté en verde, o si se encuentra algún peatón en la calzada. Para empezar la construcción de la tabla, situaremos en ella las condiciones a partir de las cuales se ha de actuar (ver el desarrollo del ejemplo en el cuadro 1).

- C1: equivale a la condición de que el semáforo está en verde; las posibilidades son SI (verde) o NO (rojo).
- C2: indica la presencia o no de un peatón en la calzada; las posibilidades son SI o NO.

Tras ello se han de reflejar en la tabla las diferentes situaciones que pueden presentarse (S1 a S4). Estas quedan patentes por medio de grupos de respuestas a las condiciones (punto 2 del cuadro adjunto).
El siguiente paso (3) consiste en presentar en la tabla los diferentes tratamientos a realizar; en este caso son dos: continuar detenido, o poner el coche en marcha. Concretamente, T1 corresponde a permanecer detenido y T2 a poner el coche en marcha.
El siguiente paso es conectar las diferentes situaciones con los tratamientos a los que dan lugar. Para ello analizaremos cada una de las situaciones y, a partir de las premisas del problema, deduciremos cuál es el tratamiento a otorgar.
La primera situación que se refleja en la tabla (S1) es «semáforo en verde» y «peatón cruzando». En tal caso, el coche ha de permanecer detenido para no arrollar al peatón.
La forma de reflejar este tratamiento en la tabla es poner un aspa en el punto donde se cruzan la columna que corresponde a dicha situación y la fila donde se encuentra el tratamiento a efectuar (ver punto 4 del cuadro adjunto).
A continuación avanzaremos en el estudio de las restantes situaciones reflejando en las tablas los tratamientos oportunos hasta concluir la tabla de decisión correspondiente al ejemplo propuesto.

CUADRO 1

1. Emplazamiento de las condiciones en la tabla.

C1	S	N	S	N
C2	S	S	N	N

2. Definición de todas las posibles situaciones.

	S1	S2	S3	S4
C1	S	N	S	N
C2	S	S	N	N

3. Tratamientos que pueden realizarse.

	S1	S2	S3	S4
C1	S	N	S	N
C2	S	S	N	N
T1				
T2				

4. Tratamiento asociado a la situación S1.

	S1	S2	S3	S4
C1	S	N	S	N
C2	S	S	N	N
T1	X			
T2				

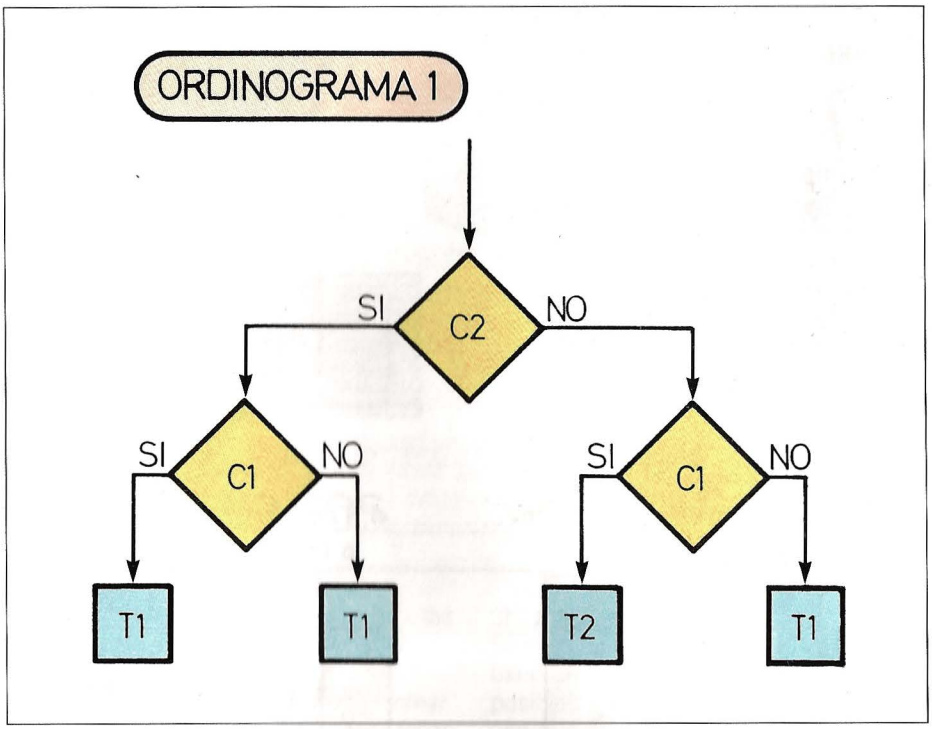
5. Tratamiento de las distintas situaciones.

	S1	S2	S3	S4
C1	S	N	S	N
C2	S	S	N	N
T1	X	X		X
T2			X	

Secuencia de formación de la tabla de decisiones relativa al ejemplo del "vehículo" descrito en el texto.

DE TABLA A ORDINOGRAMA

Una vez confeccionada la tabla de decisión, da comienzo la siguiente fase: construir el ordinograma que permitirá codificar el programa adecuado.
En el caso que nos ocupa, el ordinograma asociado a la tabla de decisión es el que se reproduce bajo el título de ordinograma 1 (ver figura).
Al observarlo, se deduce de inmediato que dicho ordinograma no es óptimo. En efecto, la segunda decisión de la rama de la izquierda puede omitirse perfecta-



mente, ya que en cualquiera de los casos conduce al mismo tratamiento.
Este hecho puede detectarse en la propia tabla de decisión (ver cuadro 2) y eliminar en ella tal redundancia.
Observando la tabla con más detalle, se descubre que las situaciones S1 y S2 conducen a un mismo tratamiento. Es obvio, pues, que para C2 verdadera (S) es posible prescindir del resultado de la condición C1; sea cual fuere su estado, siempre lleva al mismo tratamiento.
Esta situación, recibe el nombre de "indiferencia". En la tabla quedará reflejada según muestra el gráfico 2 del cuadro 2.
El nuevo ordinograma resultante aparecerá bastante más simplificado (ordinograma 2).

CLASIFICACION DE LAS TABLAS DE DECISION

De acuerdo al tipo de condiciones que se planteen, las tablas de decisión pueden ser clasificadas en tres categorías:

- TABLAS LIMITADAS
En ellas las decisiones se pueden plantear simplemente como que "se cumplen" o "no se cumplen": un SI o un NO.

- TABLAS AMPLIADAS
Son aquellas en las que las decisiones no se pueden plantear en los términos estrictos SI o NO, y es necesario recurrir a un acontecimiento de valores.

- TABLAS MIXTAS
Evidentemente, esta categoría engloba a las tablas en las que aparecen mezcladas decisiones de ambos tipos.

UN EJEMPLO EVOLUCIONADO

Para ilustrar con todo detalle el uso de las tablas de decisión, vamos a desarrollar un

CUADRO 2

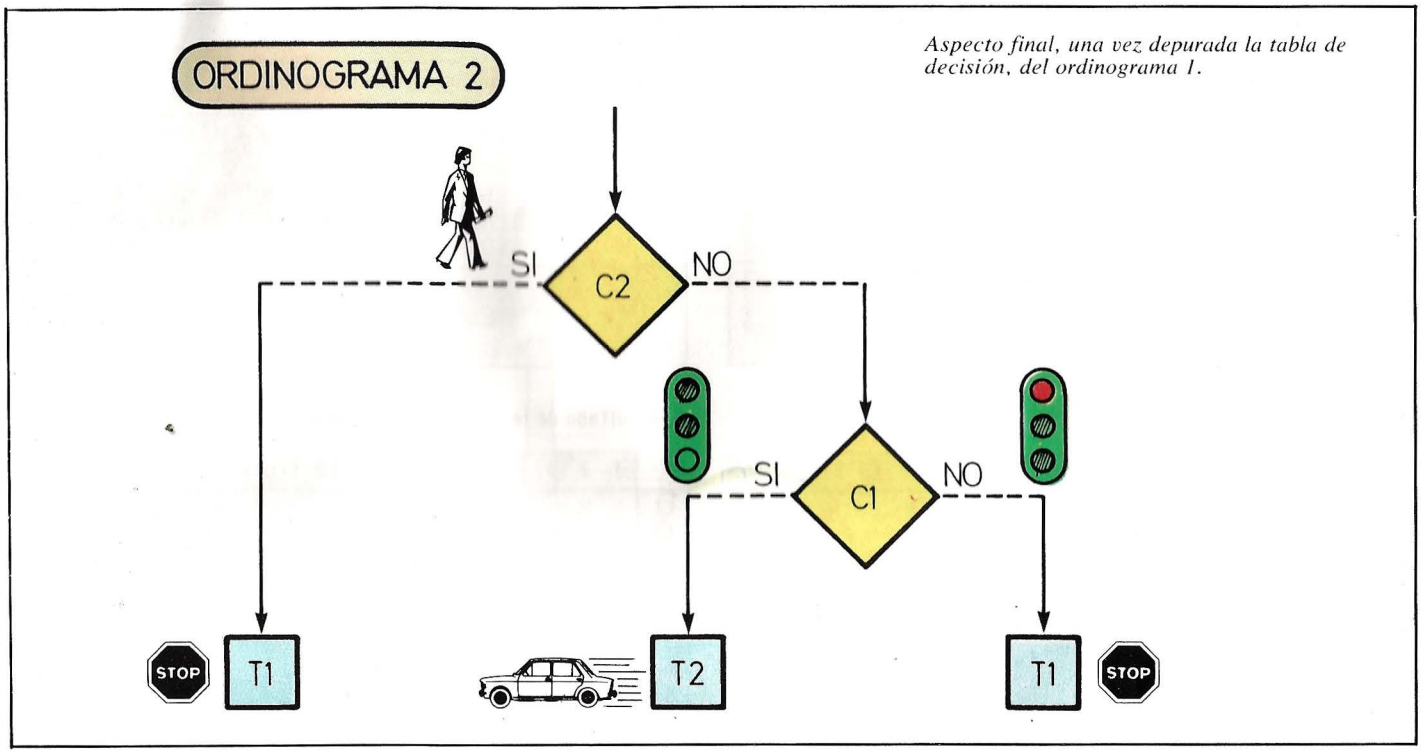
Depuración de la tabla de decisiones correspondiente al ejemplo del "vehículo" descrito en el texto.

1. Tabla de decisión sin eliminar redundancias.

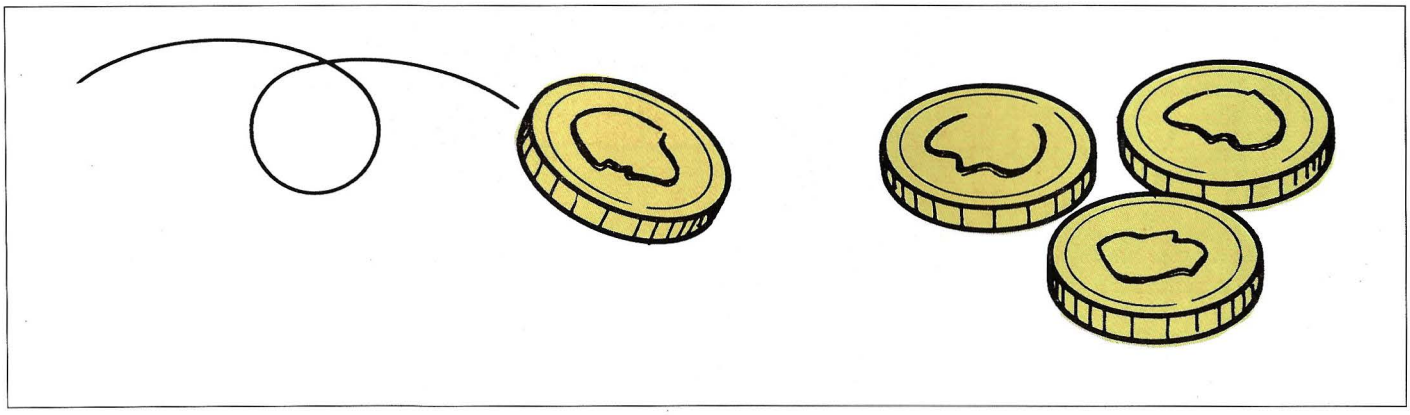
	S1	S2	S3	S4
C1	S	N	S	N
C2	S	S	N	N
T1	X	X		X
T2			X	

2. Tabla resultante una vez simplificada.

	S1 y S2	S3	S4
C1	-	S	N
C2	S	N	N
T1	X		X
T2		X	



Aspecto final, una vez depurada la tabla de decisión, del ordinograma 1.



En el segundo ejemplo, el objetivo es desarrollar un programa que gestione el resultado de un juego.

ejemplo bastante más complejo que el utilizado al principio para introducir esta técnica.

Ahora, el número de posibles situaciones es bastante mayor, hecho que permitirá observar las facilidades que ofrecen las tablas de decisión a la hora de simplificar un problema complejo en el que intervienen un gran número de decisiones.

Se desea crear un programa que indique si se ha obtenido premio, y que revele su cuantía, en el siguiente juego: se lanza una moneda al aire 4 veces y se anotan los resultados; el jugador gana 20 veces la cantidad apostada si en las cuatro tiradas sale cara, recupera la apuesta si sale cara tres veces y pierde en cualquiera de las restantes situaciones. (La construcción de la tabla en sus distintas fases, puede observarse en el cuadro 3).

Por supuesto, se empezará definiendo las condiciones:

- C1: corresponde a la primera tirada; será SI en el caso de que haya salido cara.
- C2: segunda tirada de la moneda.
- C3: tercera tirada de la moneda.
- C4: cuarta tirada de la moneda.

Tal como establecen las reglas del juego, enunciadas en un párrafo anterior, son tres los posibles tratamientos a aplicar:

- T1: corresponde a la victoria del jugador; éste cobrará 20 veces la cantidad apostada.
- T2: reintegro, con lo que el jugador recuperará su inversión.
- T3: derrota del jugador.

En estas condiciones, la estructura de la tabla ofrecerá el aspecto señalado en la figura 1 del cuadro 3.

A continuación se trasladarán a la tabla todas las posibles combinaciones de condiciones para obtener el conjunto de si-

CUADRO 3

1. Condiciones y tratamientos.

C1	
C2	
C3	
C4	
T1	
T2	
T3	

2. Definición de las posibles situaciones.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N
C2	S	S	N	N	S	S	N	N	S	S	N	N	S	S	N	N
C3	S	S	S	S	N	N	N	N	S	S	S	S	N	N	N	N
C4	S	S	S	S	S	S	S	S	N	N	N	N	N	N	N	N
T1																
T2																
T3																

3. Tratamientos asociados a las diversas situaciones.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N
C2	S	S	N	N	S	S	N	N	S	S	N	N	S	S	N	N
C3	S	S	S	S	N	N	N	N	S	S	S	S	N	N	N	N
C4	S	S	S	S	S	S	S	S	N	N	N	N	N	N	N	N
T1	X															
T2		X	X		X				X							
T3				X		X	X	X		X	X	X	X	X	X	X

4. Resultado de la primera simplificación.

	1	2	3	4	5	6	7y8	9	10	11y12	13y14	15y16
C1	S	N	S	N	S	N	-	S	N	-	-	-
C2	S	S	N	N	S	S	N	S	S	N	S	N
C3	S	S	S	S	N	N	N	S	S	S	N	N
C4	S	S	S	S	S	S	S	N	N	N	N	N
T1	X											
T2		X	X		X			X				
T3				X		X	X		X	X	X	X

5. Resultado de la segunda simplificación.

	1	2	3	4	5	6	7y8	9	10	11y12	13y14y15y16
C1	S	N	S	N	S	N	-	S	N	-	-
C2	S	S	N	N	S	S	N	S	S	N	-
C3	S	S	S	S	N	N	N	S	S	S	N
C4	S	S	S	S	S	S	S	N	N	N	N
T1	X										
T2		X	X		X			X			
T3				X		X	X		X	X	X

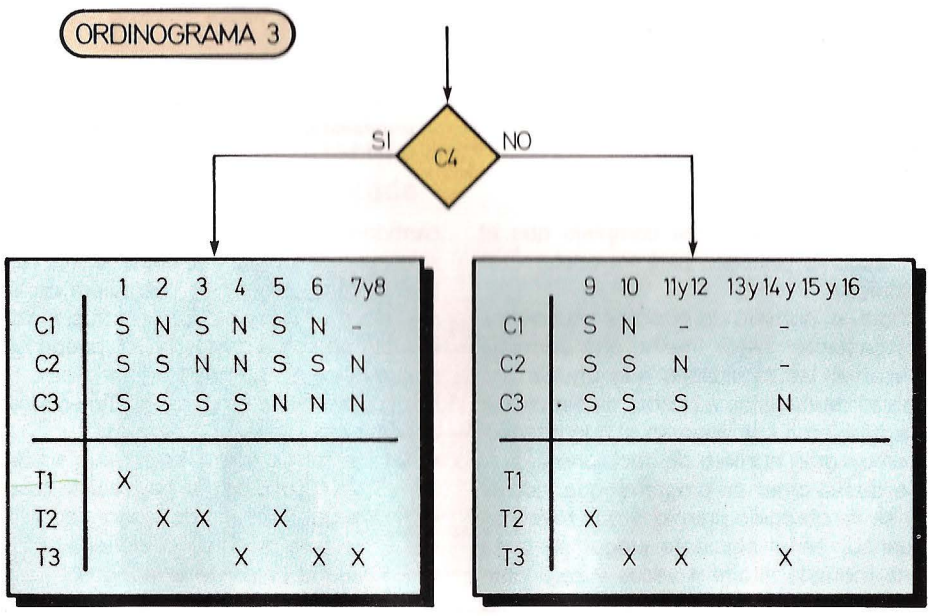
Secuencia de formación de la tabla de decisiones asociada al ejemplo "lanzamiento de monedas".

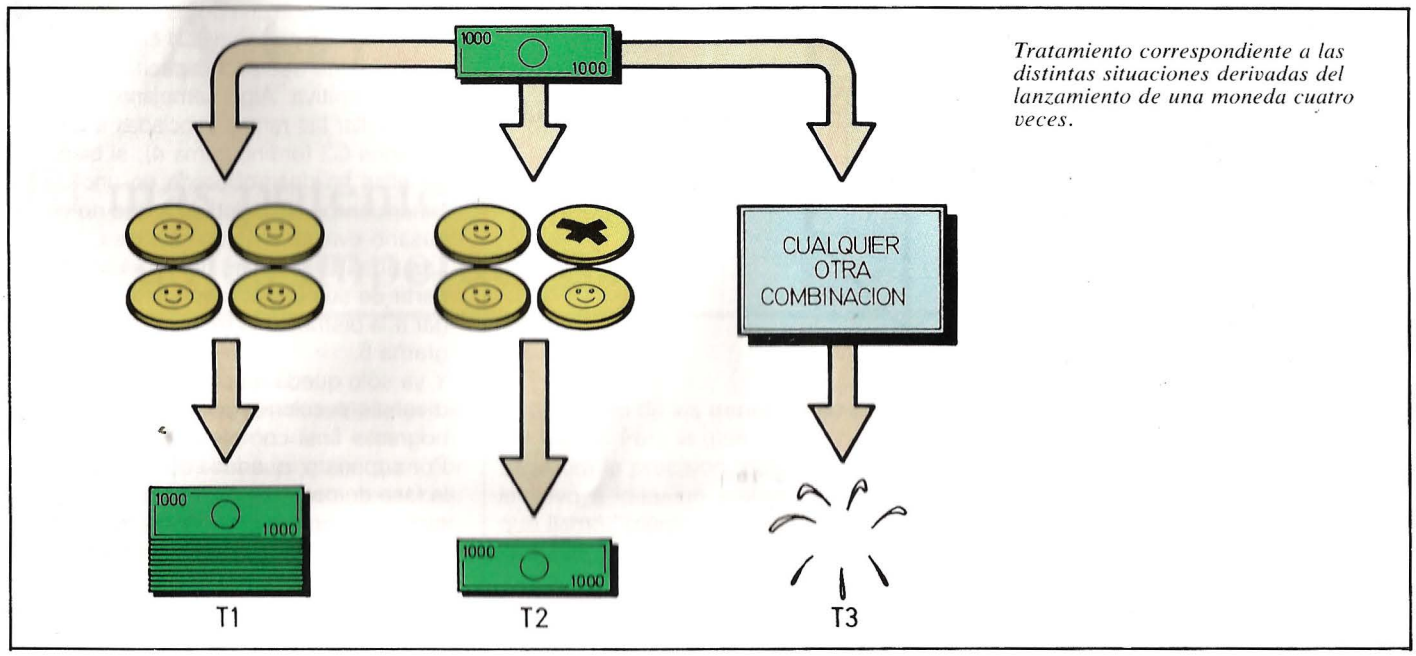
tuaciones que pueden darse en el juego (ver figura 2 del cuadro 3).

Tras ello hay que analizar cada una de las situaciones y decidir los correspondientes tratamientos (ver figura 3 del cuadro 3). Los tratamientos asociados a las diversas situaciones se marcarán con una X en las respectivas intersecciones de fila (situación) y columna (tratamiento).

Así, por ejemplo, la situación 3 (obtención de "cara" en las tiradas primera, tercera y cuarta) da lugar al tratamiento T2 (reintegro de la apuesta). También resulta obvio que la única situación que deriva en el tratamiento T1 (el jugador gana veinte veces el valor apostado) es la primera (cara en las cuatro tiradas).

Acto seguido dará comienzo la tarea de simplificar la tabla. Para ello, hay que observar qué pares de situaciones conducen a un mismo tratamiento; pares de situa-





Tratamiento correspondiente a las distintas situaciones derivadas del lanzamiento de una moneda cuatro veces.

ciones que tan sólo se diferencien en el resultado de una de las condiciones.

Por ejemplo, situaciones como la siete y la ocho implican que no es necesario evaluar la condición C1, ya que el tratamiento de la situación, una vez que se han examinado las otras tres condiciones, es indiferente del valor que adopte dicha condición. Esta primera fase de simplificación da lugar a la tabla 4 incluida en el cuadro 3.

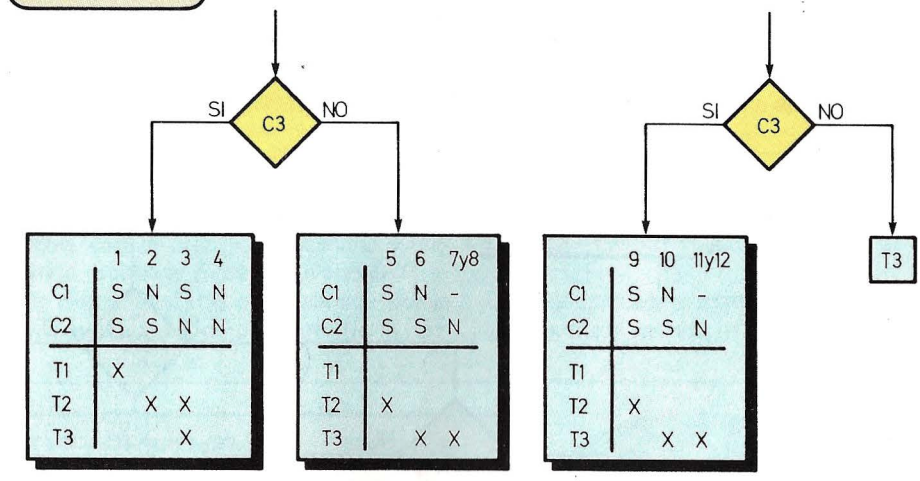
Pero conviene revisar de nuevo las situaciones por parejas, con el fin de comprobar si alguna de las parejas ya establecidas admite una nueva simplificación.

Es interesante hacer notar que las columnas correspondientes a las situaciones 13 y 14 y a la 15 y 16 son simplificables, ya que únicamente divergen en C2. Aunque no hay que acelerar la simplificación sin comprobar todos los extremos. Por ejemplo, la pareja de situaciones 7 y 8 no se puede simplificar con la 6, puesto que no sólo difieren en C2, sino también en la indiferencia de C1.

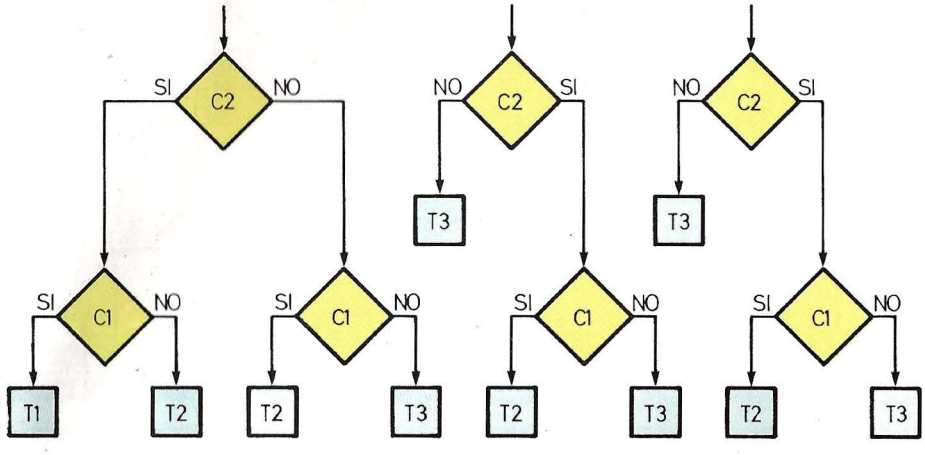
Finalmente, llegamos a la tabla 5 (cuadro 3); la tabla de decisión resultante a partir de la que se obtendrá el ordinograma asociado.

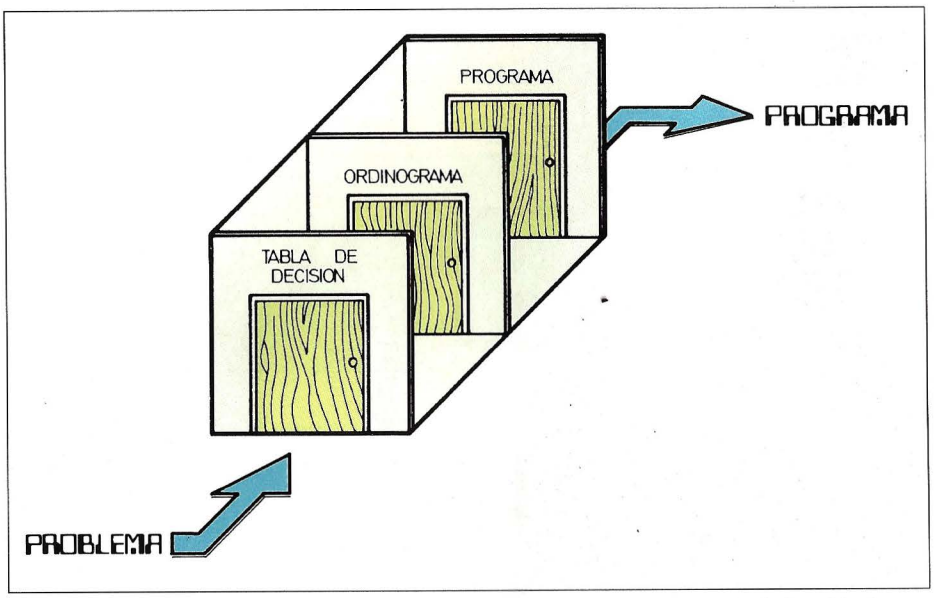
El ordinograma representativo del proceso debe ir resolviendo decisiones para llegar progresivamente a los correspondientes tratamientos. Es muy importante la elección del orden en el que estas decisiones han de ser tomadas. Un criterio a aplicar es elegir para las primeras decisiones condiciones en las que no estén implicadas indiferencias.

ORDINOGRAMA 4



ORDINOGRAMA 5





Proceso a seguir para el desarrollo eficaz y óptimo de un programa: creación de la tabla de decisión, ordinograma y programa.

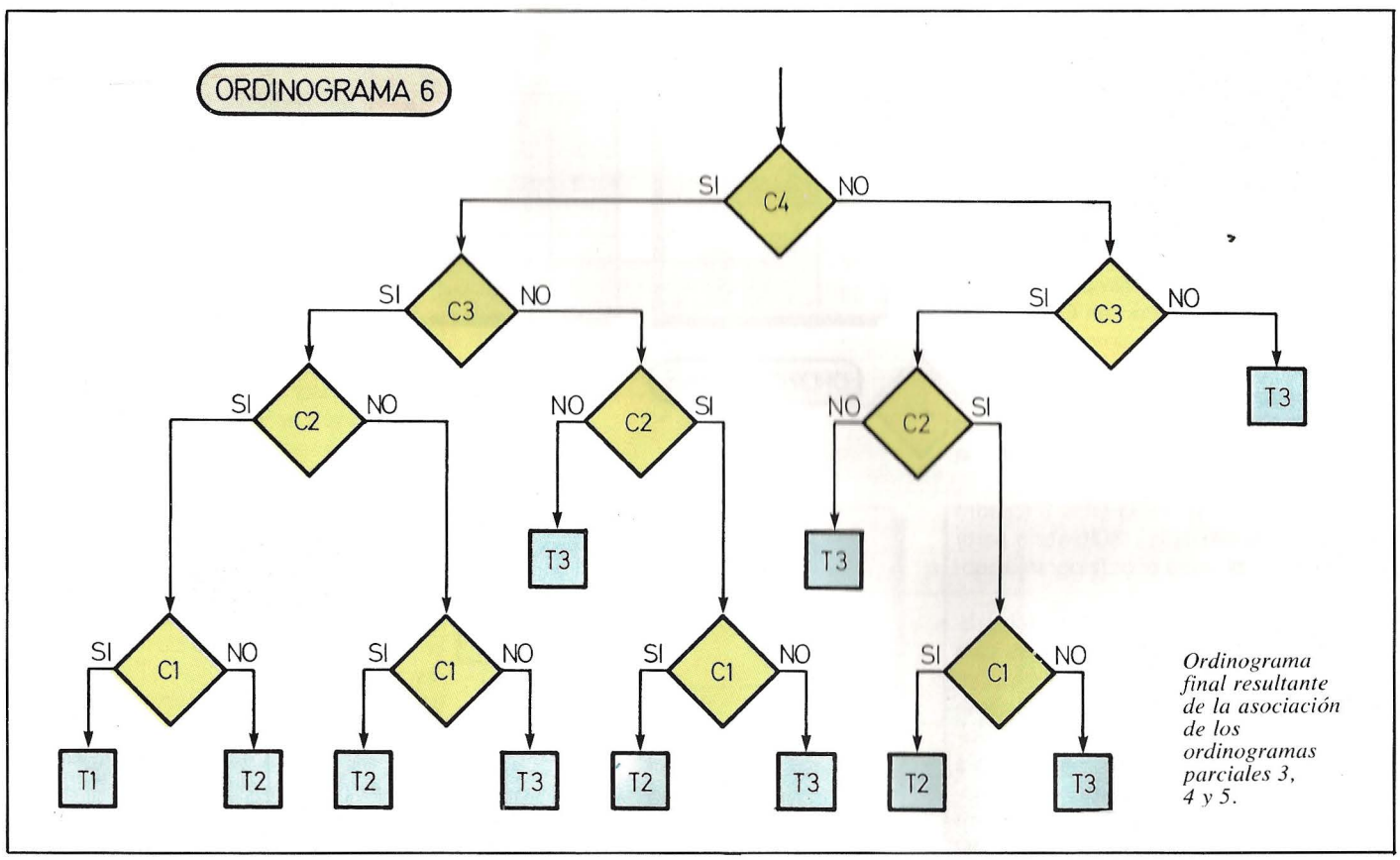
En el caso que nos ocupa, existen dos condiciones a las que no afecta ninguna indiferencia; éstas son: C3 y C4. Por lo tanto, cualquiera de ellas es válida como primera condición a evaluar. Optemos, por ejemplo, por C4 como primera condición a testear (ver ordinograma 3).

Como se observa en la correspondiente ilustración, cada una de las ramas del ordinograma que parten de C4 ha dado lugar a una tabla de decisión específica, derivada de la primitiva. Algo semejante ocurre al desarrollar las ramas asociadas a las condiciones C3 (ordinograma 4); si bien, una de ellas ha desembocado en uno de los tratamientos (T3), debido a que no es necesario evaluar más condiciones.

Las otras tres ramas deben expandirse a partir de sus tablas respectivas, dando lugar a la distribución reflejada en el ordinograma 5.

Y ya sólo queda un paso más: enlazar las diversas secciones para construir el ordinograma final completo (ordinograma 6). Por supuesto, quedaría aún por acometer la fase de codificación, en la que se redactará el programa en un lenguaje informático, atendiendo a la estructura reflejada en el ordinograma.

El ordinograma resultante presenta la sustancial ventaja de permitir la codificación del programa correspondiente de tal forma que éste presente la mínima ocupación posible de memoria. Esta es una característica inherente a los programas desarrollados a través de tablas de decisión perfectamente depuradas.



Ordinograma final resultante de la asociación de los ordinogramas parciales 3, 4 y 5.

ADA (1)

El más potente de los lenguajes imperativos

Los próximos dos capítulos de esta sección van a estar dedicados a un lenguaje que, por su historia y características, se sale de la tónica que venimos observando en los lenguajes imperativos. De igual forma que una buena comida se hace partiendo de unos buenos ingredientes, en el desarrollo de ADA se han invertido los mejores recursos: decenas de cerebros y una buena cantidad de dólares.

UNA HISTORIA CENTENARIA

Comentar la historia del ADA supone remontarse al siglo XIX. Esta fue la época en la que vivió Charles Babbage, un matemático e inventor entre otras cosas del rastrillo delantero de los trenes de las películas del Oeste. Pero la gran obsesión de Babbage era la precisión matemática, lo que le llevó a idear un artilugio mecánico al que llamó "máquina diferencial" para realizar cálculos polinómicos con seis decimales de precisión, lo cual constituyó un rotundo éxito en aquel tiempo.

Posteriormente se propuso la construcción de un nuevo ingenio mecánico al que llamó "máquina analítica", el cual sería capaz de resolver cualquier operación aritmética y lógica que se le indicara a partir de unos datos en forma de tarjetas perforadas (¡Un auténtico ordenador en pleno siglo XIX!). La máquina analítica nunca llegó a funcionar debido a dos razones: primero, los artesanos que trabajaban para él no eran capaces de fabricar las piezas con la precisión necesaria y, segundo, Babbage murió en el año 1871.

A lo largo de esta historia, Babbage recibió gran ayuda y apoyo por parte de la condesa Augusta Ada Lovelace, hija del poeta inglés Lord Byron, y cuyo primer apellido da nombre a nuestro protagonista. Ella fue la que recopiló el trabajo de Babbage a su muerte, introduciendo sus propios comentarios y notas sobre los estudios de su amigo. Gracias a su trabajo, los primeros diseñadores de ordenadores de los años cuarenta se percataron de que eran, en realidad, los segundos en llegar a la meta.

LA NECESIDAD CREA EL LENGUAJE

En la actualidad, el mayor consumidor de software del mundo es el Departamento

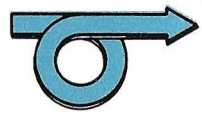
La disparidad de lenguajes utilizados en el Departamento de Defensa Norteamericano planteó la imperiosa necesidad de revisar la situación. Y de ahí nació el ADA.

de Defensa Norteamericano. La gran mayoría de sus proyectos caen dentro del grupo de los "sistemas de computadores integrados" (ver figura). Estos son sistemas mecánicos o electromecánicos en los que hay uno o más ordenadores que tienen el control sobre la mayor parte de los elementos. El éxito de las misiones del Discovery, de las sondas Pioneer o de un sofisticado avión de caza no se comprenden sin tener en cuenta que detrás de ellos hay uno o más ordenadores vigilando el correcto funcionamiento de los diversos elementos. En estos ejemplos, el ordenador no es un fin, sino que está integrado en el conjunto como medio para conseguir los objetivos planteados.

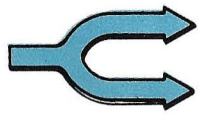
ADA surgió por la necesidad de unificar los más de 400 lenguajes y dialectos que dicho Departamento utilizaba en sus proyectos, de forma que el tiempo y dinero invertidos en el desarrollo de software para uno de ellos fuera utilizable en otro de similares características.

Si analizamos las necesidades que el Departamento de Defensa puede tener, obtendremos una imagen aproximada de lo que ADA puede ofrecer. Un programa que dirija a un misil en su trayectoria puede tener más de cien mil líneas de código fuente, por lo que su desarrollo ha de ser necesariamente una labor de equipo. El ADA proporciona las herramientas necesarias para que cada programador pueda realizar su trabajo independientemente del resto, a la vez que pueda manejar subrutinas creadas por otros sin interferir con ellas. Además, si en los cálculos que se

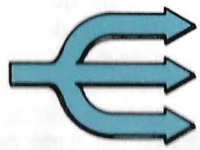
Estructuras de control del ADA



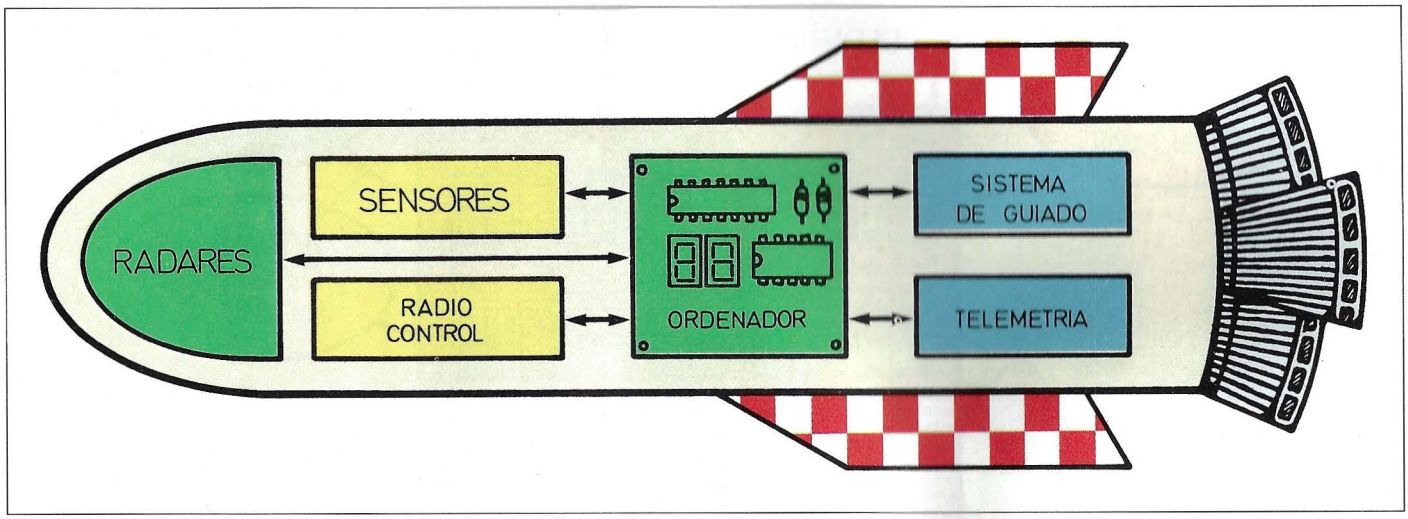
```
for CONTADOR in 1. . 10 loop
   -- hacer esto diez veces
end loop;

while X > EPSILON loop
   -- este bucle se ejecuta mientras
   -- X sea mayor que EPSILON. El va
   -- lor de X debera ser modificado
   -- aqui si se desea salir del bucle
   -- en algun momento.
end loop;

loop
   -- se ejecutan las sentencias que
   -- aparezcan aqui.
   exit when (FINAL=TRUE);
end loop;
```



```
if VARIABLE > 5  then
  PUT("Variable mayor que cinco");
else
  PUT("Variable menor o igual que cinco");
end if;
```



```
case DIA is
   when DOM | SAB => PUT("Hoy es fiesta");
   when LUN . . VIE => PUT("Hoy no es fiesta");
end case;
```



En un sistema de computador integrado la coordinación del funcionamiento de todos los elementos queda confiada al ordenador u ordenadores. Este tipo de sistemas se aplican desde al control de misiles, hasta a la gestión de centrales telefónicas automáticas, pasando por sondas interplanetarias.

realizan para guiar a este misil aparece una división por cero, lo más catastrófico sería que el ordenador se detuviera sin más; pensando en situaciones de este tipo, ADA también proporciona las herramientas necesarias para una gestión cómoda —y sobre todo segura— de las situaciones excepcionales que pudieran surgir en el transcurso de la ejecución de un programa.

ESTRUCTURA DE UN PROGRAMA EN ADA

```
with I_O_PACKAGE;
procedure CONVERSION_DE_TEMPERATURA is
   --parte de declaraciones
begin
   --sentencias
end;
```

Estructura de un programa en lenguaje ADA.

En la figura adjunta se observa la estructura de un programa en ADA. Para los conocedores del Pascal, esta estructura resultará más que conocida. Las palabras clave están en minúsculas. El prefijo:
with I_O_PACKAGE;
es necesario ya que el programa (dedicado a la conversión de grados Fahrenheit a Celsius) utilizará los procedimientos de entrada/salida (Input/Output) PUT y GET, los cuales se encuentran en dicho paquete. Hablaremos más despacio de lo que un paquete significa.
En la figura que muestra el programa completo se observa que los comentarios van precedidos por dos guiones ("--").
Como ya hemos dicho, los procedimientos PUT y GET pertenecen al paquete de entrada/salida, y prodríamos haberlos referenciado así:
I_O_PACKAGE.GET(GR_FAHR); y
I_O_PACKAGE.PUT(GR_CEL);
pero no ha sido necesario por haber incluido la cláusula:
use I_O_PACKAGE;
al principio de la parte de declaraciones.
Los ";" tienen el mismo significado que en C; esto es, como terminadores de sentencias en vez de separadores de las mismas como en Pascal. La declaración:
GR_FAHR,GR_CEL: FLOAT;
declara a estas variables como de tipo real (valores en punto flotante).

LOS TIPOS EN ADA

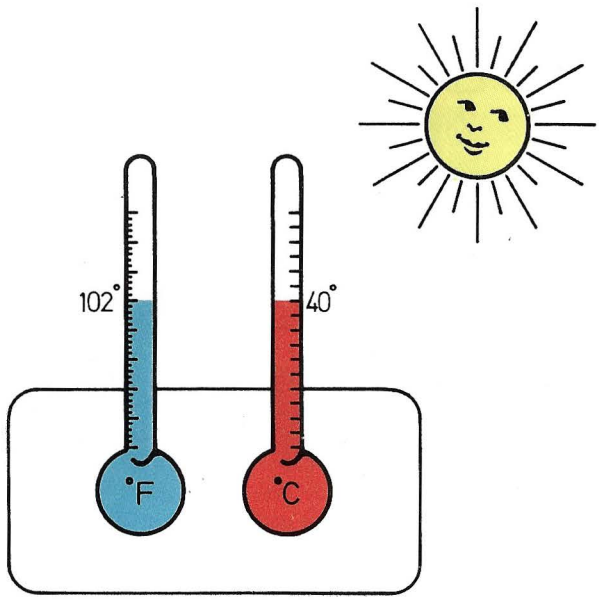
El ADA comparte con el Pascal la característica de ser un lenguaje "tipado"; esto es, al igual que existen unos tipos de datos básicos, como enteros (INTEGER), reales (FLOAT), etc., el usuario puede definir sus propios tipos.
Uno de los objetivos prioritarios del ADA es ser un lenguaje seguro, en el que nada se escape al programador en el momento de la ejecución. Así, si tenemos una variable "DIA" que representa el día de la semana en que estamos, sería estúpido que tomara el valor −5.3E01. Es deseable que el lenguaje rechace la asignación a variables de valores que no tienen sentido, como el anterior. Todo esto redunda en la confección de programas más seguros.
En nuestro caso, definiríamos el tipo "DIAS_DE_LA_SEMANA" así:
type DIAS_DE_LA_SEMANA is (LUN, MAR,MIE,JUE,VIE,SAB,DOM);
e indicaríamos que "DIA" sólo puede tomar estos valores de la siguiente forma:
DIA: DIAS_DE_LA_SEMANA;
Tal definición tendrá lugar en la parte de declaraciones, de igual forma que se hace en Pascal.

ESTRUCTURAS DE CONTROL

Las estructuras de control son las habituales en otros lenguajes estructurados. Estas estructuras son las que resume el cuadro adjunto.

Lenguajes imperativos y declarativos

Como ya se ha comentado al principio, el ADA puede considerarse el exponente más avanzado de los lenguajes imperativos. Este tipo de lenguajes se caracteriza porque, con ellos, se le indican al ordenador de forma inequívoca los pasos a seguir para la resolución de un problema: se trata de expresar un algoritmo en términos comprensibles para el ordenador.
Con los avances en los estudios sobre inteligencia artificial y los nuevos ordenadores de la quinta generación, han hecho aparición otros lenguajes cuya filosofía es distinta: se trata de los lenguajes declarativos. Con ellos se introducen al ordenador unas reglas generales sobre la forma en la que debe resolver el problema y manejar los datos. Para llevar a cabo su tarea, el ordenador seleccionará aquellas reglas que se le "antojen" adecuadas en cada momento. Cabe resumir diciendo que con un lenguaje declarativo se le dice al ordenador *qué* debe hacer, mientras que con un lenguaje imperativo se le dice *cómo* debe hacerlo.



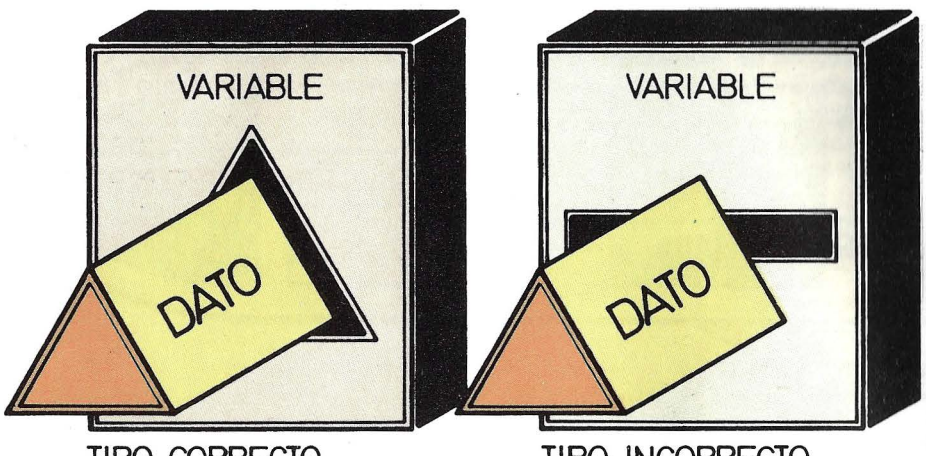

```
with I_O_PACKAGE;
procedure CONVERSION_DE_TEMPERATURA is
    use I_O_PACKAGE;
    -- Este programa lee una temperatura
    -- en grados Fahrenheit y escribe su
    -- equivalente en Celsius
    GR_FAHR,GR_CEL: FLOAT;
begin
    GET(GR_FAHR);
    GR_CEL:=(5.0/9.0)*(GR_FAHR - 32.0);
    PUT(GR_CEL);
end;
```

Aspecto de un programa confeccionado en ADA.

● for-loop
Equivalente al FOR-DO del Pascal. El bucle que aparece en el ejemplo se ejecutará para valores de "CONTADOR" desde 1 hasta 10. Presenta la particularidad en ADA de que no es necesario declarar la variable del bucle en la parte de declaraciones. Consecuentemente, el ejemplo no declara a "CONTADOR" como variable entera.

● while-loop
Plena similitud con su equivalente en Pascal.

● loop-end loop
Esta estructura no se corresponde con ninguna del Pascal. Simplemente permite ejecutar reiteradas veces una serie de sentencias. Si no se desea permanecer dentro del bucle indefinidamente, será preciso introducir un "exit when", el cual hará que el control pase a la primera sentencia que sigue a "end loop" cuando la condición entre paréntesis sea cierta. En nuestro caso (ver cuadro) habríamos declarado a "FINAL" como de tipo BOOLEAN y su valor sería actualizado en alguna sentencia del bucle.



Las variables declaradas de un tipo específico sólo podrán almacenar los valores permitidos para ese tipo. Otros valores serán rechazados.

● if-then
Prácticamente como en Pascal, a excepción del ";" que precede a "else". Hay que recordar el distinto significado de este símbolo comentado anteriormente.

● case-is
En el ejemplo se hace referencia a la variable "DIA" cuyo tipo está definido en el párrafo anterior. Su efecto será escribir "Hoy es fiesta" si "DIA" es DOM ó SAB, y "Hoy no es fiesta" en los restantes casos.
Si hay que sacar alguna conclusión de todo esto, no puede ser otra que el triunfo de las tesis sobre programación estructurada que sentó el lenguaje Pascal diez años antes que el ADA.
En el caso del ADA, hay que seguir a la palabra clave "end" con otra que refleje a qué final nos estamos refiriendo. Los que hayan programado en Pascal pueden haberse encontrado en ocasiones con una ristra de ENDs de los que, en el caso de no haber seguido una norma de indentación, no se sabe a ciencia cierta a qué estructura pertenecen.
Seguiremos viendo más coincidencias con el Pascal, redundando en la anterior afirmación. Se comprobará que las diferencias sintácticas que aparecen en el ADA están orientadas a facilitar la comprensión de los listados.

UCSD p-System (1)

Un sistema operativo concebido en PASCAL

A la hora de diseñar un sistema operativo hay que hacer frente a un primer interrogante esencial: ¿se construye un sistema operativo muy optimizado, y soportado por un tipo muy concreto de microprocesador y, por ende, de microordenador?... o bien, ¿se proyecta un S.O. menos rápido, pero que permita una mayor portabilidad del software a otras máquinas, al no depender en demasía de los condicionamientos hardware del sistema en el que está instalado?

La decisión por una de las dos alternativas depende claramente de los objetivos que se pretenden cubrir al desarrollar un S.O. Y no cabe afirmar, en principio, que ninguno de los dos enfoques sea intrínsecamente bueno o malo. De hecho, hay ejemplos de uno y otro bando en el mercado, tan importantes y renombrados como el CP/M (orientado hacia un microprocesador), Apple-DOS (orientado hacia un microordenador) y UNIX (de tendencia más universalista).

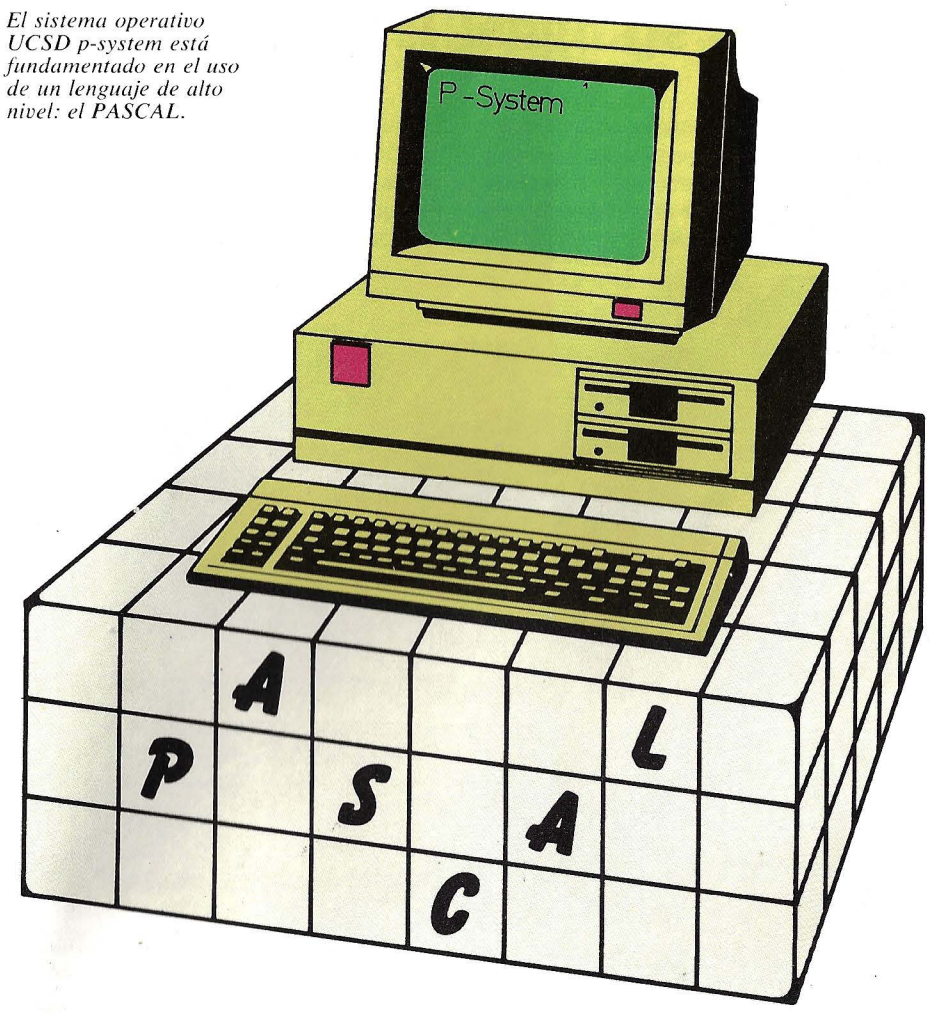
TRANSPORTABILIDAD ANTE TODO

A pesar de la cuestión abierta, bien es cierto que las tendencias actuales se dirigen hacia una mayor compatibilidad y facilidad de intercambio de información entre equipos diferentes, con el fin de aprovechar las ventajas que ofrece la conexión en red de todo tipo de ordenadores.

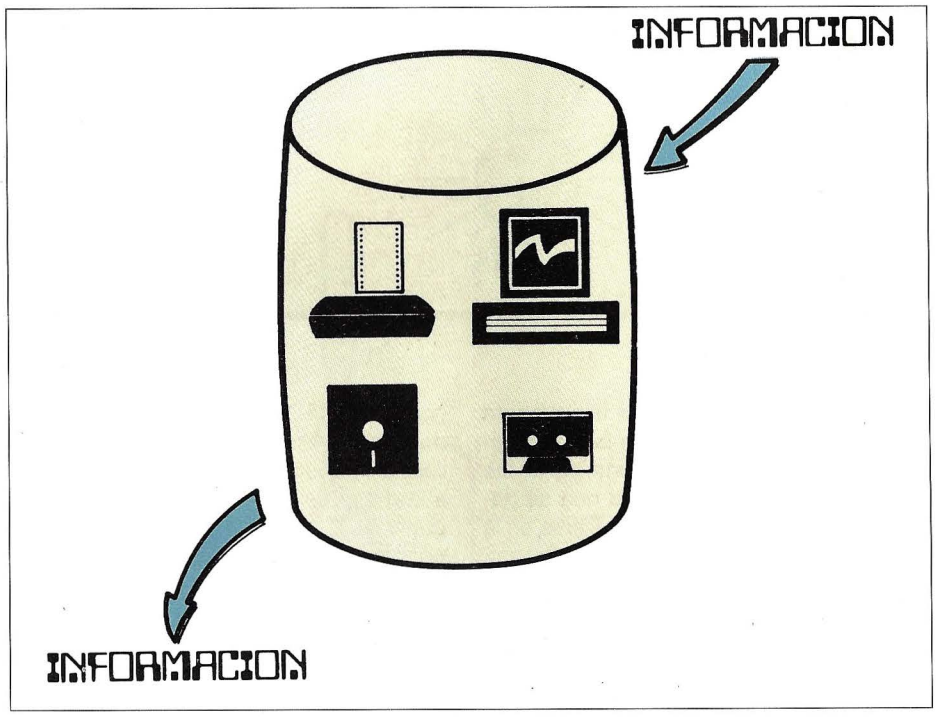
Entre los S.O. que siguen la consigna de la transportabilidad del software, cabe destacar el desarrollado en la Universidad Californiana de San Diego (UCSD) al final de los años 70. Este S.O., conocido como UCSD p-Sytem, fue creado inicialmente con el propósito de implantar la programación en PASCAL dentro del entorno de los microordenadores; tendencia que se detecta fácilmente, sin más que fijarse en la cantidad de programas de utilidad disponibles como ayuda para la confección de aplicaciones en PASCAL.

La transportabilidad del UCSD p-System se fundamenta en la capacidad del lenguaje PASCAL que permite compilar o traducir programas a un código intermedio llamado pseudocódigo o código-p, situado a la mitad de camino entre el código máquina y un lenguaje de alto nivel como, por ejemplo, el FORTRAN, BASIC o el mismo PASCAL.

El código-p obtenido es el mismo para cualquier tipo de microordenador que disponga del lenguaje PASCAL. Así pues, la



El sistema operativo UCSD p-system está fundamentado en el uso de un lenguaje de alto nivel: el PASCAL.



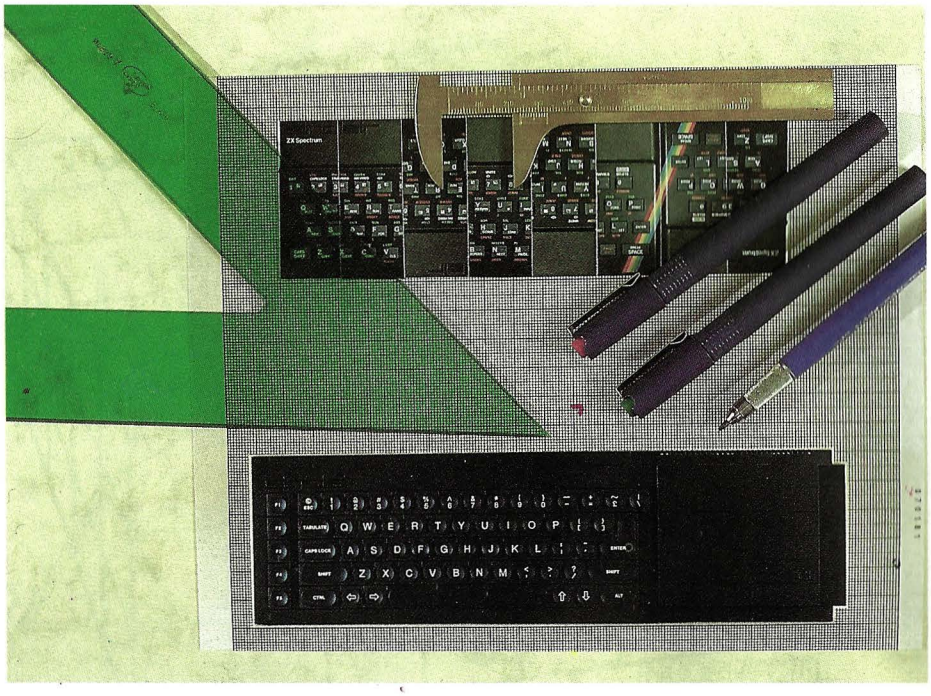
Desde el punto de vista del UCSD p-system, la gestión de la información a través del sistema se realiza con volúmenes, o dispositivos con capacidad para almacenar, transmitir o recibir información.

instalación del p-system en cualquier nuevo microordenador, sólo requiere que una parte del compilador sea sustituida, con el fin de convertir el código-p en código máquina ejecutable.

En cuanto a la zona destinada a la comunicación del sistema con el exterior, cubierta por las rutinas de entrada/salida, hay que comentar que normalmente está localizada en un módulo de código máquina llamado BIOS (Basic Input-Output System), módulo que debe ser retocado para acomodarlo a las exigencias del S.O. p-system.

La introducción de estas modificaciones en cada sistema hacen que con un mínimo cambio sea posible salvar la barrera que ofrecen los diferentes microprocesadores a la generalidad del software. Así, puede lograrse que programas desarrollados en microordenadores basados en el microprocesador 8086 de Intel, sean ejecutables sin ninguna traba en otros sistemas con distinto microprocesador, por ejemplo, el Motorola 68000.

De un modo general, cuando se hace referencia a la transportabilidad de la mayoría de los sistemas operativos, se da por supuesto que se habla de los programas fuente en lenguaje de alto nivel. La versión del programa en lenguaje de alto nivel es la misma, pero ha de convertirse en cada uno de los microordenadores que operan bajo p-system.



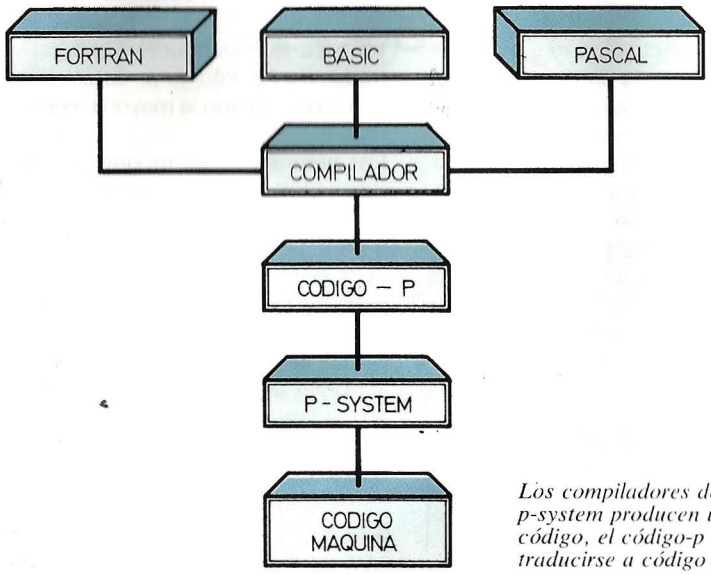
El UCSD p-system está muy bien dotado de programas de utilidad que apoyan y facilitan el desarrollo de aplicaciones de la más diversa naturaleza.

ELECCION CON MENUS

Los diversos comandos y aplicaciones disponibles bajo el UCSD p-system están organizados en menús estructurados jerárquicamente (en forma de árbol), con múltiples niveles de submenús que dan acceso a los detalles de los comandos. Este modo de funcionamiento hace que no sea necesaria una memorización de un número más o menos extenso de comandos, antes de ser capaz de actuar sobre el sistema, ya que los comandos y sus significados son visibles en todo momento.

La apariencia física de los menús difiere de la tradicional que ocupa una pantalla completa. En efecto, se asemejan realmente a mensajes del sistema, al estar indicadas todas las opciones en una sola línea de la pantalla. Ello obliga, en algunas ocasiones, a mostrar en el menú nada más que los comandos más usuales por falta de espacio. En estos casos existen submenús que complementan a los menús principales, accediéndose a ellos al responder a un menú de comandos con el signo ?.

Dado que todos los comandos se ejecutan al elegir opciones de menús, el usuario se ve liberado de especificar formulaciones, a veces de complicada notación.



Los compiladores del UCSD p-system producen un único código, el código-p que puede traducirse a código máquina.

Por ello, resulta tarea fácil recorrer las diversas ramas del árbol para llevar a buen término las oportunas acciones.

MANEJO DE FICHEROS

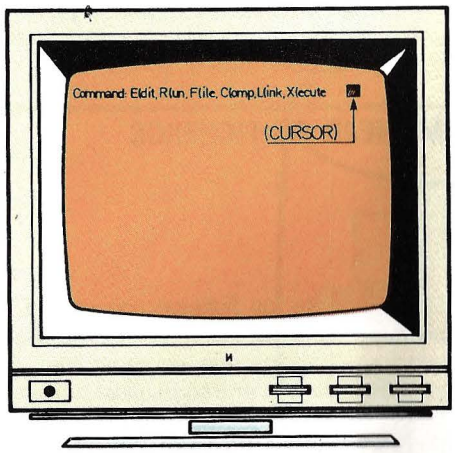
La gestión de los ficheros con el UCSD p-system está basada en unos conceptos sencillos, con los que se construyen la arquitectura de entrada/salida del sistema.
Dichos conceptos son los siguientes:

— *Dispositivo*
Bajo este nombre se agrupan diferentes periféricos conectados a la unidad central, tales como una consola, una impresora, un controlador de disco, etc.

Traducción e interpretación del lenguaje común

El uso de ordenadores para la resolución de problemas está condicionado en gran manera por el grado de comunicación entre el hombre y la máquina. Esta comunicación ha ido evolucionando desde las primeras y complicadas conversaciones en código máquina, hasta la llegada de los lenguajes de cuarta generación o "lenguajes naturales" existentes en la actualidad. No obstante, estos lenguajes siguen siendo poco flexibles, obligando al programador a utilizarlos de una forma muy precisa. La capacidad del ordenador para entender y actuar según instrucciones dadas en un lenguaje coloquial de uso común, tal como el castellano, está empezando a desarrollarse, aunque dista mucho de estar resuelta.
Dos son las principales dificultades inherentes al lenguaje común. La primera surge al ser el lenguaje común mucho más rico y con estructuras más complejas que las contempladas por los lenguajes que el ordenador entiende en la actualidad. En segundo lugar, el lenguaje común está evolucionando continuamente con nuevas palabras, nuevas construcciones de frases y nuevos significados para viejas palabras. Con harta frecuencia, palabras y frases tienen significados que difieren enormemente de su interpretación literal.
La complejidad del problema de la interpretación por parte del ordenador del lenguaje común, no ha sido freno en el empeño por buscar un sistema de interpretación. La clave para el futuro progreso parece ser que se encuentra en una combinación de limitaciones e interacción.
En contextos limitados que incluyen mucha interacción con el usuario, es posible que el ordenador entienda suficientemente el lenguaje común como para ser capaz de responder lógicamente. Existen programas que llevan a cabo conversaciones razonablemente simples con el usuario e interpretan sus preguntas, dando las respuestas apropiadas. Están empezando a aparecer en el mercado programas conversacionales simples para diferentes aplicaciones que requieren respuestas del usuario también simples. Es previsible que en los próximos años se avance sustancialmente en el tema, debido a los recientes avances tanto lingüísticos como informáticos.

La ambigüedad del lenguaje humano hace que situaciones tan sencillas como la descrita en la figura resulten problemáticas para la máquina.



Los comandos del p-system están reflejados en menús de una sola línea, lo que facilita en gran medida su aprendizaje y uso práctico.

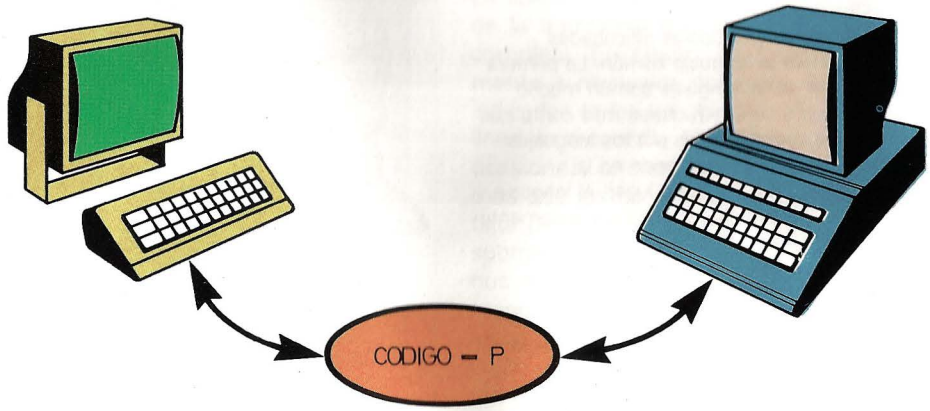
— *Número de unidad*
La identificación de los distintos dispositivos conectados al sistema exige que cada uno de ellos tenga una etiqueta que lo distinga de los demás. Esto se consigue asignando a cada dispositivo un número, el llamado número de unidad.

— *Volúmenes*
El sistema de ficheros de p-system está constituido por conjuntos de volúmenes. El volumen es un soporte capaz de almacenar información (en forma de ficheros) y de emitir y recibir información.

— *Nombres de volumen*
Cada volumen tiene asociado un nombre, al igual que los dispositivos tienen un número de unidad.
Todos los dispositivos, excepto los controladores de disco, son al mismo tiempo volúmenes. La impresora es un volumen que emite información hacia el exterior, la consola es un volumen que recibe información (desde el teclado) o la emite (desde la pantalla). (Bajo el punto de vista del p-system, la entrada de datos a la consola se interpreta como si procediesen de un fichero).
Los controladores de disco no son considerados como volúmenes, ya que la información no es almacenada en ellos, sino en el disco; y tampoco son capaces de emitir ni recibir información, pues su trabajo es actuar sobre los mecanismos de lectura/escritura.
Dentro de los diferentes volúmenes, existe uno que dada su importancia tiene un nombre especial (*). Nos referimos al volumen desde el cual se arranca el sistema. Este proceso de arranque consiste en copiar en la memoria interna el sistema operativo residente en disco y activar la CPU. Por lo demás, el volumen denominado * también se caracteriza por disponer de ficheros especiales utilizados para la configuración del sistema.
La referencia a los nombres de ficheros se lleva a cabo con los métodos ya clásicos de utilización de extensiones, referencias ambiguas (wild cards), etc.

PROS Y CONTRAS DEL UCSD P-SYSTEM

En el lado positivo del p-system cabe destacar su capacidad de concurrencia (realización de varias tareas simultáneamente), la existencia de utilidades que permiten una conexión en red local con otros tipos de ordenadores, así como su fácil aprendizaje y funcionalidad.
La biblioteca de software existente es muy amplia, con lo que está garantizada la disponibilidad de una extensa colección de utilidades y programas de aplicación instalados en un rango de equipos que va desde Apple hasta sistemas multiusuario, como los Digital.
La desventaja principal está en la lentitud con respecto a otros sistemas operativos más especializados y orientados hacia una máquina en particular. El problema de la falta de velocidad es consecuencia directa del modo de actuación del UCSD p-system, ya que aunque los programas son traducidos a código-p, éste es interpretado como lo son la mayoría de los programas en BASIC.
Por naturaleza, los programas interpretados son más lentos que los compilados o que los escritos en código máquina. En efecto, en la interpretación, una instrucción se traduce tantas veces como sea ejecutada dicha instrucción, mientras que en caso de la compilación la traducción se realiza de una vez por todas.
Aún con este serio inconveniente, su velocidad es aceptable para la mayoría de las aplicaciones, debido a que el código-p es más eficiente que el BASIC y, por lo tanto, su velocidad mayor. Si las aplicaciones que se ejecutan bajo p-system son de gestión o tienen frecuentes accesos a disco, esta falta de velocidad queda aún más enmascarada. Ciertamente, el ordenador gasta la mayor parte de su tiempo aguardando entradas por teclado, u ocupándose de operaciones de entrada/salida a disco, que son varios órdenes de magnitud más lentas que las operaciones de cálculo normales.
Aunque la filosofía del UCSD p-system apunta hacia una transportabilidad total de sus programas, esto no siempre es posible; sobre todo si nos encontramos con sistemas informáticos cuyos discos estén formateados siguiendo tendencias diferentes.
Una idea de su transportabilidad y compatibilidad lo da la lista de los diferentes equipos en los que es posible su instalación; entre ellos destacan miembros de las familias Apple, COMMODORE, DIGITAL, IBM, TEXAS, NCR...



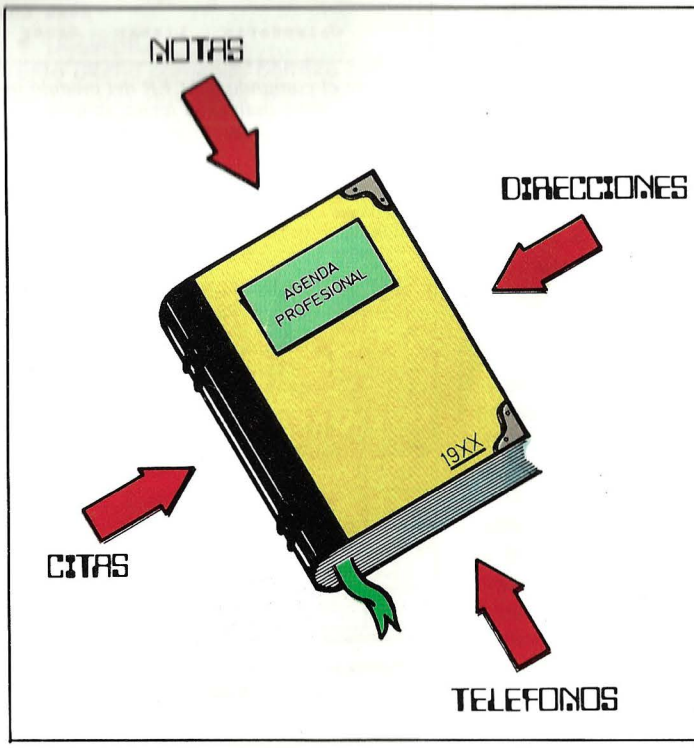
La posibilidad de disponer del mismo código-p en todos los equipos gestionados por el UCSD p-system, hace que la transportabilidad de programas sea casi absoluta.

Open Access (y 3)

Los entornos de agenda y comunicaciones

Sin duda, los paquetes integrados tienen una ventaja innata sobre los programas de único objetivo: la fácil comunicación entre módulos. A cambio, también presentan algunos aspectos negativos respecto a los programas individuales, como es la mayor necesidad de memoria principal, la casi obligatoriedad de disponer de disco rígido y, sobre todo, la menor calidad de los módulos al compararlos con programas especializados en su misma misión. OPEN ACCESS es una excepción, ya que algunos de sus módulos, como la base de datos, gráficos o agenda, igualan o superan en calidad a otros programas de objetivo único.

En este capítulo finaliza el estudio de los módulos del OPEN ACCESS, resaltando algunas peculiaridades del programa en su conjunto.



La incorporación de un módulo/agenda dentro del programa OPEN ACCESS es un elemento que realza su interés. Dado el tipo habitual de usuarios de OPEN ACCESS, la utilidad de este entorno resulta notoria.

AGENDA

La planificación personal, tanto para actividades empresariales como privadas, se ha venido realizando tradicionalmente mediante agendas en las que se apuntan citas, direcciones, teléfonos y cualquier otra anotación que su propietario considere oportuna. También suelen incluir un calendario válido para dos o tres años. Pues bien, el módulo agenda de OPEN ACCESS se encarga de ofrecer al usuario todas estas posibilidades corregidas y aumentadas. Para tener una idea de su potencia, cabe citar que puede almacenar hasta 32.000 citas; en las hojas del calendario pueden realizarse 32.000 anotaciones distintas y contempla hasta el año

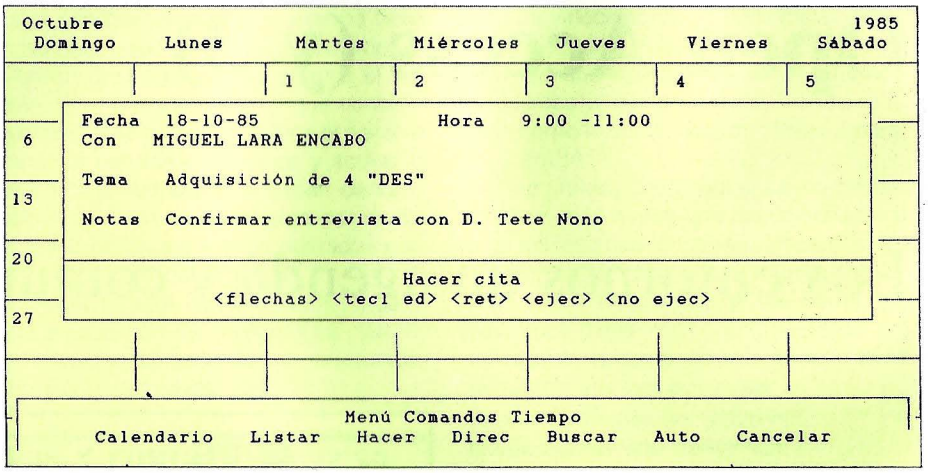
Octubre 1985

Domingo	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

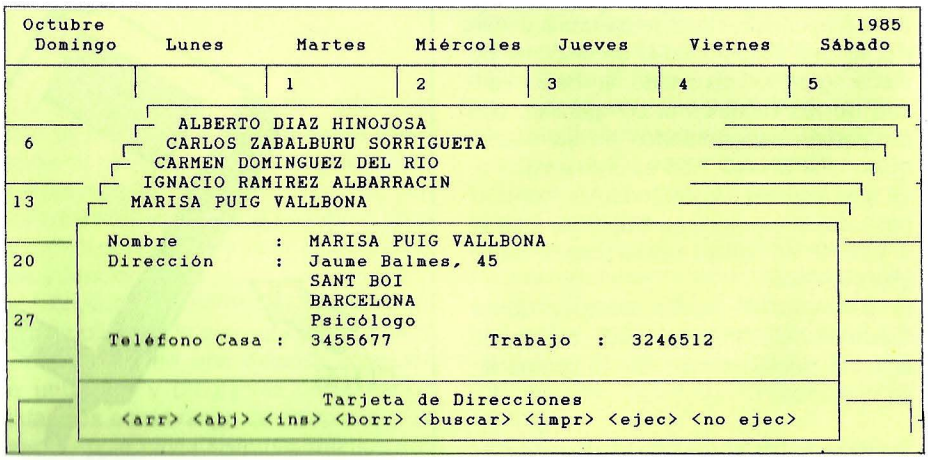
Menú Comandos Tiempo
Calendario Listar Hacer Direc Buscar Auto Cancelar

El módulo agenda de OPEN ACCESS sirve para gestionar la información que suele incluirse en una agenda convencional. Su funcionamiento se fundamenta en un calendario sobre el que se pueden realizar anotaciones y en una base de datos con direcciones y teléfonos.

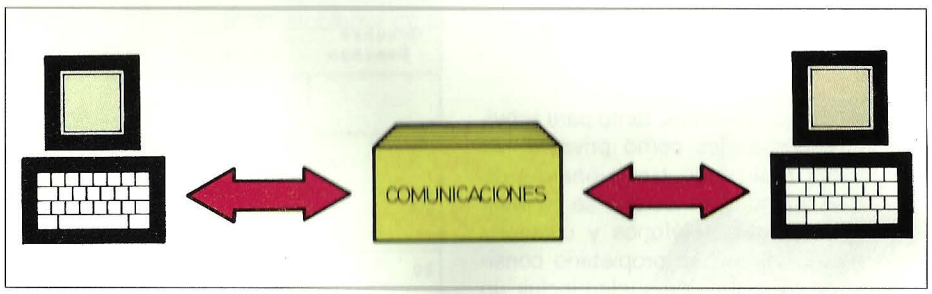
1999, más que suficiente ya que al ritmo en el que se desarrolla la Informática, es probable que para ese año el concepto de ordenador y software integrado no tenga nada que ver con el sentido actual.
Cuando el usuario accede a este módulo, ya sea pulsando el carácter "A" en el teclado, o bien a partir del menú principal de opciones, inmediatamente podrá visualizar en la pantalla un mensaje que le solicitará el nombre del propietario de la agenda. OPEN ACCESS admite que más de una persona disponga de su agenda particular mecanizada. A continuación, aparecerá el calendario correspondiente al mes en curso; para ello el programa tomará información de la fecha introducida al comenzar la sesión de trabajo. Ya en ese momento, sin necesidad de ejecutar ningún nuevo comando, el usuario estará en disposición de anotar cualquier frase en el día que desee.
En el menú correspondiente al módulo Agenda, existen once comandos distintos con las siguientes misiones:



Mediante el comando HACER del módulo agenda se pueden introducir citas. Para cada una de ellas se debe indicar la fecha, la hora, con quien se mantendrá la reunión, el tema que se tratará y las notas que se consideren oportunas.



La figura reproduce el aspecto de la pantalla del ordenador cuando se activa el comando DIRECCIONES de la agenda del OPEN ACCESS.



Mediante el módulo de comunicaciones se puede establecer una vía a través de la que fluirán los datos entre dos ordenadores distantes.

- CALENDARIO

Permite abrir la ventana de calendario donde el usuario realizará las anotaciones correspondientes. Inicialmente el calendario corresponderá al mes en curso, aunque puede ser desplazado hacia delante o hacia atrás, hasta llegar al mes deseado. Dado que cada celda correspondiente a un día es demasiado pequeña como para contener anotaciones largas, el usuario puede acceder a un cuaderno de notas asociados, donde podrá escribir frases y párrafos más extensos.

- LISTAR

Mediante ese comando el usuario puede obtener un listado de citas diarias, en el que para un determinado día obtendrá el plan de trabajo hora a hora.

- HACER

Sirve para introducir nuevas citas. Antes de seleccionarlo habrá que situar el cursor en el día correspondiente del calendario y, a continuación, aparecerá una nueva ventana en la que se podrá indicar: qué horas se reservan para la cita, con quién se realizará, el tema que se tratará y las notas que el propietario de la agenda considere oportunas.

- DIREC

Este comando se encarga de mantener un archivo de direcciones. El aspecto físico de la ventana que utiliza es muy similar al de un archivador en el que para cada persona existe una ficha que incluye: el nombre, la dirección, el teléfono del trabajo y el particular. Por supuesto, el programa permite introducir nuevas tarjetas y realizar todo tipo de búsquedas rápidas.

- BUSCAR

Tiene como misión realizar búsquedas en

el archivo de direcciones. Para ello basta con seleccionar el nombre a obtener, e inmediatamente la ficha correspondiente pasará a ser la activa.

- AUTO

Mediante el comando AUTO se pueden introducir citas que se suceden regularmente; por ejemplo, si todos los martes el propietario de la agenda tiene una reunión para la entrega del trabajo realizado en la semana anterior, podrá incluir las citas automáticamente de una sola vez.

- CANCELAR

El propio nombre del comando es autoexplicativo; con él se puede cancelar una o varias citas que, por cualquier circunstancia, hayan sido anuladas.

- HORAS

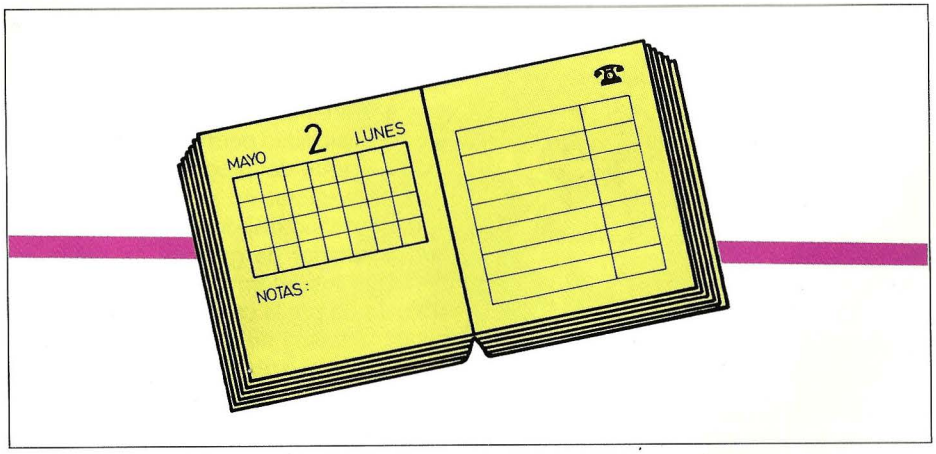
El comando horas protege al usuario contra el "estrés" Mediante él, para cada día de la semana se pueden reservar horas que, en consecuencia, no estarán disponibles para citas ni otras actividades.

- IMPRIMIR

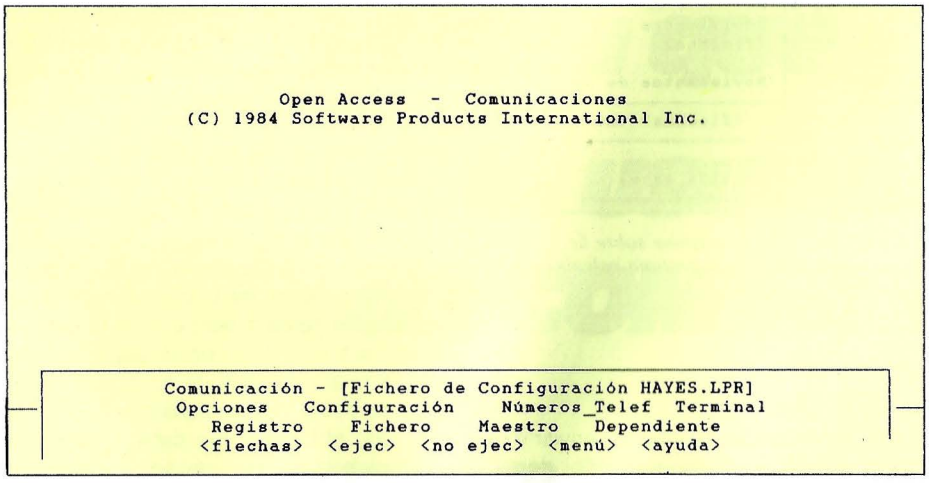
Permite obtener una copia en el papel del plan de citas diarias. Naturalmente, la ejecución de este comando requiere la existencia de una impresora correctamente configurada en OPEN ACCESS (para ello se dispone del entorno de Utilidades).

- USUARIO

Este último comando permite especificar el usuario que está trabajando con la agenda.



Uno de los módulos más atractivos del OPEN ACCESS lo constituye la agenda; un entorno capaz de simular las funciones desempeñadas por una auténtica agenda personal.



Mediante los distintos comandos del módulo de comunicaciones, OPEN ACCESS ofrece al usuario la posibilidad de gestionar el intercambio de información entre dos ordenadores.

En resumen, este módulo puede ser considerado como un elemento muy válido para la planificación de tiempos y, de alguna manera, obliga al usuario a actuar de forma metódica.

COMUNICACIONES

El último entorno del OPEN ACCESS facilita la comunicación con otros ordenadores.

En principio existen dos alternativas para establecer la comunicación entre dos ordenadores: conectarlos directamente a través de cable, o conectarlos a través de un modem acústico. La primera posibilidad resulta muy cómoda cuando la distancia física entre los dos equipos es pequeña; en cambio, cuando los ordenadores estén situados a grandes distancias, resulta imprescindible utilizar líneas telefónicas y los correspondientes modems.

Cuando el usuario accede al módulo de comunicaciones del OPEN ACCESS, este le presentará siete comandos distintos:

- CONFIGURACION

Permite crear una nueva configuración o seleccionar una ya existente. El usuario puede visualizar en la pantalla del fichero en que se encuentran las características de la configuración y modificación en el caso de que así lo desee. Finalmente este comando permite inicializar el modem y la puerta serial con determinada configuración.

- NUMEROS DE TELEFONO

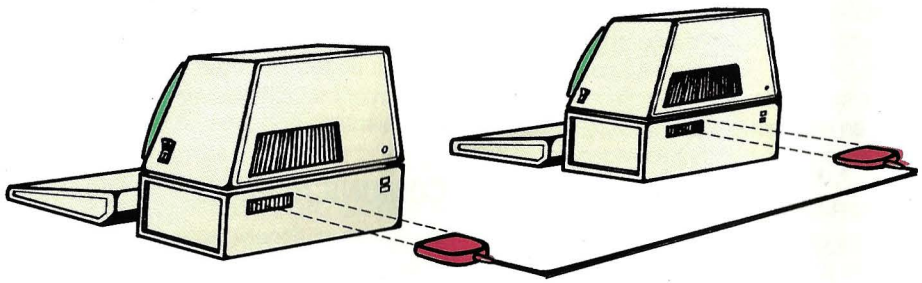
Si el usuario ha decidido establecer la comunicación a través de modems inteligentes, este comando permitirá marcar automáticamente los números telefónicos de una lista mantenida por el usuario.

- TERMINAL

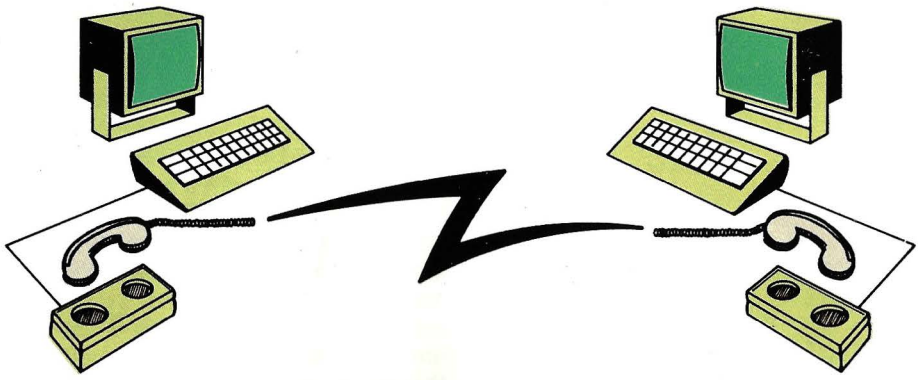
Convierte al ordenador en un terminal de otro ordenador.

- REGISTRO

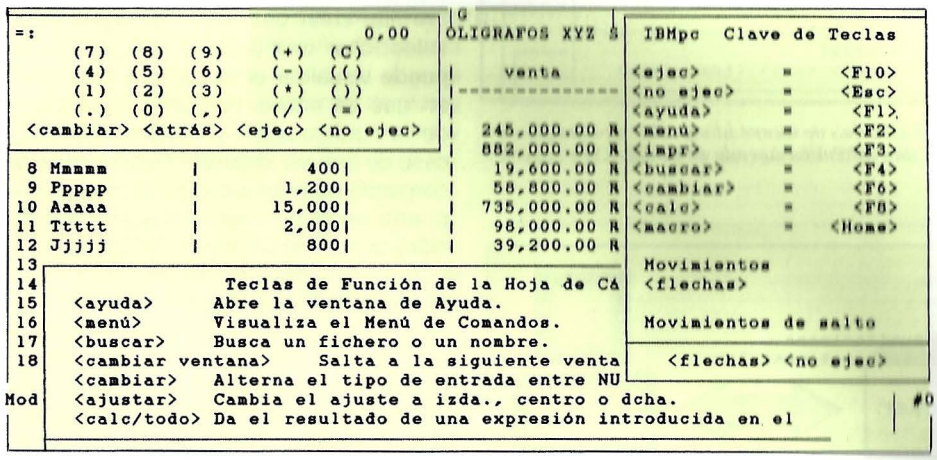
Este comando tan solo puede utilizarse cuando se esté funcionando en el modo terminal y su misión consiste en crear un fichero para almacenar la información recibida desde el ordenador principal.



Una posibilidad inicial para establecer la comunicación entre ordenadores próximos consiste en conectarlos mediante el cable apropiado...



... En cambio, cuando la distancia entre los equipos a conectar es grande, será necesario recurrir a modems que gestionen la conversión digital/analógica/digital.



En la figura aparecen algunas de las ventanas que OPEN ACCESS superpone sobre la pantalla para facilitar la labor del usuario. En este caso se ha elegido la ventana calculadora, una ventana de ayuda y la de descripción de teclas de función.

● FICHERO

El objeto de este comando es de tipo transferencia; es decir, al ser ejecutado se producirá el envío de un texto ASCII sin formatear de un ordenador a otro.

● MAESTRO

Sirve para acceder, mantener, enviar y recibir ficheros desde el ordenador dependiente; es decir, el ordenador local se inicializa como máquina primaria.

● DEPENDIENTE

Este comando se emplea en conjunción con el comando maestro. Permite que los ficheros de ordenador puedan ser accedidos y mantenidos desde el ordenador maestro; esto significa que el ordenador local se inicializa como máquina secundaria.

CARACTERISTICAS PECULIARES DEL OPEN ACCESS

Los requisitos hardware para instalar el programa OPEN ACCESS consisten en un ordenador con compatibilidad IBM-PC, dotado de una memoria principal de al menos 256 Kbytes de RAM, dos unidades de disquete o una única unidad de disquete y un disco duro; obviamente, esta última posibilidad permite una explotación más eficiente del programa.

Para la explotación del módulo gráfico será necesario disponer de algún dispositivo adaptador de gráficos. La salida impresa de los informes puede obtenerse, además de por la propia pantalla del ordenador, a través de la mayor parte de las impresoras existentes en el mercado, o a través de plotters del tipo HP-7470 o STROBE 260.

En cuanto al método de funcionamiento lógico, cabe destacar como constante en todos los módulos, la posibilidad de presentar varias ventanas superpuestas sobre la pantalla del ordenador. La situada en el primer nivel puede ser considerada como la ventana principal y dependerá del módulo que se esté utilizando; en algunos casos ésta variará según el comando ejecutado.

Sobre esta ventana y tapándola parcialmente, se puede obtener otra ventana de ayuda que presentará una descripción general de las posibles operaciones a efectuar sobre la ventana principal. En la parte superior derecha de la pantalla puede situarse una tercera ventana en la que se visualizará una descripción del cometido asignado a las teclas de funciones. Además de estas tres ventanas existen otras muchas, entre las que cabe destacar la ventana/calculadora que se situará en la parte superior izquierda y la ventana de utilidades.

Agenda telefónica

Ejercicio práctico con archivos de acceso directo

En anteriores capítulos de la obra dedicados al lenguaje BASIC, se ha hablado largo y tendido del tratamiento que pueden recibir los archivos aleatorios o de acceso directo. En esta ocasión se va a hablar de nuevo de ellos, aunque desde una perspectiva totalmente práctica; concretamente, desarrollando un programa de aplicación que resultará de plena utilidad.

Al tiempo que se confecciona el programa, se expondrán las peculiaridades que en este aspecto presenta el BASIC de los equipos ATARI; dialecto éste que se adoptará como herramienta de programación. Por lo tanto, a lo largo de los próximos párrafos se introducirán y estudiarán con detalle una serie de nuevos comandos relativos al manejo de los archivos en disco, específicos de estas máquinas.

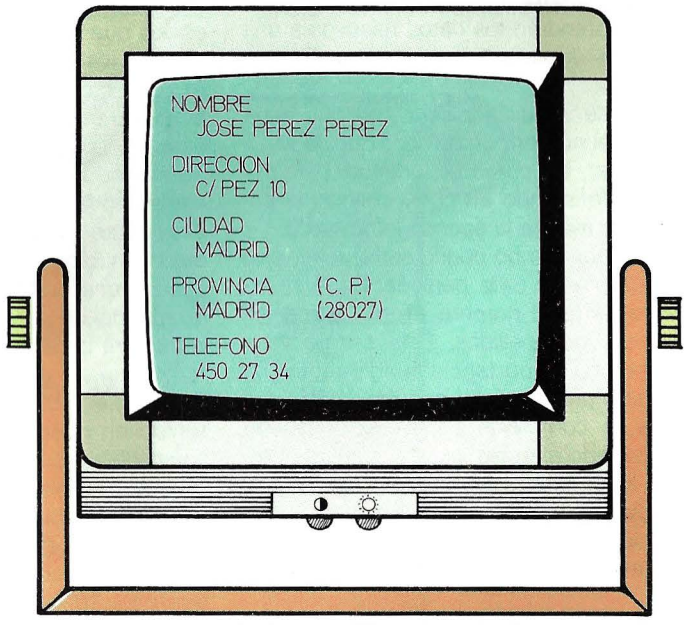
Como ya se ha precisado en otras ocasiones, los poseedores de máquinas dotadas de otra versión del lenguaje BASIC no deben echar en saco roto este capítulo. La traducción al BASIC respectivo de cada ordenador se podrá realizar muy fácilmente, sin más que establecer la equivalencia de unos pocos comandos, según se indica en las diferentes tablas de conversión publicadas a lo largo de la obra.

UNA AGENDA INFORMATIZADA

El programa-ejemplo a codificar coincide con una agenda de direcciones y teléfonos. Tal vez alguien dude de la utilidad real de este programa, pensando que a la hora de conocer una determinada dirección o un teléfono es suficiente hojear su propia agenda de bolsillo o el listín telefónico. Muy cierto. Pero ¿cuál será la solución que adoptarán en el caso de que deseen conocer cuántas personas de su agenda tienen el primer apellido igual? ¿O cuántas se llaman David para poderlas felicitar el día de su santo? Incluso puede resultar conveniente determinar cuántos amigos viven en Pontevedra, para visitarlos en un viaje inesperado a esa ciudad. O tal vez encontrar al dueño de un número de teléfono concreto que viene a la cabeza y no se consigue relacionar con ningún conocido.

Es indudable que una agenda convencional no es capaz de dar una eficaz solución a ninguna de estas cuestiones, ni a otras similares que cabría argumentar. Por supuesto que existe la agotadora posibilidad de examinar una por una todas las anota-

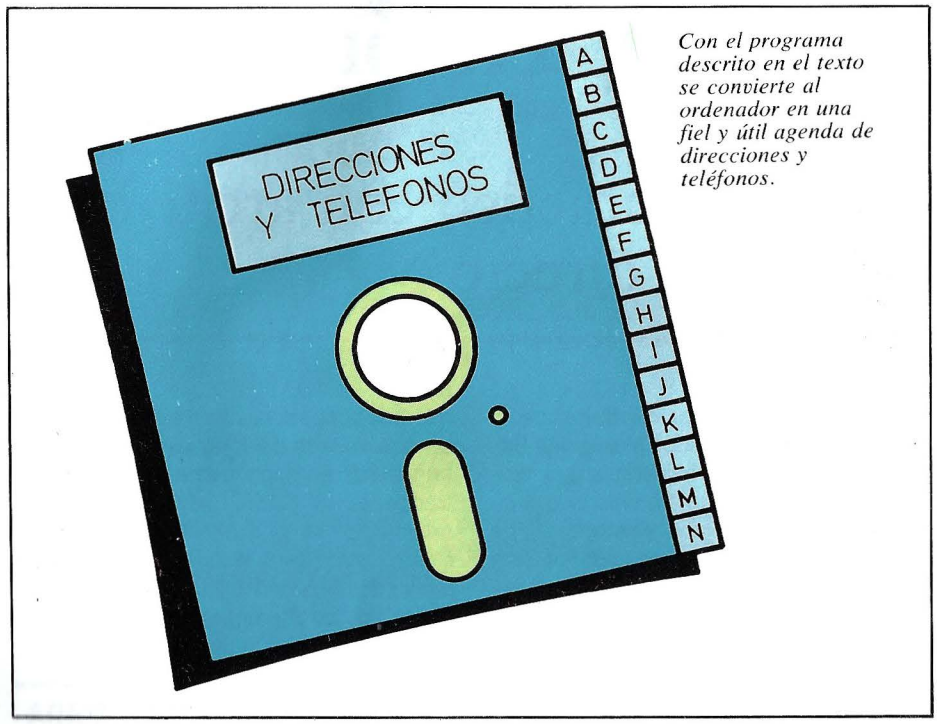
Pantalla con el menú de opciones que da entrada al programa "agenda telefónica".



Visualización en pantalla de un posible registro de almacenamiento en la agenda telefónica del programa ejemplo.

ciones realizadas en la agenda; ardua labor si el número de anotaciones es lo suficientemente extenso.
No hay que perder de vista que el acceso a la información contenida en una agenda convencional, sólo se puede realizar por la inicial del nombre de cada anotación realizada. No en balde éstas se realizan atendiendo a su clasificación alfabética. En una agenda informatizada dicho acceso puede programarse de tal forma que la búsqueda se realice por diferentes conceptos: apellido, ciudad de residencia, número de teléfono, etc. Así, el acceso a la información almacenada en ella resulta de una gran versatilidad.
Aclarado este concepto, pasemos a la acción. La analogía de la agenda convencional con los archivos informáticos es enorme. Así, una agenda constituye en sí misma un archivo, entendiendo por archivo a un conjunto de informaciones relacionadas entre sí.
La agenda, como conjunto, es divisible en otras partes que son las diferentes anotaciones para cada persona o entidad, lo que coincide totalmente con el concepto informático de registro. Estas anotaciones de la agenda estarán compuestas, a su vez, por el nombre, la dirección, la ciudad, el teléfono... Elementos que constituyen lo que en términos informáticos se denominan campos de un registro.
En definitiva, nuestra agenda informatizada estará almacenada en un archivo (en este caso en disco, para poder utilizar el acceso aleatorio), dividido en registros que contendrán los datos relativos a una persona depositados en los respectivos campos.
Antes de seguir adelante, hay que establecer el número de campos que se van a necesitar, así como su longitud, para poder ir definiendo en consecuencia la estructura real de la agenda informática.
Un primer campo podría ser el reservado al nombre de cada persona. En él se incluirá tanto el nombre de pila como los respectivos apellidos. Este campo va a tener una importancia fundamental en nuestro programa, ya que para evitar que éste se complique en exceso y resulte demasiado extenso, el campo del nombre va a servir de índice; o lo que es lo mismo, de clave interna para el acceso a la información guardada en la agenda. Los restantes campos pueden ser los siguientes:

— Campo de dirección (calle, número, piso, etc.).



Con el programa descrito en el texto se convierte al ordenador en una fiel y útil agenda de direcciones y teléfonos.

— Campo de ciudad o localidad.
— Campo de provincia y código postal.
— Campo de número de teléfono (incluyendo prefijos, etc.).

La longitud de estos campos se puede elegir libremente, ya que no está prefijado ningún valor a priori. En todo caso, lo más conveniente es "amoldarse" en cierta medida a la longitud de los bloques físicos en los que el sistema operativo de disco divide a los diferentes sectores del disco. Al respecto, no hay que olvidar que los bytes no utilizados de cada uno de estos bloques quedarán ya inaccesibles, desperdiciándose un cierto espacio de almacenamiento.
El sistema operativo de ATARI divide el disco en bloques de 256 bytes; de los cuales, una parte se utiliza para control, quedando un total de 214 bytes como zona libre para el almacenamiento de información.
Para no ser demasiado estrictos, no se ha tenido en cuenta este aspecto, y se ha asignado a todos los campos una longitud de 40 caracteres; suficiente a todas luces para poder albergar las respectivas informaciones. Si bien, es obvio que para el campo destinado a albergar el número de teléfono, 10 ó 15 caracteres serían más que suficientes.
El siguiente paso es establecer las distintas funciones que debe gestionar el programa de agenda. Por ejemplo:

— Búsqueda de un registro específico.
— Introducción de un nuevo registro.
— Listado de todos los registros.
— Modificación de un registro.
— Borrado de un registro.
— Abandonar el programa, almacenando los datos necesarios.

Estas seis funciones encierran todo el proceso de creación, actualización y almacenamiento del archivo que constituirá la agenda. A partir de ellas se puede ya definir una primera estructura del programa, que va a ser controlado por un menú de opciones coincidentes con las funciones indicadas. A partir de este menú y de la opción elegida en cada momento, el programa bifurcará hacia una serie de subrutinas que, una vez ejecutadas, devolverán el control al menú de opciones. Dicho menú constituirá el programa principal o bucle principal, que se ejecutará indefinidamente, hasta que se dé paso a la opción de salida del programa.
Se puede ya empezar a codificar el programa; comenzando por una zona de presentación que consistirá, sencillamente, en un título centrado mediante el comando POSITION. Esta es la tarea asignada a las instrucciones 10, 20 y 30.

Previamente a la codificación del bucle principal del programa, habrá que realizar una subrutina de inicialización del programa y sus variables; subrutina que será llamada por la línea 40 de la zona de presentación.
La subrutina de inicialización sólo será ejecutada una vez al principio del programa, por lo que podría haber sido escrita directamente en las líneas 30 y 50 de la presentación. No obstante, se ha preferido darle la estructura de subrutina ya que así queda más claro el programa y se ve facilitada la comprobación de su correcto funcionamiento. La subrutina de inicialización se ocupará de dimensionar y de inicializar las diferentes variables a utilizar en el programa, así como de obtener del disco la información necesaria para que sea posible acceder a la agenda con posterioridad. Por este motivo, se pospondrá la confección de dicha subrutina hasta el momento en el que hayan definido las variables necesarias, así como la estructura de los ficheros en disco. Hasta que llegue ese instante, daremos por supuesto que todas las variables han sido adecuadamente dimensionadas.
Continuando con el programa principal, el siguiente paso es crear el bucle infinito que se encargue de mantener su ejecución. Para ello se empleará una simulación de estructura REPEAT/UNTIL, construida mediante el comando BASIC FOR/NEXT:

```
60 FOR BUCLE=0 TO 1 STEP 0
```

El incremento de la variable BUCLE es 0, por lo que ésta nunca alcanzará el valor 1 que determinaría la salida del bucle; claro está, excepto que se establezca directamente el valor de BUCLE en 2, como se hará en la subrutina de salida del programa. La penúltima línea del bucle principal, como se verá más tarde, debe coincidir con la instrucción NEXT BUCLE, seguida por la orden END que evite la ejecución imprevista de las siguientes subrutinas. De esta forma se evita además el empleo del comando GOTO, que complicaría el seguimiento del código.
Acto seguido hay que visualizar en la pantalla el menú de opciones, tarea encomendada a las líneas 70 a la 140.
Una vez que se tiene en pantalla el menú de opciones, hay que solicitar al usuario que elija la opción que desee. Puesto que el BASIC de ATARI no posee el comando INKEY$, la captación de la respuesta debe realizarse con el comando equivalente GET. Este necesita la apertura del teclado como dispositivo de entrada de datos, lo que se consigue mediante una instrucción OPEN:

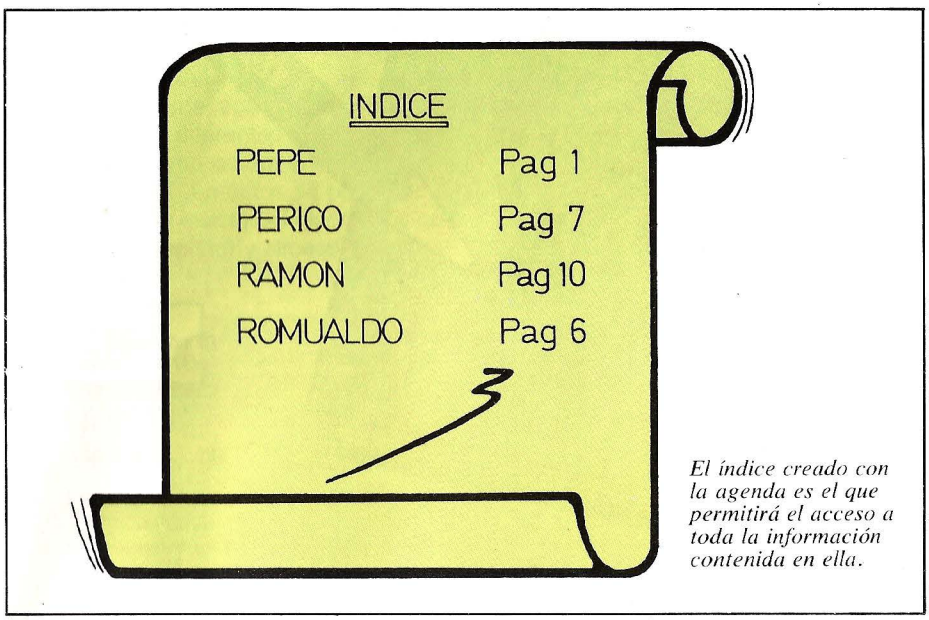
```
150 OPEN # 1, 4, 0, "K:"
```

En donde el 4 denota una operación de entrada de datos y "K:" es el código de dispositivo correspondiente al teclado.
A continuación hay que imprimir en pantalla la petición de pulsar una tecla. Como quiera que el menú ofrece seis opciones, éstas se elegirán mediante un número comprendido del 1 al 6. Desde luego, el programa debe comprobar que la elección es correcta, desechando la pulsación de teclas que no sean dígitos del 1 al 6. El comando GET obtiene el código ASCII de la tecla pulsada; código que debe estar comprendido entre el 49 y el 54 (ambos inclusive) que corresponden a los dígitos del 1 al 6. Todo ello se codifica en las líneas de programa que van de la 160 a la 190.
Cabe observar que se ha simulado de nuevo una estructura REPEAT/UNTIL en la línea 180, la cual establece la salida de este bucle asignando a la variable L el valor 2 si la tecla pulsada es correcta. Si la tecla pulsada no es correcta, se volverá a ejecutar el bucle solicitando la pulsación de una nueva tecla.
Una vez detectada la opción que desea el usuario, sólo habrá que utilizar una instrucción ON/GOSUB para ejecutar la subrutina correspondiente. La variable selectora de ON/GOSUB será T. Al código ASCII de la tecla pulsada habrá que restar el valor 48, para obtener así un número del 1 al 6 que permita la bifurcación a las subrutinas indicadas detrás de la palabra GOSUB. El bucle principal concluirá, por tanto, con las líneas:

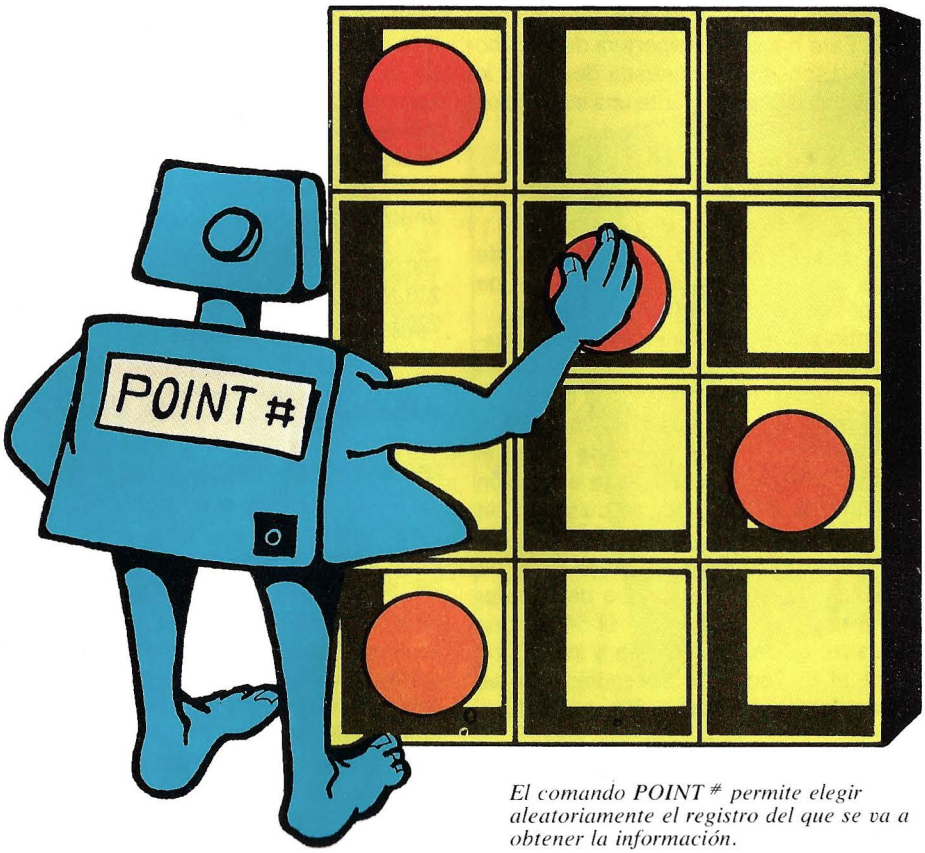
```
200 ON T-48 GOSUB 350, 800, 900, 950, 1040, 1110
210 NEXT BUCLE
220 END
```

ADQUISICION DE LOS DATOS

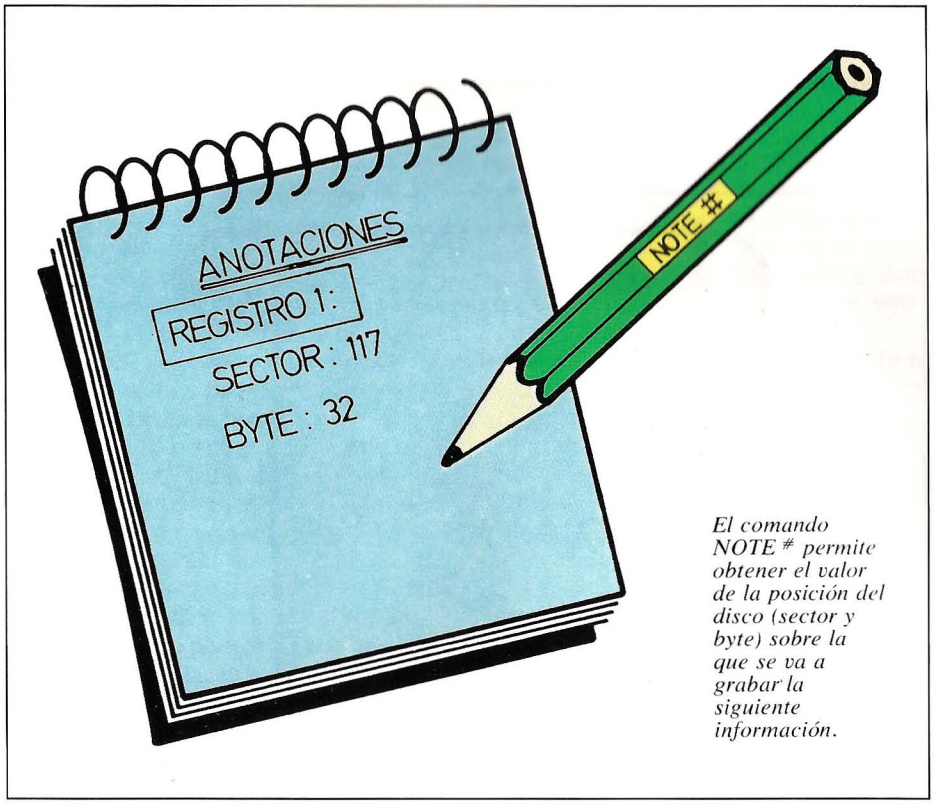
Para seguir un orden lógico —el cual no coincide con el indicado en el menú de opciones— hay que estudiar primero la subrutina de introducción en la agenda de un nuevo registro. Es obvio que sin ejecutar esta opción al principio, difícilmente se podrá buscar, modificar o eliminar algún registro. La subrutina para introducir un nuevo registro empezará inicializando las variables oportunas; ésta es, precisamente la misión de las líneas 800 a la 830.
Cada una de las variables utilizadas almacenará temporalmente la información de cada uno de los campos. Así, NOMBRE$ contendrá el nombre completo correspondiente a este nuevo registro, CALLE$



El índice creado con la agenda es el que permitirá el acceso a toda la información contenida en ella.



El comando POINT # permite elegir aleatoriamente el registro del que se va a obtener la información.



El comando NOTE # permite obtener el valor de la posición del disco (sector y byte) sobre la que se va a grabar la siguiente información.

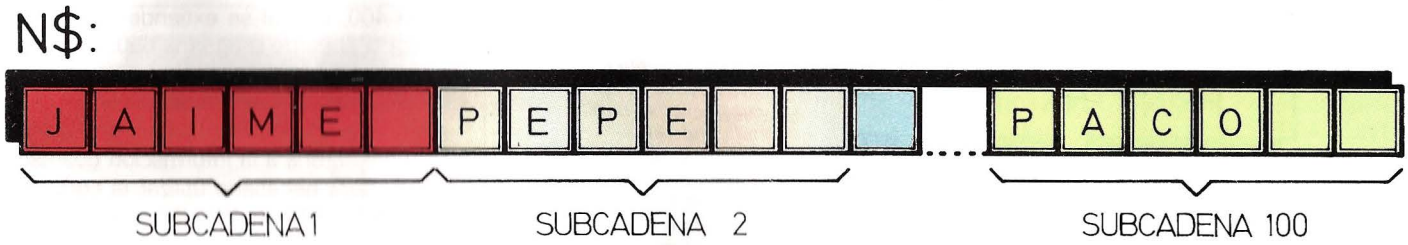
la dirección, CIUDAD$ la localidad de residencia, PROV$ la provincia y TEL$ el número de teléfono. Todas ellas se inicializan con el valor de B$, que es una cadena constante de 40 caracteres blancos. El motivo es que otras funciones de la agenda utilizarán también estas variables y pueden contener otras informaciones no deseadas.

La línea 820 llama a la subrutina de la línea 670, que se encarga de la introducción por parte del usuario, mediante el teclado, de los datos del nuevo registro. Dicha subrutina será también utilizada por la función de modificación de registros.

Abandonemos pues, momentáneamente, la subrutina de introducción de nuevo registro, para estudiar esta última (líneas de la 670 a la 790). Resulta de muy fácil asimilación, ya que se limita a introducir mediante comandos INPUT los datos requeridos, para, a continuación, pedir confirmación al usuario de que los datos son correctos. De ser así, se vuelve a interrumpir la estructura REPEAT/UNTIL, nuevamente simulada (línea 770) mediante el FOR de la línea 700. La entrada de datos con las instrucciones INPUT se realiza después de haber ejecutado la subrutina de la línea 610, la cual imprime en pantalla las cabeceras y valores actuales de los diferentes campos de un registro de la agenda; éstos serán utilizados después como referencia del dato cuya entrada se está solicitando con el comando INPUT. Por este motivo se vuelven a inicializar en blanco las variables de los diferentes campos, para la función de modificación de registros que se verá más adelante. Todo ello puede parecer un tanto complicado; si bien, hay que considerar que de esta forma se evita la necesidad de escribir varias veces las mismas líneas en distintas zonas del programa.

Continuando con la subrutina de entrada de un nuevo registro, la cual se "aparcó" momentáneamente, cabe señalar que en su línea 830 se efectúa una llamada a una nueva subrutina, localizada ésta entre las líneas de programa 480 y 530. Su misión es la de transformar, carácter a carácter, el contenido de la variable NOMBRE$, de forma que quede en mayúsculas (línea 510) y desprovisto de todo carácter distinto de una letra del abecedario (línea 520).

El objetivo de esta transformación no es otro que crear un índice estandarizado que permita acceder a través del mismo a los restantes campos de información de la



Las matrices multidimensionales de caracteres se pueden simular en el ATARI "trozeando" artificialmente una cadena suficientemente larga.

agenda. Para no dilatar en exceso el programa, sólo se va a codificar la creación de un índice para la localización de registros por su nombre; aunque, bien es cierto, que también se podría realizar de la misma forma para acceder al registro por el campo de dirección, de ciudad o de teléfono, sin más que crear un índice para cada uno de estos campos. Así pues, en la variable NOM$ se obtendrá el nombre en mayúsculas y sin signos extraños del registro en curso. Este valor será introducido en la cadena N$, que constituirá el índice de los nombres de los registros que estén almacenados en el disco. De la actualización de este índice con el nuevo nombre se encarga la siguiente línea:

```
840 N$(P*40-39, P*40)=NOM$ : N$(4001)=" "
```

La variable N$ se supone previamente dimensionada con 4001 caracteres. Esto es así porque el BASIC del ATARI no admite matrices multidimensionales de cadenas de caracteres; de ahí que sea necesario simularlas mediante una única cadena que será artificialmente "cortada" en los trozos oportunos.
Como la longitud máxima del nombre es de 40 caracteres (longitud adoptada para ese campo), y se ha previsto una capacidad máxima de 100 registros en la agenda (cantidad que puede ser modificada a voluntad), esto hace un total de 40*100= =4000 caracteres, que serán necesarios para el índice. Cada subcadena, dentro de la cadena N$, será localizada gracias a la variable P, que contendrá en todo momento el número de posición del siguiente registro libre, con lo que el nombre para el índice del nuevo registro guardado hasta ahora en NOM$, se almacenará en N$(P*40-39, P*40). El hecho de ordenar que el último carácter de N$ sea " ", se debe a que al asignar un valor a una subcadena de N$, la longitud de ésta queda recortada a P*40, lo que podría dar lugar a errores al querer buscar posteriormente una subcadena en N$.

CREACION DEL ARCHIVO DE ACCESO ALEATORIO

Una vez actualizado el índice, sólo queda almacenar el registro en el archivo del disco; tarea ésta cuya ejecución corresponde a las líneas de la 850 a la 890.
En la línea 850 se abre el archivo AGENDA.DAT en modalidad APPEND (ver cuadro adjunto), con lo que el nuevo registro se podrá incorporar al final del archivo. Pero ¿qué ocurre si el programa es la primera vez que se ejecuta y no existe todavía ese archivo en el disco? Pues que se producirá un error de "archivo no encontrado". El comando TRAP de esta misma línea, hará que al producirse ese error se derive la ejecución del programa a la línea 890, en la que se creará por vez primera el archivo AGENDA.DAT.
Los archivos de acceso aleatorio del ATARI no son diferentes a los de acceso secuencial, como ocurre en otras versiones del BASIC. En éstos se puede obtener la posición exacta del disco en la que se graba un registro y almacenarla en alguna variable, para, posteriormente, acceder directamente a esa posición. El comando del BASIC-ATARI que se encarga de tal cometido es NOTE, cuyo formato es el siguiente:

```
NOTE # <canal>, <var. 1>, <var. 2>
```

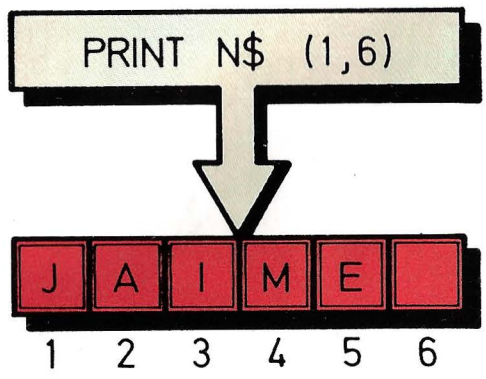
El parámetro <canal> es el número identificador de un archivo previamente abierto en las modalidades WRITE, APPEND o UPDATE, mientras que <var. 1>, <var. 2>, son dos variables numéricas.
El funcionamiento de este comando es el siguiente: la unidad de disco (o más bien el D.O.S.) posee un puntero que señala al sector y al byte concreto de dicho sector al que se está accediendo en cada momento. El comando NOTE accede a ese puntero y almacena el número de sector en la primera variable específica (<var. 1>) y el número del byte de ese sector en <var. 2>. Una vez leídos estos valores, no queda más que grabar la información mediante un PRINT# o un PUT#. Como se puede ver en el programa de agenda, la línea 860 obtiene con NOTE la posición del disco en la que se va a grabar la información y después la almacena en la matriz numérica bidimensional INDICE(,), para saber dónde encontrarla cuando sea necesario. Seguidamente, "imprime" en esa posición los diferentes campos del registro y cierra el archivo, con lo que la

NOTE

Obtiene la posición del puntero de la unidad de disco en la que se va a efectuar una grabación de un registro de acceso aleatorio.

Formato: <n. 1.> NOTE #<canal>, <var. 1>, <var. 2>

Ejemplos:
```
10 NOTE #2, SECTOR, BYTE
20 NOTE #1, IND(1), IND(2) : PRINT #2; A$
```



El fraccionamiento de una cadena de caracteres se realiza indicando entre paréntesis el principio y el final de la porción a obtener. Esta técnica equivale a la puesta en práctica por el comando MID$.

información queda ya a buen recaudo; tras ello actualiza la variable P para el siguiente acceso y se devuelve al ordenador el control de errores mediante TRAP 40000. Con ello finaliza la subrutina de introducción de un nuevo registro y se regresa al programa principal para presentar en pantalla el menú de opciones.

EN BUSCA DEL REGISTRO PERDIDO

Para recuperar un registro grabado en disco se pulsará la tecla 1, con lo que el programa saltará a la subrutina de la línea 350, la cual se extiende hasta la 430.
En primer lugar se pide el nombre que se desea buscar, almacenándolo mediante un INPUT en la variable NOMBRE$. La subrutina 480, llamada en la siguiente línea, reconvertirá el nombre —en la variable NOM$— dándole el formato adecuado para que sea posible localizarlo en el índice constituido por la variable N$. La variable POS empieza con valor 1, e irá recorriendo uno a uno todos los caracteres ocupados de N$, hasta hallar una equivalencia entre el nombre a buscar (almacenado en NOM$) y una subcadena de N$; dicha misión se encomienda a la subrutina llamada en la línea 380 y que ocupa las líneas comprendidas entre la 440 y 470.
Una vez localizado el nombre en cuestión, se halla el valor de la variable CLAVE. Este coincidirá con el número de registro en el que está guardada la información solicitada. Acto seguido se actualiza POS, por si ese nombre no es el deseado y hay que seguir buscando. Esto es así debido a que dos registros pueden tener el mismo nombre, por lo que al solicitar ese nombre obtendrán los dos.
La subrutina que nos ocupa devolverá el control al punto de llamada cuando encuentre el nombre buscado, o bien cuando no encuentre ese nombre en todo el índice; es este último caso, retornará con el valor de CLAVE 0. De acuerdo al valor devuelto como clave, la subrutina de búsqueda de registro imprimirá el mensaje de "no encontrado" y volverá al programa principal cuando sea CLAVE=0. No obstante, si el contenido de CLAVE es distinto de 0, pasará a leer ese registro del disco mediante la subrutina llamada en la línea 400, la cual se extiende desde 540 hasta la línea de programa 600.
Esta subrutina abre el archivo AGENDA.DAT que contiene la información de la agenda, en modo READ. Para acceder de forma aleatoria a la información guardada en él, será necesario utilizar el comando del BASIC-ATARI: POINT#, complementario del NOTE#, y cuyo formato es el siguiente:

POINT #<canal>, <var. 1>, <var. 2>

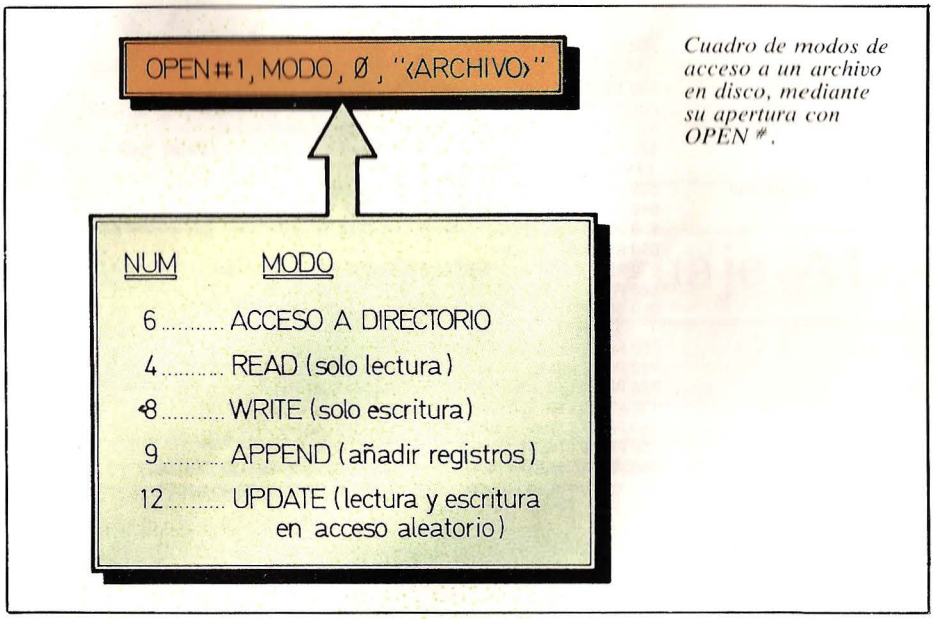
Este realiza la operación contraria a NOTE#, de tal forma que permite situar el puntero de la unidad de disco en la posición que se desee; posición que se indicará mediante el contenido de las variables <var. 1> y <var. 2>, para el sector y el byte de dicho sector, respectivamente.
Como se recordará, esta posición estaba almacenada en la variable INDICE(,), la cual será leída mediante el valor de CLAVE como se muestra en la línea 580.
Una vez posicionado el puntero, sólo queda ejecutar las correspondientes instrucciones INPUT# para leer la información del disco. Tras presentar la información leída en la pantalla mediante la subrutina 610, se devuelve el control al programa principal.
Las restantes opciones utilizarán las mismas subrutinas que ya se han explicado hasta aquí, por lo que resultará muy sencillo comprender su funcionamiento. Por ejemplo, para listar todos los registros, sólo habrá que crear un bucle que recorra todos los registros uno a uno, e ir leyendo la información del disco y presentándola en pantalla. Esta subrutina es la comprendida entre las líneas 900 y 940.
Tras presentar en pantalla un registro, se esperará a que el usuario pulse una tecla cualquiera para visualizar el siguiente.
La subrutina de modificar un registro coincidirá también con una combinación de las subrutinas ya estudiadas. Concretamente, la zona básica de la misma se sitúa entre las líneas de programa 950 y 1030. En ella se pide en primer lugar el nombre del registro a modificar; éste es buscado de forma similar a como se hacía en la opción de encontrar un registro. Después se visualiza en pantalla y mediante instrucciones INPUT se modifican los campos necesarios. Una vez modificado, se deposita el contenido en el archivo empleando el comando POINT# y los PRINT#; la única salvedad es que en este caso el archivo se abre en modalidad UPDATE. Previamente

POINT

Posiciona el puntero del disco en el registro de acceso aleatorio al que se desea acceder.

Formato: <n. 1.> POINT #<canal>, <var. 1>, <var. 2>

Ejemplos: 30 POINT #3, IND(1), IND(2)
50 POINT #3, SECTOR, BYTE : INPUT #3; A$



Cuadro de modos de acceso a un archivo en disco, mediante su apertura con OPEN #.

se habrá actualizado el índice N$, con lo que se da por terminada esta subrutina.

La subrutina de eliminar un registro reduce su actuación a borrar a éste del índice (variables INDICE( ) y N$), con lo que ya no será accesible. Dicha eliminación se opera desplazando hacia atrás, y uno a uno, los valores que lo siguen en las variables índice; si bien, no se elimina el archivo del disco, por lo que será fácilmente recuperable en el caso de que se haya ordenado su descarte por equivocación. La subrutina al efecto es la situada entre las líneas 1040 y 1100.

Finalmente, la opción de salida del programa se encargará de almacenar en el disco el archivo secuencial "INDICE.DAT", las variables P, INDICE( ) y N$, ya que hasta ahora éstas sólo residían en la memoria principal del ordenador y son esenciales para recuperar la información de la agenda. Por este motivo, no se debe ordenar la ruptura del programa en ningún punto del mismo; siempre hay que abandonar el programa ejecutando la opción 6 de salida. De otra forma se podría perder toda la información actualizada.

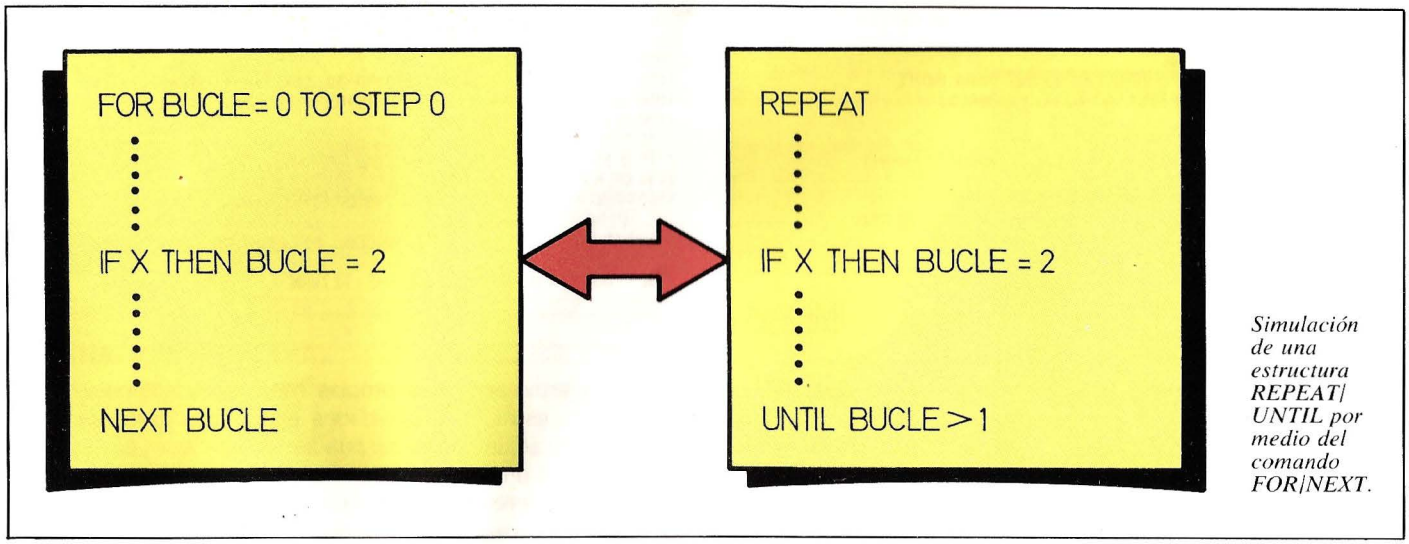
La grabación de los índices para el acceso se podría haber realizado tras cada actualización del archivo AGENDA.DAT, bien por añadir un nuevo registro, o por modificar o eliminar alguno ya existente; no obstante, ello haría que el manejo de la agenda resultara más lento y tedioso. La subrutina de salida está emplazada entre las líneas 1110 y 1160 del programa.

Como puede observarse, una vez grabadas las variables de índice, se ejecuta un GRAPHICS 0 que inicializa el sistema, y se asigna a BUCLE el valor 2 que concluirá la ejecución de la estructura REPEAT/UNTIL simulada en el programa principal; con ello se alcanzará la instrucción END.

A estas alturas sólo queda por comentar la subrutina de inicialización (líneas 230 a la 340), relegada hasta el final para su mejor comprensión. Ahora se conocen ya las distintas variables utilizadas en el programa.

En ella se empieza dimensionando las distintas variables utilizadas en el programa, de acuerdo con la dimensión requerida y prefijada en las líneas precedentes. La línea 250 de la mencionada subrutina pone en práctica un artificio, específico de los ordenadores de ATARI, que permite una rápida inicialización de cadenas de caracteres con un valor concreto (en este caso caracteres blancos para B$ y N$).

A continuación, se obtienen del archivo AGENDA.DAT los valores para las variables P, INDICE( ) y N$, que permitirán el acceso aleatorio al archivo AGENDA.DAT que memoriza la información de la agenda. En el caso de tratarse de la primera ejecución del programa —aún no existe en el disco el archivo INDICE.DAT—, se producirá un error de "fichero no encontrado"; error que es manipulado mediante el comando TRAP, de forma similar a como se explicó para la subrutina de creación del archivo AGENDA.DAT. En este caso no se creará el archivo INDICE.DAT, sino que se asignará a la variable P el valor 1, que revelará que no existen todavía registros en la agenda.



Simulación de una estructura REPEAT/UNTIL por medio del comando FOR/NEXT.

AGENDA ELECTRONICA

```
10 REM AGENDA ELECTRONICA
20 PRINT CHR$(125) : POKE 752, 1 : REM BORRAR PANTALLA Y ELIMINAR
   CURSOR
30 POSITION 10, 9 : PRINT "AGENDA TELEFONICA"
40 GOSUB 230 : REM INICIALIZAR VARIABLES
50 POKE 752, 0 : REM HACER APARECER CURSOR
60 FOR BUCLE=0 TO 1 STEP 0
70 PRINT CHR$(125) : REM BORRAR PANTALLA
80 POSITION 8, 1 : PRINT "MENU AGENDA TELEFONICA"
90 PRINT : PRINT : PRINT "1. ENCONTRAR REGISTRO"
100 PRINT : PRINT "2. AGREGAR NUEVO REGISTRO"
110 PRINT : PRINT "3. LISTAR TODOS LOS REGISTROS"
120 PRINT : PRINT "4. MODIFICAR REGISTRO"
130 PRINT : PRINT "5. ELIMINAR REGISTRO"
140 PRINT : PRINT "6. SALIDA DEL PROGRAMA"
150 OPEN #1, 4, 0, "K:"
160 FOR L=0 TO 1 STEP 0
170 POSITION 7, 16 : PRINT "SELECCIONE UN NUMERO ";: GET #1, T :
    PRINT CHR$(T)
180 IF T>48 AND T<55 THEN L=2
190 NEXT L : CLOSE #1
200 ON T-48 GOSUB 350, 800, 900, 950, 1040, 1110
210 NEXT BUCLE
220 END
230 REM SUBRUTINA DE INICIALIZACION
240 DIM NOMBRE$(40), T$(1), NOM$(40), CALLE$(40), CIUDAD$(40),
    PROV$(40), TEL$(40), B$(40)
250 DIM N$(4001), INDICE(100, 1) : N$=" " : N$(4001)=" " : N$(2)=N$ :
    B$=" " : B$(40)=" " : B$(2)=B$
260 TRAP 340 : OPEN #3, 4, 0, "D : INDICE.DAT"
270 INPUT #3; P
280 FOR I=1 TO P-1
290 INPUT #3; NOMBRE$ : N$(I*40-39)=NOMBRE$
300 NEXT I : N$(4001)=" "
310 FOR I=1 TO P-1
320 INPUT #3; T : INDICE(I, 0)=T : INPUT #3; T : INDICE(I, 1)=T
330 NEXT I : CLOSE #3 : TRAP 40000 : RETURN
340 CLOSE #3 : P=1 : TRAP 40000 : RETURN
350 REM SUBRUTINA BUSCAR REGISTRO
360 PRINT : PRINT "INTRODUZCA NOMBRE A BUSCAR" : INPUT NOMBRE$
370 GOSUB 480 : POS=1
380 FOR L=0 TO 1 STEP 0 : GOSUB 440 : REM BUSCAR NOMBRE EN INDICE
390 IF CLAVE=0 THEN PRINT : PRINT "NOMBRE NO ENCONTRADO";: L=2 :
    FOR K=1 TO 200 : NEXT K
400 IF CLAVE<>0 THEN GOSUB 540 : PRINT : PRINT "SIGO BUSCANDO ?
    (S/N) ";: OPEN #1, 4, 0, "K:" : GET #1, T : T$=CHR$(T) : CLOSE #1
410 IF CLAVE <>0 AND (T$="N" OR T$="n") THEN L=2
420 NEXT L
430 RETURN
440 REM SUBRUTINA BUSCAR NOMBRE
450 CLAVE=0 : FOR I=POS TO (P-1)*40-LEN(NOM$)
460 IF N$(I, I+LEN(NOM$)-1)=NOM$ THEN CLAVE=INT(I/40)+1 : POS=I+1 :
    I=(P-1)*40-LEN(NOM$)
470 NEXT I : RETURN
480 REM SUBRUTINA MODIFICAR NOMBRE PARA INDICE
490 NOM$=" " : N=1 : FOR L=1 TO LEN(NOMBRE$)
500 T$=NOMBRE(L, L) : T=ASC(T$)
510 IF T>=97 THEN T=T-32
520 IF T>64 AND T<91 THEN NOM$(N, N)=CHR$(T) : N=N+1
530 NEXT L : RETURN
540 REM SUBRUTINA LECTURA DE DATOS
550 IF P=1 THEN PRINT : PRINT " AGENDA VACIA" : RETURN
560 PRINT CHR$(125);: REM LIMPIAR PANTALLA
570 OPEN #2, 4, 0, "D : AGENDA.DAT"
580 POINT #2, INDICE(CLAVE, 0), INDICE(CLAVE, 1)
590 INPUT #2; NOMBRE$ : INPUT #2; CALLE$ : INPUT #2; CIUDAD$
600 INPUT #2; PROV$ : INPUT #2; TEL$ : CLOSE #2 : GOSUB 610 : RETURN
610 REM SUBRUTINA IMPRIMIR REGISTRO EN PANTALLA
620 PRINT CHR$(125); "NOMBRE" : PRINT " "; NOMBRE$ : PRINT "DIRECCION"
    : PRINT " "; CALLE$
630 PRINT "CIUDAD" : PRINT " "; CIUDAD$
640 PRINT "PROVINCIA (C. P.)" : PRINT " "; PROV$
650 PRINT "TELEFONO" : PRINT " "; TEL$
660 RETURN
670 REM SUBRUTINA INTRODUCIR DATOS POR TECLADO
680 PRINT CHR$(125)
690 GOSUB 610 : NOMBRE$=B$ : CALLE$=B$ : CIUDAD$=B$ : PROV$=B$ :
    TEL$=B$
700 FOR L=0 TO 1 STEP 0
710 POSITION 2, 1 : INPUT NOMBRE$ : NOMBRE$(40)=" "
720 PRINT : PRINT : INPUT CALLE$ : CALLE$(40)=" "
730 PRINT : PRINT : INPUT CIUDAD$ : CIUDAD$(40)=" "
740 PRINT : PRINT : INPUT PROV$ : PROV$(40)=" "
750 PRINT : PRINT : INPUT TEL$ : TEL$(40)=" "
760 PRINT : PRINT : PRINT "SON CORRECTOS LOS DATOS ? (SI/NO) ";:
    OPEN # 1, 4, 0, "K:" : GET #1, T : CLOSE #1 : T$=CHR$(T)
770 IF T$="S" OR T$="s" THEN L=2
780 NEXT L
790 RETURN
800 REM SUBRUTINA INTRODUCIR NUEVO REGISTRO
810 NOMBRE$=B$ : CALLE$=B$ : CIUDAD$=B$ : PROV$=B$ : TEL$=B$
820 GOSUB 670 : REM ENTRADA POR TECLADO
830 GOSUB 480 : REM MODIFICAR NOMBRE PARA INDICE
840 N$(P*40-39, P*40)=NOM$ : N$(4001)=" "
850 TRAP 890 : OPEN #2, 9, 0, "D : AGENDA.DAT"
860 NOTE #2, SECT, BYTE : INDICE(P, 0)=SECT : INDICE(P, 1)=BYTE
870 PRINT #2; NOMBRE$ : PRINT #2; CALLE$ : PRINT #2; CIUDAD$ :
    PRINT #2; PROV$; PRINT #2; TEL$
880 CLOSE #2 : P=P+1 : TRAP 40000 : RETURN
890 TRAP 40000 : CLOSE #2 : OPEN #2, 8, 0, "D : AGENDA.DAT" : GOTO 860 :
    REM CREAR ARCHIVO SI NO EXISTIA YA
900 REM SUBRUTINA LISTAR REGISTROS
910 FOR IND=1 TO P-1
920 CLAVE=IND : GOSUB 540
930 PRINT : PRINT " PULSE UNA TECLA PARA SEGUIR ";: OPEN #1, 4, 0,
    "K:" : GET #1, T : CLOSE #1
940 NEXT IND : RETURN
950 REM SUBRUTINA MODIFICAR REGISTRO
960 PRINT : PRINT "INTRODUZCA NOMBRE A MODIFICAR" : INPUT NOMBRE$
970 GOSUB 370 : IF CLAVE=0 THEN RETURN : REM BUSCAR REGISTRO A
    MODIFICAR
980 GOSUB 690 : REM ENTRADA DE DATOS POR TECLADO
990 GOSUB 480 : N$(CLAVE*40-39, CLAVE*40)=NOM$ : N$(4001)=" "
1000 OPEN #2, 12, 0, "D : AGENDA.DAT"
1010 POINT #2, INDICE(CLAVE, 0), INDICE(CLAVE, 1)
1020 PRINT #2; NOMBRE$ : PRINT #2; CALLE$ : PRINT #2; CIUDAD$ :
     PRINT #2; PROV$ : PRINT #2; TEL$
1030 CLOSE #2 : RETURN
1040 REM SUBRUTINA ELIMINAR REGISTRO
1050 PRINT : PRINT "INTRODUZCA NOMBRE A ELIMINAR" : INPUT NOMBRE$
1060 GOSUB 370 : IF CLAVE=0 THEN RETURN : REM BUSCAR REGISTRO A
     MODIFICAR
1070 FOR L=CLAVE TO P-1
1080 N$(L*40-39, L*40)=N$((L+1)*40-39, (L+1)*40) : N$(4001)=" "
1090 INDICE(L, 0)=INDICE(L+1, 0) : INDICE(L, 1)=INDICE(L+1, 1)
1100 NEXT L : P=P-1 : RETURN
1110 REM SALIDA DEL PROGRAMA
1120 IF P=1 THEN GRAPHICS 0 : BUCLE=2 : RETURN
1130 OPEN #3, 8, 0, "D : INDICE.DAT"
1140 PRINT #3; P : FOR L=1 TO P-1 : NOMBRE$=N$(L*40-39, L*40) :
     PRINT #3; NOMBRE$ : NEXT L
1150 FOR L=1 TO P-1 : PRINT #3; INDICE(L, 0) : PRINT #3; INDICE(L,1) :
     NEXT L
1160 CLOSE #3 : GRAPHICS 0 : BUCLE=2 : RETURN
```

De inmediato se retornará al programa principal para comenzar el bucle que visualiza el menú y solicita al usuario que elija una opción.

Este programa pretende tan sólo ilustrar claramente el manejo de archivos de acceso aleatorio, por lo que en ciertos aspectos está simplificado. Con su estructura actual —reflejada en el listado adjunto—, resulta ya perfectamente útil en la práctica; no obstante, su máximo rendimiento se obtendrá al introducir el usuario sus propias mejoras, adaptándolo a sus necesidades específicas. Estas modificaciones pueden concretarse, por ejemplo, en permitir el acceso por otros campos diferentes al de nombre, o ampliar la capacidad de la agenda a más de 100 registros.

ADA (y 2)

Características del lenguaje ADA

En el capítulo anterior se observó que ADA y Pascal tenían muchas cosas en común. Los próximos párrafos confirmarán de nuevo esta idea, a la vez que esbozarán las dos características fundamentales que hacen que el ADA se salga de lo corriente.

ESTRUCTURAS DE DATOS

Todo lenguaje estructurado que se precie —y el ADA lo es— debe permitir al programador representar en su programa los datos de la realidad de una forma fiel y cómoda de manejar. Esto irá en beneficio de una mayor corrección y facilidad a la hora de realizar posibles cambios en los programas.

Al igual que en el Pascal, en ADA es posible manejar ARRAYs y RECORDs, y con estos últimos construir listas encadenadas y árboles binarios. Así, por ejemplo, la siguiente línea:

B: array (1..100) of INTEGER;

declara un vector (B) de cien elementos enteros.

ADA proporciona potentes herramientas para la inicialización de arrays sin tener que realizar bucles, manejar secciones de los mismos, e incluso hacer declaraciones de arrays cuya dimensión máxima esté sin especificar. Los arrays de caracteres se llaman "string" y se declaran como sigue:

PALABRA: STRING (1..10);

y ahora podríamos hacer cosas como:

PALABRA:="TU MICRO ";

El ADA también incluye operadores especiales para tratar los arrays de caracteres.

Los RECORDs son en ADA muy similares a como lo son en Pascal. En la figura se observa un ejemplo de declaración de un registro que se podría utilizar en la gestión de una empresa de no más de 1000 empleados; el equivalente en Pascal de la declaración de la variable "NUMERO" sería:

NUMERO: 1..1000;

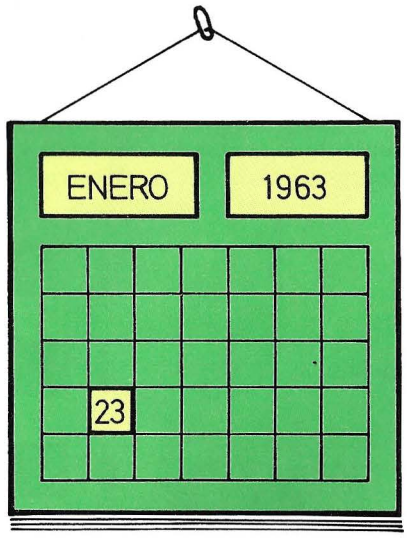
Los RECORDs pueden tener estructuras alternativas. (Los parientes pascalianos de este tipo de registros son los RECORDs variantes). El acceso a los campos del registro se realiza también a través del ope-

```
type EMPLEADO is
   record
      NOMBRE: string (1..80);
      NUMERO: range 1..1000;
      DIRECCION: string (1..160);
   end record;
```

Los registros permiten agrupar una serie de variables de tipos distintos dentro de un marco común.

```
type NOMBRE_MES is (ENE,FEB,MAR,ABR,MAY,JUN,JUL,
                    AGO,SEP,OCT,NOV,DIC);

type FECHA is
  record
    MES: NOMBRE_MES;
    DIA: INTEGER;
    AÑO: INTEGER;
  end record;

NACIMIENTO: FECHA;

NACIMIENTO:=(ENE,23,1963);
```

NACIMIENTO :=



Ejemplo de asignación directa de todos los campos de un registro.

```
procedure SUMAR_UNO (RESULTADO: out INTEGER; X in INTEGER) is
      UNO: constant INTEGER :=1;      --declaración de una constante
begin
      RESULTADO:= X + UNO;
end;

function SUMAR_UNO (X: INTEGER) return INTEGER is
      RESULTADO: INTEGER;
begin
      RESULTADO:= X + 1;
      return RESULTADO;
end;
```

Los procedimientos y funciones son las herramientas básicas que permiten estructurar los programas.

rador ".". Como muestra la correspondiente figura, podemos inicializar un registro completo en una sola sentencia, sin necesidad de tener que hacerlo campo a campo.

PROCEDIMIENTOS Y FUNCIONES EN ADA

Son equivalentes a sus homónimos del Pascal, aunque en el ADA tienen nuevas características que los/las hacen más potentes. En la figura adjunta se tiene una declaración de cada tipo.
Los parámetros que se pasan a un procedimiento pueden ser de tres tipos:

- tipo "in"

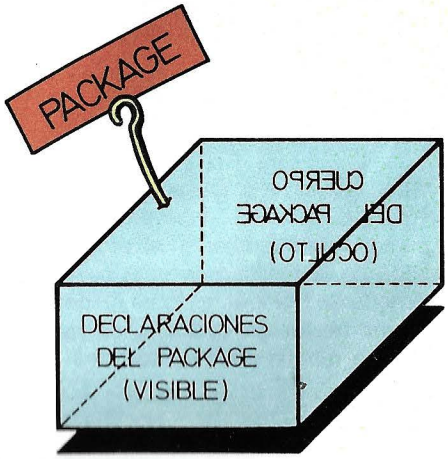
Constituyen la entrada al procedimiento. Dentro de él son como constantes y sus valores no pueden ser modificados.

- tipo "OUT"

Son la salida del procedimiento.

- tipo "in out"

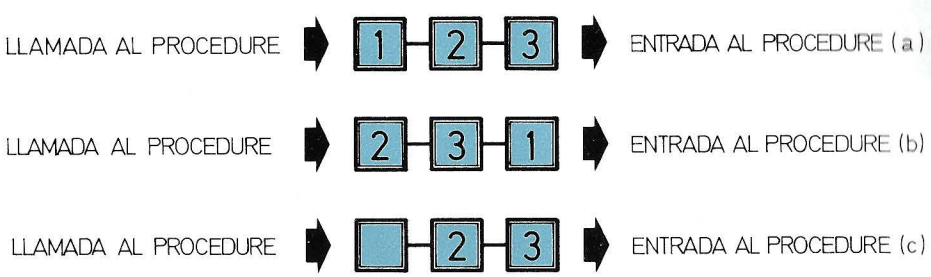
Son considerados como variables cuyos valores se pueden modificar durante la ejecución del procedimiento, y estas modificaciones quedan reflejadas a la salida del mismo.
La declaración de "UNO" que tiene lugar en el procedimiento ilustrado sería equivalente en Pascal a:
CONST UNO=1;
En cuanto a las funciones, éstas sólo admiten parámetros del tipo "in" (aunque no es necesario decirlo explícitamente como ocurre en el ejemplo). El uso de las funciones es análogo al que se hace en Pascal; apareciendo sus nombres en expresiones sintácticamente correctas y cuidando la compatibilidad del tipo devuelto.
Pero en ADA los procedimientos y funciones dan todavía más que hablar. Una llamada a "SUMAR_UNO" podría consistir en:
SUMAR_UNO(INCREMENTADO, NUMERO);
la cual se denomina "llamada posicional": los parámetros de la llamada se sustituyen en la declaración en el mismo orden.
La alternativa que ofrece el ADA consiste en hacer:
SUMAR_UNO(X=>NUMERO, RESULTADO=> INCREMENTADO);
Con ello especificamos el paso de los parámetros explícitamente, rompiendo la norma posicional de la llamada anterior.
Además, los parámetros del tipo "in" pueden tomar valores por defecto. Si cambiáramos la cabecera de "SUMAR_UNO" por:
procedure SUMAR_UNO(X: in INTEGER :=5; RESULTADO: out INTEGER)
una llamada podría consistir ahora en:
SUMAR_UNO(INCREMENTADO);
y al acabar la ejecución del procedimiento la variable "INCREMENTADO" tendría el valor 6 (ver figura).



El usuario de un "package" sólo puede acceder a la parte visible del mismo. Su cuerpo (body) queda oculto por las declaraciones.

LOS PAQUETES (PACKAGES)

En el capítulo precedente se mencionó que las aplicaciones para las que está concebido el lenguaje ADA pueden ser programas de enorme longitud.
En esta categoría de programas, la posibilidad de estructurarlos a base de procedimientos y funciones puede resultar insuficiente, ya que su complejidad puede des-



Los parámetros destinados a los procedimientos del ADA pueden ser pasados tal y como es costumbre hacerlo en PASCAL (a), cambiando su orden (b), o incluso omitiendo algún parámetro en la llamada si en la declaración éste tiene algún valor por defecto (c).

```
package ESTADISTICA is

   PI: constant FLOAT :=3.141592654;
   E: constant FLOAT :=2.718281828;

   procedure FACTORIAL (N: in INTEGER; FACT: out INTEGER);

   procedure DISTR_NORMAL (X: in FLOAT; VALOR: out FLOAT);

end;
```

Ejemplo de declaración de la parte visible de un "package".

bordar la capacidad de "modularización" que proporcionan las citadas estructuras.
En ADA existe otra forma —no excluyente con la conocida— de estructurar los programas: el "package". En uno de ellos se pueden incluir una serie de tipos, datos, procedimientos y funciones que lógicamente estén relacionados. En términos generales, un package ofrece un servicio; y como cualquier servicio, puede precisar de otros datos y servicios internos que no son visibles al usuario del servicio original. Un package se compone de dos partes: la parte visible y el propio cuerpo del package. Sólo la primera será accesible a la persona que quiera utilizarlo.
Suponga que se trata de redactar un programa que necesita algunas funciones matemáticas que ADA no proporciona directamente. Para empezar, estructuraremos la parte principal en forma de package; de esta forma nuestro trabajo será útil a otra persona que pueda necesitarlo en el futuro. En la figura se observa el posible aspecto de la parte visible de nuestro package; es decir, aquella que podremos utilizar en el programa. En ella realizamos la declaración de dos constantes y dos procedimientos: uno para el cálculo del factorial de un número y otro para calcular la función de distribución normal estándar. Estos elementos será posible emplearlos en aquellos programas a los que se indique que utilicen el package "ESTADISTICA", como se verá más adelante.
En una de las figuras aparece tan sólo la parte visible. En el cuerpo (body) del package será preciso realizar la declaración completa de los procedimientos parcialmente declarados en la parte visible. Ello se hace tal como ilustra la figura siguiente. Lo comentado en la primera parte acerca del I_O_PACKAGE es aplicable a cualquier otro paquete. Si en nuestro programa quisiéramos utilizar los datos o procedimientos definidos en el paquete "ESTADISTICA" añadiríamos al principio del mismo:

```
use ESTADISTICA;
```

y de esta forma eliminaríamos la necesidad de tener que referenciar a "FACTORIAL" así:

```
ESTADISTICA.FACTORIAL(X,Y);
```

Ahora, bastaría una llamada del tipo:

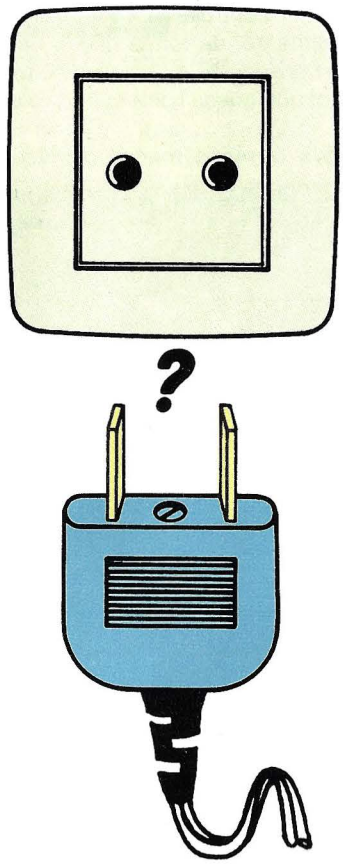
```
FACTORIAL(X,Y);
```

PROCESAMIENTO EN PARALELO

La posibilidad de controlar dos o más eventos cuyas acciones ocurren simultá-

La portabilidad del ADA

Los lenguajes a los que ADA va a sustituir son utilizados sobre ordenadores muy distintos, con diversos tipos de procesadores y mecanismos de entrada/salida también diferentes. Invertir la cantidad de recursos que se han invertido sin tener en cuenta este hecho sería una locura. Por ello, hay que añadir a las características ya comentadas del ADA su portabilidad; esto es, la posibilidad de que se puedan ejecutar en distintos ordenadores programas escritos en este lenguaje.
Normalmente, los mayores problemas de portabilidad se centran en la distinta configuración de los mecanismos de entrada/salida de los ordenadores. La posibilidad de estructurar los programas en paquetes hace que, en la mayoría de las ocasiones, baste con modificar el I_O_PACKAGE para adaptar el programa al ordenador en cuestión.



```
package body ESTADISTICA is

   procedure FACTORIAL (N: in INTEGER; FACT: out INTEGER) is
         -- declaraciones de "factorial"
   begin
         -- sentencias de "factorial"
   end;

   procedure DISTR_NORMAL (X: in FLOAT; VALOR: out FLOAT) is
         -- declaraciones de "distr_normal"
   begin
         -- sentencias de "distr_normal"
   end;

begin
      -- puede ser necesaria alguna sentencia para inicializar
      -- el package, ESTAS se introducen aquí y se ejecutan
      -- cuando se crea el propio cuerpo del package.
end;
```

Este puede ser el cuerpo del paquete al que se hace referencia en la figura anterior.

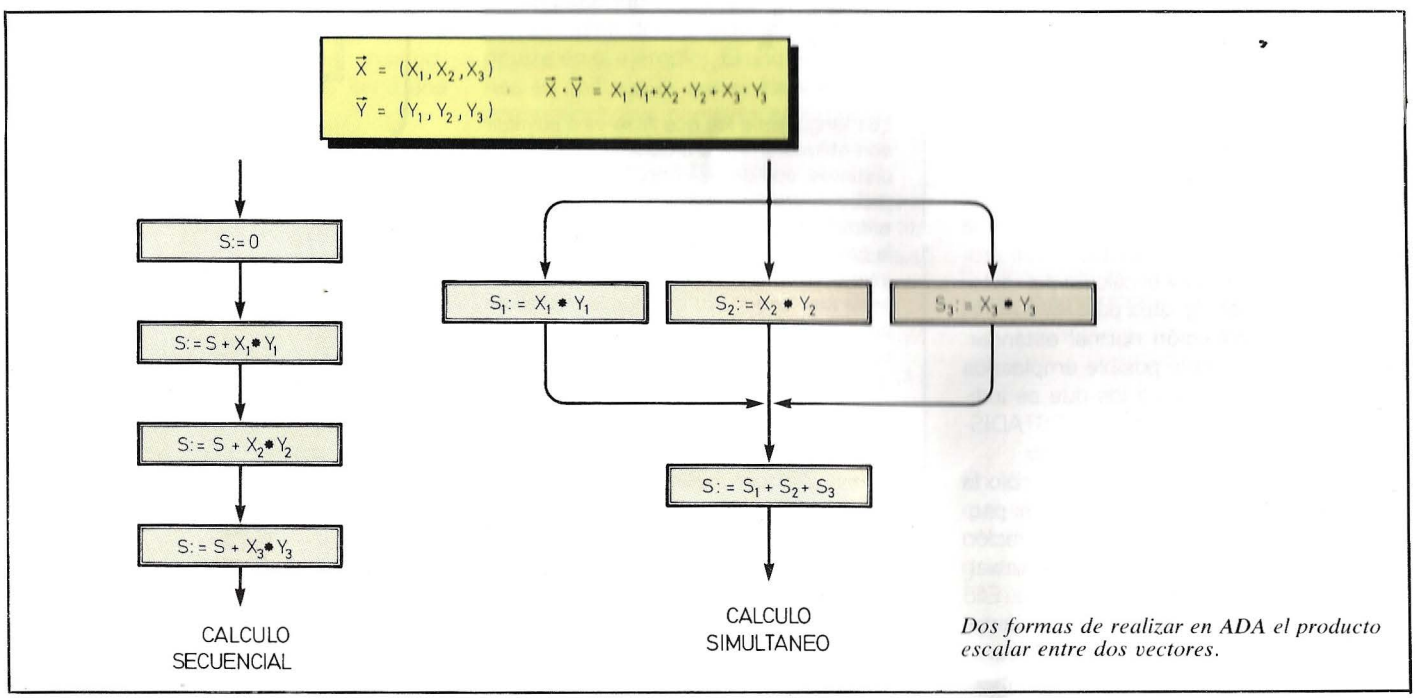
neamente tiene gran importancia en el tipo de trabajos para los que ADA está pensado. Suponga que se están dirigiendo dos misiles hacia un objetivo concreto. En esta situación sería muy interesante poder controlar los dos misiles simultáneamente, de forma que si uno de ellos ha conseguido su objetivo se pueda desviar al que queda hacia un nuevo objetivo.

Bajando a terrenos menos conflictivos, suponga ahora que se desea calcular el producto escalar de dos vectores de tres coordenadas (ver figura). La única forma que conocemos hasta ahora para realizar este cálculo es efectuar las sumas y los productos *secuencialmente,* como refleja la primera zona de la mencionada figura. Sin embargo, en ADA puede hacerse que los productos de las coordenadas se realicen *simultáneamente* y, una vez obtenidos estos valores, sumarlos en el siguiente paso. El beneficio en tiempo de ejecución para un vector de tres coordenadas no compensa la complejidad adicional de realizar un procesado en paralelo; pero si el vector fuera de 1000 coordenadas, un enfoque como el descrito puede resultar muy ventajoso.

Estas posibilidades "software" que nos proporciona el ADA están basadas en los adelantos hardware de los últimos años. Un ordenador que pueda soportar eficientemente el tipo de cálculo expresado más arriba ha de poseer más de una CPU, de forma que cada una de ellas se ocupe, con simultaneidad, de un proceso.

El uso de procesos paralelos para la resolución de un problema trae consigo a menudo muchos quebraderos de cabeza. Volviendo a nuestro ejemplo, una CPU no tardará lo mismo en realizar el producto 1×1 que el producto 1543.43×12E5, por lo que la necesidad de sincronizar las diversas tareas encomendadas a cada procesador es una necesidad evidente. Además, hay que contemplar la posibilidad de que se comentan errores en alguna CPU mientras se están realizando los cálculos, lo que complica aún más tanto el diseño del hardware como del software.

En ADA los procesos paralelos se expresan mediante "task" (tareas), cuya apariencia es muy similar a la de los packages. Las "tasks" están acompañadas por una serie de sentencias que facilitan su sincronización.

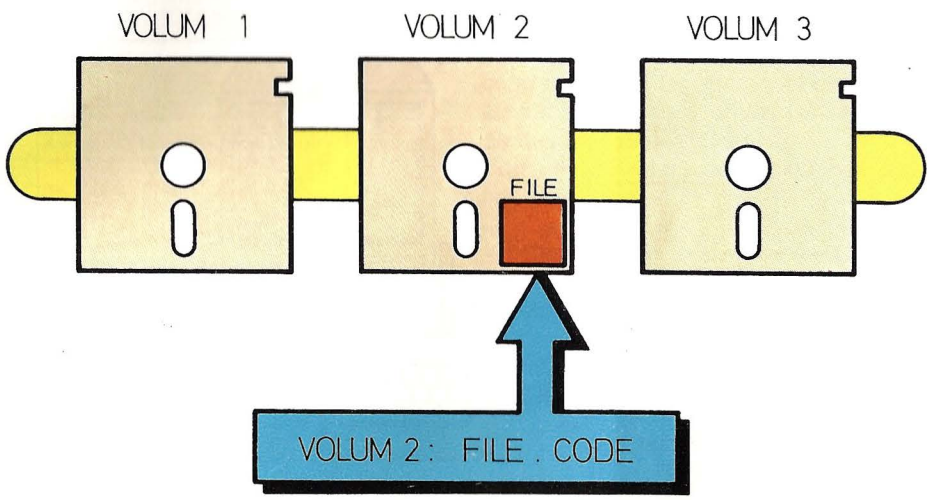


Dos formas de realizar en ADA el producto escalar entre dos vectores.

UCSD p-System

La gestión de ficheros con el FILER

La organización de la información con el UCSD p-system se realiza de acuerdo a una filosofía muy simple pero verdaderamente eficaz: todo objeto que sea capaz de almacenar información (en forma de ficheros), o bien de recibirla y enviarla, es etiquetado como un volumen. El flujo de información se reduce, en definitiva, a intercambios de datos entre volúmenes.
Así pues, una impresora es un volumen que envía información al exterior del sistema, mientras que el teclado es también un volumen cuya función es la de introducir datos en el sistema.
Los disquetes se comportan igualmente como volúmenes, dado que su actividad (almacenar información) está comprendida en la definición de volumen. A cada disco se asigna un nombre de volumen con el cual puede ser referenciado unívocamente.



La identificación de un fichero se consigue al citar el volumen al que pertenece (VOLUM 2), el nombre del fichero (FILE) y su tipo (CODE).

NOMENCLATURA

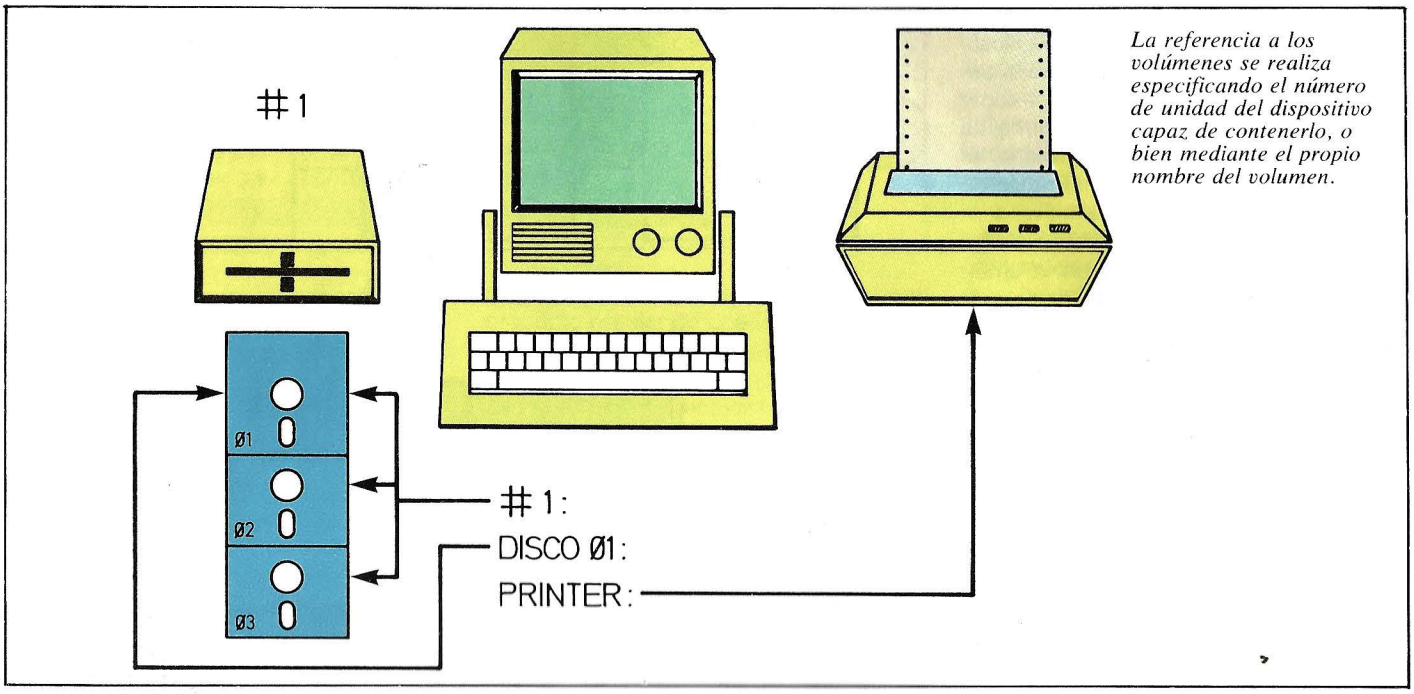
Para identificar la información contenida en los distintos volúmenes del sistema, deben existir ciertos convenios con los que se pueda acceder a la información por sus nombres. Así se evitará el grave inconveniente que supondría la exigencia de conocer las localizaciones exactas en donde reside la información, a través de los diferentes sectores y pistas.
Los ficheros se reconocen por su nombre y por el nombre del volumen en el cual están contenidos, actuando como separador de los dos identificativos el signo dos puntos (:).

La impresora no es considerada como una entidad aparte por el UCSD p-System; sencillamente, es considerada como un volumen más sobre el que se vuelca información.

Los diversos volúmenes se identifican de dos formas. La primera consiste en asignar un nombre de volumen con el que éste pueda ser referenciado, mientras que la segunda se basa en la asignación de un número de unidad precedido por el signo #. Este segundo caso se diferencia del primero en que, al contrario que el nombre de volumen, el número de unidad no identifica a un solo volumen, sino que aplica el volumen residente en ese instante en el dispositivo indicado por el número de unidad.

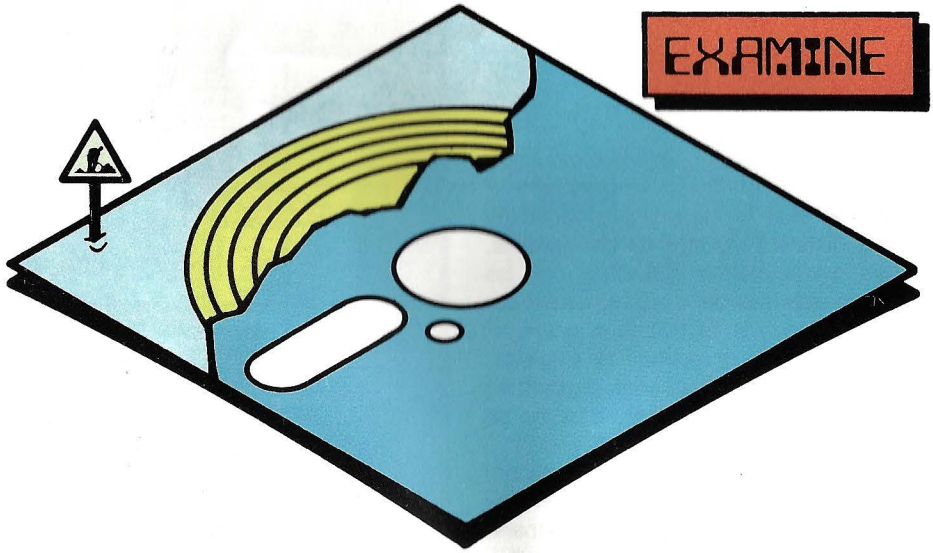
ACCESO A FICHEROS

Cuando se especifica el nombre de un fichero, el sistema operativo lleva a cabo las siguientes operaciones para leer los datos contenidos en el fichero. En primer lugar, el sistema operativo revisa las diferentes unidades de disco disponibles para encontrar el nombre de volumen correcto. A continuación, si se ha encontrado el volumen indicado, se consulta el directorio del mismo para comprobar que existe en él un nombre de fichero idéntico al especificado. Acto seguido, si se ha comprobado la existencia del nombre del fichero en el directorio, el sistema operativo lee la información contenida en el directorio, conociendo de este modo en qué lugares del volumen está almacenada la información del referido fichero; el propio sistema operativo se encargará de leer

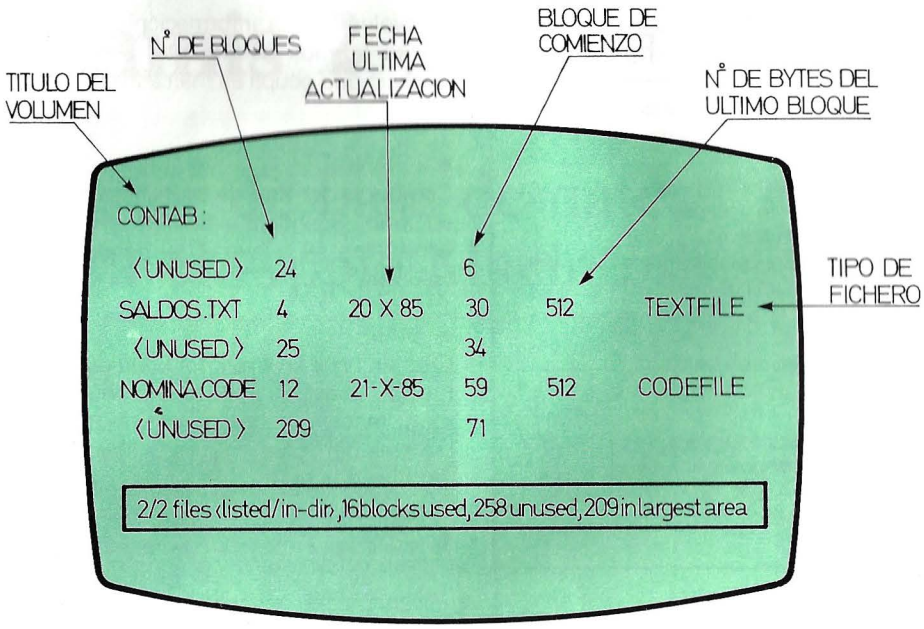


La referencia a los volúmenes se realiza especificando el número de unidad del dispositivo capaz de contenerlo, o bien mediante el propio nombre del volumen.

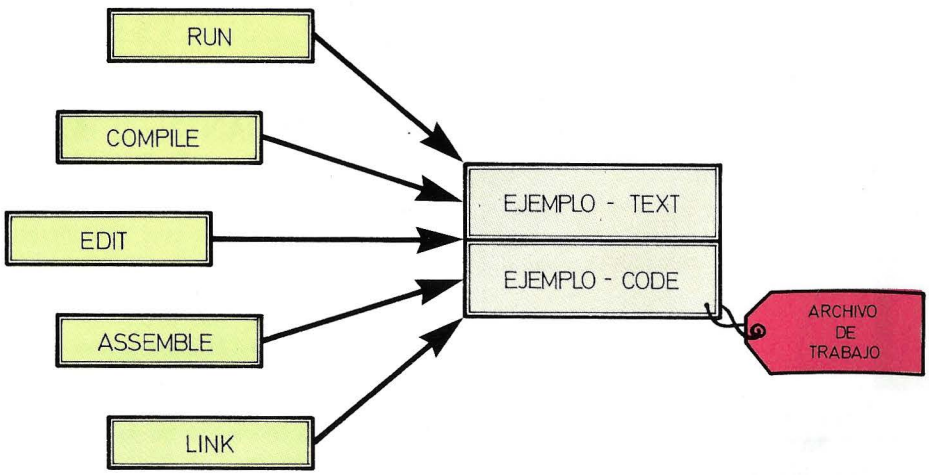
De acuerdo a estas reglas serían válidas las siguientes especificaciones de ficheros:
VENTAS : ENERO.TEXT ó
4: ENERO.TEXT
Dentro de los volúmenes cabe distinguir dos tipos: el volumen de arranque o "boot volume", diferenciado por representarse con un asterisco (*) y cuyo contenido hace posible el arranque del sistema, y el volumen en curso, o "prefix volume", que es el que está activo en un momento dado.
Como se ha podido comprobar en el ejemplo, los nombres de los ficheros admiten extensiones (coletilla a continuación del punto) con las que se realiza una detección más rápida del contenido de los ficheros. De igual forma, la referencia a más de un fichero a la vez es posible por medio de referencias ambiguas (wildcards).



El sistema operativo puede regenerar ciertas zonas del disquete que no estén muy dañadas, aplicando el comando EXAMINE.



El comando EXTENDED DIRECTORY proporciona una información completa acerca del volumen en curso.



Cuando un fichero va a ser utilizado en múltiples ocasiones, es un buen método etiquetarlo como archivo de trabajo. De esta forma no será preciso repetir, cada vez que se escriba un comando, el nombre del fichero sobre el que debe actuar.

la información residente en las posiciones oportunas.

EL FICHERO DE TRABAJO

Dentro de los distintos ficheros que cohabitan en los soportes de memoria, cabe mencionar a uno de ellos que si bien no es diferente de los demás en cuanto su estructura, sí lo hace en cuanto a su función. El fichero al que nos referimos es el denominado fichero de trabajo, en el cual reside una copia del fichero que se ha especificado para su acceso. Su empleo resulta muy útil cuando se hagan operaciones sobre un mismo fichero, puesto que no habrá necesidad de indicar en los comandos sobre qué fichero deben ejecutarse. En efecto, si no se indica el nombre de ningún fichero, el sistema toma por defecto el fichero de trabajo.

EL FILER

El tratamiento de los ficheros en UCSD p-system se efectúa bajo el monopolio de un programa, denominado FILER, que centraliza todas las operaciones a realizar en los volúmenes y en los ficheros residentes en dichos volúmenes.
Con el FILER se pueden realizar operaciones como éstas:

— Crear o borrar el fichero de trabajo.
— Cambiar el nombre, copiar y borrar ficheros.
— Copiar un fichero de un volumen a otro.
— Imprimir un fichero.
— Identificar áreas defectuosas en un disquete.

La ejecución del FILER se logra al elegir del menú principal de comandos la opción que lo identifica (F). De inmediato aparecerá otro menú que refleja los distintos comandos disponibles en el entorno del FILER. Estos son los siguientes:

- VOLUMES

Muestra los números o nombres de los volúmenes que están en línea en ese momento, así como los nombres de los volúmenes de arranque y en curso.

- PREFIX

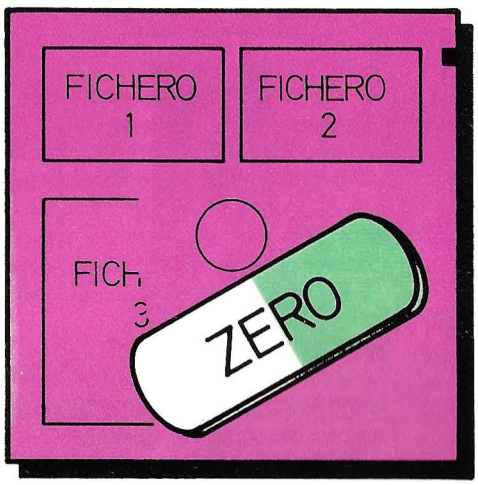
Muestra o cambia el nombre del volumen en curso. Cuando se arranca el sistema coinciden el volumen en curso y el volumen de arranque, siendo necesario ejecutar este comando si se desea acceder a otros volúmenes.

- EXTENDED DIRECTORY

Muestra el contenido del directorio de un volumen. Especifica el nombre del volumen, así como los nombres de los ficheros en él contenidos, el número de bloques que ocupa cada fichero, sus fechas de creación o última modificación, el número del bloque en donde comienza cada fichero, la cantidad de bytes contenidos en el último bloque y, por último, el tipo de cada uno de los ficheros.

RESUMEN DE COMANDOS DEL FILER

BAD BLOCKS	Inspecciona los bloques de un disco, identificando los defectuosos.
CHANGE	Cambia el nombre de un fichero o de un volumen.
DATE	Define la fecha del sistema.
EXAMINE	Intenta reparar los bloques defectuosos de un disco.
EXTENDED DIRECTORY	Lista los nombres de los ficheros de un volumen.
GET	Designa a un fichero como fichero de trabajo.
KRUNCH	Reordena el contenido de un volumen, agrupando espacios libres.
LIST DIRECTORY	Lista los nombres de los ficheros de un volumen.
MAKE	Crea un nuevo fichero.
NEW	Limpia el fichero de trabajo.
PREFIX	Designa un nuevo volumen como volumen-prefijo.
QUIT	Termina Filer.
REMOVE	Borra ficheros de un volumen.
SAVE	Graba con un nuevo nombre el fichero de trabajo.
TRANSFER	Copia un fichero en otro volumen o dispositivo.
VOLUMES	Lista los volúmenes existentes.
WHAT	Refleja el estado del fichero de trabajo.
ZERO	Inicializa el directorio de un volumen.



Con el comando ZERO se borra toda la información contenida en un volumen.

- LIST DIRECTORY

Es una versión reducida del comando anterior. Muestra tan sólo los nombres de los ficheros, su tamaño en bloques y su fecha de creación o última actualización.

- GET

Designa a un fichero como fichero de trabajo, sin necesidad de cambiar su nombre ya que basta con etiquetarlo como tal. Su utilidad se aprecia cuando se trabaja frecuentemente sobre un mismo fichero.

- WHAT

Revela la existencia o inexistencia del fichero de trabajo; en el caso de que exista, muestra su nombre y estado.

- NEW

Borra el fichero de trabajo del sistema, es decir, borra los ficheros de trabajo actuales o los ficheros SYSTEM.WRK.TEXT y SYSTEM.WRK.CODE, que son los ficheros de trabajo existentes en el volumen de arranque y que son tomados por defecto al iniciar la sesión.

- SAVE

Cambia de nombre los ficheros SYSTEM.WRK.TEXT y SYSTEM.WRK.CODE, permitiendo así guardar el contenido del fichero de trabajo.

- REMOVE

Borra un nombre de fichero del directorio del volumen. La información del fichero no es borrada físicamente, sino que el espacio que ocupa es marcado como utilizable.

- CHANGE

Cambia el nombre de un fichero o de un volumen. Sustituye el nombre antiguo del fichero por el nuevo en el directorio del volumen.

- MAKE

Crea un fichero al insertar un nuevo nombre de fichero en el directorio de un volumen; reserva un determinado número de bloques del volumen para la constitución del nuevo fichero.

- ZERO

Inicializa el directorio y el nombre del volumen; es decir, crea un volumen en blanco. Este paso previo es necesario para poder poblar un volumen con ficheros.

- TRANSFER

Permite mover información con las siguientes alternativas: copiar un fichero en otro volumen, imprimir un fichero, mostrar un fichero por pantalla, realizar copias de respaldo o de seguridad de volúmenes enteros, etc.

- KRUNCH

El nombre de este comando refleja perfectamente su tarea, la cual se concreta en reordenar el volumen comprimiendo los datos de manera que las zonas sin utilizar se reúnan en un solo bloque. Ello facilitará el acceso a la información.

- BAD BLOCKS

Comprueba la existencia de áreas defectuosas en el disquete, ocasionadas por el continuo uso de la superficie magnética que lo recubre.

- EXAMINE

Sirve para examinar los bloques defectuosos localizados con el comando anterior. El modo de operar es el siguiente: se lee un bloque defectuoso, se escribe y, a continuación, se vuelve a escribir; esta operación se realiza un determinado número de veces, comprobándose si lo escrito es igual a lo leído.

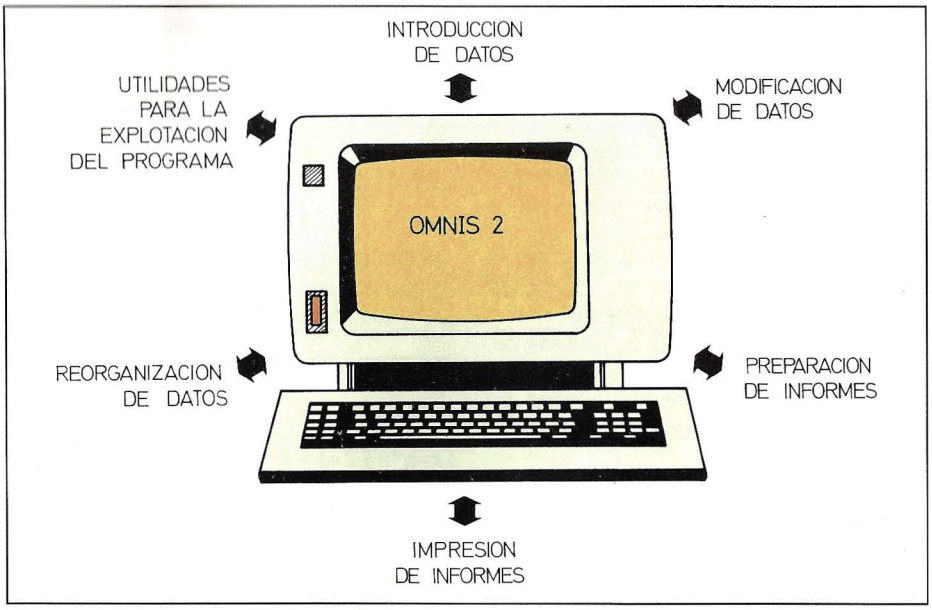
- DATE

Fija la fecha para que sea posible marcar los ficheros con una fecha de creación o última actualización.

Omnis 2 (1)

Gestor de base de datos para el Apple Macintosh

A lo largo de los dos próximos capítulos se analizará una nueva base de datos, concretamente, se trata del programa OMNIS en su versión 2 (ya está anunciada la versión 3, aunque aún no se encuentra disponible en el mercado). Las características generales de OMNIS 2 están orientadas a facilitar la gestión de información. Para ello, como se verá más adelante, permite definir pantallas para la entrada de datos, introducir y/o modificar información en ficheros, crear e imprimir informes, reorganizar los datos almacenados y, en definitiva, utilizar el ordenador para sustituir a los tradicionales archivadores de fichas manuscritas.



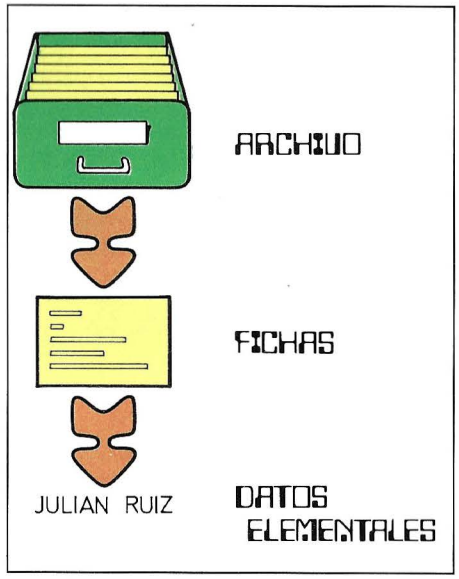
Las funciones asumidas por el OMNIS 2 son las tradicionales en cualquier base de datos. En la figura se destacan las operaciones más importantes que el programa ofrece a sus usuarios.

CARACTERISTICAS TECNICAS DE OMNIS 2

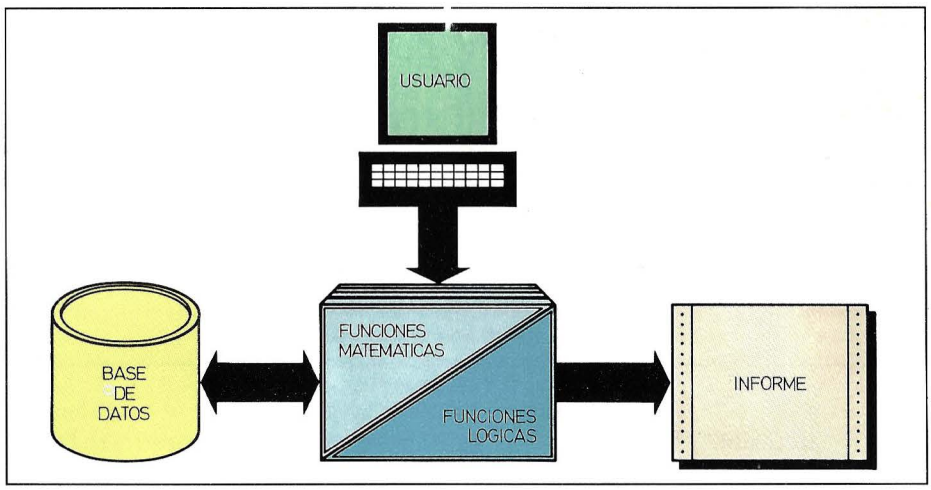
Puede definirse al programa OMNIS 2 como un sistema para la gestión de información. El elemento básico de trabajo para el programa lo constituyen los *archivos* en donde se almacenarán los datos referentes a entes homogéneos. Al respecto, el programa permite que el usuario defina sus propios archivos de aplicación, como pueden ser: archivo de direcciones, teléfonos de clientes y proveedores, archivo de libros y revistas, archivo de clientes... Y, así, tantos otros como el propio usuario desee.

Además de los anteriores archivos de aplicación, el programa necesita uno para su uso específico; se trata del archivo denominado BIBLIOTECA. En él se almacenan todos los datos que OMNIS 2 necesita para reconocer y ejecutar las órdenes del usuario.

Continuando con la estructura de la información utilizada por este programa, el siguiente concepto es el de *ficha*. En ella se reflejará la información correspondiente a cada uno de los entes del archivo al que pertenece. Por último, en un tercer nivel, nos encontramos con los *datos elementales* que representan las distintas anotaciones puntuales que se almacenarán en cada ficha. OMNIS 2 limita a 120 el número máximo de datos incluibles en una ficha. La búsqueda de información se realizará a partir de algunos de los datos contenidos en las fichas; a estos datos ele-



La estructura lógica de la información utilizada por el OMNIS 2 coincide plenamente con la utilizada en los ficheros convencionales: archivo, fichas y datos elementales.



Para la reproducción de informes a partir de la base de datos se pueden utilizar funciones matemáticas o lógicas. Estas permitirán el uso de nuevos datos en función de otros incluidos en la base de datos.

mentales se les denomina índices. El usuario puede especificar un máximo de 10 índices por los que se localizará preferentemente la información; no obstante, las búsquedas se pueden realizar por datos que no sean de tipo índice, aunque, en tal caso, el tiempo requerido será superior.

Para calcular nuevos datos a partir de los ya existentes en un archivo, OMNIS 2 incorpora una serie de funciones matemáticas y lógicas. Otra característica importante de este producto consiste en la posibilidad de diseñar sobre la pantalla hasta 12 páginas en las que se visualizarán los datos.

La seguridad de los datos se puede garantizar mediante "palabras secretas"; de esta forma, si el usuario del programa solicita visualizar o modificar un dato protegido, éste sólo le será mostrado si es capaz de teclear correctamente la palabra secreta.

OMNIS 2 también permite realizar ciertas operaciones que podríamos clasificar como de "pseudoproceso de textos"; tal es el caso de la producción de cartas personalizadas, la emisión de etiquetas adhesivas, la preparación de facturas, etc.

OMNIS 2 SOBRE MACINTOSH

Dado que el ordenador más utilizado para explotar este programa es el Apple Macintosh, todas las características sobre su funcionamiento se referirán a la versión preparada para dicho ordenador.

Además del ratón, el cual puede utilizarse para seleccionar entre las opciones que OMNIS 2 presenta en forma de menús, existen algunas teclas que sirven para controlar el programa.

1. TECLA DE COMANDO

Sirve para indicar que el carácter pulsado simultáneamente tendrá un significado especial; es decir, no formará parte de una palabra sino que desencadenará la ejecución de un comando. Por supuesto, el usuario puede elegir entre ejecutar un comando pulsando esta tecla y la letra que lo identifica, o seleccionándolo directamente sobre el menú.

2. TECLA DE RETROCESO

Como su propio nombre indica, se utiliza para corregir errores tipográficos cuando se desea modificar o borrar caracteres escritos previamente.

3. TECLA DE TABULACION

Su principal misión consiste en facilitar la entrada de datos. Al ser accionada provoca un salto en el cursor, de forma que éste queda situado en la primera posición del dato que sigue el último tecleado.

4. TECLA RETURN

Como es tradicional en prácticamente todos los ordenadores, y en todos los programas, pulsar la tecla RETURN equivale a dar entrada a la confirmación del comando o dato tecleado previamente.

En cuanto a la distribución que hace OMNIS 2 de la pantalla del ordenador, cabe destacar que siempre se visualizan en ella las opciones disponibles en cada momento; de esta forma, el usuario tendrá

Si el administrador/propietario de la base de datos la protege total o parcialmente, la única forma de acceder a la información protegida será conociendo la palabra secreta de acceso.

suficiente información sobre lo que puede o no puede hacer en cada instante.

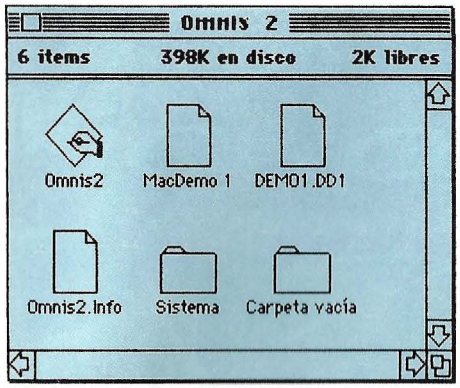
PASOS TIPICOS EN UNA SESION DE TRABAJO

- Paso-1 (Preparación)

Obviamente, antes de realizar ninguna operación, deben formatearse los disquetes en los que se almacenará la información de la base de datos.

- Paso-2 (Carga del programa)

En este caso se trata de pasar el programa que reside normalmente en un soporte externo a la memoria principal del ordenador. Para ello es suficiente con introducir en la correspondiente unidad el disquete que contiene el programa. En menos de un minuto aparecerá el menú inicial del programa, sobre el que el usuario podrá optar por entrar en el entorno de trabajo del OMNIS 2.



Reproducción de la llave de entrada a OMNIS 2 (icono, en terminología Macintosh) que aparece sobre la pantalla del ordenador al ser cargado el programa en la memoria principal.

- Paso-3 (Selección de la base de datos)

El usuario, mediante un nuevo menú, debe indicar la base de datos con la que desea trabajar. Esta puede ser alguna de las ya existentes, o una completamente nueva que desee crear.

- Paso-4 (Construcción de las pantallas)

Después de haber seleccionado "Crear y Modificar" en el menú de opciones, el programa permitirá definir el formato de las pantallas necesarias para introducir y recuperar datos.

- Paso-5 (Introducción y consulta)

A partir de este momento será posible incluir nueva información en la base de datos o consultar la ya existente. Para ello, el menú de opciones ofrece la posibilidad de seleccionar un comando denominado "Introducir y Consultar".

- Paso-6 (Selección e impresión de informes)

En algunas sesiones de trabajo será necesario producir un informe escrito a partir

Máquinas de Turing: tortugas en un mundo de liebres

Está claro que a un ordenador se le pide que sea capaz de procesar instrucciones muy potentes y que su velocidad impresione. Pues bien, el inglés A. M. TURING dedicó todos sus esfuerzos a diseñar y estudiar una máquina capaz de procesar sólo instrucciones muy simples, y a una velocidad muy pequeña. A estas máquinas se las conoce universalmente como máquinas de Turing.

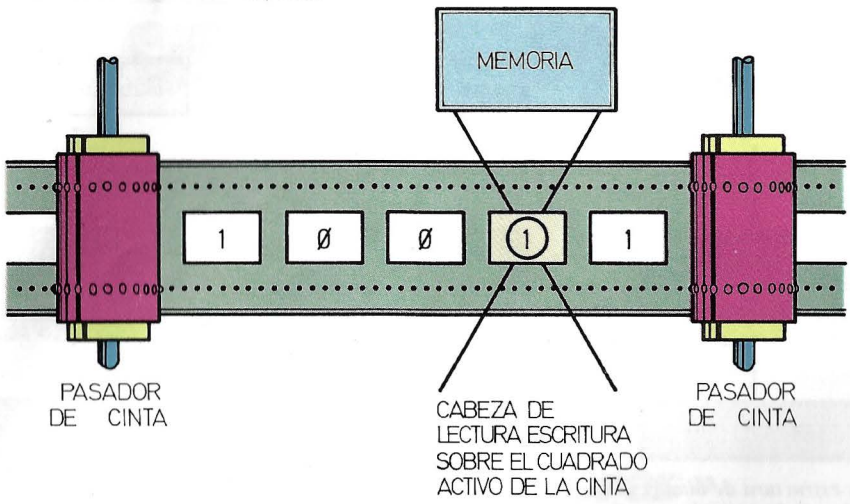
Probablemente el lector estará pensando que esto se trata de una broma. ¿Cómo se puede considerar interesante una máquina torpe y lenta? La respuesta es muy sencilla: la máquina de Turing representa la imagen más clara de lo que una máquina puede hacer. Y por añadidura, con tan sólo seis instrucciones de bajísimo nivel, son capaces de resolver problemas de gran complejidad.

Una máquina de Turing puede considerarse como un artilugio mecánico capaz de trabajar con una larguísima cinta dividida en cuadrados o casillas sucesivas, sobre los que puede escribir o leer un dígito del alfabeto binario; es decir: {0,1}. La máquina trabaja en cada instante sobre un cuadrado concreto y, a partir de él, puede hacer las siguientes operaciones:

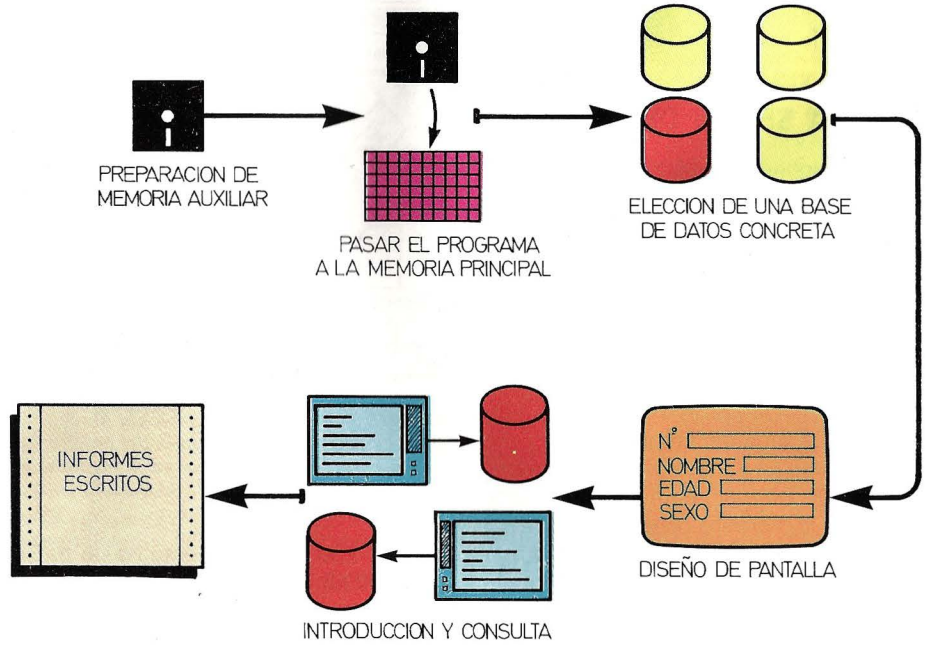
1. Escribir un "0" sobre el cuadrado o casilla.
2. Escribir un "1" sobre el cuadrado.
3. Borrar cualquiera de estos dígitos, si es que previamente estaba ya escrito.
4. Mover la cinta al cuadrado de la izquierda.
5. Mover la cinta al cuadrado de la derecha.
6. Parar.

Parece increíble, pero con una máquina de Turing dotada de este lenguaje superelemental y considerando que la cinta sea de tamaño infinito, puede llegarse a la máquina universal; es decir, a una máquina capaz de resolver cualquier problema.

Para finalizar esta breve descripción sobre las máquinas de Turing, citaremos que además de la cinta y de las instrucciones, es necesario incorporar una memoria para su buen funcionamiento. Eso sí, su capacidad no puede ser más limitada: dicha memoria tan sólo debe ser capaz de almacenar el carácter uno o el carácter cero.



de los datos almacenados en la base. OMNIS 2 ofrece al efecto el comando "Crear y Modificar Informes", éste permite la definición o modificación de las características que marcan la naturaleza del informe a obtener. En cambio, si sólo se desea imprimir un informe para el que ya se habían definido sus características en otra sesión precedente, bastará con ordenar la ejecución del comando "Imprimir Informe" desde el menú de opciones.

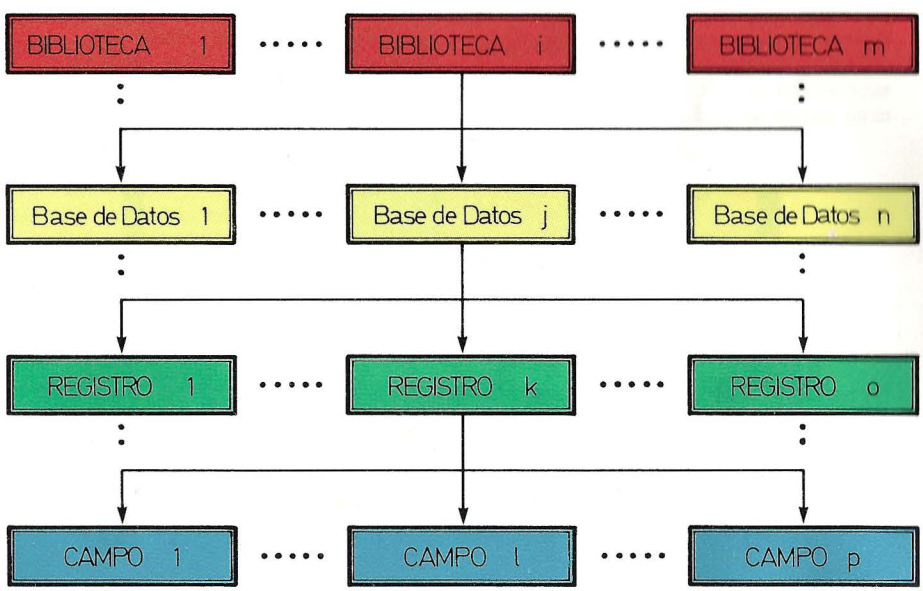


La figura sintetiza los pasos tradicionales para llevar a cabo una sesión de trabajo completa con OMNIS 2.

ESTRUCTURA DE ALMACENAMIENTO

Sin duda uno de los conceptos más importantes de toda base de datos es la estructura que utiliza para almacenar los datos. Como ya adelantábamos al hablar de las características técnicas de OMNIS 2, este programa se basa en la existencia de bibliotecas en las que se agruparán las distintas bases de datos. De esta manera, el usuario estará en condiciones de organizar todos sus archivos, incluyéndolos en la misma biblioteca si están relacionados, o en bibliotecas diferentes si no tratan temas comunes.

En cualquier caso, tanto para la creación de una biblioteca como para una base de datos, es necesario indicar un nombre que las identificará. El espacio de disco utilizado para almacenar un archivo de biblioteca nunca será demasiado grande; en un caso exagerado la biblioteca puede llegar a tener un máximo de 50 bloques y teniendo en cuenta que cada bloque está formado por 512 bytes (es decir 0,5 K), el tamaño final de este tipo de archivo oscilará entre 0,5 y 25 Kbytes.



La estructura de almacenamiento utilizada por OMNIS 2 favorece tanto la organización lógica del usuario como la organización física de los datos.

El objeto de utilizar una organización estructurada en bibliotecas y bases de datos, no sólo se fundamenta en la organización lógica de la información desde el punto de vista del usuario; también existe un motivo de mucho peso para optar por este tipo de estructura: el acceso a los datos será muy rápido.

En el fondo, cuando alguien utiliza una base de datos, lo que desea es aumentar la rapidez en la búsqueda de la información. Existen además otras ventajas adicionales, como es la comodidad, la seguridad, etc. Pero, en cualquier caso, si los tiempos necesarios para buscar un dato no se vieran reducidos, la utilidad de este tipo de procesos se vería muy depreciada.

Para finalizar este primer capítulo dedicado a OMNIS 2, intentaremos demostrar que su estructura de almacenamiento favorece la rápida localización de los datos. La biblioteca contiene una serie de datos relativos a la situación física de las bases de datos y a ubicación en el soporte de almacenamiento. De esta forma, cuando el usuario activa una base de datos, el programa transfiere los parámetros que la definen desde la biblioteca hasta la memoria principal. A partir de ese momento, cuando el usuario solicite localizar un dato, OMNIS 2 utilizará cómodamente la referida información para acelerar su búsqueda.

Locomotive BASIC

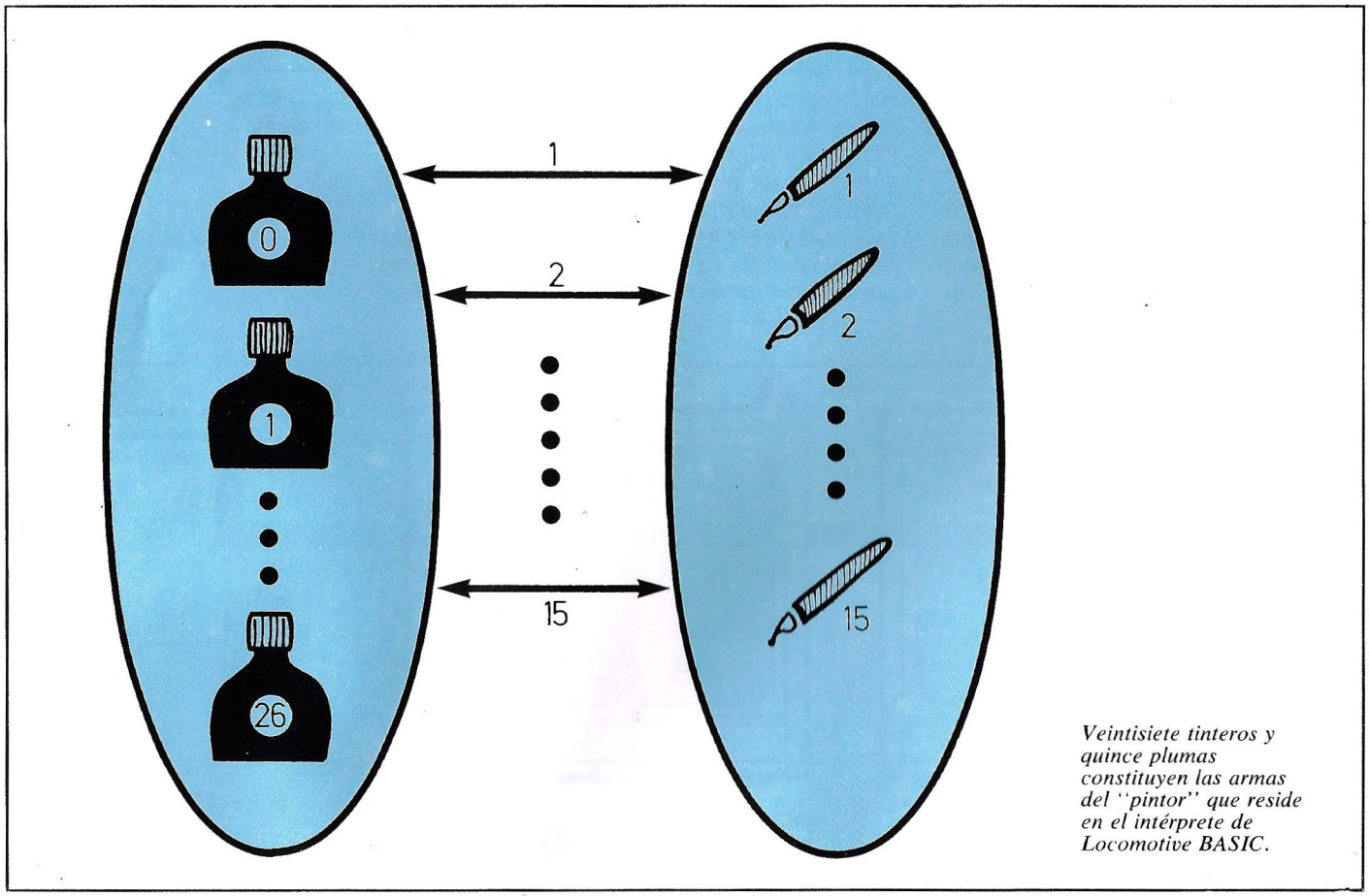
El dialecto BASIC de los ordenadores Amstrad

Los ordenadores Amstrad (CPC-464, CPC-664 y CPC-6128) están todos ellos dotados de intérpretes BASIC creados por la firma inglesa Locomotive. Como ya es habitual en el mundo de los intérpretes de este popular y omnipresente lenguaje de programación, cada dialecto incorpora ciertas características que lo distinguen. A través de los próximos párrafos se describirán las particularidades más significativas del Locomotive BASIC; algunas tan notables como la posibilidad de definir ventanas en la pantalla y gestionar las interrupciones bajo el control del propio intérprete BASIC.

LA PANTALLA

El usuario puede trabajar con tres formatos diferentes de pantalla o "modos", de los que sólo uno de ellos estará activo en cada momento. Los modos 0, 1 y 2 se corresponden con pantallas de 20, 40 y 80 columnas respectivamente. En todos los modos de pantalla el número de líneas horizontales es de veinticinco.

Se pueden manejar hasta 27 colores, aunque el número de ellos visualizables en cada momento depende del modo de definición de la pantalla: a mayor definición, menor número de colores disponibles para visualizar, ya que el ordenador debe almacenar en un espacio de 16K toda la



Veintisiete tinteros y quince plumas constituyen las armas del "pintor" que reside en el intérprete de Locomotive BASIC.

información relativa a la pantalla. A mayor número de caracteres imprimibles, menor es el espacio que queda para guardar la información sobre el color.
Siempre se dispone de quince plumas con las que escribir los caracteres, elegibles mediante una instrucción PEN, a las cuales puede asociarse la tinta que el usuario desee a través de la instrucción INK. Los colores del papel sobre el que se escribe y del borde de la pantalla son también definibles, pudiendo incluso hacer que sean parpadeantes y variar el período de parpadeo.
Aparte de las posibilidades cromáticas, disponibles en la gran mayoría de los ordenadores personales, los ordenadores Amstrad poseen otra facultad no tan generalizada; ésta es la de definir y gestionar hasta ocho ventanas independientes de texto sobre el monitor o TV.
Una ventana es un espacio definido sobre la pantalla del monitor que el programador puede gobernar como si se tratara de otra "pequeña pantalla", definiendo los colores del papel y de los caracteres de forma independiente para cada una de ellas. La ventana número cero (≠0) está asignada a todo el espacio físico de la pantalla del monitor.
Una pantalla se define a través de la instrucción WINDOW. Por ejemplo, la orden:

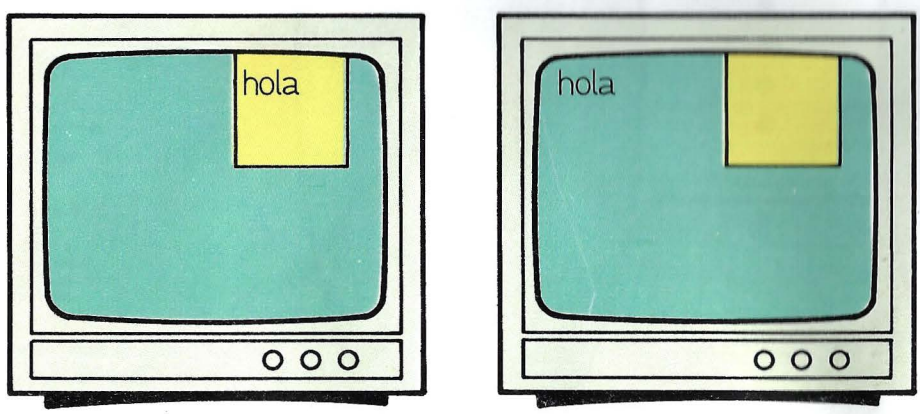
El intérprete BASIC de la firma inglesa Locomotive constituye el medio de diálogo con los ordenadores Amstrad. La fotografía muestra el modelo CPC-464.

WINDOW #3,10,20,1,20

define la ventana número 3. Las cuatro últimas cifras del argumento indican el tamaño de la ventana, especificando el borde izquierdo, derecho, superior e inferior, respectivamente. Los comandos típicos de interacción con la pantalla, como PRINT, INPUT o LOCATE (para situar el cursor en un punto determinado de la misma), deben ir seguidos por un número indicativo de la pantalla sobre la que deba reflejarse su acción. Si no es así, la máquina da por supuesto que las órdenes en cuestión hacen referencia a la ventana cero. Por ejemplo:

PRINT #3, "HOLA"

escribirá la cadena de caracteres HOLA en la esquina superior derecha de la pantalla, dentro de la ventana anteriormente defi-



Las ventanas son una potente herramienta a disposición del programador de aplicaciones.

nida; mientras que:

PRINT "HOLA" ó PRINT #0, "HOLA"

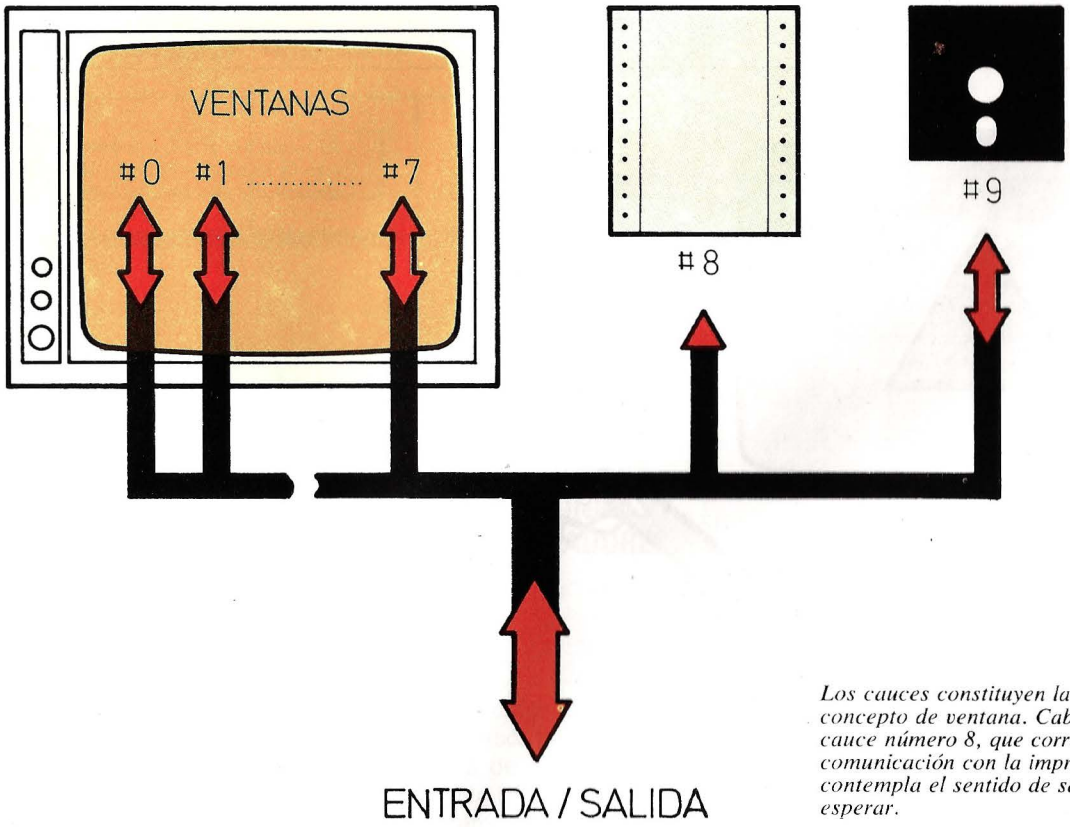
lo hará en el espacio definido para toda la pantalla.
En realidad, las ventanas son la particularización a un caso concreto de un concepto más amplio: el de "cauce" (ver figura). Toda la actividad de entrada/salida, que se lleva a cabo fundamentalmente a través de PRINT e INPUT, se puede asignar a un cauce concreto. Cuando el número de cauce está comprendido entre 0 y 7, se trata de una ventana de texto sobre la pantalla. Si este número es el 8, la salida se dirigirá hacia la impresora y si es el 9 se corresponde con el canal de comunicación con la unidad de disco o cinta. Por lo tanto, para obtener un listado por impresora, la orden a ejecutar es:

LIST #8

la cual dirige el listado hacia el cauce número ocho.
Las ventanas son una excelente herramienta para confeccionar programas de utilidad, en los que un factor decisivo es la facilidad que el usuario encuentre en su manejo: puede haber ventanas dedicadas a la presentación de menús, otras que muestren el estado en el que se encuentra la ejecución del programa, y otras cuya misión sea recoger las entradas que realiza el usuario a través del teclado.
También existe la posibilidad de trabajar en modo gráfico, con una resolución máxima de 640×400 pixels. Mediante los comandos adecuados se pueden elegir los colores del papel y de la pluma de gráficos.
Como viene siendo habitual, el Locomotive Basic también incluye todo un repertorio de comandos para dibujar líneas y puntos y mover el cursor gráfico a voluntad del programador. Aunque se echan de menos instrucciones típicas para el dibujo de círculos, sí se dispone de comandos para dibujar líneas discontinuas (MASK) y para rellenar superficies delimitadas por líneas dibujadas con anterioridad (FILL); desgraciadamente, este último comando sólo está presente en el Basic del CPC 664.

LOS SONIDOS

Las posibilidades que en este campo ofrece el Locomotive Basic son ingentes. En primer lugar se dispone de tres canales independientes de sonido, lo que da la posibilidad de conectar el ordenador a un amplificador estéreo para aprovechar al máximo sus características y mejorar la calidad ofrecida por el pequeño altavoz incorporado. Un canal se corresponde con el altavoz izquierdo, otro con el derecho y el tercero de ellos con ambos a la vez (ver figura).
Todos los sonidos se controlan a través de la instrucción SOUND, la cual admite hasta un total de siete parámetros. El primero de ellos especifica la situación de los canales anteriormente comentados; el segundo el período del tono emitido; el tercero su duración, expresada en centé-



Los cauces constituyen la extensión del concepto de ventana. Cabe observar que el cauce número 8, que corresponde a la vía de comunicación con la impresora, sólo contempla el sentido de salida como era de esperar.

simas de segundo con un valor comprendido entre 1 y 32767, y el cuarto especifica el volumen con un valor entre 0 y 15. Si este último no se define, el ordenador tomará el valor 12 por defecto.
Cada canal tiene asociado una cola que puede contener hasta cinco sonidos en espera de ser ejecutados. Suponga que tecleamos la siguiente orden seguida de una pulsación sobre ENTER:

```
SOUND 1,284,3000
```

Esta hará sonar por el altavoz incorporado al equipo la nota LA durante unos cinco minutos. Mientras está sonando esta nota es posible operar normalmente con el ordenador ya que, al pulsar ENTER, la orden fue almacenada en la cola de sonido del canal 1 para que la gestionara el chip correspondiente, pudiendo incluso teclear

```
SOUND 1,478,200 <ENTER>
```

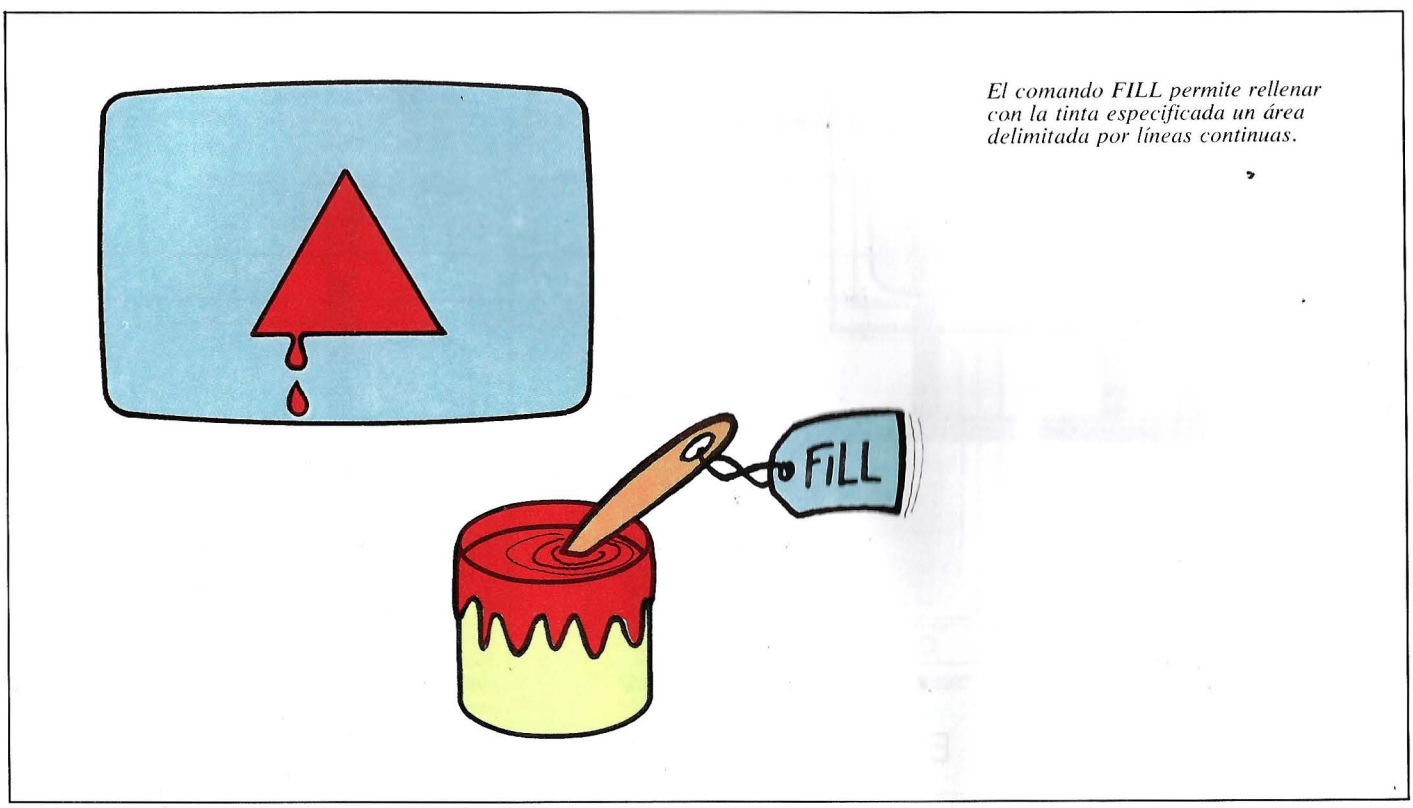
cuyo efecto es pasar a la cola de sonido del canal 1 un DO de duración igual a dos segundos. Al acabar la ejecución del LA, transcurridos los 5 minutos, el chip de sonido recogerá el DO de la cola y lo ejecutará.
Además se dispone de los comandos necesarios para sincronizar sonidos por los diversos canales disponibles, y controlar el estado de las colas para tomar decisiones en función de sus contenidos en la ejecución de programas.
Por si esto fuera poco, se pueden definir los contornos de las envolventes de volumen y de tono, con las que los sonidos generados dejan de ser fríos pitidos, a través de las instrucciones ENV y ENT respectivamente. De igual forma que una envolvente de volumen especifica la variación relativa de amplitud del sonido emitido en el tiempo, una envolvente de tono realiza la misma definición sobre el período del tono emitido.
Se pueden definir hasta quince envolventes de cada tipo, identificando a cada una por un número. Si el sonido generado por una instrucción SOUND va a hacer uso de alguna envolvente, el número de ésta se indica como 5.° parámetro para las de volumen y en el 6.° para las de tono.
Un ejemplo de envolvente de volumen puede ser:

```
ENV 1,5,3,4,5,-3,8
```

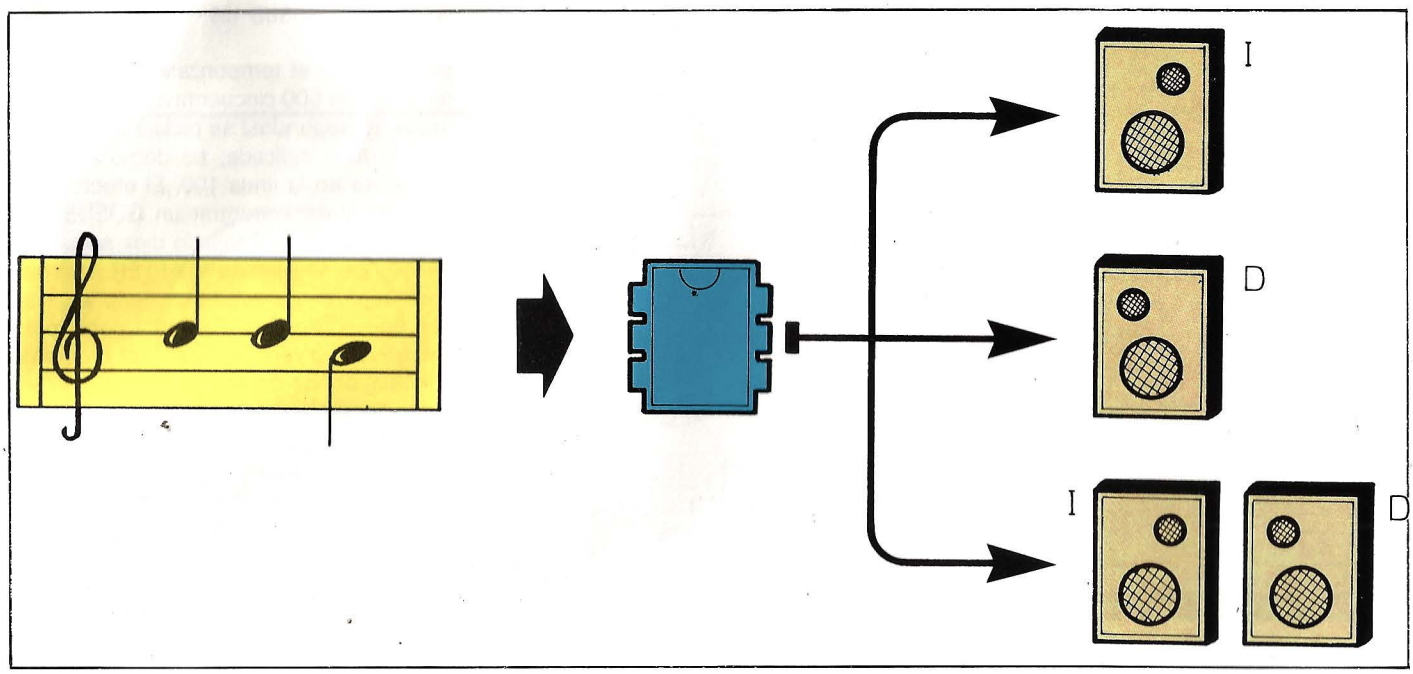
El primer parámetro especifica el número de la envolvente de volumen (1) por el que será referenciada en las instrucciones SOUND que la utilicen. Los siguientes parámetros hacen que la nota crezca en cinco etapas, aumentando el volumen tres unidades en cada una de ellas, siendo la duración de cada una de cuatro centésimas de segundo. A continuación, el volumen caerá en cinco etapas de tres unidades de volumen cada una, durando cada una 8 centésimas. Observemos que, puesto que una envolvente de volumen —y de tono también— especifica la duración de la nota, el correspondiente parámetro de la instrucción SOUND puede tomar el valor cero. Por ejemplo:

```
SOUND 1,284,0,0,1
```

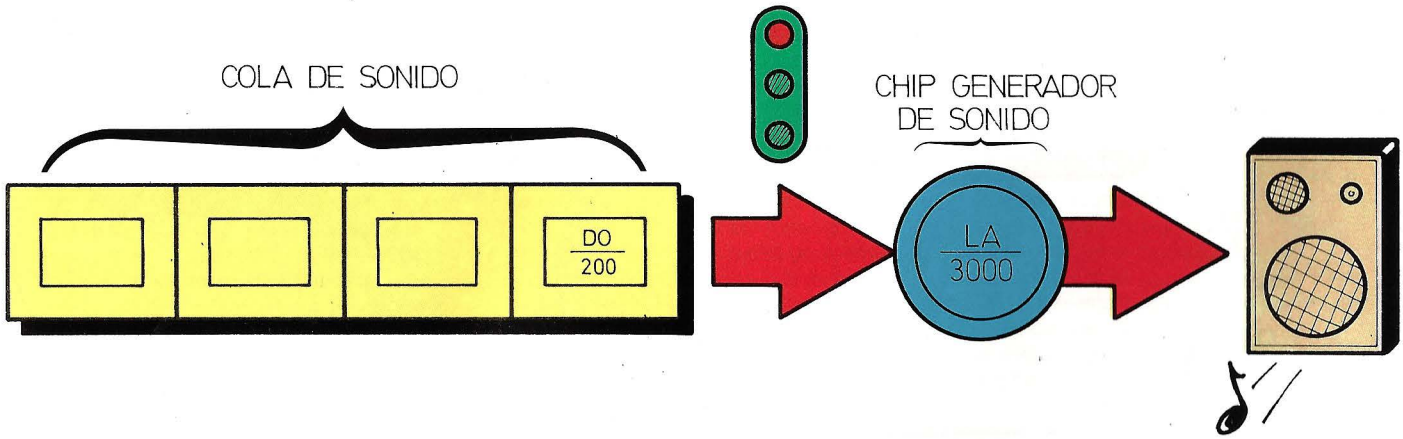
envía un LA por el canal 1, estando controladas su amplitud (que comienza en cero) y su duración por la envolvente de volumen número 1 (último parámetro).
Para finalizar la cuestión del sonido, comentar que, a base de variar el valor del parámetro de tono de la instrucción SOUND, se pueden barrer ocho octavas musicales, y que su último parámetro se utiliza para añadir ruido a los tonos genera-



El comando FILL permite rellenar con la tinta especificada un área delimitada por líneas continuas.



Es posible conectar a los Amstrad un amplificador estéreo que aproveche todas las posibilidades sonoras que se ofrecen. De no realizar esta conexión, toda la discusión sobre los canales carece de sentido ya que los sonidos son enviados siempre al pequeño altavoz incorporado, con independencia del canal especificado.



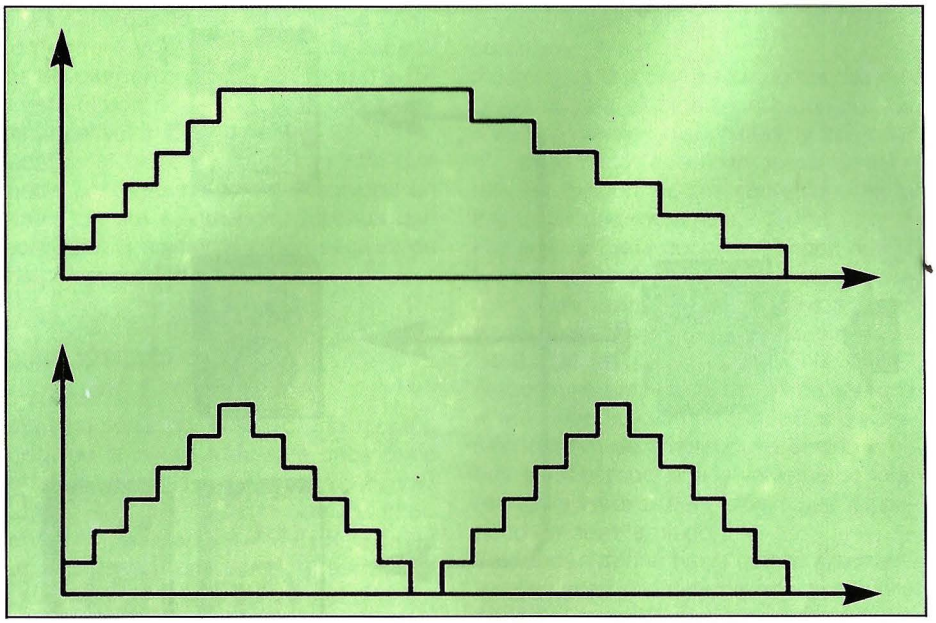
Cada canal de sonido tiene asociada una cola en la que van alojándose los sonidos a emitir; ello libera la atención de la CPU que podrá ocuparse de otras tareas.

dos. Tal posibilidad resulta útil para imitar sonidos secos, como los producidos por instrumentos de percusión o por una locomotora de vapor. También cabe resaltar que la versatilidad de los comandos enunciados hace pensar en la posibilidad de sintetizar la voz. De hecho, existe un programa denominado "3-D Voice Chess" que consiste en un juego de ajedrez que brinda la posibilidad de que el ordenador recite sus jugadas a la vez que las ejecuta. El resultado es, sin embargo, bastante pobre debido a las dificultades que presenta la síntesis de voz humana.

INTERRUPCIONES Y TEMPORIZADORES

En el Locomotive Basic encontramos una característica poco común entre los ordenadores de su rango: la posibilidad de gestionar las interrupciones desde el propio intérprete Basic.

Una interrupción es un evento interno o externo a la máquina que obliga a suspender momentáneamente la ejecución del programa en curso, para atender a la causa que produjo dicha interrupción. Las interrupciones tienen distinta prioridad según la importancia de los eventos que las producen: no es lo mismo atender la interrupción que puede generar el operador al pulsar la tecla ESCAPE o ^C al ver que su programa se ha metido en un bucle sin salida, que la que produce un hipotético circuito detector de fallo de la alimentación, en cuyo caso el interés primordial es intentar salvar en memoria permanente



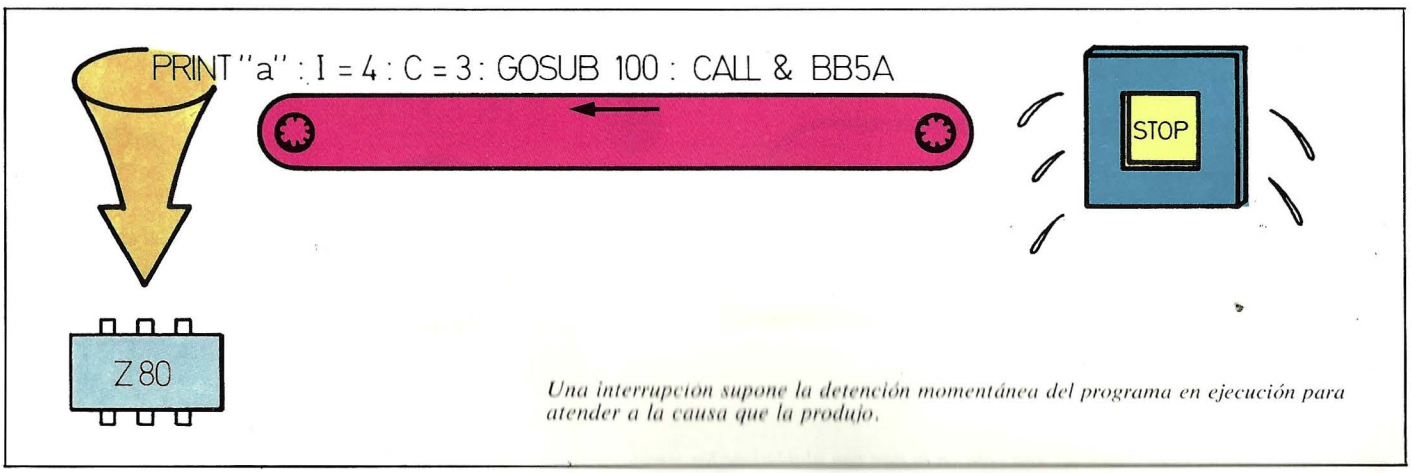
Los sonidos que emiten los instrumentos musicales son bastante mas complejos que simples pitidos. Respecto a su amplitud, se caracterizan por un período de tiempo en el que ésta sube muy pronunciadamente, manteniéndose después a un nivel constante, para, a continuación, decaer suavemente. La envolvente de volumen de la figura expresa esta característica.

AFTER 500,1 GOSUB 100

pone a cero el temporizador número 1 y después de 500 cincuentavos de segundo (unos 10 segundos) se pasará el control a la subrutina indicada; es decir, a la que comienza en la línea 100. El efecto sería análogo al de encontrar un GOSUB 100 después de haber pasado diez segundos desde que se ejecutó el AFTER. Una vez procesada dicha subrutina (esto es, una vez que aparece el RETURN correspondiente) se volverá al punto del programa principal donde ocurrió la interrupción.

El funcionamiento de EVERY es análogo al anterior, con la particularidad de que la subrutina especificada será ejecutada repetidas veces. Como vemos, los temporizadores "compiten" por arrancar el control al programa principal; de ahí que sea importante establecer una jerarquía de prioridades para resolver los casos de conflicto que pudieran aparecer entre ellos.

La acción de los temporizadores puede ser inhibida sobre una sección crítica del



Una interrupción supone la detención momentánea del programa en ejecución para atender a la causa que la produjo.

(disco por ejemplo) lo más que se pueda de lo que en ese momento estaba en memoria principal, relegando a un segundo plano a las restantes interrupciones (ver figura).

Existen cuatro temporizadores que permiten generar interrupciones en la ejecución del programa principal. Su acción consiste en bifurcar a una subrutina específica cada vez que se cumple el tiempo prefijado, volviendo de nuevo al programa principal una vez que la rutina seleccionada ha finalizado con un RETURN. Además, existe una quinta manera de ocasionar una interrupción: realizando un "Break", lo que se consigue pulsando la tecla de ESCape dos veces consecutivas.

El orden de prioridad de las interrupciones es, de mayor a menor:

— Break ([ESC] [ESC])
— Temporizador número 3
— Temporizador número 2
— Temporizador número 1
— Temporizador número 0

Los temporizadores se controlan fundamentalmente con las órdenes AFTER ("después de") y EVERY ("cada") que hacen exactamente lo que sus respectivos nombres indican.

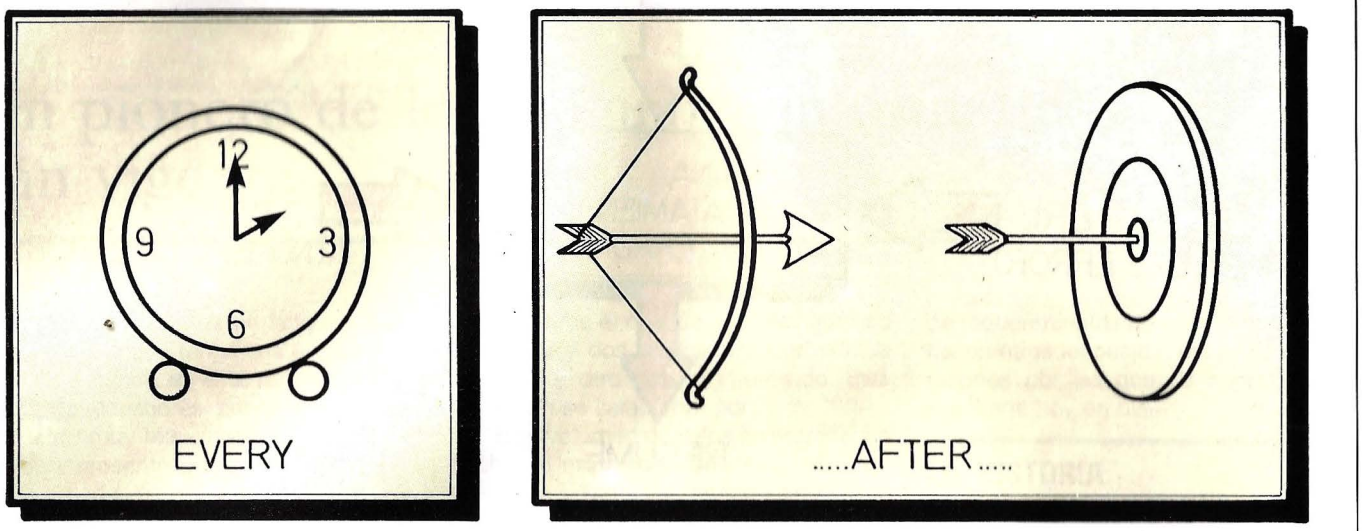
Por ejemplo, la ejecución de la instrucción

programa a través de las instrucciones DI (Disable Interrupts) y EI (Enable Interrupts). Cuando se ejecuta una instrucción DI, los intentos de arrebatar el control por parte de los temporizadores son anulados, volviendo a la situación normal al ejecutarse una orden EI.

La propia interrupción "Break" puede ser controlada por el programa en ejecución a través de una sentencia del tipo:

ON BREAK GOSUB 300

la cual bifurca hacia la subrutina indicada cuando se pulsa dos veces consecutivas la tecla ESCape. Incluso se puede inhibir por completo la posibilidad de que el ope-

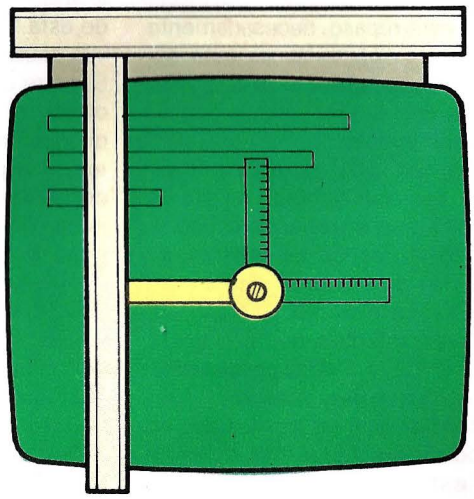


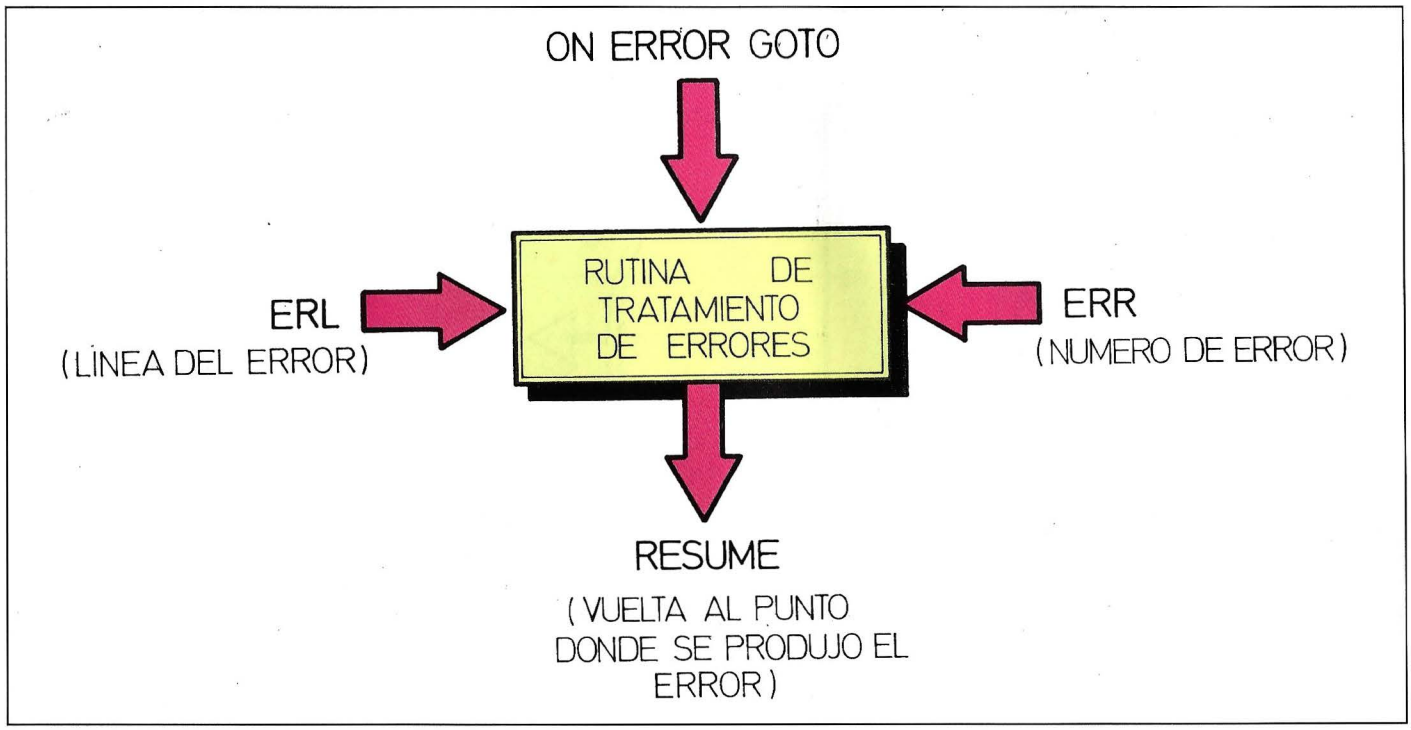
Una clara diferencia entre las órdenes EVERY y AFTER.

Editores de líneas y de pantalla completa

Los editores que habitualmente acompañan a los sistemas operativos, a los programas monitores o incluso a los intérpretes de lenguajes, suelen ser de dos tipos: de líneas o de pantalla completa ("full-screen"). En ambos casos, la finalidad del editor es permitir la creación y modificación de bloques de texto o de programas que deben ser sometidos a tratamiento por parte del ordenador.

Mientras que en un editor de líneas sólo es posible modificar o crear la línea que está actualmente en edición, un editor de pantalla permite moverse por cualquier punto de la misma, pudiendo realizar los cambios oportunos sin más esfuerzo que el de colocar el cursor en el punto deseado mediante las teclas pertinentes. La razón por la que la mayoría de los intérpretes de Basic incorporan un editor de líneas en vez del más cómodo de pantalla es doble. En primer lugar, desarrollar un editor de líneas es más sencillo y ocupa menos espacio en memoria que su compañero. Además, al ser el Basic un lenguaje en el que la "línea" tiene una importancia especial —hasta el punto de que todas van numeradas, cosa que no ocurre en Pascal o C, por ejemplo—, utilizar un editor de estas características no es tan problemático como en los lenguajes citados.





*Una rutina de tratamiento de errores debe recoger información del tipo y lugar donde ocurrió el error para tratar a los diferentes casos de la manera oportuna. Toda rutina de este tipo acabará con una instrucción **RESUME** que devolverá el control al lugar donde se produjo el error una vez subsanado.*

rador interrumpa el programa con la instrucción ON BREAK CONT.

EL INTERPRETE EN GENERAL

Después de este repaso, necesariamente superficial, a las características más sobresalientes de este dialecto BASIC, quedan por comentar aún una serie de pequeños detalles de interés.

Aparte de la conocida estructura FOR-NEXT para realizar bucles, existe la WHILE-WEND, que se muestra especialmente útil en la gestión de ficheros en disco/cinta. Hay que recordar que un bucle FOR-NEXT es siempre ejecutado al menos una vez, mientras que si a la entrada de un WHILE-WEND la condición es falsa, todo el bloque será saltado.

La posibilidad de tratar los errores que se puedan producir durante la ejecución del programa está ampliamente contemplada. El programador puede crear mensajes de error a la medida de la aplicación que esté desarrollando. Se pueden realizar rutinas de tratamiento de errores (ON ERROR GOTO) cuya misión consistirá en que el propio ordenador se recupere sin detener la ejecución del programa. Para ello se dispone de instrucciones específicas (ERR y ERL) que devuelven información sobre el código del error producido y la línea en que ocurrió, respectivamente; de esta forma podrá determinarse el lugar y la causa del fallo.

Si entre el gran surtido de comandos que ofrece el intérprete no se encuentra el que se necesita en un momento dado, existe la posibilidad de que el programador se lo construya a medida, a través de lo que en la jerga de Locomotive se denomina un RSX (Resident System eXtension —Ampliación del sistema residente—).

Los RSX van precedidos por una barra vertical (|) y tienen que ser programados en código máquina. En la publicación "CPC464 Firmware Manual" se comenta la forma de programar estos comandos para poder utilizarlos desde el Basic. Por ejemplo, dada la falta de un comando en el intérprete para el dibujo de círculos, se podría pensar en la creación de un RSX cuya utilización fuera algo parecido a:

```
|CIRCLE,200,150,25
```

en el que los dos primeros parámetros sitúan el centro y el último especifica el radio. Desgraciadamente, el hecho de dotar a los Amstrad con nuevos comandos no es tarea sencilla; se necesita un buen conocimiento del lenguaje ensamblador del microprocesador Z80, así como una buena documentación sobre la forma de crearlos, la cual resulta escasa en la publicación antes citada.

Para finalizar, aparte del editor de líneas que proporcionan casi todos los intérpretes Basic para la creación de programas, existe lo que han dado en llamar un "editor de pantalla", el cual dista mucho de ser tal. En realidad lo que se ofrece es la posibilidad de copiar sobre la línea que está editándose el contenido total o parcial de otra línea que en ese momento aparezca en la pantalla, simplificando así la tarea de teclear una serie de líneas consecutivas con mínimas diferencias entre ellas.

COBOL

Un pionero de los lenguajes informáticos aún vigente

El COBOL es un lenguaje que, aunque tenga una escasa presencia en el terreno de los ordenadores personales, ocupa un puesto muy relevante en el panorama pasado y presente de la informática.

EL PROPOSITO DEL COBOL

El COBOL lleva la orientación que se le quiso dar desde su nacimiento en su propio nombre. En efecto, la palabra COBOL está formada por las iniciales de "COmmon Business Oriented Language", esto es: "lenguaje general orientado a los negocios".

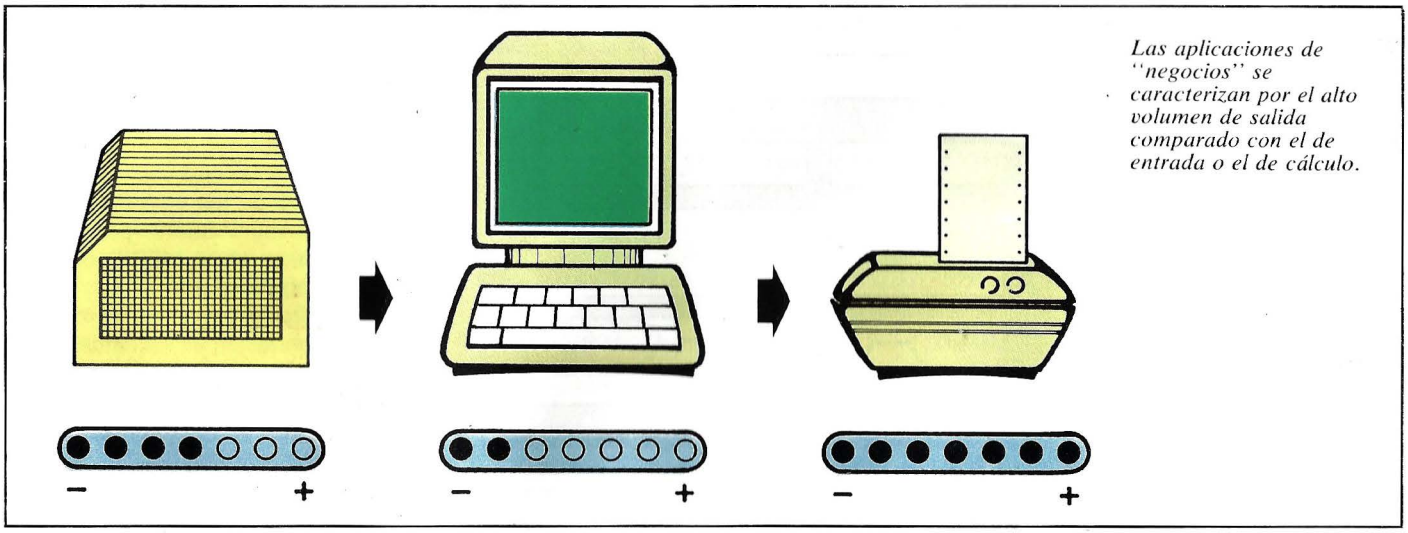
Una aplicación típica del COBOL puede consistir en la gestión de la nómina de empleados de una empresa. La tarea encomendada al programa será que, cada vez que sea ejecutado a fin de mes, calcule el sueldo que va a recibir cada empleado en base a aumentos, cotizaciones, etc., imprima el cheque que recibirá cada trabajador y dos resguardos, uno para la empresa y otro para el interesado. Esta aplicación se caracteriza por:

1. Bajo volumen de datos de entrada. De hecho, el programa sólo necesitará nuevos datos si se ha contratado a alguien por primera vez en el transcurso del mes; aunque si la empresa es muy grande, el número de nuevos contratados puede resultar elevado.
2. Poco volumen de cálculo. Las operaciones a realizar con los sueldos se reducen a sumas, restas y algún tanto por ciento.
3. Gran volumen de salida. Aunque en la impresión de los cheques existen pocas variantes, no sucede lo mismo con los resguardos, en los que pueden ir especificados los trabajos realizados, horas ocupadas y un sinfín de datos aplicables tan sólo a una parte de los empleados.

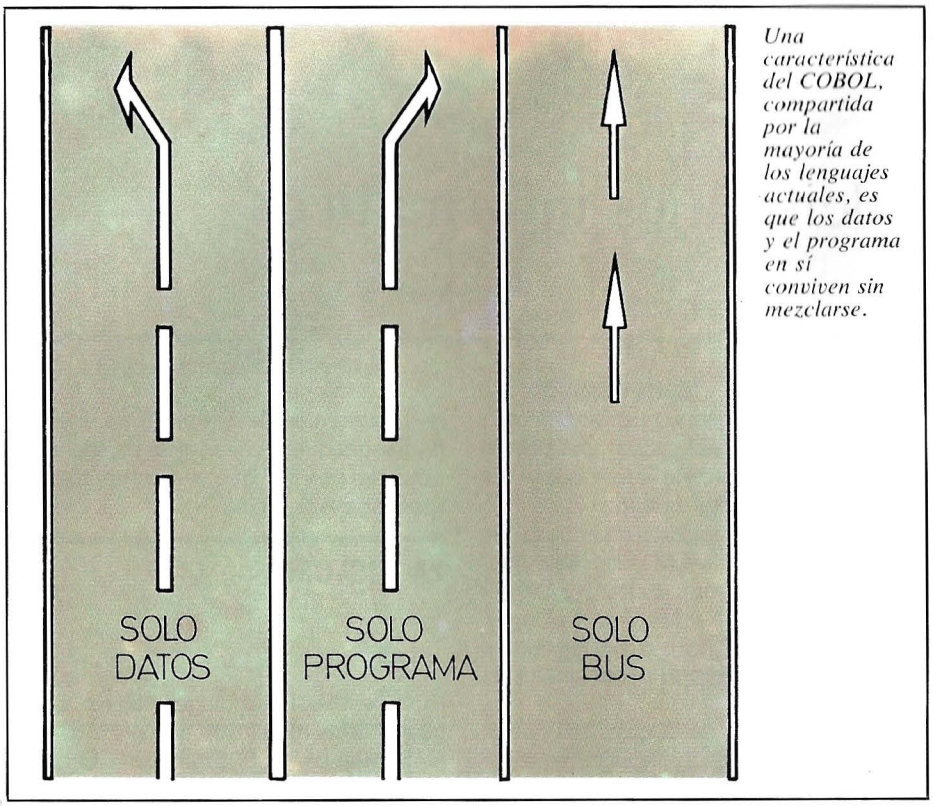
Estas tres características son en gran parte comunes a la mayoría de las aplicaciones informáticas que requieren los "negocios". COBOL se ajusta a este tipo de requerimientos de una forma que no lo hacen otros lenguajes; ésta es una de las razones por las que su importancia se mantiene hoy en día.

LA HISTORIA

Aunque nuestro protagonista es apenas cuatro años más joven que el FORTRAN, encontramos en él características que lo alejan mucho de su compañero científico. Su nacimiento se produjo a raíz de una reunión convocada por el Departamento de Defensa de los Estados Unidos y en la que estaban representadas las mayores empresas informáticas de la época (algunas sobreviven hoy, como IBM, Honeywell o Burroughs), instituciones y el ejército. Esta convocatoria recibió el nombre de CODASYL, siglas de COnference on Data SYstems Languages (Conferencia sobre los lenguajes de sistemas de datos).



Las aplicaciones de "negocios" se caracterizan por el alto volumen de salida comparado con el de entrada o el de cálculo.

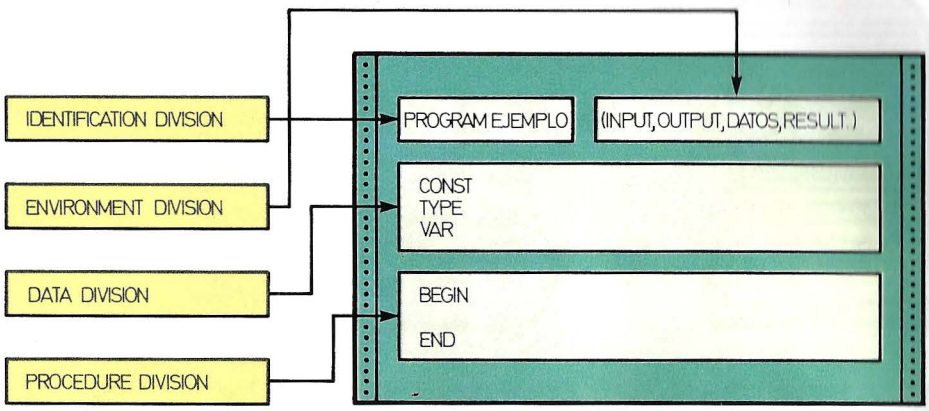


Una característica del COBOL, compartida por la mayoría de los lenguajes actuales, es que los datos y el programa en sí conviven sin mezclarse.

De CODASYL surgió el COBOL 60. Posteriormente fueron apareciendo variantes de este COBOL, manteniendo siempre la misma estructura y sentencias aunque añadiendo otras nuevas; así se llegó hasta el COBOL ANSI 74 y el COBOL 80. Más recientemente, han aparecido el CIS-COBOL, con facilidades para el manejo de pantallas, y el RM-COBOL, especialmente diseñado para su uso en sistemas basados en microprocesador.

Con el COBOL nació un concepto que se ha mantenido en la mayoría de los lenguajes que aparecieron con posterioridad: la separación del programa propiamente dicho y de los datos del mismo. Esto trae consigo indudables beneficios en la depuración y mantenimiento de los programas (ver cuadro adjunto).

En la actualidad, el COBOL mantiene su vigencia debido en parte al gran número de aplicaciones que han sido desarrolladas con él y a la experiencia adquirida por los programadores; aunque también constituye una razón primordial el hecho de que los requerimientos de las aplicaciones de negocios se adaptan muy bien a sus prestaciones.



La equivalencia entre las Divisions del COBOL y la estructura de un programa en PASCAL, aquí representada, no es ni mucho menos exacta, y debe ser considerada como una primera aproximación al significado de cada una de ellas en un programa COBOL.

Por lo demás, la circunstancia de que una empresa cambie el ordenador que está utilizando no significa tirar por la ventana todo el software desarrollado hasta la fecha. Los programas escritos en COBOL tiene el "don" de la transportabilidad, esto es, son fácilmente implementables sobre distintas máquinas.

ESTRUCTURA DE UN PROGRAMA COBOL

En la figura adjunta se muestran las cuatro "divisions" (zonas o divisiones) en las que se divide todo programa COBOL, intentando buscar un equivalente Pascal a las mismas. Estas cuatro divisiones deben aparecer en el programa en el mismo orden que refleja la figura. Sus propósitos son los que se describen a continuación.

- *Identification Division*

El compilador no producirá ningún código objeto a partir de ella. Es algo parecido a tener una serie de líneas REM en BASIC. Su contenido especifica datos concernientes al programador, propósito y estructura del programa, fecha de realización, etc.

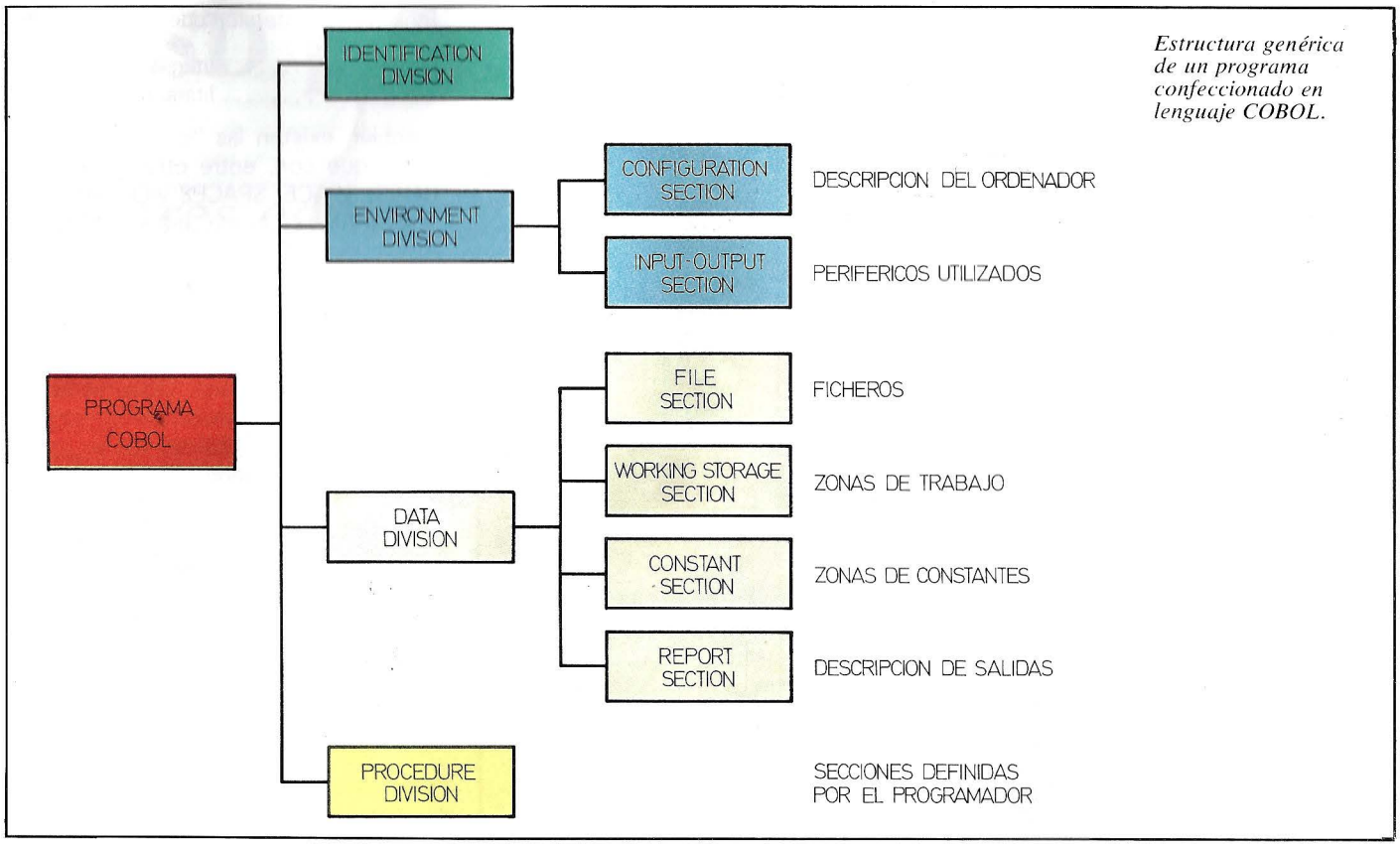
- *Environment Division*

Adapta el programa a la configuración del sistema. Recordemos que en el FORTRAN las sentencias READ y WRITE llevaban un primer argumento que especificaba el dispositivo sobre el que se iba a realizar la operación. Este tipo de información, entre otras, iría en esta división.

- *Data Division*

Describe los ficheros, las estructuras de datos y las variables que van a ser procesadas: longitud de los campos en variables de tipo ALPHABETIC, posición del punto decimal en las NUMERIC, organización de los registros, etc.

La Data Division proporciona las características y organización de los datos de una manera que es, en gran medida, independiente del equipo que se utilice (cuya definición está en la Environment Division) y casi por completo independiente de la forma en la que se van a procesar los datos (Procedure Division).



Estructura genérica de un programa confeccionado en lenguaje COBOL.

- *Procedure Division*

Especifica las acciones que hay que realizar sobre la Data Division. Constituye el programa en sí.

En la figura está representada la estructura final que, a grandes rasgos, presenta un programa COBOL.

ALGUNOS EJEMPLOS DE SENTENCIAS

Aunque en la jerga del COBOL la palabra "sentencia" carece de significado, la utilizaremos aquí en el sentido que le es habitual en los lenguajes conocidos.

El nombre de una variable en COBOL está compuesto por hasta treinta caracteres entre letras, dígitos y guiones (estos últimos especialmente útiles en nombres de variables constituidos por más de una palabra). Al menos uno de ellos ha de ser una letra y no puede acabar en guión.

Nombres válidos serían pues:

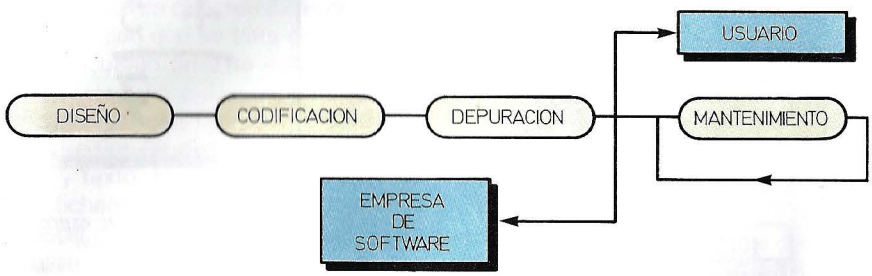
Ciclo de vida de un programa

Se ha mencionado que el hecho de especificar la estructura de datos y el programa en sí (el algoritmo a seguir) en dos partes diferenciadas del programa ayuda a la introducción de futuros cambios y a la corrección de errores.

El ciclo de vida de un programa comprende todas las fases que van desde el diseño de la estructura del mismo hasta que le es entregado al usuario, incluyendo la corrección de los posibles errores que se detecten una vez en funcionamiento.

La susodicha partición del programa en dos zonas diferenciadas es útil prácticamente en todas las fases, pero lo es especialmente en la de mantenimiento. En esta última se corrigen los errores que el usuario encuentra en el funcionamiento del programa que se le entregó.

No hay que olvidar que un gran proyecto informático es un esfuerzo que puede ocupar a muchas personas durante un largo período de tiempo. Cuando aparecen los errores en la fase de mantenimiento, los componentes del equipo de programación pueden no ser los mismos, por lo que el hecho de que el propio programa se "autoexprese" a través de sus estructuras de datos y algoritmos resultará de gran ayuda.

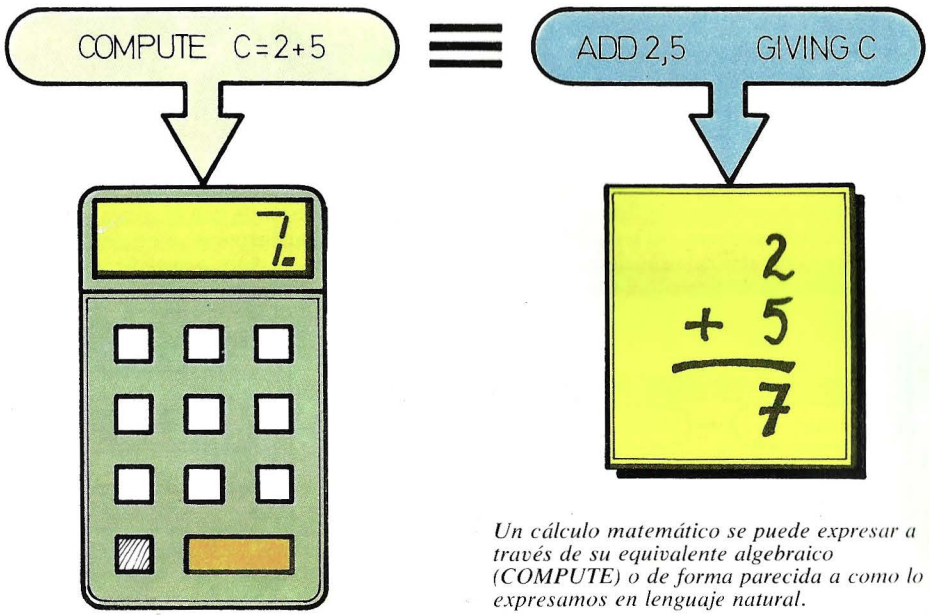


PALABRAS RESERVADAS DEL COBOL

ACCEPT	FOR	READ
ADD	FROM	REEL
AFTER	GIVING	REMAINDER
ALL	GO TO	REPLACING
ALPHABETIC	GREATER	REWRITE
AND	IF	ROUNDED
BEFORE	INPUT	RUN
BY	INSPECT	SEARCH
CHARACTER	INTO	SENTENCE
CLOSE	INVALID	SET
COMPUTE	I-O	SIZE
COUNT	KEY	START
DATE	LEADING	STRING
DAY	LESS	SUBTRACT
DELETE	LOCK	TALLYING
DELIMITED	MOVE	THROUGH
DEPENDING	MULTIPLY	THRU
DISPLAY	NEGATIVE	TIME
DIVIDE	NEXT	TIMES
DOWN	NOT	TO
ELSE	NUMERIC	UNIT
END	OPEN	UNSTRING
END-OF-PAGE	OR	UNTIL
EOP	OUTPUT	UP
EQUAL	OVERFLOW	USE
ERROR	PAGE	VARYING
EXCEPTION	PERFORM	WHEN
EXIT	POINTER	WRITE
EXTEND	POSITIVE	
FIRST	PROCEDURE	

SALDO-TOTAL
CANTIDAD-EN-EXISTENCIA
8WXY

Lo que normalmente reciben el nombre de constantes en otros lenguajes se llaman en COBOL literales, y pueden ser de dos tipos: numéricos y no numéricos, en cuyo caso van encerrados entre comillas simples.



Un cálculo matemático se puede expresar a través de su equivalente algebraico (COMPUTE) o de forma parecida a como lo expresamos en lenguaje natural.

Ejemplos de literales pueden ser:

123.45 ..... **literal numérico**
'SOFT' ..... **literal no-numérico**

También existen las "constantes figuradas", que son, entre otras, ZERO, ZEROES, SPACE, SPACES y QUOTE. Si en la Data Division hemos definido la variable ABC como capaz de contener seis caracteres, al hacer:

```
MOVE SPACES TO ABC.
```

la variable ABC pasa a contener sólo espacios en blanco (seis). Observemos el punto con el que finaliza esta sentencia. En COBOL, las sentencias van separadas entre sí por un punto y un espacio como mínimo.

Las operaciones aritméticas se pueden realizar dejando el resultado en uno de los operandos, o especificando la variable que va a recibir tal resultado. Por ejemplo:

```
ADD 1 TO CONTADOR. ..... CONTADOR=
                         CONTADOR+1
ADD A,B TO C.      ..... C=A+B+C
ADD A,B,C GIVING D. .... D=A+B+C
```

De forma análoga se expresan restas (SUBTRACT), multiplicaciones (MULTIPLY) y divisiones (DIVIDE). Hay una manera de abreviar la escritura de los cálculos a través de la sentencia COMPUTE (ver figura).

Los siguientes ejemplos son equivalentes a los anteriores:

```
COMPUTE CONTADOR=CONTADOR+1.
COMPUTE C=A+B+C.
COMPUTE D=A+B+C.
```

Las secciones en las que el programador divide a la Procedure Division son una especie de subprogramas que pueden ser invocados a través de las sentencias PERFORM o GO TO.

La entrada/salida se realiza a través de las sentencias ACCEPT, DISPLAY, OPEN, READ, WRITE y CLOSE, estando las cuatro últimas dedicadas al manejo de ficheros. Por ejemplo, podemos especificar:

```
DISPLAY 'HOLA COMO ESTAS'.
```

y aparecerá HOLA COMO ESTAS en el dispositivo de salida especificado en la Configuration Division. Como en todo literal no-numérico no pueden aparecer comillas; para escribir la misma frase entrecomillada deberíamos utilizar la constante figurada QUOTE:

```
DISPLAY QUOTE 'HOLA COMO ESTAS' QUOTE.
```

UCSD p-System (2)

Los tres editores del sistema operativo

En el sistema operativo UCSD p-System, el usuario cuenta con la posibilidad de elegir entre tres editores: un editor de líneas y dos modalidades de editor "full screen" o de pantalla completa. Estos editores son los siguientes:

— YALOE
— Editor estándar "full screen"
— L2

YALOE es el nombre constituido a partir de las siguientes siglas "Yet Another Line Oriented Editor". En efecto, se trata de un editor orientado a la línea, por lo que resulta interesante en aquellos casos en los cuales no se dispone de medios adecuados para la introducción de datos al ordenador. El editor estándar de pantalla completa es el que va asociado normalmente con el sistema operativo UCSD-p. Como todo editor de tipo "full screen" permite al usuario extender la actividad de edición a todo el contenido de la pantalla, lo cual exige contar con el oportuno periférico de visualización, así como con particiones de memoria adecuadas. Este editor permite la localización y sustitución de grupos de caracteres, así como la ejecución de funciones orientadas a una mayor comprensión de los programas y textos sujetos a edición.

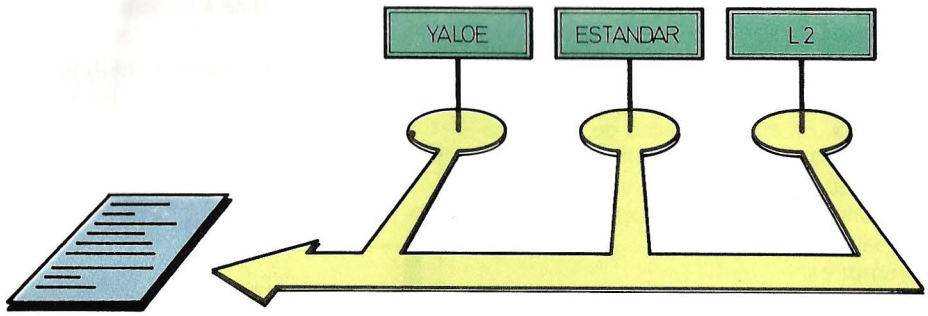
Para su entrada en actividad, el editor estándar exige la asignación de un área de memoria (buffer de editor) en la que sea posible cargar los diversos ficheros a tratar. El contenido de los referidos ficheros es modificado mientras reside en esa zona de memoria; luego, una vez concluida su edición, el contenido de la misma pasa a actualizar los correspondientes ficheros residentes en un soporte de memoria secundaria (en disco flexible, por ejemplo).

En este punto se hace patente una insuficiencia del editor estándar del UCSD p-System, cual es el hecho de admitir tan sólo ficheros que tengan un tamaño igual al del buffer del editor; de tal forma que no es posible cargar ficheros de superior tamaño. Esta característica deprecia su calidad respecto a los editores de otros sistemas operativos, como el CP/M o MS-DOS.

El editor L2 del sistema operativo UCSD p-System aporta una solución a las objeciones que acabamos de precisar. Mantiene igualmente un buffer de edición; pero ahora éste puede ser más reducido, puesto que se mantiene un fichero auxiliar en el disco que permitirá ir operando con sucesivas porciones, oportunamente cargadas en el buffer de edición. Con este método no exite ya el límite impuesto por el tamaño de los ficheros a editar.

En los próximos párrafos vamos a describir las principales características del editor estándar, dado que se trata de la alternativa más común en este sistema operativo.

Como dato general, cabe señalar que éste puede operar en dos modalidades: programa y texto. En la primera de ellas se tratan ficheros que contienen programas en Pascal, mientras que en la segunda se procesan ficheros de texto en lenguaje natural, tal como ocurre con un procesador de textos convencional, aunque sin muchas de las capacidades de éste último.

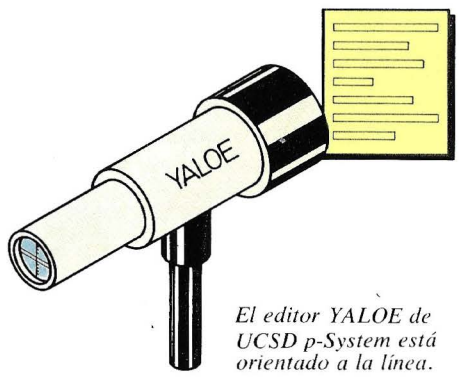


El usuario del sistema operativo UCSD p-System tiene a su disposición varias herramientas de apoyo para las tareas de edición.

ACCESO AL EDITOR ESTANDAR

El acceso al editor se logra tecleando la palabra EDIT. Hecho esto, se abren tres posibilidades:

- Carga del fichero de trabajo. Sólo



El editor YALOE de UCSD p-System está orientado a la línea.

puede producirse si en el directorio raíz está definido un fichero de nombre:

SYSTEM . WRK . TEXT.

Este proceso es automático y no se produce si el fichero no existe.

- Carga de un fichero específico. Para ello es preciso teclear el nombre del fichero sin el sufijo TEXT. Esta opción desencadenará la carga del fichero en el buffer del editor.
- Carga de un fichero en blanco. Para ello es necesario, únicamente, accionar la tecla RETURN.

Si no se ha cometido ningún error, bastará con pulsar la tecla de ESCAPE para salir del editor y pasar al modo general de comandos. En cualquier caso de los indicados aparecerá el prompt del editor, el cual contiene un resumen de los diferentes comandos que permiten controlar el funcionamiento del editor. El aspecto de este prompt es:

> Edit: A(djst C(py D(Lete F(ind I(nsrt J(mp R(place Q(uit X(chng Z(ap)

CARACTERISTICAS ESPECIALES DEL EDITOR ESTANDAR

Este editor estándar del UCSD p-System ofrece una serie de posibilidades que distinguen su funcionamiento de otros editores. Cualquier usuario de un editor sabe que uno de los aspectos más tediosos, cuando es necesario efectuar correcciones, reside en la necesidad de desplazar el cursor del editor hacia atrás. Pues bien, este editor ofrece tal posibilidad de movimiento hacia atrás, lo cual se indica por la dirección que muestra el símbolo > localizado en el prompt del editor.

Pulsando la tecla < es posible cambiar el sentido de desplazamiento, con lo cual se da un toque de flexibilidad a las operaciones. Pulsando la mencionada tecla se devolverá el desplazamiento a su sentido usual.

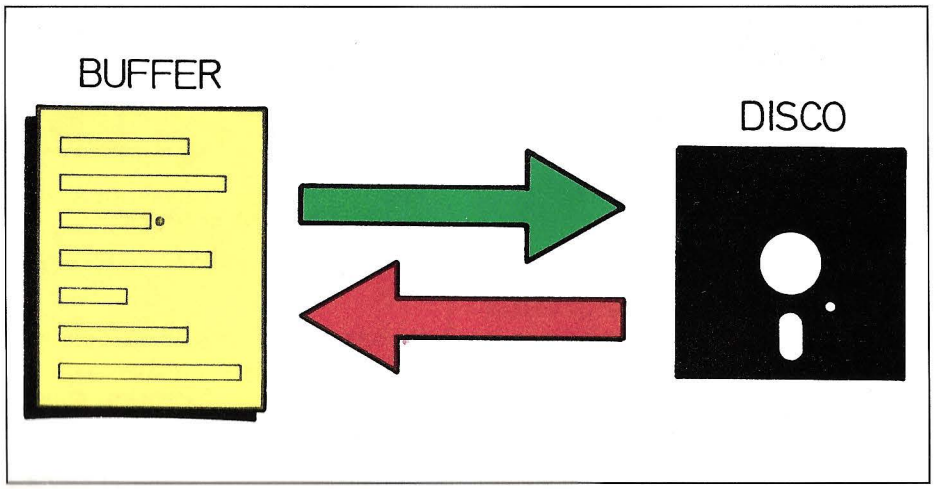
Otra característica singular de este editor es el hecho de que permite programar el efecto de las teclas de control del cursor en forma de autorrepetición predefinida. Así, para la operación de la barra espaciadora, por ejemplo, bastará con pulsar un número cualquiera y a continuación dicha barra, para que el cursor se desplace un número de espacios igual al número introducido. Igual efecto puede conseguirse con las teclas que controlan el desplazamiento del cursor en cualquier sentido (flechas), con el tabulador, etc.

DESPLAZAMIENTO DEL CURSOR

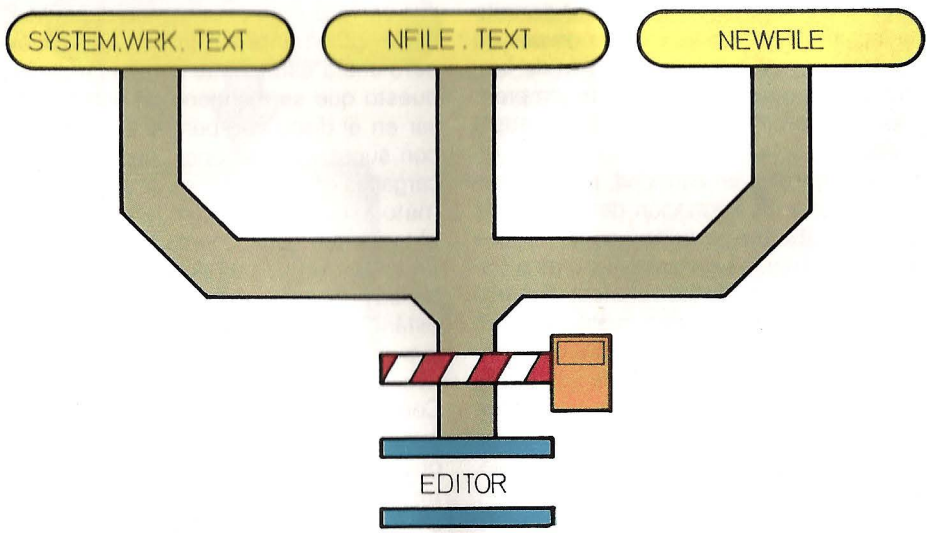
Una vez que se tiene un fichero cargado en el buffer del editor, pueden dar comienzo las operaciones sobre su contenido. Para ello es necesario desplazar el cursor a través de la pantalla, con objeto de situarlo en aquellos puntos en los que deba realizarse un proceso de edición. Los métodos de desplazamiento a disposición del usuario son los siguientes:

— Teclas de desplazamiento en cualquier dirección (flechas).
— Barra espaciadora.
— Teclas "Backspace", "RETURN" y "TAB".
— Comandos Jump y Page.

La característica más relevante de las órdenes de desplazamiento es la de admitir argumentos numéricos en la forma seña-

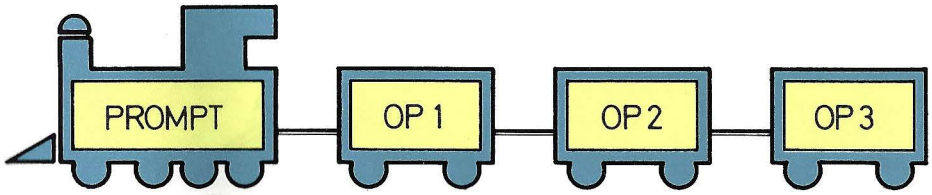


El editor L2 actualiza el buffer de edición continuamente desde la memoria de masa.



Al acceder al editor del sistema operativo UCSD p-System cabe la posibilidad inicial de elegir entre tres posibles caminos.

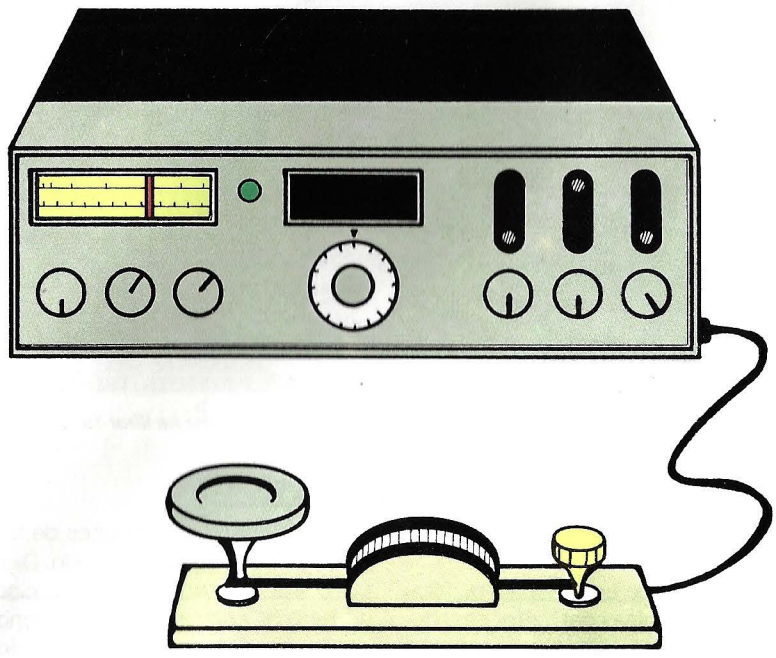
lada en el apartado anterior. En todo caso, la barra espaciadora sólo admite desplazamientos en el interior de la línea en curso; aunque, igualmente acepta argumentos numéricos.
La tecla Backspace cumple una función similar a la barra espaciadora, si bien se desplaza en sentido inverso a la anterior; concretamente, su sentido de movimiento depende de los criterios fijados por el usuario.
La tecla RETURN permite saltar verticalmente de línea en línea, posicionando el cursor en el primer carácter en blanco de la siguiente. La tecla TAB hace que el cursor se desplace un determinado número de espacios hacia adelante. Por defecto este número es de 8 espacios (1, 9, 17, 25), aunque al igual que en el caso de los comandos anteriores, admite argumentos numéricos para el control del número de espacios a saltar.
El comando JUMP permite posicionarse en el buffer del editor. Su activación permite al usuario saltar con inmediatez al principio o al final de dicho buffer. Si previamente se han definido los oportunos marcadores de posición en el interior del buffer, dicho comando permitirá posicionarse sobre ellos. El salto se producirá a la marca inmediatamente posterior a la posición actual en la que se encuentra el cur-



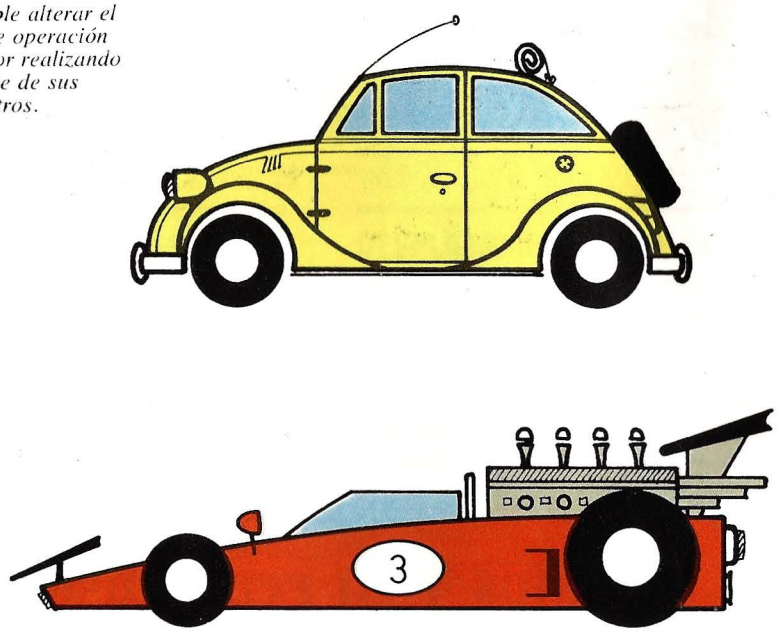
El prompt del sistema operativo UCSD p-System muestra la lista de opciones y comandos directamente accesibles.

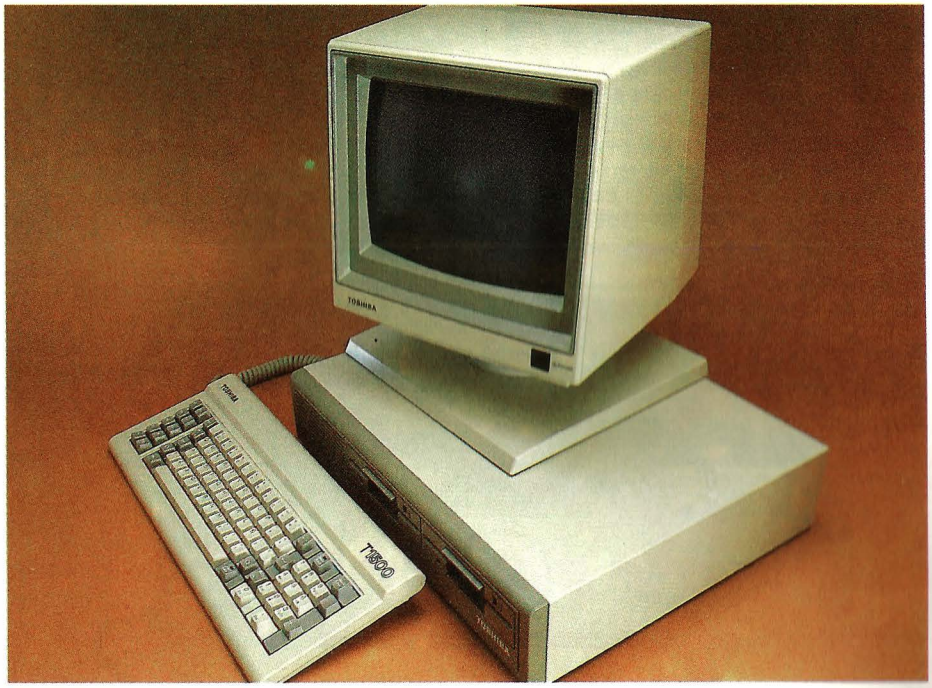
Formas de proceso de datos y su repercusión en el hardware

El tratamiento de datos admite dos variantes fundamentales cuya elección depende, esencialmente, de las necesidades del usuario del equipo informático. Tal decisión implica un desplazamiento de los requisitos técnicos del hardware que ha de soportarlo.
La primera alternativa es el denominado proceso "on-line". Esta técnica permite que se realicen las transacciones y se actualicen los bancos de datos en el mismo instante en el que se lleva a cabo la operación. Un ejemplo puede ser la reserva de un billete de avión en cualquier oficina preparada para ello. Dado que en este modo de operación las respuestas a las peticiones del usuario han de ser sumamente rápidas, es necesario que la CPU sea capaz de procesar un gran número de transacciones a gran velocidad. Igualmente, se necesitan unidades de disco cuyos tiempos de acceso a la información almacenada en ellos sean los más bajos posibles.
También es imprescindible que los sistemas de transmisión de información, terminal-CPU y CPU-unidad de disco, sean de gran capacidad y velocidad; resultaría ilógico tener una CPU muy potente pero rodeada de sistemas de transferencia de información lentos e ineficaces.
El proceso de datos en modo "batch" supone la antítesis del proceso de datos "on-line". En él se submiten las tareas a realizar, las cuales se emplazan en una serie de colas dependiendo de sus respectivas prioridades. La CPU se ocupa de procesarlas una tras otra, con los consiguientes retrasos. Obviamente, los requisitos especificados para el proceso de datos on-line serían también deseables en una máquina orientada al proceso batch. Sin embargo, por la propia naturaleza de esta técnica, es posible rebajar las exigencias y trabajar con unidades de proceso, sistemas de comunicaciones y periféricos de almacenamiento de inferior capacidad y velocidad de proceso.
Normalmente, el proceso en modo batch es el que se emplea en operaciones tales como confección de nóminas o gestión de ficheros de personal, tareas que normalmente no requieren una respuesta instantánea y que pueden extenderse en el tiempo.



Es posible alterar el modo de operación del editor realizando un ajuste de sus parámetros.





El usuario del UCSD p-System cuenta con tres posibles editores para facilitar su labor de confección de programas o documentos de texto sobre la pantalla.

sor, dentro del conjunto global del buffer de edición.

El comando PAGE se emplea para gestionar las distintas páginas del buffer del editor. En su definición interna, éste se divide en lo que se denominan páginas, que no son más que participaciones de memoria de un tamaño predeterminado. Dicho buffer puede contener dos, tres, cinco o el número de páginas que correspondan por su tamaño. Por medio del comando PAGE queda a disposición del usuario un método por medio del cual es posible pasar de una página a otra sin mayores complicaciones.

Normalmente, cuando se carga en memoria el fichero que se desea tratar, la página que aparece en la pantalla es la situada en la cabecera del buffer de edición. Una vez que el usuario haya efectuado los cambios oportunos en esta página, necesitará pasar a la siguiente para proseguir las operaciones. Para ello le bastará con pulsar la tecla P y accederá a la pantalla la siguiente página residente en memoria. También es posible ordenar el retroceso a la página previa, para lo cual tan sólo hay que seguir un procedimiento similar, con la única salvedad de que el cursor que señala el desplazamiento relativo de los diferentes comandos ha de estar apuntando hacia atrás. Este tipo de desplazamiento sobre la pantalla del ordenador recibe el nombre de "scrolling", y para su control también es posible recurrir al empleo de argumentos numéricos. Si se pulsa una tecla numérica, por ejemplo la tecla 3, y a continuación la tecla P se producirá un salto de 3 páginas hacia adelante o hacia atrás, dependiendo de la orientación que posea el indicador de desplazamiento localizado en el prompt del editor.

MODOS DE OPERACION DEL EDITOR. EL COMANDO SET

El editor estándar del sistema operativo UCSD p-System puede operar, según se ha visto, en dos modos básicos: el modo programa, a través del cual es posible el tratamiento de ficheros que contienen programas en Pascal, y el modo texto, a través del cual se pueden editar ficheros en lenguaje natural.

Tal selección de comportamiento se establece cambiando ciertos parámetros del editor; parámetros que están a disposición del usuario a través del comando SET por medio de sus diferentes opciones internas: autoindentación y rellenado, control de márgenes y gestión de párrafos.

Normalmente, el editor se coloca automáticamente en el modo PROGRAM con objeto de controlar la operación con ficheros de programas. La oportuna modificación de los parámetros permitirá el paso al modo TEXT.

Omnis 2 (2)

Diseño de una base de datos

Para completar el estudio de la base de datos OMNIS 2, vamos a profundizar en los aspectos prácticos del programa. En primer lugar se incluirá un sencillo ejemplo, que pretende ilustrar la forma en la que se puede diseñar una base de datos y sus pantallas asociadas. A continuación, se trabajará con la anterior base de datos, introduciendo y consultando la información en ella contenida.

DISEÑO DE LA BASE DE DATOS

Dentro del menú de opciones, OMNIS 2 ofrece el comando "Crear y Modificar", cuya misión consiste en definir las características de una nueva base de datos o modificar las de una ya existente. Esto es, si no existe una base de datos con el nombre indicado, el programa presentará una pantalla en blanco; por el contrario, si existe la referida base aparecerá en la pantalla la estructura del archivo, entendiendo por estructura tanto las posiciones donde se introducen y presentan datos (CAMPO), como los textos explicativos que describen el contenido de cada campo (ETIQUETA). En el caso de que el número de datos sea excesivo como para presentarlos en una única pantalla, se podrán utilizar varias pantallas para completar el diseño.
En la zona derecha de la pantalla se pueden hacer visibles o invisibles los dos comandos que facilitan el diseño:

- CAMPO

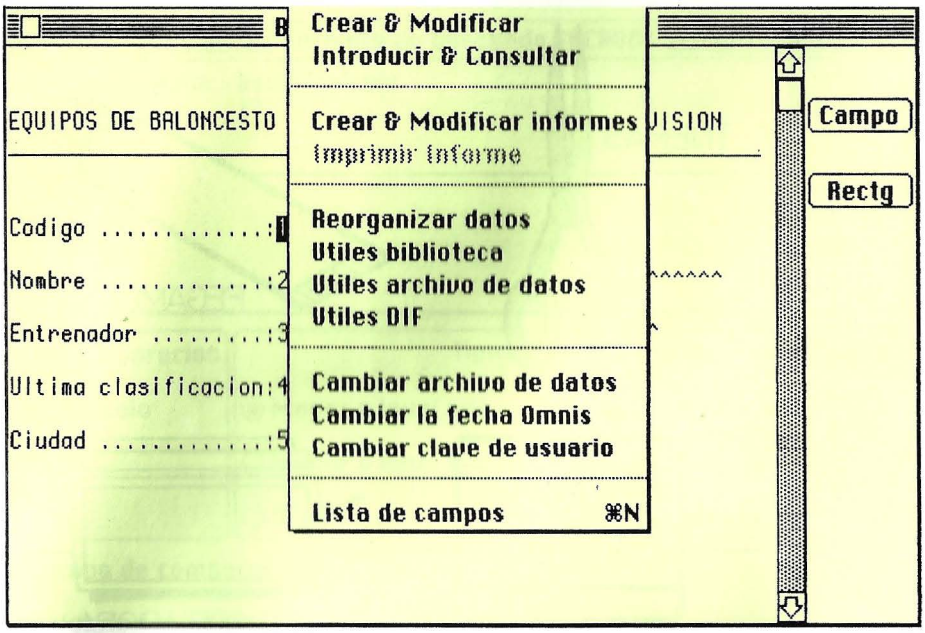
Permite introducir la descripción del campo y todos los atributos que lo definen; es decir: el número de orden del dato, su nombre, la longitud, el número de decimales, su naturaleza (caracteres, números, dato lógico, fecha o secuencia)... y, en definitiva, todos los atributos que el usuario puede necesitar para describir las características del dato.

- RECTANG

El comando rectángulo permite dibujar rectángulos negros que resaltarán algunos campos o permitirán aumentar la estética de la pantalla. La forma en la que el usuario define la ubicación del rectángulo no puede ser más sencilla: basta con indicar su vértice superior izquierdo y su vértice inferior derecho.
Además de los dos comandos anteriormente señalados para el diseño de la base de datos, se debe utilizar el ratón para situar las etiquetas y definir la primera posición de cada campo. En algunas ocasiones, puede resultar muy útil la selección de alguno de los comandos de edición: cortar, copiar, pegar y borrar; sus propios nombres son autodefinitorios y sirven, en general, para desplazar texto desde una parte a otra de la pantalla, e incluso desde una pantalla a otra.
Como última posibilidad para la creación y modificación de formatos, vamos a describir los comandos que ofrece del menú desplegable (pull-down) al efecto:

- BORRADO DE NOMBRES SIN USO

Cuando se ejecuta este campo, el programa solicita confirmación del usuario; en el caso de que ésta sea afirmativa, se anularán todos los nombres no utilizados.



El diseño de la base de datos se ha realizado seleccionando la opción "crear y modificar" dentro del menú denominado "opciones" (de tipo "pull-down"), el cual aparece desplegado en la figura.

● IMPRIMIR UNA LISTA DE CAMPOS
Su misión consiste en producir una lista con todos los campos incluidos en la base de datos; de esta forma el usuario podrá documentar todos los archivos mecanizados que utilice.

● DEFINIR LOS CARACTERES DEL RECTANGULO
Con este comando se seleccionan los caracteres, horizontal y vertical, que permiten dibujar un borde alrededor del rectángulo definido.

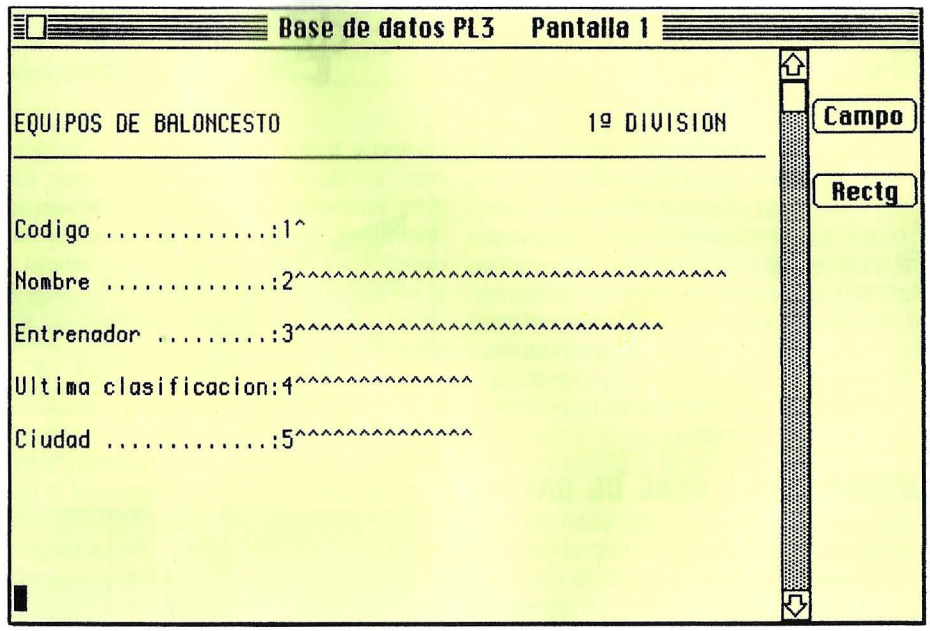
● PANTALLA DE 80 COLUMNAS
Mediante este comando se puede extender el área de datos de la pantalla hasta 80 columnas; en el caso de que no sea necesario una base datos diseñada sobre 80 columnas puede mantenerse el formato de 60 columnas.

INTRODUCCION Y CONSULTA DE DATOS

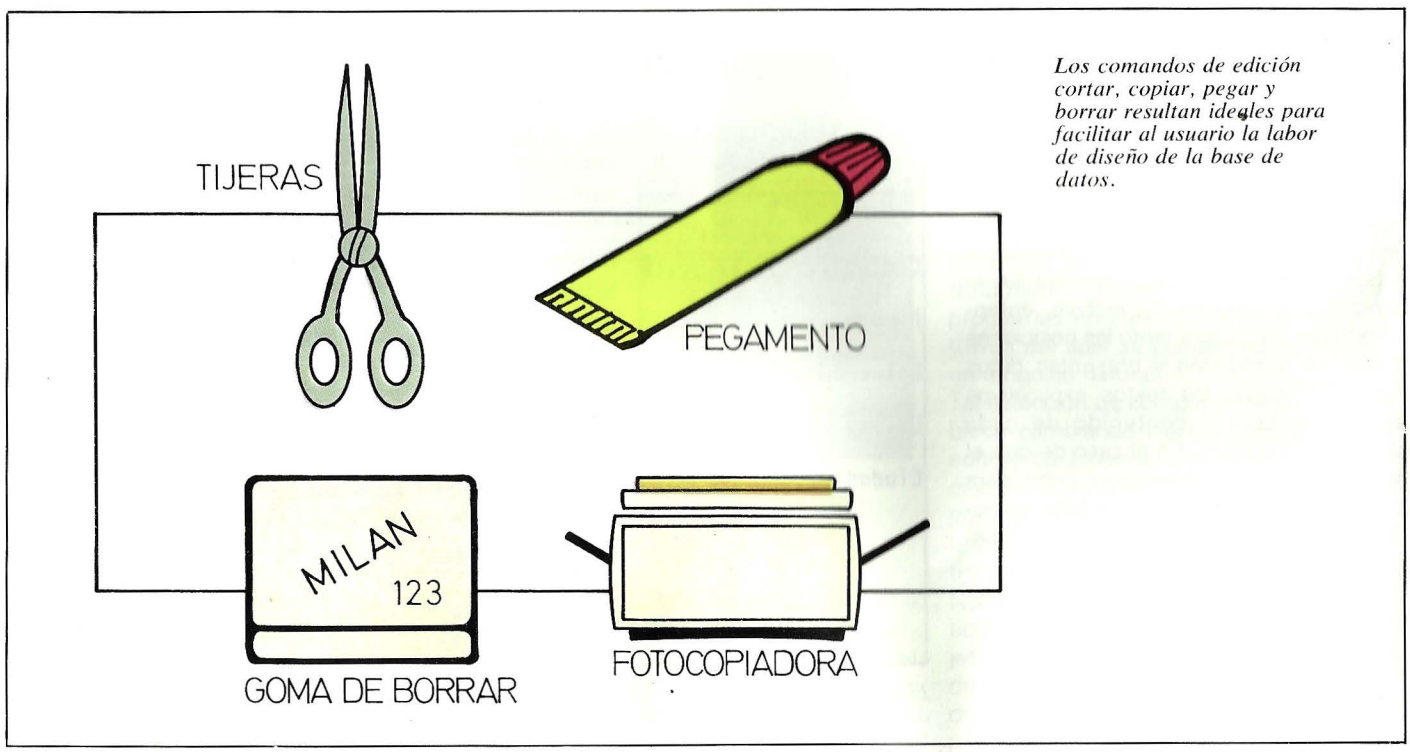
Si el usuario de un ordenador se toma la molestia de definir las características de una base de datos (en el caso de OMNIS 2, según hemos descrito en el párrafo anterior), lo menos que ésta le puede ofrecer a cambio es la posibilidad de introducir datos y consultarlos cómodamente después. A continuación, vamos a detallar algunos de los comandos que ofrece OMNIS 2 para este tipo de operaciones.

● COMANDOS DE PANTALLA
Se corresponden con los botones de selección que aparecen al lado derecho de la pantalla y representan las opciones básicas disponibles por el usuario:

1. SIGUIENTE
Presenta el siguiente registro de la base



Además de los textos de título y cabecera, los dos principales conceptos para el diseño los constituyen las etiquetas, que explican al significado de los datos, y los campos en donde éstos se introducirán.



Los comandos de edición cortar, copiar, pegar y borrar resultan ideales para facilitar al usuario la labor de diseño de la base de datos.

de datos; por supuesto siempre a partir de la última localización realizada.

2. ANTERIOR
Su misión es complementaria a la del comando anterior; es decir, presenta el registro anterior al último localizado.

3. ENCUENTRA
Permite localizar automáticamente un registro específico a partir de uno de los campos indexados en la base de datos.

4. INSERTA
Si se ordena la ejecución del comando INSERTA, el programa presentará una pantalla con las etiquetas oportunas, para que el usuario introduzca un nuevo registro en la base de datos.

5. EDITA
Permite editar el registro actual para visualizarlo y, en su caso, modificarlo.

6. BORRA
Su misión consiste en eliminar toda la información referente al registro actual. Antes de producir la eliminación física, el programa solicita confirmación del usuario.

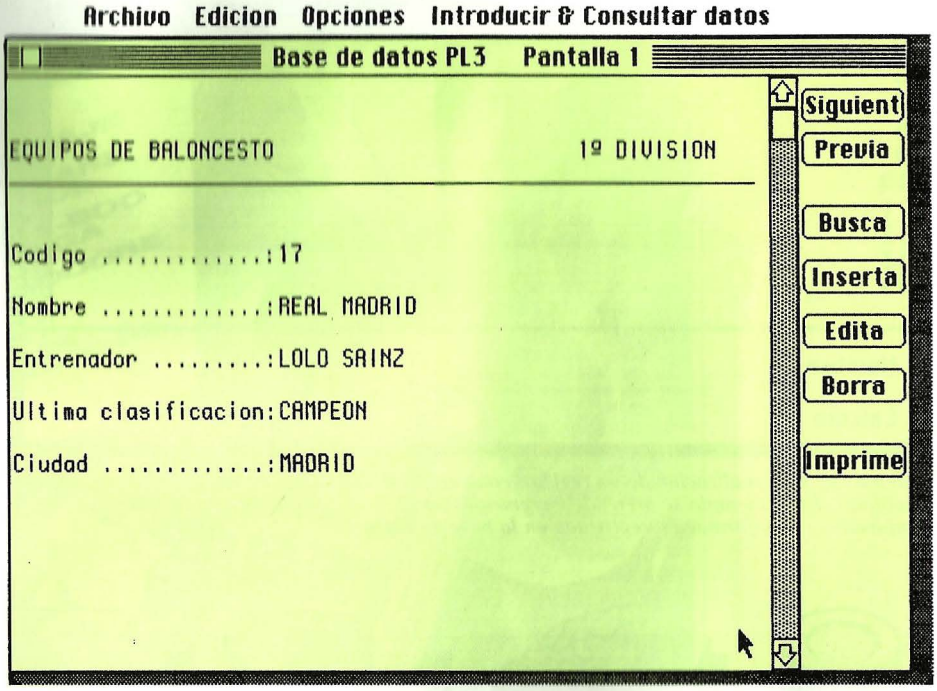
7. IMPRIME
Para completar esta lista de comandos elementales sólo falta permitir la impresión del registro activo; precisamente ésta es la labor encomendada al comando IMPRIME.

• COMANDOS DE BUSQUEDA
Además de los comandos elementales descritos en el punto anterior, existe una pantalla especial, con comandos asociados, para facilitar las búsquedas de información.
La pantalla de formato para la búsqueda presenta una serie de líneas, numeradas desde 1 hasta 50, en cada una de las cuales se deben introducir tres conceptos:

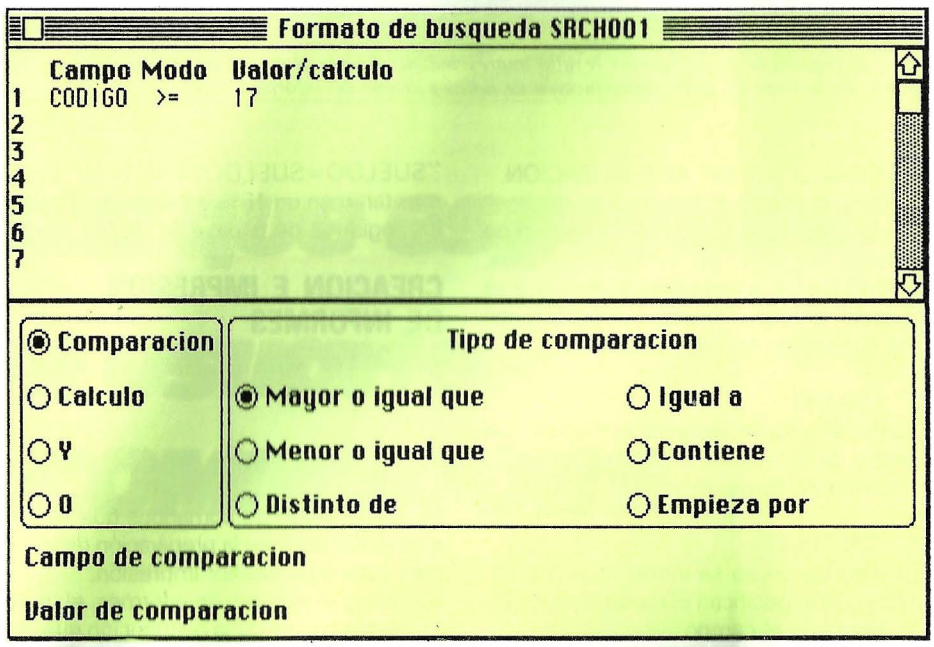
1. CAMPO
Debe contener el nombre de alguno de los campos de la base de datos.

2. MODO
Define el tipo de comparación que debe usarse para comprobar si un cierto campo tiene el valor deseado. Existen seis tipos de comparaciones: mayor o igual que, menor o igual que, distinto de, igual a, contiene y empieza por. También en este concepto se pueden incluir la conjunción (y) y la disyunción (o) para ligar las distintas líneas.

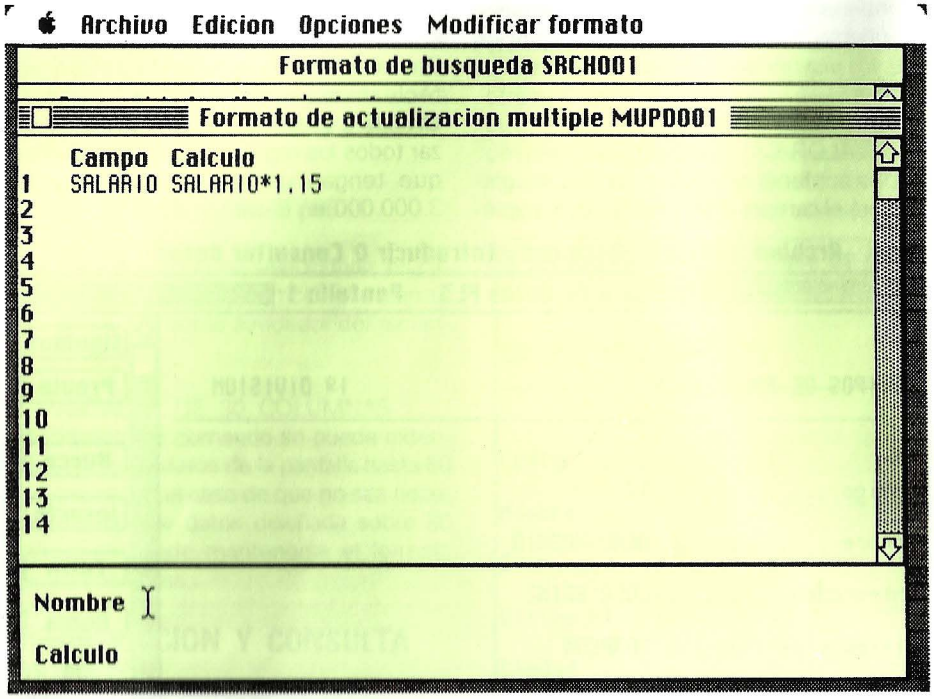
3. VALOR/CALCULO
Debe contener el valor con el que se comparará el campo, o en su defecto, la expresión que se desea calcular en función de algunos campos de la base de datos.
En resumen, si en una de estas líneas tecleamos: "SUELDO>=3.000.000" OMNIS 2 entenderá que se desean localizar todos los registros de la base de datos que tengan un valor mayor o igual a 3.000.000 en el campo SUELDO.



La definición de la base de datos influirá en las posibles entradas válidas para cada uno de los campos. En la figura se muestra un ejemplo de entrada de acuerdo al diseño de la figura anterior.



Reproducción del aspecto de la pantalla del ordenador para la busqueda en la base de datos de los registros que satisfagan la restricción impuesta.



Además de la actualización de un registro concreto, OMNIS 2 permite realizar actualizaciones múltiples. En el ejemplo se pretende incrementar un 15 % el dato de salario de todos los empleados con información registrada en la base de datos.

Para la impresión de un informe resulta imprescindible realizar al menos tres pasos previos: diseño de la base de datos, introducción de datos y diseño del informe.

- POSIBILIDAD DE ACTUALIZACION MULTIPLE

Para permitir al usuario la actualización de muchos registros de forma simultánea, OMNIS 2 ofrece una pantalla en la que se pueden distinguir líneas numeradas con dos conceptos en cada una de ellas:

1. CAMPO
Donde el usuario tecleará el nombre del campo de la base de datos que se desea actualizar de forma múltiple.

2. CALCULO
En este concepto se introducirán los cálculos que especifican el modo en que va a ser afectado el campo.
Continuando con el ejemplo anterior, al teclear en una línea: "SUELDO=SUELDO*1.15", se incrementarán en un 15% los salarios de todos los registros de la base de datos.

CREACION E IMPRESION DE INFORMES

El último grupo de comandos que vamos a describir permite la preparación de informes para su posterior impresión.
Mediante el formato de informes, el usuario puede introducir la distribución que desea para ubicar los datos a imprimir en el informe; fundamentalmente, se pueden utilizar los siguientes tipos de líneas imprimibles:

— ENCABEZAMIENTO
Se imprimirá al principio de cada página del informe.

— ENCABEZAMIENTO DE DETALLE
Permite definir cabeceras que "explicarán" el significado de las líneas que incluyen los datos.

— LINEAS DE DETALLE
Se imprimirá en cada uno de los registros que se incorporen en el informe.

— LINEAS DE SUBTOTAL
Esta sección se imprime siempre que cambien los valores de los campos que se determinen para subtotalizar. Se pueden utilizar hasta un máximo de 9 tipos de subtotales en el mismo informe.

— LINEA DE TOTAL
Se imprimirá al final del informe sumando los campos que el propio usuario haya decidido.
Para gestionar la edición de informes escritos, OMNIS 2 ofrece seis comandos al usuario:

1. FORMATO NUEVO
Permite que el usuario introduzca el nombre del formato que se desea preparar.

2. RENOMBRE
Su misión se limita a cambiar el nombre de un formato de informe diseñado previamente.

3. BORRA
Se puede utilizar para borrar un formato de informe que, bien por su diseño defectuoso, o bien por haber caído en desuso, no vayan a ser utilizados de nuevo.

4. IMPRIME
Obviamente, su misión consiste en producir la impresión del informe solicitado por el usuario.

5. MODIFICA
Este comando se utiliza para cambiar la distribución del informe.

6. CAMPOS
El último comando de esta sección permite establecer el tipo concreto de clasificación, subtotalización y saltos de página que se van a utilizar en un cierto informe.

Ficheros en los Amstrad

La coexistencia CP/M y AMSDOS

Una característica primordial de los ordenadores Amstrad es su capacidad de disponer del sistema operativo CP/M, el cual abre las puertas de un nuevo universo de aplicaciones.

EL SISTEMA DE DISCOS

Los modelos CPC664 y CPC6128 de la familia Amstrad incorporan en la parte derecha del teclado una unidad de disco de tres pulgadas, mientras que en el CPC464 este espacio es ocupado por un casete convencional. Para poder disfrutar en este último modelo de las prestaciones del sistema de discos, es necesario conectarle una unidad adicional a través del conector correspondiente.

También es posible conectar una segunda unidad de disco a cualquiera de los modelos Amstrad; ésta puede elegirse para trabajar con discos de tres pulgadas o de cinco y un cuarto. Cada cara de los discos, independientemente de su tamaño, tiene capacidad para almacenar unos 170K de información; aunque ello parezca mucho a simple vista, dicha capacidad puede resultar escasa para trabajar con algunos programas en CP/M, o para mantener una base de datos algo extensa.

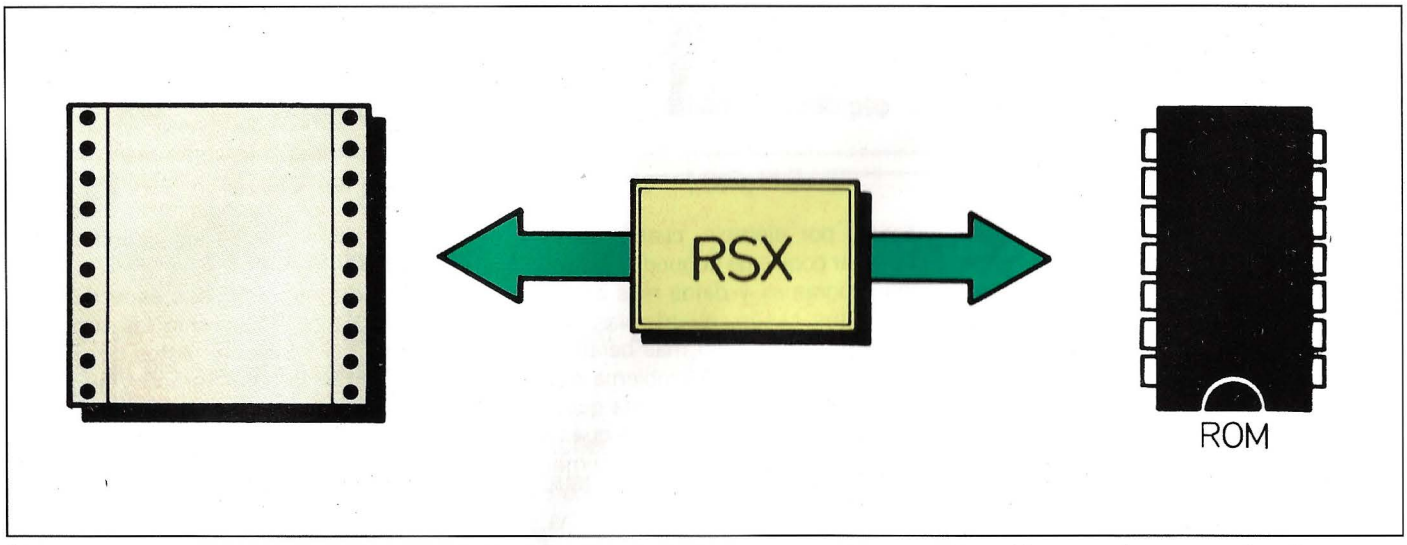
Nada más conectar el aparato, el sistema de discos queda bajo el mando del AMSDOS (acrónimo para "AMStrad Disk Operating System") el cual ofrece una serie de comandos, algunos de ellos muy parecidos a los del CP/M, para la gestión de ficheros y de los directorios.

Para pasar el control al CP/M se teclea un comando específico de AMSDOS; a partir de entonces se dispone de las órdenes típicas del CP/M, ya descritas en otro lugar de esta obra.

AMSDOS, UN S.O. DESDE BASIC

Como se ha comentado en el párrafo anterior, AMSDOS está disponible desde el momento en el que lo está el intérprete BASIC; esto es, desde que se conecta el ordenador. Esto sucede así ya que los comandos que proporciona el AMSDOS son, realmente, una serie de RSX implementados en ROM.

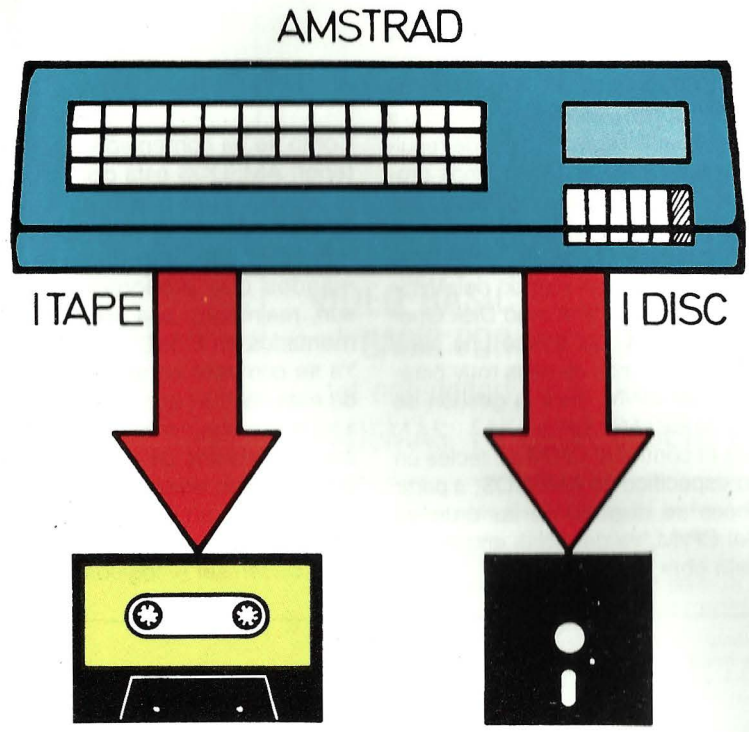
Ya se comentó en el capítulo precedente de esta sección que el usuario podía crear a su gusto nuevos comandos que siempre iban precedidos de una barra vertical. En la mayoría de las ocasiones, estos comandos residirán en la RAM del equipo, por lo que cada vez que se quiera hacer uso de ellos deben ser recogidos previamente de



Una vez que se ha diseñado un RSX, éste puede "enlatarse" en ROM de forma que esté siempre disponible nada más conectar el equipo.

RSH	Acción
|DISC.IN	Indica que las operaciones de lectura se harán sobre el disco.
|DISC.OUT	Especifica que las operaciones de escritura se realizarán sobre el disco.
|DISC	Señala que las operaciones de lectura y escritura se realizarán sobre disco.
|TAPE.IN	Indica que las operaciones de lectura se harán con la cinta.
|TAPE.OUT	Señala que las operaciones de escritura se harán con la cinta.
|TAPE	Tanto las operaciones de lectura como de escritura se realizarán sobre cinta.

La posibilidad de dos tipos de almacenamiento externo obliga a la existencia de comandos que especifiquen hacia cuál de ellos se han de dirigir las operaciones de entrada y salida.



cinta o disco. Otra opción consiste en que una vez programados puedan convertirse en parte indeleble del conjunto de comandos que proporciona el equipo si son grabados en ROM; precisamente esto es lo que ha ocurrido con los RSX que constituyen el sistema operativo AMSDOS.

Pese a que las ventajas que presenta el sistema de discos respecto al de cintas son incuestionables, hay ocasiones en las que disponer de un almacenamiento masivo sobre casetes puede resultar interesante; por ejemplo, cuando se trata de obtener copias de seguridad (back-ups) de los programas y datos más importantes. En estos casos, guardar las copias en cinta resulta mucho más barato que hacerlo en disco, y el problema del mayor tiempo de acceso a la cinta queda compensado por la esperanza de que no habrá que utilizarlas muy frecuentemente. Al efecto, AMSDOS proporciona los comandos oportunos para "redirigir" las entradas y salidas, bien sea hacia el disco o hacia la cinta. Estos comandos están reflejados en el cuadro adjunto.

En la correspondiente tabla aparecen los restantes comandos que proporciona AMSDOS para la gestión de los ficheros y del espacio en disco. Se observa la presencia del comando |USER, que permite realizar hasta 16 particiones del disco (|USER,0... |USER,15), lo cual resulta especialmente útil cuando la escasez de discos obliga a tener directorios excesivamente poblados. Mediante este comando es posible dividir el disco en secciones para programas BASIC, ficheros en código máquina, ficheros fuente Pascal, etc., de forma que en todo momento una lectura del directorio por pantalla sólo muestre los ficheros que sean útiles en ese instante, mejorando la presentación.

Los nombres de ficheros que maneja el AMSDOS son análogos en formato a los que se utilizan en CP/M; es decir:

<nombre de fichero>. <extensión>

donde "nombre de fichero" es un campo de ocho caracteres como máximo y "extensión" de tres, también como máximo. Los símbolos comodín ("*" y "?") son tratados de forma análoga a como lo son en CP/M.

Aparte de los comandos específicos del AMSDOS, se encuentran los comandos propios del intérprete; éstos no van precedidos por la barra vertical y se utilizan para la carga en memoria y grabación en disco de programas: SAVE, LOAD, MERGE, etc.

La función de volcar un programa que está en memoria sobre disco o cinta se realiza a través de la orden SAVE. Por ejemplo:

SAVE "unfich"

hará que, en el caso de utilizar el disco como dispositivo de destino, aparezca en el mismo un fichero, denominado "UNFICH.BAS" (la extensión la pone automáticamente el sistema), conteniendo el programa BASIC que en ese momento reside en memoria. Si la orden fuera:

SAVE "unfich",P

el fichero quedaría protegido, de tal forma que sólo sería posible su ejecución, sin posibilidad de obtener un listado.
Al ejecutar la orden:

SAVE "unfich",A

el fichero se graba en modo ASCII; es decir, tal y como se ve al sacar un listado.
Para grabar una zona de memoria en general (que puede ser el contenido de la pantalla o un bloque de código máquina) el parámetro a utilizar es "B", en cuyo caso también habría que especificar el comienzo y la longitud de la zona de memoria a volcar. Por ejemplo:

SAVE "pantalla",B,&C000,&4000

crea en disco el fichero "PANTALLA.BIN" (de nuevo, la extensión es adjudicada directamente al detectar el parámetro "B") que contiene la información sobre la pantalla visualizada en ese momento.

CP/M, LA ESTRELLA DEL CONJUNTO

Sin duda, lo que hace que los ordenadores Amstrad se encuentren en la frontera entre el juguete y la utilidad práctica es la posibilidad de trabajar en CP/M; sistema operativo que ofrece un entorno ideal para los que velan sus primeras armas en informática, e incluso para la ejecución de aplicaciones profesionales.
Tanto el CPC464 como el CPC664 operan con la versión 2.2 de este sistema operativo, con el serio inconveniente de la escasa memoria RAM que queda para los programas de usuario, concretamente 44K. Por esta razón, los grandes paquetes de software diseñados para CP/M, como el WordStar, no pueden ejecutarse en estos aparatos. No obstante, si son ejecutables programas más pequeños, como compiladores de Fortran y Cobol de la firma Microsoft.
Esta limitación de la memoria RAM disponible bajo CP/M queda resuelta en el modelo CPC6128, que incorpora 128K de RAM, el doble que sus predecesores, además de la versión 3.0 de CP/M. Esta última incorpora algunos comandos nuevos, así como un paquete destinado a aprovechar las capacidades gráficas del aparato desde el CP/M, las cuales no pueden ser utilizadas en los modelos precedentes.
La compatibilidad entre los ficheros que manejan AMSDOS y CP/M sólo se mantiene a nivel de ficheros de texto. En consecuencia, estando bajo el control del AMSDOS, no se puede realizar un LOAD de un fichero .COM, aunque sí se pueden

Principales comandos del AMSDOS

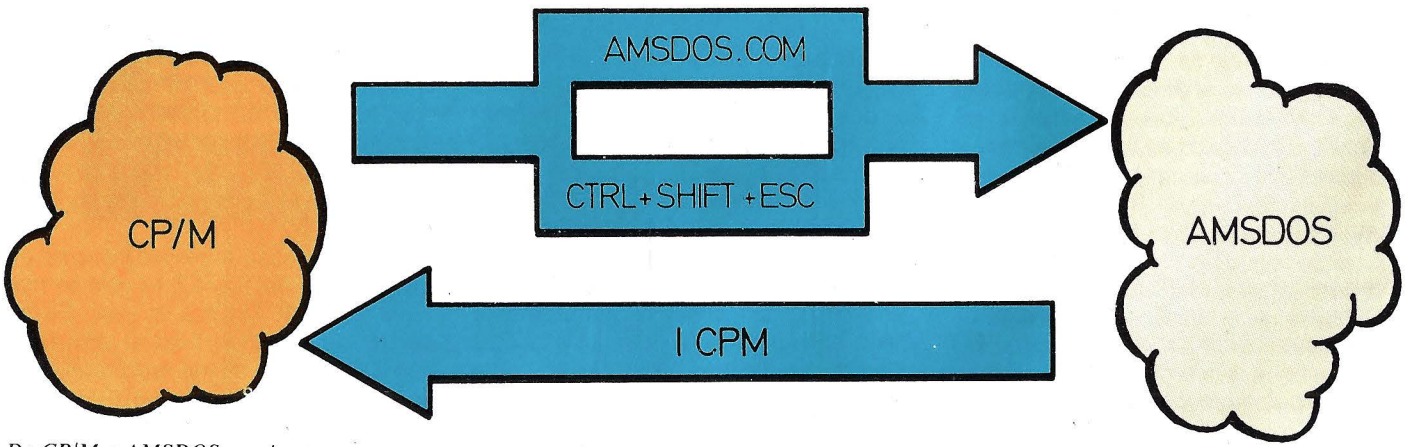
RSH	Acción	Ejemplo
| DIR	Listado por pantalla del directorio del disco.	| DIR,"*.BAS"
| A	La unidad de disco implícito será la "A".	| A
| B	Como el anterior para la unidad "B".	| B
| CPM	Pasa el control al CP/M.	| CPM
| ERA	Borrado de ficheros.	| ERA,"PRUEBA.*"
| REN	Cambio del nombre de un fichero.	| REN,"NUEVO.BAS","VIEJO.BAS"
| USER	Definición de un área de usuario.	| USER,5

Principales comandos del AMSDOS.

Pese a que la existencia de un directorio en disco hace innecesario que los ficheros cuyo soporte sea este medio incluyan una cabecera, con el fin de guardar la compatibilidad con los ficheros creados en cinta aquellos cuyo destino sea el disco también incorporarán una cabecera.

manipular los ficheros de texto que hubieran sido creados por un editor desde CP/M, por ejemplo. Ello se debe a que los ficheros generados por los programas almacenados con el comando SAVE sin especificar la opción "A" desde ASMDOS, van precedidos por una cabecera en la que se incluyen diversas características del fichero: tipo (Basic, binario, ASCII, etc.), protección, longitud, zona de carga... Mientras que los ficheros en CP/M no llevan este tipo de información. Por esta razón, al interntar desde el AMSDOS cargar un fichero generado por CP/M y detectar la ausencia de cabecera, ocurre un error.

Como se ha comentado, los ficheros que son grabados con la opción "A" no incorporan dicha cabecera, por lo que pueden



De CP/M a AMSDOS.. y viceversa.

PROGRAMA 1

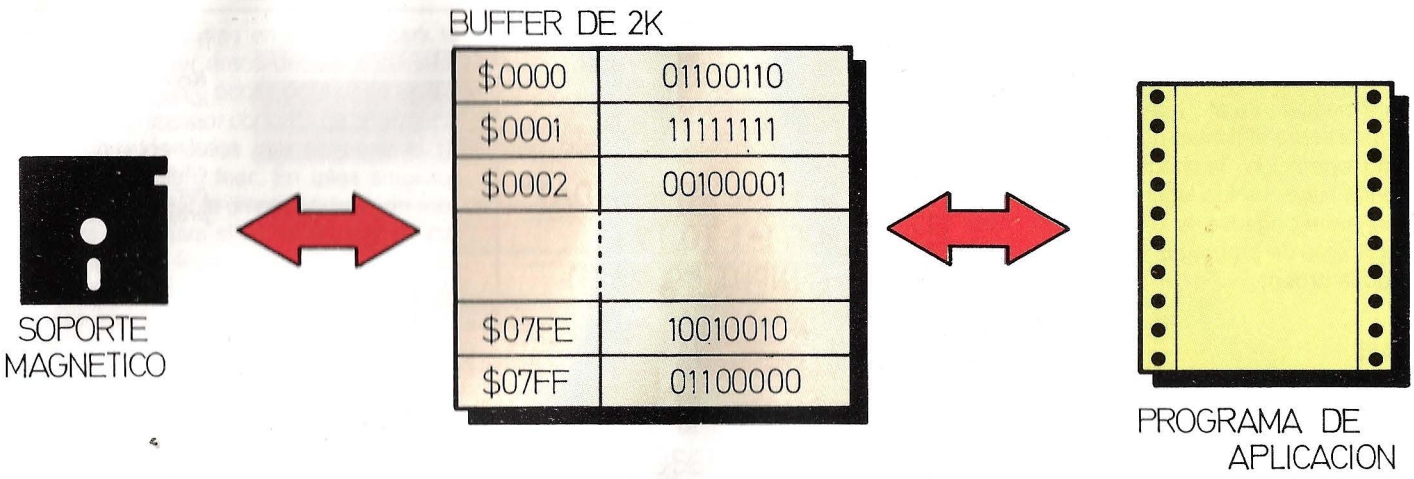
```
10 INPUT "Comando ", a$
20 IF a$="directorio" THEN |DIR;"*.*"
30 IF a$="a cpm" THEN |CPM
40 GOTO 10
```

Con este sencillo programa, el usuario puede redefinir a su medida algunos de los comandos del AMSDOS.

ser utilizados sin problemas en el entorno CP/M.

Entre los comandos CP/M que se encuentran en el disco de sistema, hay algunos que están orientados a facilitar el uso de este sistema operativo con los ordenadores Amstrad. Por ejemplo, destacan CSAVE y CLOAD, para transferir contenidos de ficheros de cinta a disco y viceversa, y FILECOPY para transferir ficheros de un disco a otro sin necesidad de disponer de una segunda unidad. Entre los ausentes cabe mencionar a SAVE, que se utiliza para guardar páginas de memoria en disco, y que puede ser de utilidad a los programadores avanzados en CP/M.

Para pasar al AMSDOS desde el CP/M, hay que ejecutar un pequeño fichero de apenas unos bytes de longitud llamado AMSDOS.COM el cual devuelve el control al intérprete Basic. Otra posibilidad alter-



Los bytes que están sobre el disco no dan un valor a las variables BASIC directamente, sino que van llenando una zona de memoria y de ahí interaccionan con el programa a medida que son necesarios.

nativa es efectuar un reset del aparato, presionando simultáneamente las teclas CTRL, SHIFT y ESC. Si se desea pasar a CP/M desde el control del AMSDOS, hay que teclear la orden:

```
|CPM
```

que, al igual que los restantes comandos del AMSDOS, se trata de un RSX al ir precedido por una barra vertical (ver figura).

GESTION DE FICHEROS

Todos los comandos de AMSDOS son interpretados desde el Locomotive Basic, lo cual permite al programador operar con ellos desde el propio intérprete (ver ejemplo de redefinición de comandos en el programa 1).

El usuario puede crear y gestionar sus propios ficheros desde el Basic. Para ello, antes de utilizarlos debe abrirlos. Una vez que haya terminado su manipulación tendrá que cerrarlos. El Locomotive Basic impone la restricción de que sólo puede estar abierto un fichero para lectura y otro para escritura en cada instante; además, todos los ficheros serán de acceso secuencial.

Para gestionar un fichero, ya sea de entrada o de salida, se reservan 2K de memoria RAM que actuarán como buffer entre el alojamiento final en RAM y el soporte magnético que representa el disco o la cinta.

El programa 2 incluye un ejemplo en el que se ha abierto un fichero para almacenar el contenido del array "a$". El primer paso a dar consiste en crear dicho fichero con la orden OPENOUT, la cual va seguida del nombre y la extensión de tal fichero.

Hay que destacar que el nombre que aparecerá al listar el directorio del disco será exactamente el que figura entre las dobles comillas. Por lo tanto, si no da una extensión al fichero, como sucedería de ejecutar:

```
OPENOUT "PRUEBA"
```

en el directorio no aparecerá extensión alguna para el mismo. Lo contrario ocurre al utilizar el comando SAVE, con el que las extensiones se asignan automáticamente.

Como se indicó en el capítulo anterior, dedicado al Locomotive Basic, las órdenes básicas de entrada/salida —fundamentalmente PRINT e INPUT— pueden ir seguidas por un número que especifica el cauce por el que se realizará la operación. Concretamente, en el programa adjunto el cauce asignado es el número nueve, que corresponde a la vía de comunicación con el sistema de disco/cinta.

Una vez que se ha enviado toda la información al fichero de destino, hay que cerrarlo con la orden CLOSEOUT. El hecho de que no haya que especificar ahora el

PROGRAMA 2

```
DIM a$(10)
...
...
...
OPENOUT "PRUEBA.DAT"
FOR i=1 TO 10
   PRINT #9, a$(i)
NEXT i
CLOSEOUT
```

Almacenamiento del "array" a$ en un fichero.

fichero que se cierra se debe a que, como se ha comentado unas líneas más arriba, sólo es posible tener abierto un fichero para salida; de ahí que no exista duda alguna sobre el fichero que hay que cerrar. Esta operación, realizada sobre el disco podría haber tenido lugar sobre la cinta si, con anterioridad a la ejecución de este segmento de programa, se hubiera ejecutado la orden:

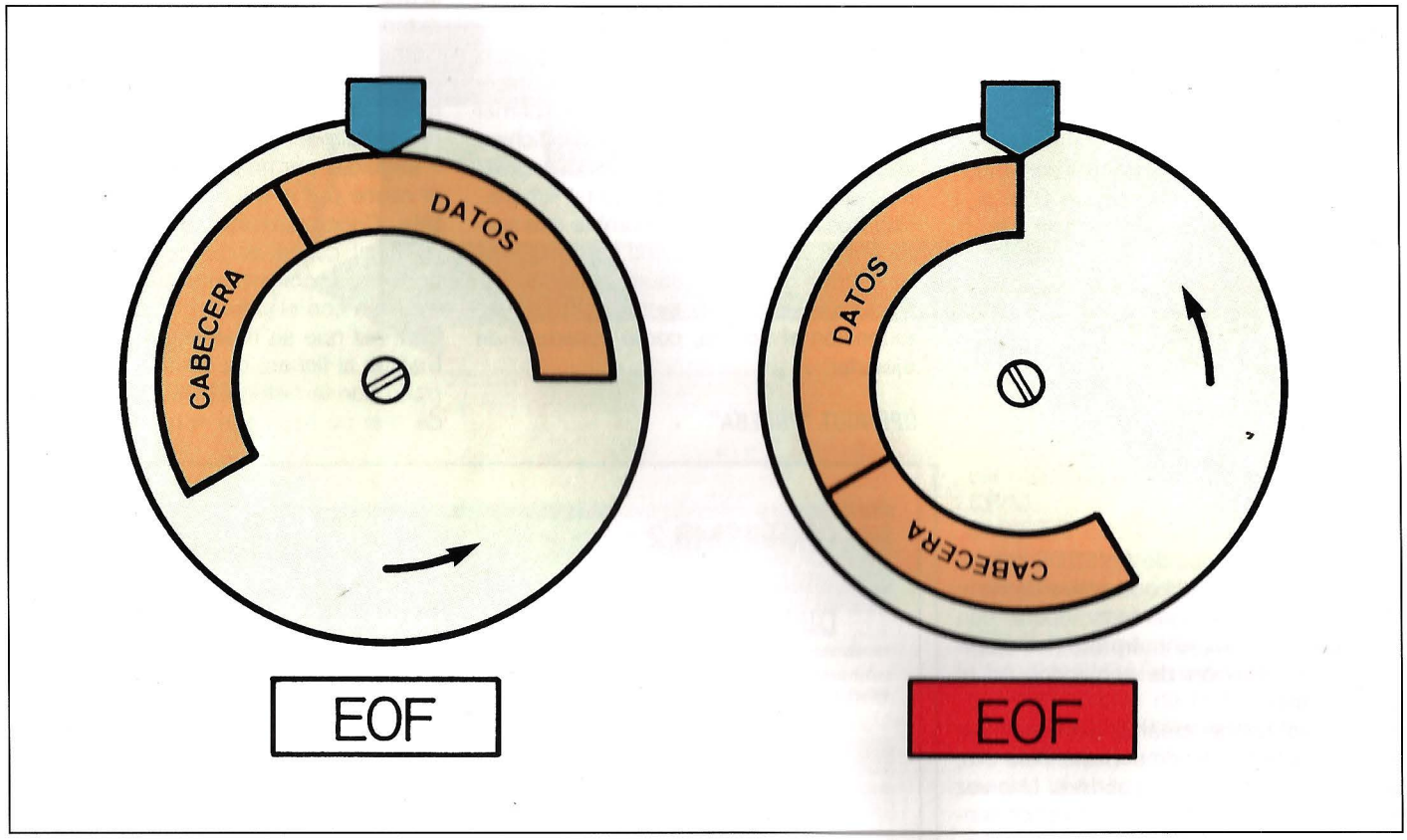
```
|TAPE
ó
|TAPE.OUT
```

bien incluyéndola como línea del programa, o bien tecleándola directamente con anterioridad a la ejecución del mismo. Cuando se conecta el ordenador por primera vez, toda la entrada/salida a través

PROGRAMA 3

```
OPENIN "PRUEBA.DAT"
FOR i=1 TO 10
   INPUT #9, a$ (i)
NEXT i
CLOSEIN
```

Recuperación de los valores de a$ almacenados en un fichero por medio del programa 2.



El valor de la variable EOF permite reconocer el momento en el que se llega al final del fichero.

del cauce número nueve está orientada hacia el disco.

Una vez que se han grabado los datos sobre el disco, lo más normal es pensar en recuperarlos; al respecto, el programa 3 ofrece una alternativa. En él se observa la forma de abrir un fichero para lectura (OPENIN), para después cerrarlo con CLOSEIN. De nuevo, es innecesario especificar el nombre del fichero que se cierra por la restricción ya comentada.

Existe una alternativa más potente que la

reflejada en la rutina en cuestión para la tarea de recoger datos del disco o cinta. En realidad, habrá pocas ocasiones en las que el programador conozca de antemano el número de datos que contiene el fichero que quiere leer. En tales situaciones, sería deseable disponer de algún medio que informara si se ha llegado al final del mismo e ir leyendo los datos mientras realmente se puedan ir cogiendo. Al efecto se dispone de la variable EOF, la cual es cierta cuando se llega al final del fichero abierto para lectura. En el programa 4 se muestra el uso típico de esta variable. El empleo de la estructura WHILE-WEND acompañando a la variable EOF es un buen ejemplo de matrimonio perfecto, el cual ha sido copiado de otros lenguajes más evolucionados como son C o Pascal.

Los ficheros creados pueden guardar cualquier tipo de información, ya sea numérica o literal, quedando bajo la responsabilidad de quien posteriormente utilice el fichero el recoger la información de la misma forma en la que fue depositada. Como se observa en los programas 5 y 6, la información se graba y se recupera en forma de parejas (dato-literal, dato-numérico). Un intento de realizar la entrada de los datos en el programa a través de la orden:

```
INPUT#9,b(i),a$(i)
```

puede traer malas consecuencias para la correcta ejecución del programa.

PROGRAMA 4

```
OPENIN "PRUEBA.DAT"
i=1
WHILE NOT EOF
    INPUT #9, a$(i)
    i=i+1
WEND
CLOSEIN
```

Uso típico de la variable EOF.

UNA APLICACION: BASIC EN ESPAÑOL

Los ficheros creados a partir de la orden OPENOUT tienen la particularidad de que, en el caso de que su contenido sean programas Basic, es posible cargarlos en memoria y ejecutarlos como si realmente hubieran sido tecleados desde el propio editor del Locomotive Basic.

En el cuadro final se muestra un ejemplo claramente ilustrativo. En un primer paso se crea el fichero "EJEMPLO.BAS" a través de OPENOUT y CLOSEOUT cuyo contenido es:

```
10 PRINT PI
```

Esto es justamente una línea de un programa Basic la cual, en el caso de aparecer en un programa, imprimiría por pantalla el valor 3.14159265. Por lo tanto, lo que contiene el fichero recién creado es un pequeño programa en Basic que escribirá dicho valor por pantalla. A continuación, podemos cargar el fichero "EJEMPLO.BAS" y ejecutarlo, obteniendo el efecto deseado.

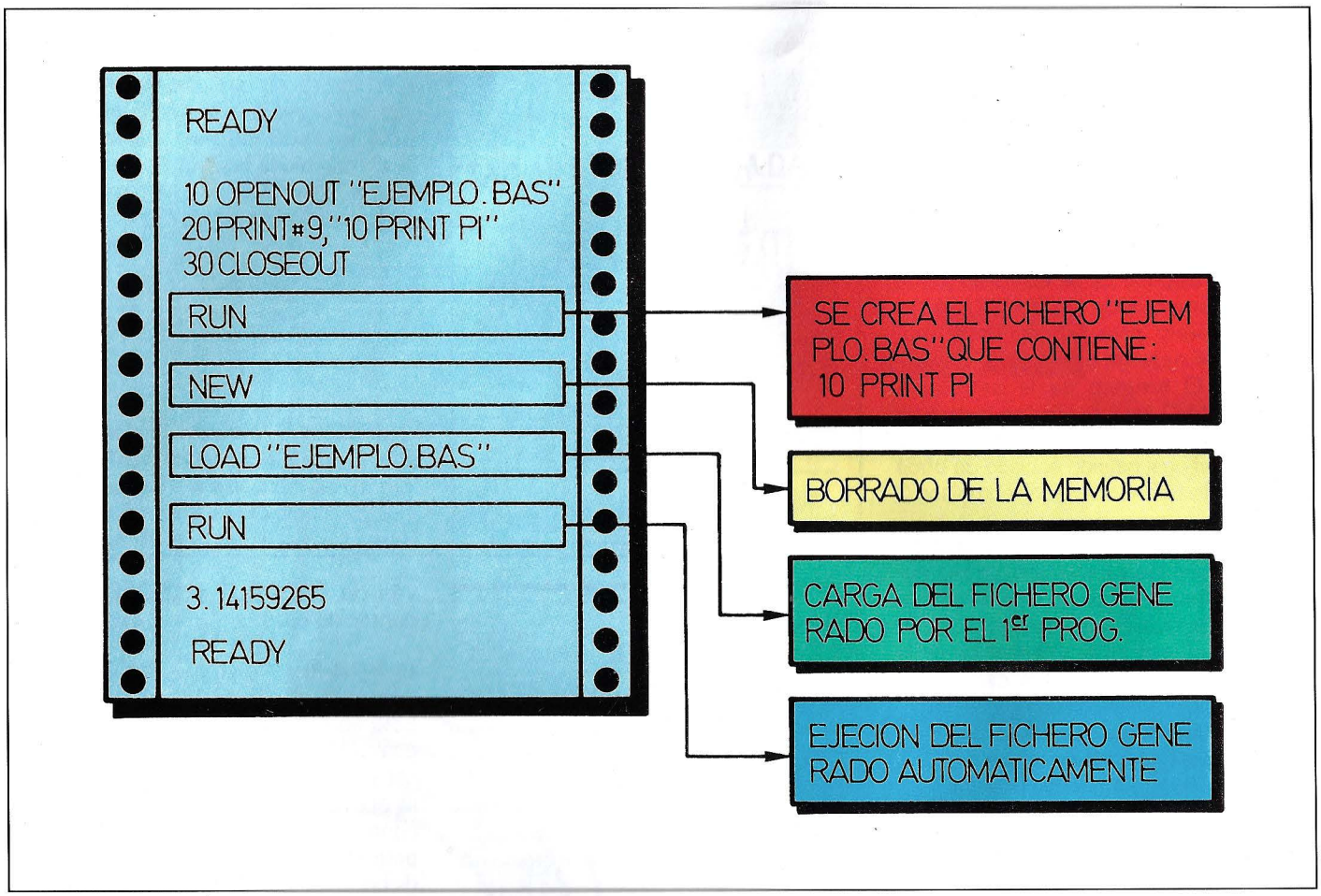
PROGRAMA 5

```
DIM a$(5), b(5)
...
...
...
OPENOUT "DATOS.DDD"
FOR i=1 TO 5
    PRINT #9, a$(i), b(i)
NEXT i
CLOSEOUT
```

PROGRAMA 6

```
OPENIN "DATOS.DDD"
i=1
WHILE NOT EOF
    INPUT #9, a$(i), b(i)
    i=i+1
WEND
CLOSEIN
```

Grabación y recuperación de datos en un fichero.



Actuación de un fichero creado por medio de la orden OPENOUT.

El resultado es análogo al que se obtendría si dicho programa se hubiera cargado en memoria por medio del editor, y luego se hubiera pasado a disco ejecutando un SAVE.
Todo esto permite pensar en un programa capaz de leer cadenas cuyo contenido sea un programa Basic en español, y de generar un fichero .BAS con el equivalente en auténtico Basic. Posteriormente, este fichero puede ser cargado en memoria y ejecutado por el intérprete. Un array de strings (lin$) contendría el programa fuente de forma parecida a:

```
lin$(1)="10 PARA I=1 HASTA 10"
lin$(2)="20 ESCRIBE I"
lin$(3)="30 SIGUIENTE I"
```

El "compilador" de Basic-español a Basic-Real debería analizar los contenidos de lin$(1), lin$(2), etc., y generar sus equivalentes en inglés, para después grabarlos en disco como sigue:

```
PRINT #9,"10 FOR I=1 TO 10"
PRINT #9,"20 PRINT I"
PRINT #9,"30 NEXT I"
...
```

El fichero creado de esta forma puede ahora ser cargado en memoria y ejecutado como si de un programa Basic normal se tratara; con la particularidad de que su generación se ha realizado de forma automática, sin necesidad de haber pasado previamente por el editor del intérprete Basic.

PROLOG

El lenguaje de la inteligencia artificial

Con PROLOG se abre un nuevo camino en el panorama de la revolución informática. No se trata de modificar los cánones de la programación, como hizo el advenimiento de la programación estructurada; ni de aprovechar las posibilidades de los ordenadores con varias CPUs en el tema del procesamiento paralelo, como han hecho ADA y otros lenguajes. PROLOG supone un cambio radical de la forma en la que se entendía la programación hasta ahora.

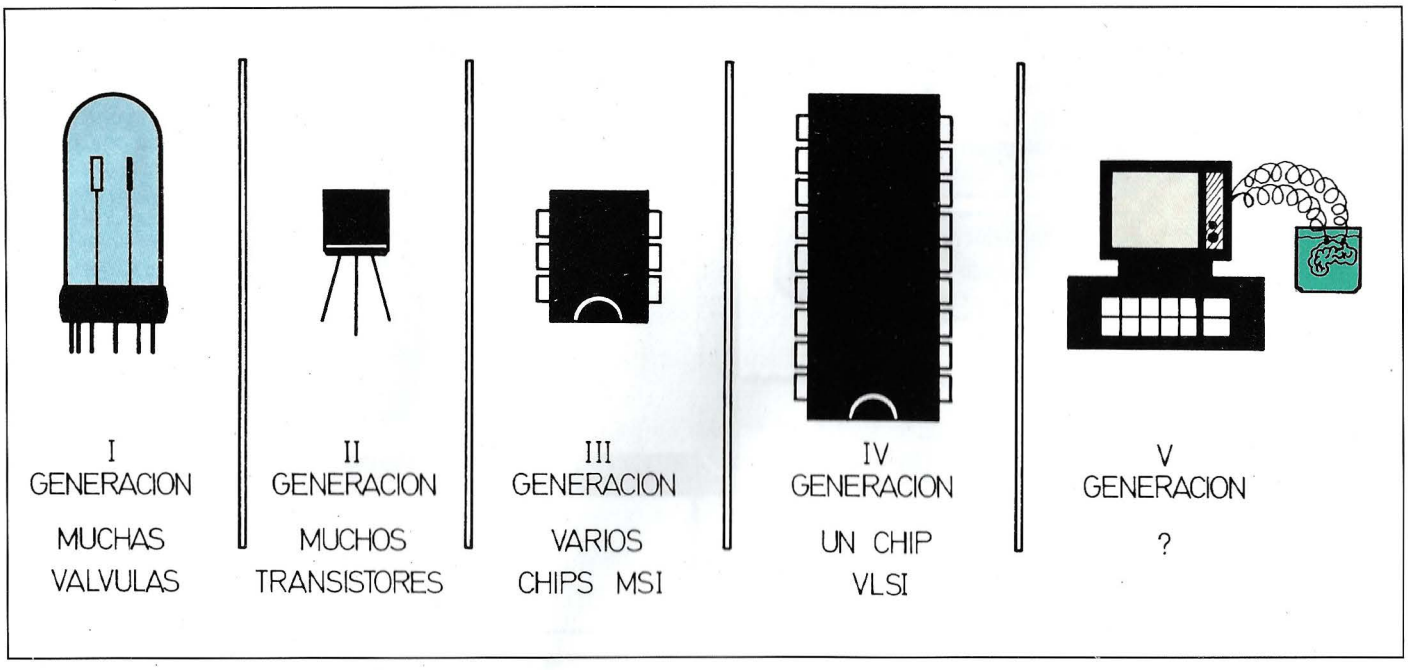
LA QUINTA GENERACION DE ORDENADORES

Como se observa en la figura, en la evolución de estas máquinas que llamamos ordenadores han existido cuatro etapas caracterizadas por una tecnología de fabricación. La quinta generación de ordenadores se caracterizará no por la aplicación de una nueva tecnología, sino por un cambio de las estructuras y funciones de las nuevas computadoras.

El objetivo fijado consiste en conseguir en un plazo de diez años —que comenzó en 1982— un prototipo de los ordenadores del futuro. Ordenadores que serán capaces de llevar a cabo y de mejorar incluso algunos aspectos de la actividad inteligente del hombre, los cuales dependen del propio conocimiento acumulado y de la capacidad de razonamiento humano. La pregunta a la que se espera dar respuesta al finalizar este plazo es: ¿cómo podemos dotar a las máquinas de las cualidades específicas de la raza humana?

Para cumplir el citado objetivo, aparte de tener que realizar un cambio profundo en la estructura y funcionamiento de los ordenadores tradicionales, se necesitarán nuevos lenguajes de programación que sean capaces de expresar este nuevo modo de pensar. De ahí la aparición de los lenguajes "declarativos", en contraposición con los "imperativos" como el BASIC, PASCAL o ADA.

Los programas escritos en esta nueva categoría de lenguajes se construyen a base del enunciado de una serie de hechos generales y de reglas que relacionen estos hechos; todos juntos serán manipulados convenientemente por el ordenador para producir una respuesta. Comparemos este punto de vista con el clásico que consiste en establecer de forma inequívoca una serie de sentencias (imperativos) que el ordenador seguirá inexorablemente para llegar a la solución. PROLOG y



De la válvula a la máquina inteligente en cinco generaciones de ordenadores.

sus derivados caen de lleno dentro del grupo de los lenguajes declarativos.

EL LENGUAJE PROLOG

PROLOG significa "PROgramming in LOGic", lo cual da una idea aproximada de lo que se pretende con él. Una de sus variantes llamada "micro-PROLOG" es una implementación que permite ejecutar programas en PROLOG en microordenadores; básicamente, consiste en una revisión sintáctica del original. En los ejemplos que siguen utilizaremos la sintaxis del micro-PROLOG por ser más sencilla de entender, aunque nos referiremos a él como si de PROLOG se tratase.
En un programa PROLOG cabe distinguir dos partes que, aunque no se diferencian físicamente en el listado, tienen cometidos absolutamente distintos (ver figura): un conjunto de relaciones y una serie de reglas.
Suponga que queremos establecer una base de datos que describa las relaciones familiares entre un grupo de individuos. En dicha base existirán cosas como:

Enrique es padre de Enriquito
Enrique es padre de María
...

De hecho, estas dos oraciones son prácticamente "sentencias" (aunque el término carece de sentido en PROLOG) del nuevo lenguaje. Para incluir estas relaciones en nuestro programa PROLOG diríamos:

```
&.add(Enrique padre-de Enriquito)
&.add(Enrique padre-de María)
```

Con el símbolo "&." PROLOG invita a introducir un comando. El usuario habría tecleado lo que va a su derecha. "add" es una palabra reservada con la que se indica que lo que sigue entre paréntesis debe pasar a formar parte de la base de datos. Observamos que una de las posibles formas de presentar a PROLOG una relación ofrece el siguiente aspecto:

(nombre de individuo)(nombre de relación)(nombre de individuo)

que concuerda plenamente con los ejemplos anteriores.

Una vez que se tienen en la base de datos las relaciones arriba expuestas, pueden ya realizarse preguntas sobre ellas.
Por ejemplo, a través de:

```
&.which(x : Enrique padre-de x)
```

estamos pidiendo al PROLOG que nos diga los hijos de Enrique que él conozca. La pregunta se enuncia: "cuáles son los x tales que (:) Enrique es padre de x". Su respuesta sería:

```
Enriquito
María
No (more) answers
&.
```

También podemos pedir al PROLOG que nos confirme algún hecho, lo cual se hace a través de "is". Por ejemplo:

```
&.is(Enrique padre-de Enriquito)
```

a lo que se nos contestará con el término:

```
YES
&.
```

Para preguntar si Pepita tiene padre diríamos:

```
&.is(x padre-de Pepita)
NO
&.
```

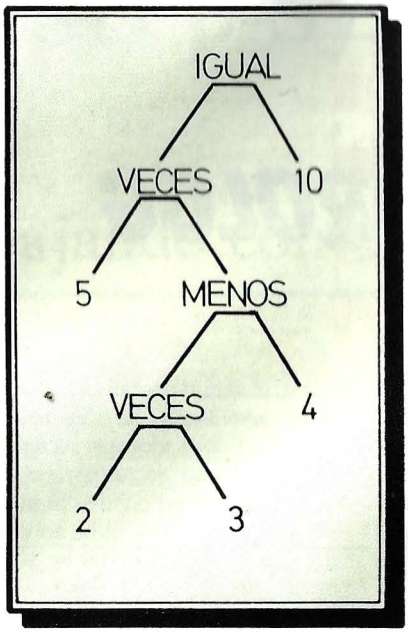
Lo cual es lógico examinando las relaciones que tenemos en la base de datos.
Sin embargo, el poder del PROLOG se concentra en la capacidad para manejar reglas que permiten establecer nuevas relaciones sin tener que declararlas explícitamente como hemos hecho hasta ahora. Suponga que borramos la base de datos original e introducimos:

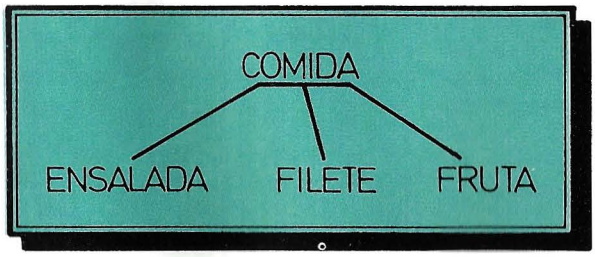
```
&.add(Enrique padre-de Enriquito)
&.add(Enriquito padre-de Isabel)
```

Estas dos relaciones definen implícitamente la relación "abuelo-de" entre Enrique e Isabel. Para comunicárselo explícitamente al PROLOG puede utilizarse la siguiente formulación:

```
&.add(Enrique abuelo-de Isabel)
```

El programa elaborado en PROLOG se almacena en una base de datos relacional, en la que se incluyen tanto las propias relaciones como las reglas para jugar con ellas.





10 = 5 x ((2 x 3) - 4)

El PROLOG almacena los conocimientos relacionales en forma de árboles etiquetados, en los que cada nudo tiene una etiqueta. En la figura se observa cómo es posible representar una igualdad matemática y cómo expresar el hecho de que una comida se compone de tres platos.

Los sistemas expertos

Otro aspecto al que PROLOG se adapta especialmente bien es el campo de los Sistemas Expertos. Estos son, básicamente, programas capaces de almacenar y asimilar el conocimiento que previamente han adquirido de un experto humano en un área determinada, para que, con posterioridad, otras personas no tan especializadas puedan pedir consejo y recibir explicaciones del propio programa. Hasta la fecha se han construido un buen número de Sistemas Expertos que funcionan con mayor o menor fortuna en diversas áreas de conocimiento. Destacan por su cantidad y calidad los dedicados a aplicaciones médicas, como MYCIN, versado en la diagnosis de enfermedades infecciosas de la sangre, y ONCOCIN, experto en el tratamiento de algunos tipos de cáncer.

Los Sistemas Expertos se enfrentan con problemas de representación del conocimiento, de imprecisión en el mismo, de comprensión del lenguaje natural cuando hacen las preguntas los "no expertos", etc. El PROLOG se perfila como la herramienta básica para su construcción.

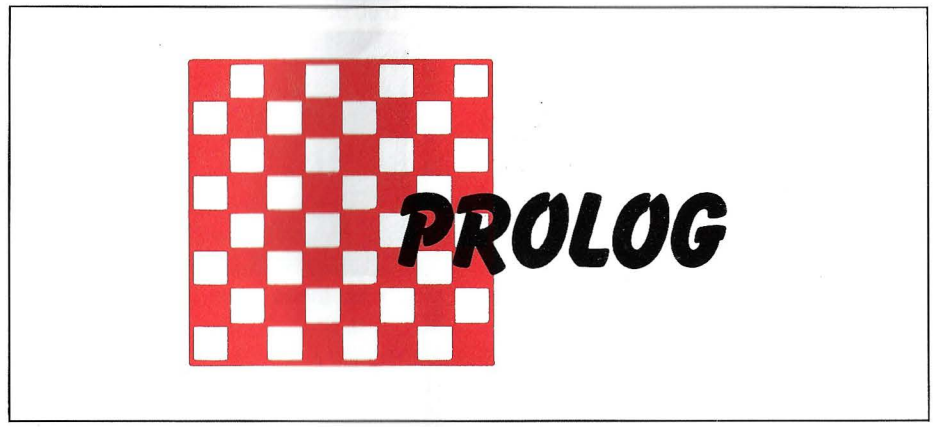
Aunque pensando en una base de datos con un gran número de relaciones "padre-de", indicar explícitamente al PROLOG las relaciones "abuelo-de" puede resultar cuando menos engorroso. Nos gustaría expresar algo de este estilo: "Si x es padre de z y z es padre de y entonces x es abuelo de y". Esto puede conseguirse introduciendo la siguiente regla:

```
&.add(x abuelo de y if
                x padre-de z and
                z padre-de y)
```

A partir de entonces, podríamos preguntar:

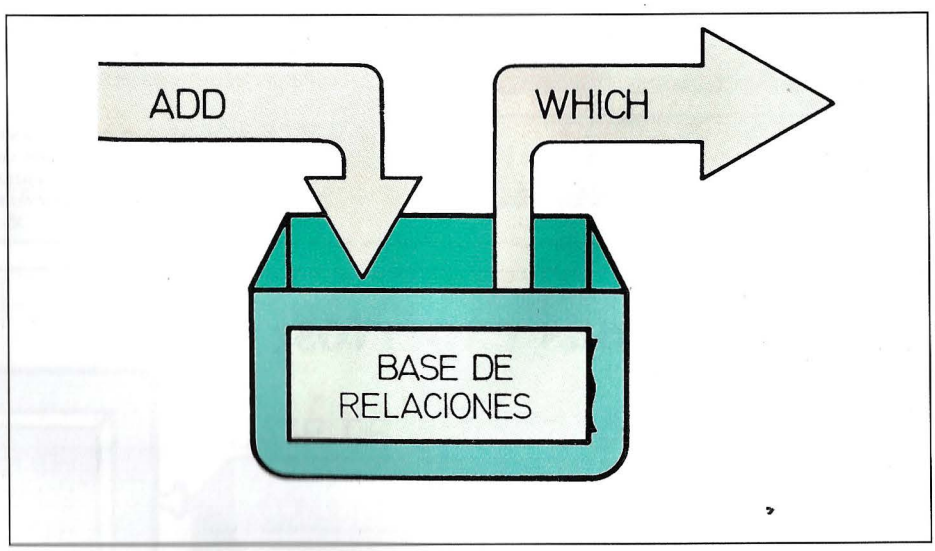
```
&.which(x y : x abuelo-de y)
Enrique Isabel
No (more) answers
&.
```

También existe la posibilidad de realizar cálculos numéricos en PROLOG, manejar listas para crear estructuras de datos complejas y manipular conjuntos de elementos, entre otras muchas cosas. Hay un sinfín de caminos por explorar con esta nueva herramienta.



El ajedrez es uno de los juegos en los que el PROLOG tendría mucho que decir.

LAS APLICACIONES



Los componentes fundamentales del PROLOG son la base de relaciones y los comandos básicos para acceder a ella.

Como ya se ha comentado, el PROLOG encuentra su medio de trabajo ideal en aplicaciones de Inteligencia Artificial: entendimiento del lenguaje natural, construcción de Sistemas Expertos, representación del conocimiento en general, juegos de lógica, resolución general de problemas, mecanismos de aprendizaje, etc. Habida cuenta de que PROLOG representa internamente las relaciones que le introduce el programador en forma de árboles (ver figura), podemos entender mejor la faceta de comprensión del lenguaje natural. Los estudios en este campo están orientados a que algún día seamos capaces de dialogar realmente con un ordenador. Nuestra charla no tendrá como base los comandos que la máquina conoce de antemano, sino que podremos dirigirnos a ella en los mismos términos en los que nos dirigimos a cualquier persona.

Veamos un ejèmplo. En BASIC, la única forma de que disponemos para que el ordenador escriba la letra A es a través de:

```
PRINT "A"
```

Sin embargo, una persona no iniciada en informática encontraría más sencillo decir cosas como:

```
ESCRIBE LA LETRA A   o
IMPRIME LA PRIMERA VOCAL   o
SACA POR PANTALLA "A",
```

sin tener que ceñirse a una serie de estructuras y comandos que tienen poco o nada que ver con nuestra forma habitual de hablar.

Para lograr este objetivo se debe hacer un análisis sintáctico y semántico de la frase introducida, identificando verbos (que expresan acción) y complementos del mismo (que expresan los objetos sobre los que actúa el verbo o viceversa). Los lingüistas de la corriente estructuralista determinaron que toda oración del lenguaje natural se corresponde con una estructura en forma de árbol, muy parecida a la que PROLOG utiliza para representar el conocimiento. La conexión es pues evidente.

UCSD p-System (y 3)

Trabajando con el editor estándar

El editor estándar del sistema operativo UCSD p-System está gobernado por toda una serie de parámetros, controlados de tal forma que el usuario puede modificarlos y cambiarlos según sus necesidades.
Básicamente, el editor puede operar en dos modos fundamentales: el modo PROGRAM, en el cual prepara ficheros conteniendo programas en Pascal, y el modo TEXT, en el que se editan textos en lenguaje natural, operando como un procesador de texto de reducidas características. Ambos modos se definen a través del comando SET, el cual permite controlar los parámetros del editor.
Cada vez que se activa el editor, éste entra por defecto en modo PROGRAM.

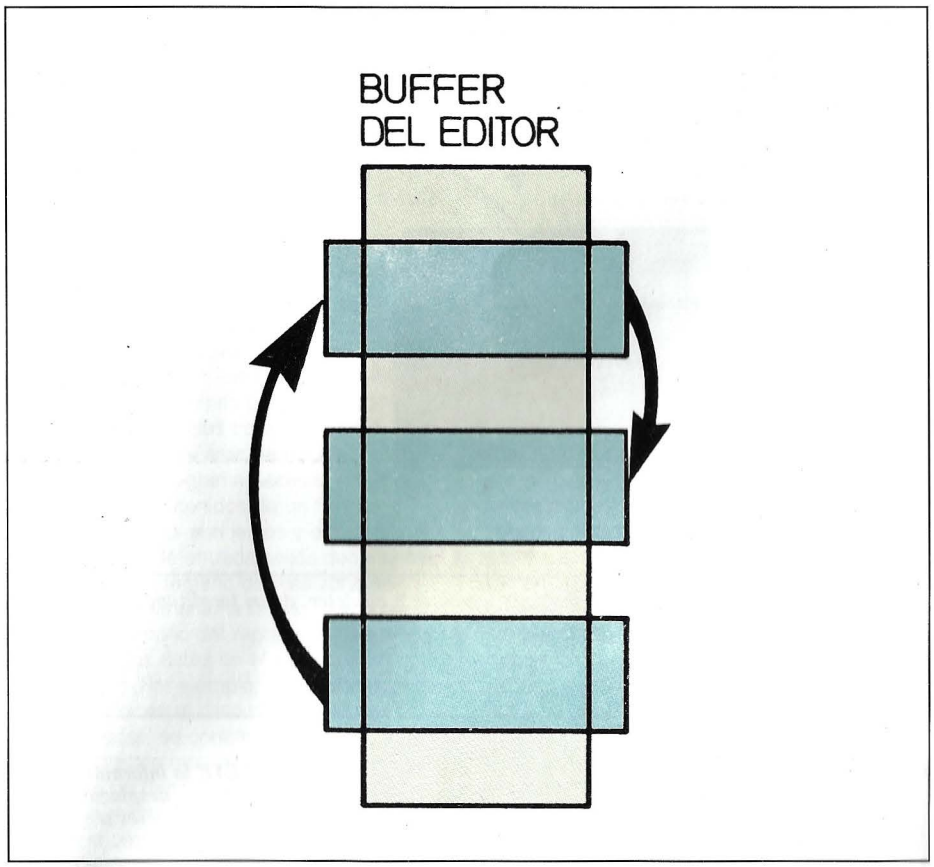
OPERACION DEL EDITOR EN MODO PROGRAM

Dada su amplitud, no se detallarán en los próximos párrafos la totalidad de los comandos o modos de trabajo del editor a disposición del usuario, sino únicamente los más relevantes.

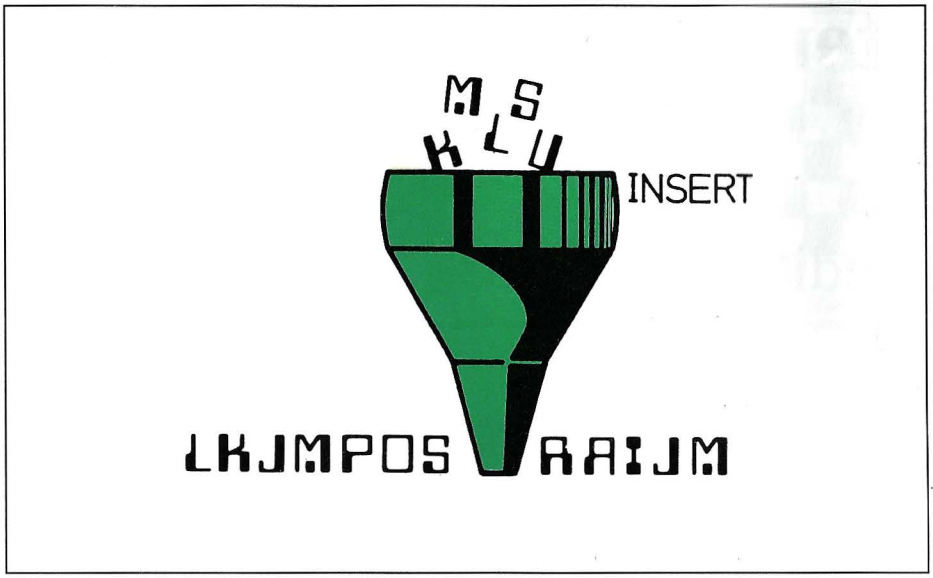
- INSERT

El cometido de este comando es permitir la entrada de información en el buffer del editor, sin destruir la información ya existente en el mismo. Este procedimiento puede emplearse también para introducir nuevos datos en un fichero de texto ya creado, o incluso para generar un nuevo fichero de texto. En cualquier caso, el volumen de información a introducir queda limitado por el propio tamaño del buffer de texto y de la memoria principal del sistema.

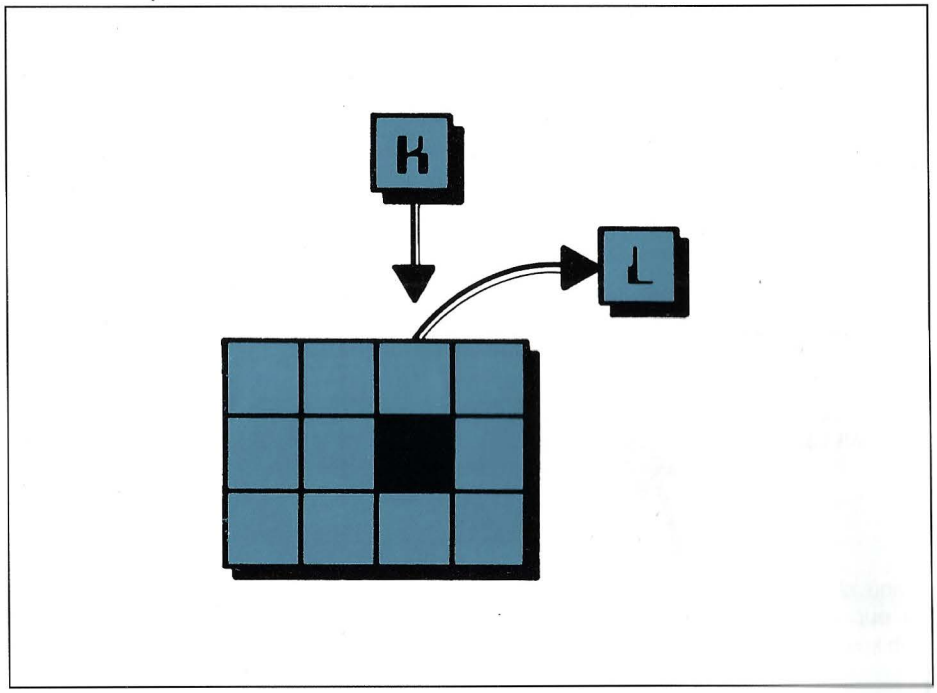
Para seleccionar la opción INSERT basta con situar el cursor en la posición en la que se desean introducir nuevos datos y pulsar la tecla I. En ese preciso instante aparecerá el prompt de INSERT sustituyendo al del editor. Dicho prompt muestra las opciones básicas a disposición del usuario. Estas son, en breve síntesis:
— Backspace: permite eliminar el último carácter incorrecto. Al activar dicha tecla, el cursor se desplaza hacia atrás y borra el último carácter.
— Delete: facilita el borrado desde el punto en el que se encuentra el cursor hasta el final de la línea previamente insertada.
— Ext: pone fin al proceso de inserción y actualiza el buffer del editor.
— Escape: accionando la tecla de escape se interrumpirá el proceso de inserción, borrándose la información introducida y devolviéndose el buffer del editor a las condiciones en que se encontraba antes de iniciar el proceso.
El modo INSERT tiene además dos características especiales. En primer lugar per-



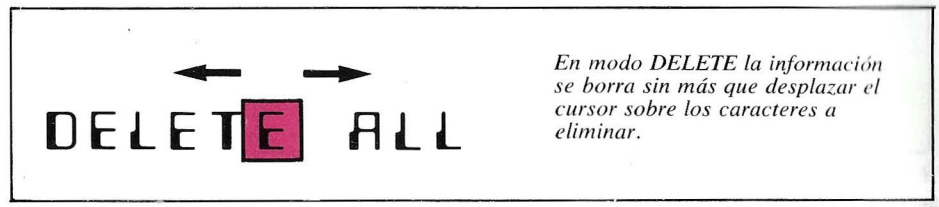
El editor estándar del UCSD p-System ofrece medios para desplazar el campo de visión sobre el buffer.



El comando INSERT permite introducir nueva información sin destruir la que anteriormente se encontraba en el buffer de edición.



La opción EXCHANGE permite reemplazar, carácter a carácter, datos localizados en el buffer de texto.



En modo DELETE la información se borra sin más que desplazar el cursor sobre los caracteres a eliminar.

mite operar con autoindentación (al pulsar RETURN para pasar a la línea siguiente, el cursor se situará automáticamente bajo el primer carácter no en blanco de la línea que se acaba de introducir).
Esta posibilidad es sumamente útil al escribir programas escritos en Pascal, dado que facilita la indentación del texto del programa, con las consiguientes ventajas que ello supone para la comprensión del mismo. Otra característica de INSERT se concreta en permitir el "desbordamiento" (overflow) de las líneas, escribiendo más allá del margen de la pantalla, y permitiendo por tanto líneas de mayor longitud.

- EXCHANGE (Intercambio)

En este modo de operación, el sistema operativo UCSD p-System facilita el intercambio de caracteres del buffer del editor, uno a uno. Su operación es algo restringida, ya que sólo actúa en una sola línea y no pueden intercambiarse bloques completos de texto sino tan sólo caracteres.
Para entrar en este modo de operación es preciso pulsar la tecla X. El correspondiente prompt sustituirá al del editor. Las opciones que ofrece el modo EXCHANGE son las siguientes:
— Backspace: permite retroceder un carácter, reponiendo el carácter eliminado en caso de error.
— Ext: facilita la salida de este modo de operación, actualizando el buffer del editor.
— Esc: se produce al pulsar la tecla de Escape, y origina la salida de este modo de operación sin actualizar el fichero editado y manteniendo el buffer de editor en las condiciones iniciales.

- DELETE

Como su propio nombre indica, este modo de operación permite eliminar elementos contenidos en el buffer del editor, borrando los datos carácter a carácter. Este proceso se produce desplazando el cursor sobre el texto a eliminar. No obstante, el texto borrado no es retirado del buffer mientras no se indique taxativamente esta opción. Para facilitar las tareas de borrado, el usuario puede modificar el sentido de desplazamiento del cursor determinado por el marcador de dirección.
La salida del modo DELETE sólo puede ordenarse en base a las dos opciones que siguen:
— Ext: se produce la actualización del buffer de editor.
— Esc: no se actualiza el buffer de editor,

sino que éste se mantiene en las condiciones de partida antes de entrar en el proceso.

● COPY
Este comando permite copiar información de los textos que actualmente se encuentran en memoria; permite asimismo trasladar zonas de un fichero al buffer de texto. Para acceder a este comando basta con pulsar la tecla C, lo cual hará que aparezca el prompt de este comando; en él están reflejadas las diferentes opciones seleccionables:
— Esc: al pulsar la tecla de escape se abandonará el modo COPY, pasando al modo general de trabajo del editor; no será modificado el buffer del editor.
— From file: al seleccionar esta opción, lo cual se realiza pulsando la tecla F, permite cargar en el buffer de edición la totalidad o sólo una parte de un fichero residente en un periférico de almacenamiento masivo.
— Buffer: esta opción, seleccionada por medio de la tecla B, hace que se inserte la información contenida en un buffer denominado de copia, a partir del punto en que se encuentra el cursor. El buffer de copia es alimentado con caracteres cada vez que se ejecuta una acción DELETE o un comando INSERT.

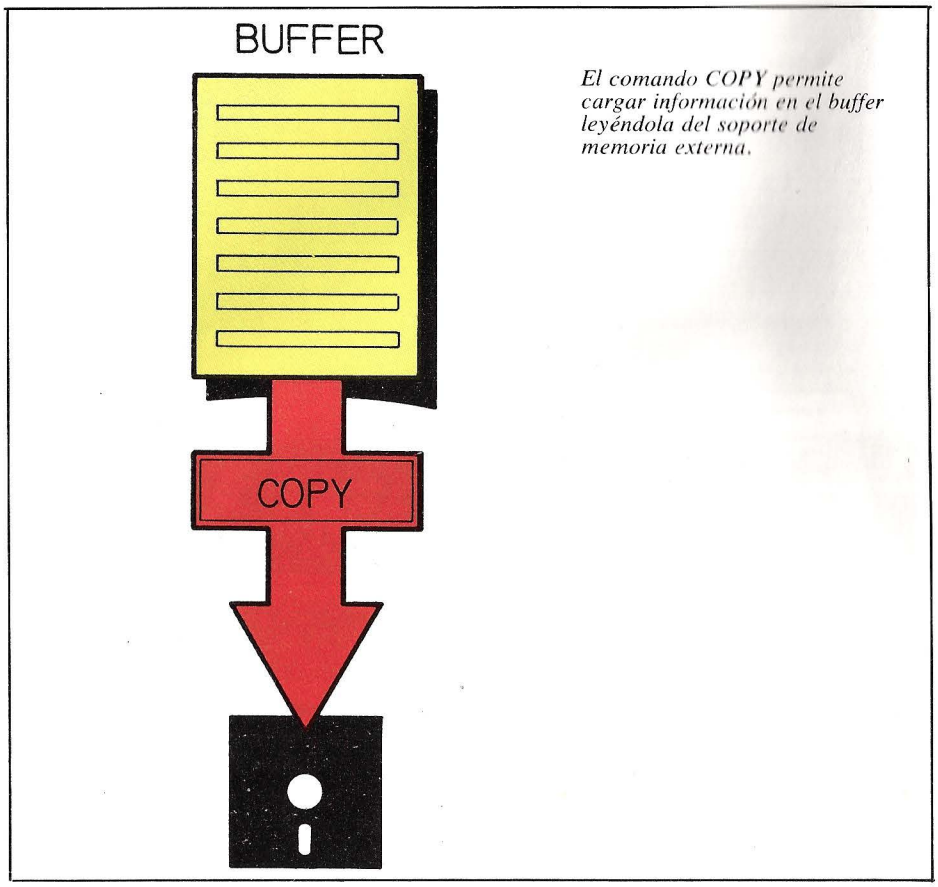
● ADJUST
La misión de este comando es desplazar las líneas que constituyen el programa residente en el buffer de editor de manera que sea más legible. El desplazamiento de las líneas se logra actuando sobre las teclas de control del cursor. Otras opciones permiten el centrado de texto en la pantalla.

● Comando QUIT
Este comando pone fin a la actividad del editor, regresando el sistema al modo comando. Su ejecución admite cuatro alternativas:
— Almacenar el contenido del buffer de editor en el volumen raíz, bajo el nombre del fichero genérico SYSTEM.WRK.TEXT el cual, si ya existe, es sustituido por el último fichero en curso.
— Almacenar los datos en cualquier volumen, dentro de un fichero cuyo nombre determine el propio usuario.
— Salida al modo general de comando sin escribir la información en ningún fichero.
— Retorno al modo de editor sin escribir

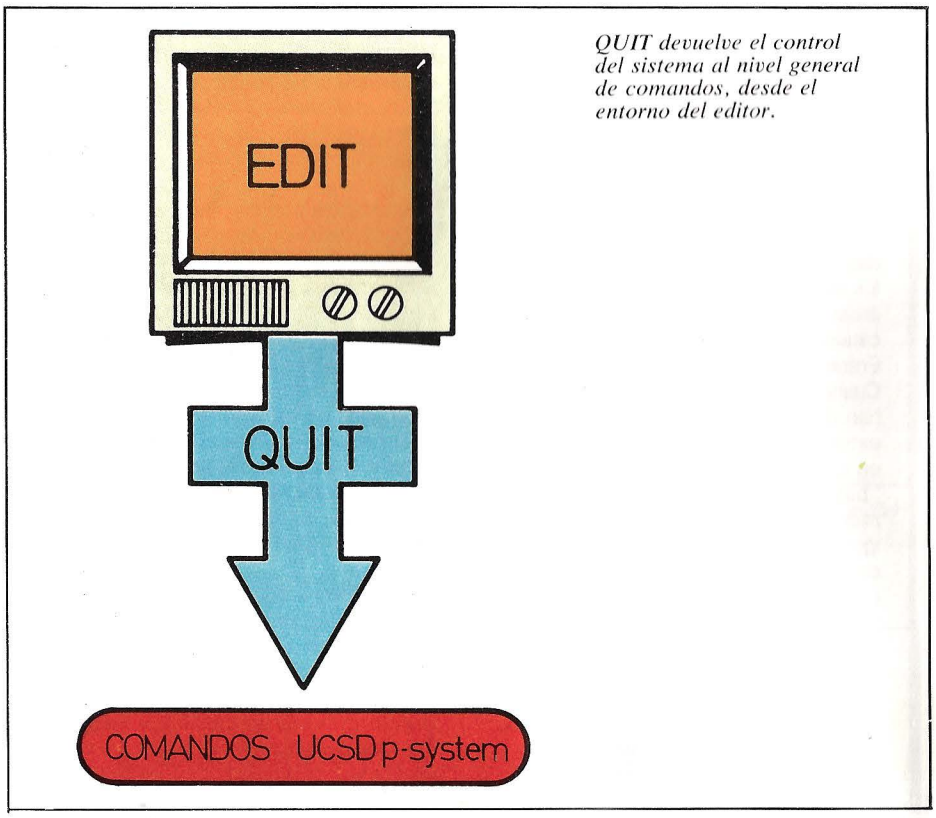
El editor del sistema operativo UCSD p-System, operando en modo "program", apoya la edición de ficheros cuyo contenido sean programas en lenguaje Pascal.

Formas de almacenamiento interno de los datos

Cuando un programa está en activo, los datos que utiliza son extraídos de diferentes puntos del sistema informático; ya sea unidades de cinta, discos magnéticos o disquetes. Una vez leídos los datos del soporte externo, éstos son depositados en diversas áreas de la memoria central del ordenador; áreas que dependen de las propias características de los datos. En términos generales, estas áreas de la memoria primaria pueden catalogarse en dos tipos: zonas de Entrada/Salida y área de trabajo.
Cuando un programa necesita datos que han de ser obtenidos de un fichero externo, éstos son leídos y depositados en un área de entrada/salida denominada "buffer". El tamaño de esta zona de memoria primaria ha de ser, como poco, igual al tamaño del registro físico que da cobijo a los datos en el periférico en cuestión. Por ejemplo, el tamaño mínimo del buffer en el caso de leer tarjetas perforadas, se concreta en 80 caracteres (información que, normalmente, puede leerse de una sola tarjeta).
Por la misma razón, es necesario contar con la presencia de un buffer para los datos de salida. En él se irán almacenando los datos que deban enviarse a otros periféricos: unidad de disco, impresora, terminal...
El área de trabajo ocupa gran parte de la memoria primaria disponible, ya que su misión es almacenar las informaciones constantes y variables que un ordenador necesita para la ejecución del programa en curso. Así, en el área de trabajo se almacenarán datos constantes, como pueden ser encabezamientos de informes, tablas numéricas, códigos postales. Una subárea dentro de esta entidad de almacenamiento es la encargada de contener todas aquellas informaciones variables que se generan en el transcurso de un programa; por ejemplo: variables que actúan como contadores, acumuladores o, simplemente, como variables de operación.
En resumen, las áreas de entrada y salida actúan como zonas de almacenamiento intermedio de la información que fluye entre la unidad central y los dispositivos periféricos, mientras que el área de trabajo se ocupa de memorizar los datos internos implicados en la ejecución de un programa.



El comando COPY permite cargar información en el buffer leyéndola del soporte de memoria externa.



QUIT devuelve el control del sistema al nivel general de comandos, desde el entorno del editor.

ningún fichero, con objeto de proseguir el trabajo.

OPERACION DEL EDITOR EN MODO TEXT

En el modo TEXT, el editor estándar del UCSD p-System opera básicamente con los mismos comandos descritos para el modo PROGRAM. Sin embargo, los parámetros que gobiernan este proceso difieren en su definición.

El control de los nuevos matices se realiza a través de la opción ENVIRONMENT asociada al comando SET.

Una vez activado el comando SET, la opción ENVIRONMENT se selecciona pulsando la tecla E. De inmediato se presenta una pantalla en la cual se visualiza el estado actual de los parámetros que gobiernan al editor, para que el usuario decida sobre su modificación. Las características seleccionables son:

— Autoindentación: cuando está en opción "TRUE", al operar en modo INSERT se producirá una indentación igual a la de la línea previa. Si es "FALSE" cada línea irá justificada a la izquierda.

— Filling: si su valor es "TRUE", cuando se salga una palabra de la línea ésta pasará automáticamente a la siguiente. En caso contrario se producirá una situación de "overflow".

— Márgenes: sólo pueden controlarse cuando la autoindentación está desactivada y el modo Filling en activo.

— Margen de párrafo: sólo produce indentación en la primera línea de un párrafo, con las mismas condiciones que en el caso de los márgenes.

El cambio de estos parámetros se realiza —una vez seleccionada la opción ENVIRONMENT— poniendo en práctica la secuencia de operaciones que se detalla a continuación:

— Pulsar la letra correspondiente al código de la característica a modificar.

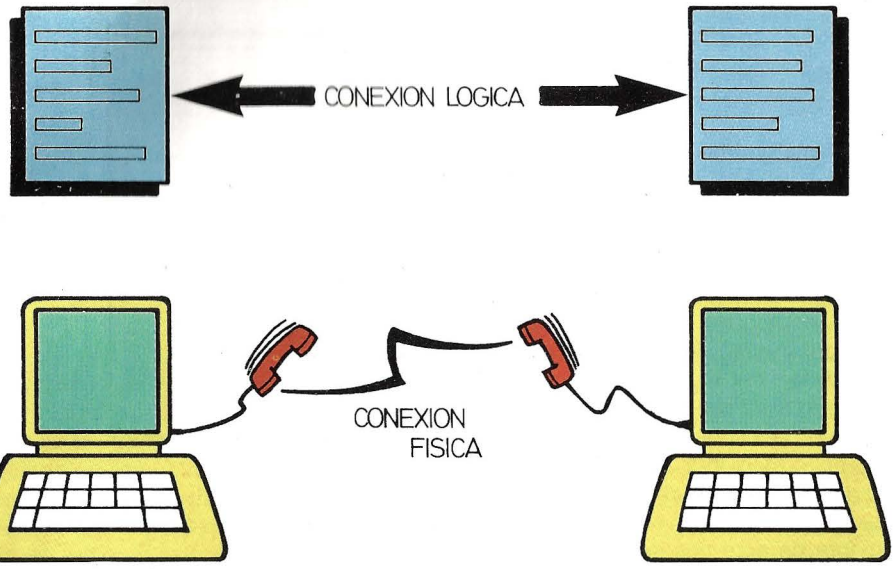
— Si el dato a alterar es "TRUE" o "FALSE", pulsar T o F respectivamente.

— En el caso de que sea necesario introducir un número, éste debe teclearse seguido por una acción sobre la barra espaciadora. En ese preciso instante, el cursor volverá al prompt de ENVIRONMENT.

File Transfer

Un programa de comunicaciones para IBM-PC y compatibles

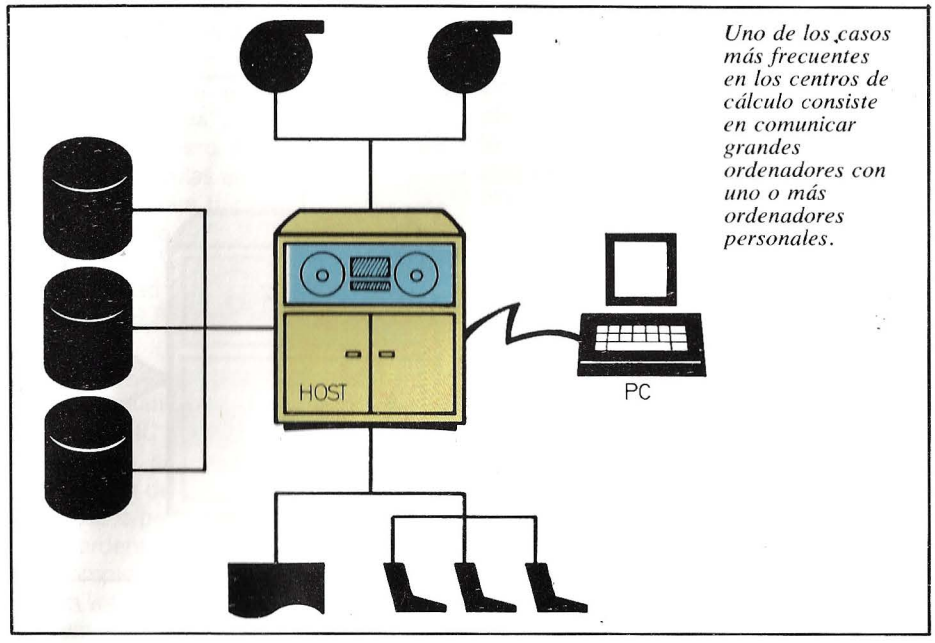
Un caso particular dentro de las posibilidades de comunicación entre equipos informáticos, consiste en la conexión de un gran ordenador con un ordenador personal. Desde luego, al hablar de comunicación entre ordenadores, en este caso, no nos referimos al aspecto físico de dicha conexión, sino al aspecto lógico. En efecto, nuestro interés se centra en el estudio de los programas (software) que permiten que dos ordenadores se entiendan, dando por supuesto que ambos equipos se encuentran conectados mediante cualquiera de las posibilidades existentes: por cable, línea telefónica, satélite, fibra óptica, micro-ondas, rayo laser...
En este capítulo se examina uno de los programas más utilizados para la comunicación de un ordenador "grande", del tipo IBM-43XX, con un ordenador personal, también IBM o compatible.



Dentro del mundo de las comunicaciones existen dos conceptos complementarios: la conexión física, formada por vías de transmisión, y la conexión lógica, formada por programas que facilitan la "comprensión" entre los dos equipos.

LA COMUNICACION ENTRE GRANDES Y PEQUEÑOS ORDENADORES

Antes de comenzar a describir las características técnicas del paquete FILE TRANSFER, conviene aclarar algunos conceptos de carácter general sobre el tipo de comunicación que nos ocupa.
La conexión se realizará entre dos ordenadores distintos: uno "grande" y otro "pequeño". Existen distintas acepciones para cada uno de ellos; así, al "grande" se le suele denominar indistintamente: HOST, MAIN FRAME, ORDENADOR, etc.; mientras que el "pequeño" recibe el calificativo de PC, ordenador personal, microordenador... A lo largo del presente capí-



Uno de los casos más frecuentes en los centros de cálculo consiste en comunicar grandes ordenadores con uno o más ordenadores personales.

tulo nos referiremos siempre a amb os equipos como HOST y PC, respectivamente.
¿Cuál sería el objetivo más ambicioso para comunicar un PC con un HOST? La respuesta parece clara: que cada uno de ellos pudiera interactuar con el otro como consigo mismo y, además, que esta comunicación no supusiera un decremento en la capacidad de ninguno de los dos equipos. Esta posibilidad no está plenamente conseguida, aunque, en la práctica, se pueden establecer conexiones plenamente satisfactorias para cualquier necesidad. Dentro de este abanico de posibilidades existen dos fundamentales:

1. **Pantalla virtual**

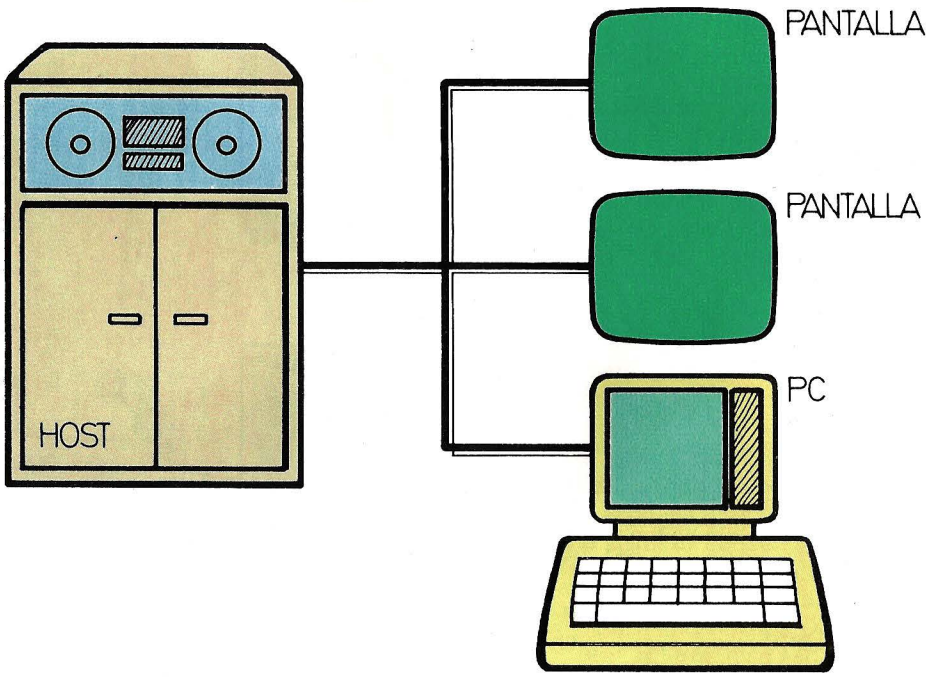
Si se opta por este tipo de conexión lógica, el PC se comportará como un terminal más del HOST. Con esta alternativa se consigue trabajar desde un modesto ordenador personal con todas las prestaciones de un gran ordenador. Según las necesidades del usuario, éste podrá optar entre trabajar con el PC de forma convencional y, por consiguiente, de modo autónomo, o mediante un programa especializado simular una pantalla virtual conectada al HOST. Tal vez el mayor inconveniente de este tipo de programas de comunicación estriba en que no se puede utilizar simultáneamente el PC como terminal y como ordenador personal.

2. **Transmisión de información**

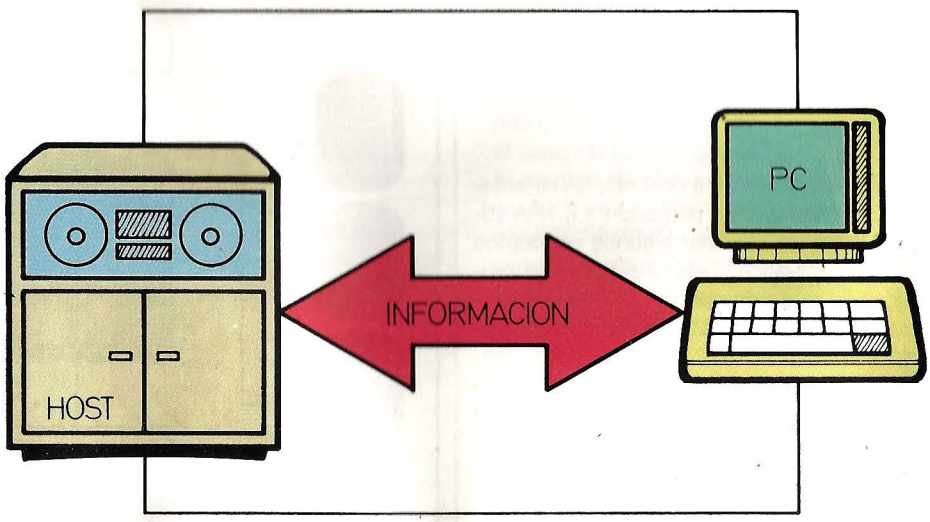
La segunda alternativa más extendida consiste en facilitar la transmisión del objeto tratado tanto por el HOST como por el PC: la información. Evidentemente, cuando hablamos de información no sólo nos referimos a datos en su sentido más restringido, sino también a cualquiera de los "objetos" manejables por ambos equipos.
Esta segunda posibilidad, que a priori parece inferior a la primera, ofrece posibilidades positivas, tanto para la explotación de programas como para el desarrollo de los mismos.
Ambos procedimientos se basan en la utilización de programas especializados en comunicaciones. La mayoría de las empresas que simultanean la utilización de equipos grandes y pequeños cuentan con programas de ambos tipos, con lo que disponen de una nueva posibilidad para la utilización de sus PC's.

FILE TRANSFER

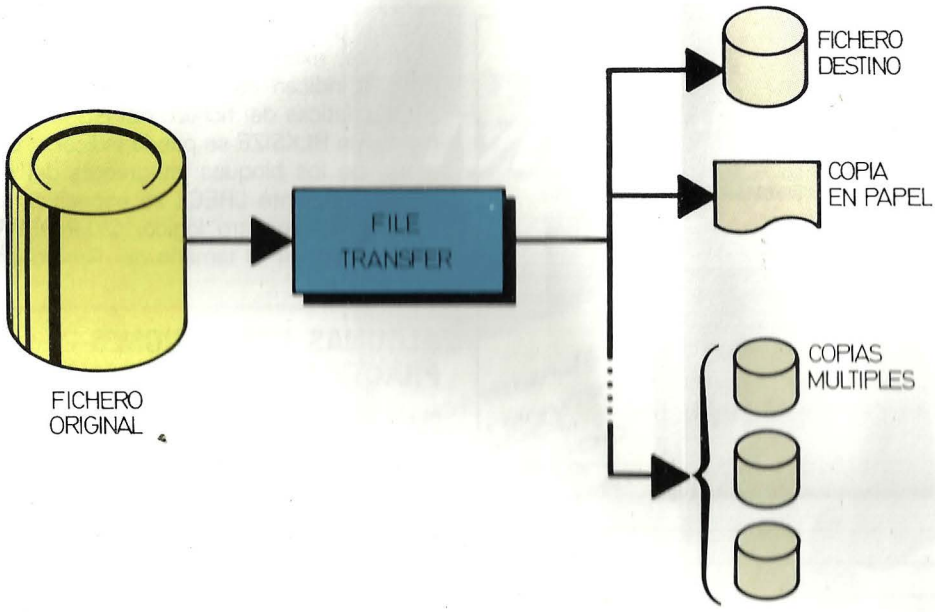
Como su propio nombre indica, FILE TRANSFER es un sistema dedicado a la transmisión de información. No puede ser considerado como un programa convencional, sino que más bien cabe catalogarlo como un sistema integrado por diversos módulos que, residiendo en la HOST, facilitarán la comunicación con un PC. El uso de uno u otro módulo dependerá tanto del tipo de PC como de la operación que se desee realizar.
Existen distintas versiones de este programa. Como ejemplo más significativo



Una de las alternativas para la conexión entre un gran ordenador y un PC consiste en hacer que éste se comporte como una pantalla más de las del Host.



La otra alternativa para la conexión Host/PC se basa en la posibilidad de intercambiar información, aunque respetando la autonomía del PC respecto del Host.



El programa FILE TRANSFER ofrece diversas posibilidades para efectuar la transmisión. En la figura se presentan algunas de ellas.

Parámetros de FILE TRANSFER

```
PCXFER DSNAME('dsname')/SYSOUT('class')/INTRDR
   TO/FROM  PCFILE('file specification')
   OLD/NEW/MOD/SHR
   ASIS
   BLKSIZE('nnnn')
   COPIES('integer')
   CYLINDERS
   DEST('ddddddd')
   FOLD
   FORMS('form number')
   LRECL('integer')
   NOKEY
   PASSWORD('password')
   RECFM('recfm')
   SPACE('nn')
   SPACE2('nn')
   TRACKS
   UNNUM
   VOLUME('vvvvvv')
```

Las características de las operaciones a efectuar se especifican mediante parámetros, en la figura se puede apreciar la lista completa de parámetros admitidos por FILE TRANSFER.

supondremos que nuestro objetivo es comunicar un ordenador IBM, dotado del sistema operativo MVS, con un ordenador personal IBM o compatible equipado con la correspondiente tarjeta interface.
Como principales características del programa, cabe destacar las siguientes:
1. Los tiempos utilizados para la transferencia de información son óptimos, ya que el programa está codificado en ASSEMBLER y se apoya en utilidades del sistema operativo residente en el HOST.
2. La utilización de recursos es eficiente ya que la transferencia se realiza en bloques de 1 ó 2 Kbytes con lo que se minimiza el impacto sobre el resto de los usuarios del HOST y se maximiza la capacidad de comunicación.
3. Capacidad para el tratamiento de ficheros secuenciales o particionados con prácticamente cualquier tipo de formato de registro: fijo, variable, bloqueado, desbloqueado e incluso indefinido.
4. Posibilidad de transferir parámetros de tipo "impresora"; es decir: número de formulario, número de copias y conversión de minúsculas en mayúsculas.
5. Facilidad para transferir directamente ficheros desde el HOST hasta la impresora del PC.
6. Capacidad para la submisión de trabajos desde el PC para su ejecución en el HOST.
7. Conversión automática entre distintos tipos de codificación, fundamentalmente entre EBCDIC y ASCII, y viceversa.

En resumen, se puede afirmar que con la participación de este programa es posible conseguir la perfecta integración de uno o varios ordenadores personales dentro de una compleja estructura informática, sin perder las ventajas intrínsecas de estos pequeños equipos.

PARAMETROS DE FILE TRANSFER

Para conseguir cualquiera de las posibilidades apuntadas en el párrafo anterior, el usuario debe aportar una serie de parámetros al programa. Mediante ellos se dan entradas al programa que de esta manera conocerá las características de la información a transferir. A continuación, vamos a describir algunos de los parámetros del programa FILE TRANSFER:

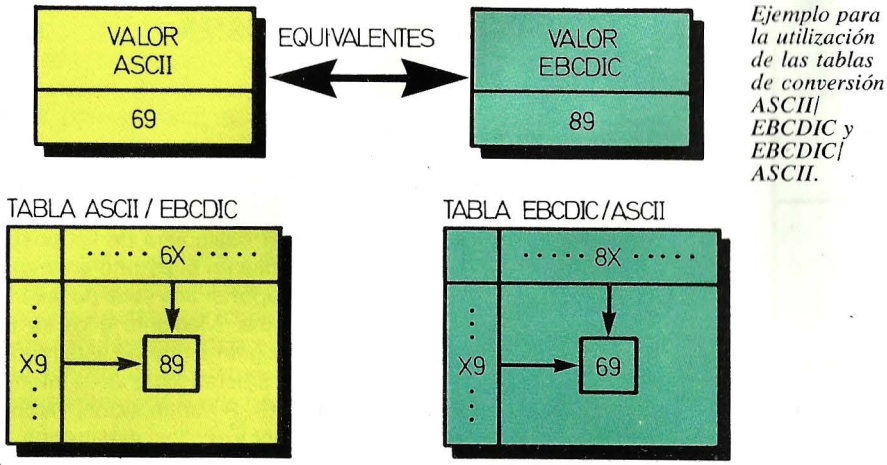
1. DSNAME
Permite especificar el nombre del fichero del HOST que recibirá o enviará la información. En el primer caso, dicho fichero puede o no existir previamente; pero en el segundo caso la existencia resulta imprescindible para realizar la transferencia al PC.

2. TO/FROM
Mediante este parámetro se indica el sentido de la transferencia. Si la información "va" del HOST al PC, se debe especificar TO; en el caso contrario, es decir, para transferir información del PC al HOST, se utilizará FROM.

3. PCFILE
Sirve para indicar el nombre del fichero del PC implicado en la transferencia. En el caso de haber especificado en el parámetro anterior TO, es posible indicar "PRN": como valor del parámetro PCFILE. En tal situación, la transferencia se realizará directamente sobre la impresora del PC.

4. ASIS
Suele ser muy usual que los PC's trabajen con código ASCII, mientras que el HOST, en el caso de IBM, suele utilizar el código EBCDIC. En consecuencia, para que la información transferida sea interpretable por el destinatario, deben traducirse los códigos ASCII a EBCDIC, o EBCDIC a ASCII, según el sentido de la transmisión. Para ello basta con especificar el parámetro ASIS.

5. OTROS PARAMETROS
Con los anteriores cuatro parámetros pueden especificarse las principales caracte-



Ejemplo para la utilización de las tablas de conversión ASCII/ EBCDIC y EBCDIC/ ASCII.

Tabla de conversión - ASCII a EBCDIC

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	00	10	40	F0	7C	D7	79	97	20	30	41	58	76	9F	B8	DC
x1	01	11	5A	F1	C1	D8	81	98	21	31	42	59	77	A0	B9	DD
x2	02	12	7F	F2	C2	D9	82	99	22	1A	43	62	78	AA	BA	DE
x3	03	13	7B	F3	C3	E2	83	A2	23	33	44	4F	80	AB	BB	DF
x4	37	3C	5B	F4	C4	E3	84	A3	24	34	45	64	8A	AC	BC	EA
x5	2D	3D	6C	F5	C5	E4	85	A4	15	35	46	65	8B	4A	6A	EB
x6	2E	32	50	F6	C6	E5	86	A5	06	36	47	66	8C	AE	BE	EC
x7	2F	26	7D	F7	C7	E6	87	A6	17	08	48	67	8D	AF	BF	ED
x8	16	18	4D	F8	C8	E7	88	A7	28	38	49	68	8E	B0	CA	EE
x9	05	19	5D	F9	C9	E8	89	A8	29	39	51	69	8F	B1	CB	EF
xA	25	3F	5C	7A	D1	E9	91	A9	2A	3A	5F	70	90	B2	CC	FA
xB	0B	27	4E	5E	D2	AD	92	C0	2B	4A	53	71	9A	B3	CD	FB
xC	0C	1C	6B	4C	D3	E0	93	6A	2C	04	54	72	9B	B4	CE	FC
xD	0D	1D	60	7E	D4	BD	94	D0	09	14	55	73	9C	B5	CF	FD
xE	0E	1E	4B	6E	D5	5F	95	A1	0A	3E	56	74	9D	B6	DA	FE
xF	0F	1F	61	6F	D6	6D	96	07	1B	E1	57	75	9E	B7	DB	FF
	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx

Tabla de conversión - EBCDIC a ASCII

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	00	10	80	90	20	26	2D	BA	C3	CA	D1	D8	7B	7D	5C	30
x1	01	11	81	91	A0	A9	2F	BB	61	6A	7E	D9	41	4A	9F	31
x2	02	12	82	16	A1	5E	B2	BC	62	6B	73	DA	42	4B	53	32
x3	03	13	83	93	A2	AB	E5	BD	63	6C	74	DB	43	4C	54	33
x4	9C	9D	84	94	A3	AC	B4	BE	64	6D	75	DC	44	4D	55	34
x5	09	85	0A	95	A4	AD	B5	BF	65	6E	76	DD	45	4E	56	35
x6	86	08	17	96	A5	AE	B6	C0	66	6F	77	DE	46	4F	57	36
x7	7F	87	1B	04	A6	AF	B7	C1	67	70	78	DE	47	50	58	37
x8	97	18	88	98	A7	B0	B8	C2	68	71	79	E0	48	51	59	38
x9	8D	19	89	99	A8	B1	B9	60	69	72	7A	E1	49	52	5A	39
xA	8E	92	8A	9A	9B	21	7C	3A	C4	CB	D2	E2	E8	EE	F4	FA
xB	0B	8F	8B	D5	2E	24	2C	23	C5	CC	D3	E3	E9	EF	F5	FB
xC	0C	1C	8C	14	3C	2A	25	40	C6	CD	D4	E4	EA	F0	F6	FC
xD	0D	1D	05	15	28	29	5F	27	C7	CE	5B	5D	EB	F1	F7	FD
xE	0E	1E	06	9E	2B	3B	3E	3D	C8	CF	D6	E6	EC	F2	F8	FE
xF	0F	1F	07	1A	B3	AA	3F	22	C9	D0	D7	E7	ED	F3	F9	FF
	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx

Tabla para la conversión de caracteres ASCII a EBCDIC y viceversa.

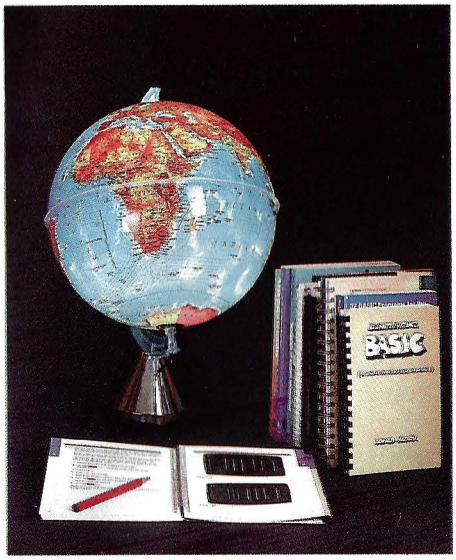
rísticas de la transmisión deseada. No obstante, existen muchos otros con los que se indican con mayor precisión las características del fichero del HOST; así, mediante BLKSIZE se puede indicar el tamaño de los bloques integrantes del fichero, mediante LRECL se especifica el tamaño del registro lógico, CYLINDERS permite definir el tamaño del fichero en cilindros, etc.

ALGUNAS UTILIZACIONES PRACTICAS DE FILE TRANSFER

Para finalizar este capítulo se detallan algunas de las principales utilizaciones del programa FILE TRANSFER.

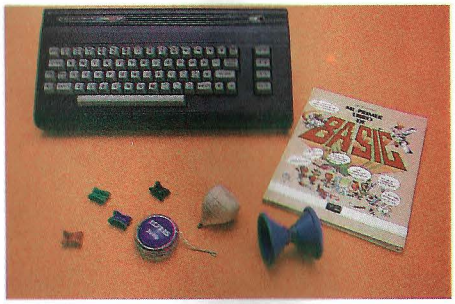
- Suponda que en el PC se dispone de un paquete integrado para la gestión de información; suponga también que en el HOST se dispone de una aplicación muy compleja que maneja grandes volúmenes de información, produciendo a partir de ellos resultados que deben ser analizados con detalle por el usuario. En este caso, mediante el programa FILE TRANSFER, se pueden "enviar" los resultados desde el HOST hasta el PC, en donde el usuario se apoyará en el sistema integrado para su estudio.
- Otro caso bien distinto podría ser el siguiente. En el PC se dispone de un programa desarrollado por el propio usuario; programa que debe ser ejecutado con un gran número de datos residentes en el HOST. En esta tesitura, la solución más práctica consiste en transferir el programa desde el PC hasta el HOST, en donde se ejecutará. Para ello resulta perfectamente válido el programa FILE TRANSFER.
- Como último ejemplo práctico de posible uso de FILE TRANSFER, vamos a suponer que en distintos puntos geográficos se dispone de PC's dedicados a recoger información, realizando con ella determinados procesos de carácter local. Posteriormente, se desea realizar un estudio con la globalidad de los datos existentes en cualquiera de los puntos de entrada de datos. En esta situación, la utilidad de FILE TRANSFER se limitaría a la recopilación en el HOST de los datos ubicados en los PC's. A continuación, el tratamiento de los mismos resultará muy sencillo.

Basic

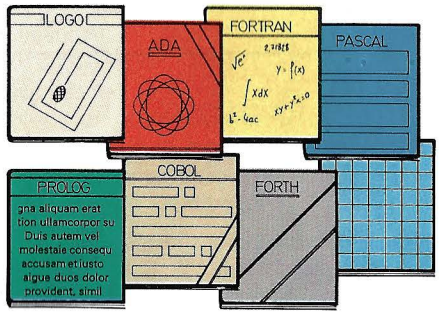


Indice

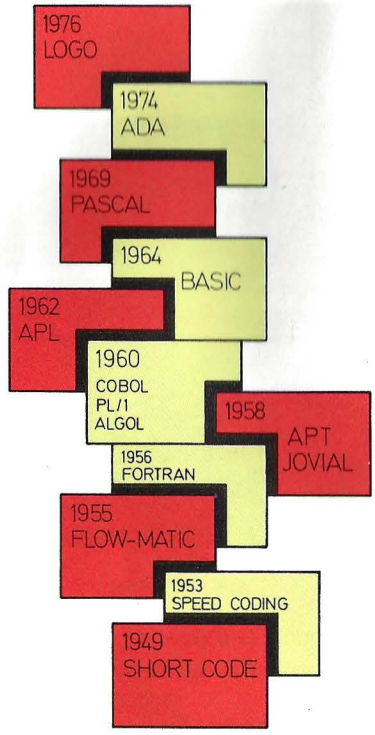
Indice

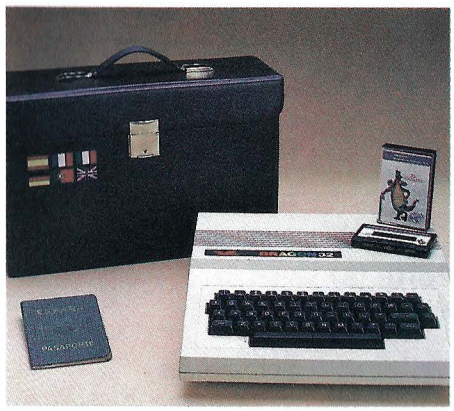


Lenguajes

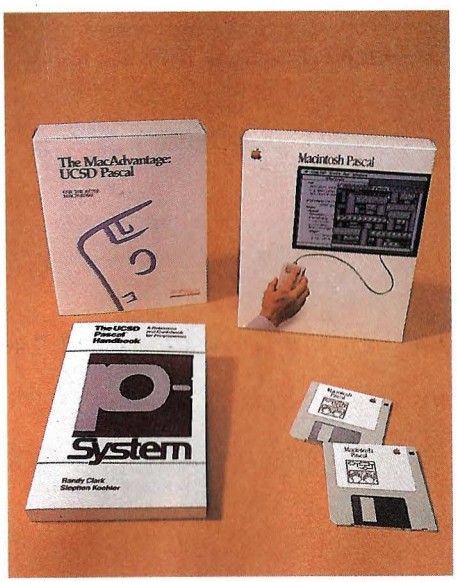


Indice



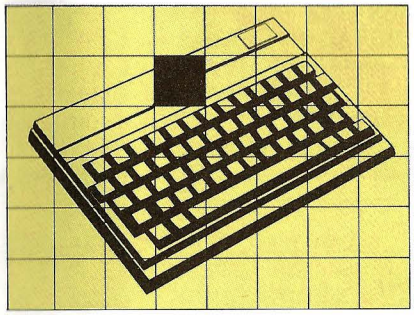


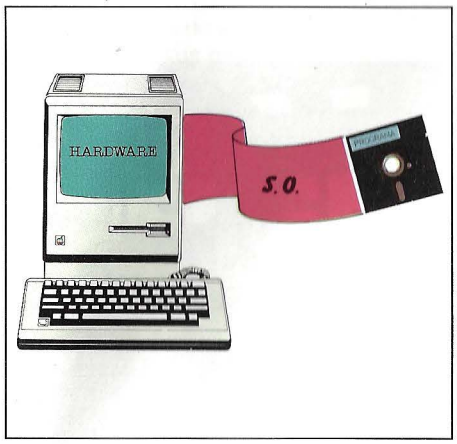
Indice



Indice

Sistemas operativos





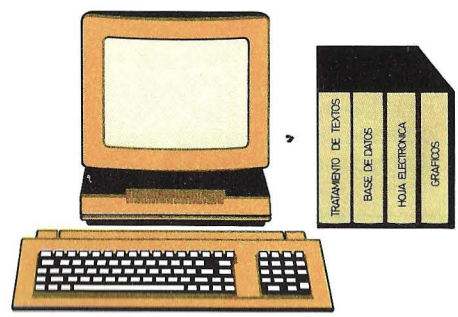
Indice

Indice

S.O.

Indice

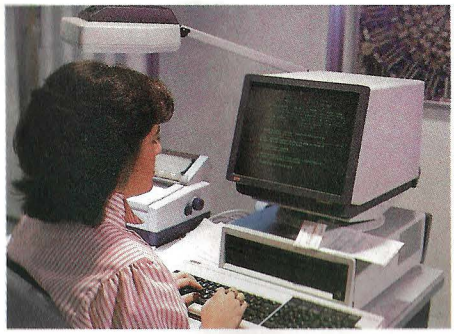
Aplicaciones



Aplicaciones

Indice

Aplicaciones



Indice