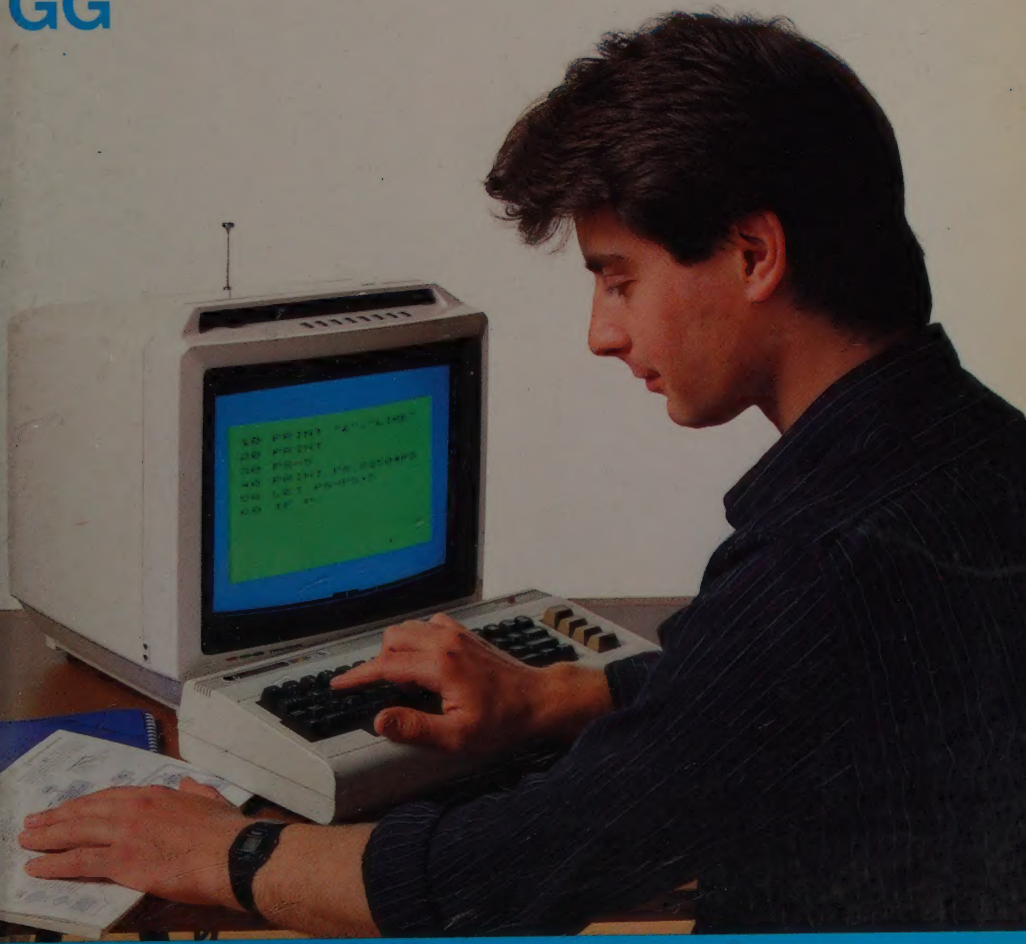
PRIMEROS PASOS EN BASIC

S. Curran - R. Curnow

GG



Colección SU ORDENADOR PERSONAL

PRIMEROS PASOS EN BASIC

Editorial Gustavo Gili, S. A.

08029 Barcelona Rosellón, 87-89. Tel. 259 14 00
28006 Madrid Alcántara, 21. Tel. 401 17 02
1064 Buenos Aires Cochabamba, 154-158. Tel. 361 99 98
03100 México, D.F. Amores, 2027. Tels. 524 03 81 y 524 01 35
Bogotá Diagonal 45 N.° 16 B-11. Tel. 245 67 60
Santiago de Chile Santa Victoria, 151. Tel. 222 45 67

PRIMEROS PASOS EN BASIC

S. Curran - R. Curnow

Asesor de esta colección
Richard Pawson,
Editor de Microcomputer Printout y Business Micro

GG

Título original
First Steps in BASIC
The Clear and Simple Home Computer Series
was conceived, edited and designed by
Frances Lincoln Limited,
Apollo Works, 5 Charlton King's Road
London NW5 2SB

Versión castellana de Jordi Abadal Berini, Ingeniero Industrial

1.ª edición 1984
2.ª edición 1985

Printed in Spain
ISBN: 84-252-1186-7 Colección
ISBN: 84-252-1184-0 Esta obra
Depósito legal: B 3.225-1985
Fotocomposición: TECFA, S.A. - Barcelona
Impresión: Imprenta Juvenil, S.A.
Maracaibo, 11 - Barcelona

Indice

Introducción

Con este libro le enseñaremos qué es el BASIC, el lenguaje de casi todos los ordenadores domésticos. Le enseñaremos cómo utilizar el BASIC en un ordenador, a escribir programas sencillos y a adaptar programas escritos por otra gente para que funcionen en su máquina.

Hablaremos de dos ordenadores: nuestro ordenador y su ordenador. Si todavía no tiene un ordenador u ocasionalmente tan sólo puede utilizar uno en la escuela, colegio o trabajo, no se preocupe. Puede seguir parte o incluso todo el libro utilizando nuestro ordenador como ejemplo.

Sin embargo, aprender a programar no es fácil si no se tiene un ordenador para practicar. Pensará que no lo ha hecho muy bien cuando descubra, al tener un ordenador, que ninguno de sus programas funciona. Le sugerimos que si no tiene un ordenador, lea primero este libro e intente algunos de los ejercicios, para conseguir un poco de práctica en la programación en BASIC.

El descubrir un poco lo que es el BASIC es una buena manera de saber si se quiere realmente, o se necesita, comprar un ordenador. ¿Será capaz de programar al ordenador para que haga lo que usted quiere que haga? ¿Se divertirá utilizándolo? Lea el Apéndice 1, para descubrir cómo varían los ordenadores según la forma de utilizar el BASIC. Utilice esta información, junto con los artículos de máquinas parecidas que aparecen en las revistas que mencionamos en la página 150, con información de libros más generales que traten sobre pequeños ordenadores o con información de su tienda de ordenadores para ayudarle a escoger qué máquina debe comprar. Entonces, en cuanto tenga uno, querrá empezar a programar lo más pronto posible.

No intentamos reemplazar a los manuales que acompañan a los ordenadores domésticos. Lo que queremos es complementarlos y explicarle a usted algunas

de las cosas que los manuales no le dicen, sobre cómo es realmente el aprender a programar y cómo debe usarlo para plantear y escribir sus propios programas.

¿Qué ordenador utilizamos?

Todos los ordenadores domésticos tienen diferencias entre sí. Aunque casi todos ellos pueden utilizar el lenguaje BASIC, todos utilizan distintos «dialectos» de él. Algunos se diferencian sólo ligeramente entre ellos; otros se diferencian bastante. Se habrá dado cuenta que la mayoría de los libros de BASIC le dicen con qué máquina se supone que usted los utilizará. Nosotros no.

Naturalmente, hemos tenido que escoger una versión particular de BASIC para enseñarle a usted en este libro, y un ordenador en particular —con un determinado teclado, determinados caracteres que aparecen en la pantalla, etc.— sobre el que hablar. De hecho, nuestro ordenador es realmente un híbrido. Al planear y escribir este libro hemos utilizado varios ordenadores domésticos populares, cada uno de los cuales utiliza una versión diferente del lenguaje BASIC, y hemos tomado algunas características comunes a partir de las cuales hemos desarrollado una forma de BASIC que utilizaremos aquí. Lo hemos escogido porque:

a) Las órdenes utilizadas tienen equivalentes en casi todos los ordenadores domésticos.

b) Puede adaptarse fácilmente a su propia máquina.

c) Utilizándolo, usted verá cómo escribir programas que nosotros creemos que encontrará interesantes.

Quizás ha oído hablar del término «microsoft BASIC». El microsoft es probablemente la más directa y fácilmente comprensible forma de BASIC. Nuestro BASIC es, fundamentalmente, sólo esto.

Debido a la manera en que hemos escrito el libro, su ordenador puede ser cualquier ordenador que entienda el BASIC, desde el más pequeño al más grande. No tendrá ninguna dificultad en descubrir las diferencias entre su ordenador y el nuestro y cómo adaptar el BASIC que utilizamos de forma que funcione en su ordenador. Para asegurarse de que no se pierde intentándolo hacer, hemos incluido un Apéndice especial (Apéndice 1) que le da cantidad de información sobre los puntos que diferencian las distintas versiones del

BASIC y sobre cómo encontrar la información necesaria en el manual de su ordenador.

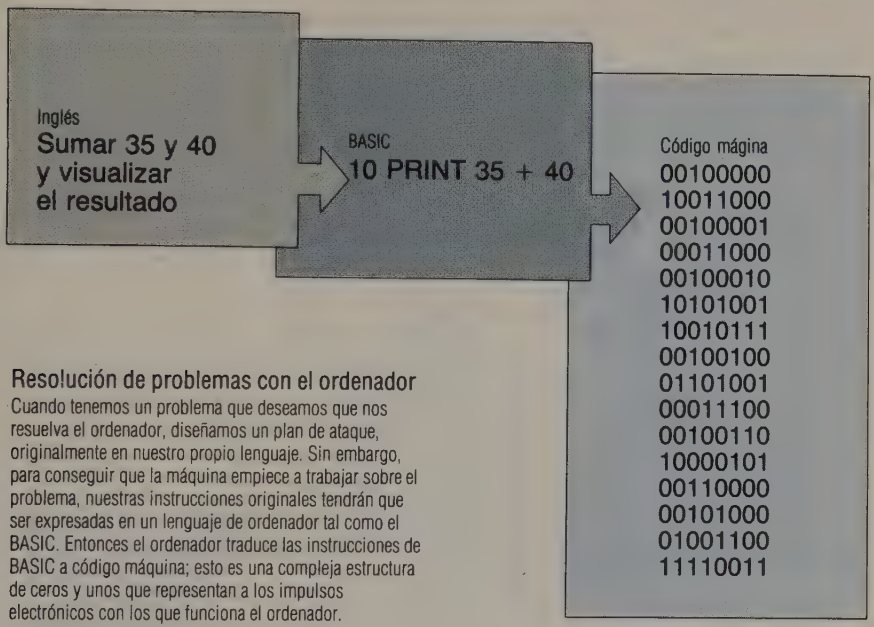
¿Qué es el BASIC?

No queremos confundirle con muchos detalles técnicos, por lo que a continuación le damos una explicación muy corta sobre lo que significa programar en BASIC.

Cuando usted piensa en algo que desea que el ordenador haga para usted —un cálculo que quiere que haga, una lista que quiere que guarde, una figura en la pantalla que quiere que dibuje o cualquier otra cosa— entonces usted lo piensa sencillamente en español. Así es como funcionamos. Pero no es así como funciona el ordenador. Este funciona a base de impulsos electrónicos. Vemos cómo funciona teniendo en cuenta que es binario, es decir, con dos estados. Cuando hay un pulso de electricidad, éste es un estado (le llamamos «1»); cuando no hay impulso será el otro estado (le llamaremos «0»). Se han desarrollado gran cantidad de formas ingeniosas de construir estructuras de unos y ceros y hacer que el ordenador juegue con ellas. Utilizando estas ingeniosas formas, el ordenador hace todas las cosas que nosotros esperamos que haga.

¡Afortunadamente, no hay que decirle al ordenador lo que tiene que hacer utilizando series de unos y ceros! En lugar de esto, en el interior del ordenador hay circuitos eléctricos que utilizan un lenguaje tal como el BASIC, lo que le permite entender este lenguaje y traducir nuestras sentencias de BASIC a series de unos y ceros.

Los lenguajes de ordenador, a diferencia del español, tienen un vocabulario limitado. Si quiere resolver un problema utilizando un ordenador o quiere conseguir que la máquina realice alguna tarea para usted, el problema —o la especificación de la tarea— debe ser descompuesto en una secuencia de instrucciones paso a paso que pueda ser escrita en el lenguaje del ordenador. Las instrucciones que tendrán que ser tecleadas en el ordenador constituyen el programa del ordenador. Por lo tanto, aprender a programar es aprender a analizar un problema y convertirlo en instrucciones que el ordenador pueda manejar, traducirlas a un lenguaje de ordenador y, naturalmente, probarlas para ver que funcionan.



Resolución de problemas con el ordenador

Cuando tenemos un problema que deseamos que nos resuelva el ordenador, diseñamos un plan de ataque, originalmente en nuestro propio lenguaje. Sin embargo, para conseguir que la máquina empiece a trabajar sobre el problema, nuestras instrucciones originales tendrán que ser expresadas en un lenguaje de ordenador tal como el BASIC. Entonces el ordenador traduce las instrucciones de BASIC a código máquina; esto es una compleja estructura de ceros y unos que representan a los impulsos electrónicos con los que funciona el ordenador.

Los diseñadores de lenguajes han hecho el BASIC (y varios de los otros lenguajes que utilizan los ordenadores) lo más cercano posible al lenguaje en el cual pensamos. Pero de todas maneras no es lo suficientemente cercano. El ordenador tiene una mentalidad muy literal, y todo lo que escribimos en BASIC tiene que ser traducido a una serie de unos y ceros. Como resultado, debemos estar seguros de que lo que le decimos al ordenador es claro y sin ambigüedades y que seguimos las reglas del lenguaje.

A primera vista, estas reglas —dónde hay que utilizar una coma, dónde un punto y coma, etc.— pueden parecer difíciles de manejar. Pero a medida que usted se acostumbre a programar, encontrará que utilizarlas viene de una forma natural. Cuanto más practique, más fácil encontrará el expresar sus ideas en BASIC.

El diagrama de la página 11 resume el camino a seguir desde que planteamos un problema hasta que conseguimos que el ordenador lo resuelva para nosotros.

¿Entiende el ordenador lo que usted le dice?

Si no se ciñe a las reglas del BASIC, su ordenador se lo dirá muy pronto. He aquí algunas reglas que le ayudarán a darse cuenta de cuándo el ordenador le dice que ha cometido algún error:

1) Si su ordenador hace lo que usted esperaba —o, mientras está empezando, lo que nosotros le decimos que espere—, no le da ningún mensaje o un mensaje tal como «OK» o «READY», entonces es que lo está haciendo bien.

2) Si su ordenador:

a) hace lo que usted esperaba, pero le da también un mensaje de error (probablemente lo reconocerá, pero si no está usted seguro, en el capítulo 5 hay una descripción de los mensajes de error),

b) no hace nada en absoluto (¡a menos que usted esperase esto!),

c) hace algo que usted no esperaba

entonces es que ha cometido un error en su sentencia o programa de BASIC.

Si ha cometido un error, entonces intente teclear una vez más su sentencia de BASIC, caso de que haya cometido algún error al escribirlo. Si el ordenador sigue sin aceptarlo, entonces es que hay algún otro error. Es casi seguro que el error está en su BASIC, y no en su ordenador. En este caso, no se preocupe. Este tipo de cosas le suceden a todos los programadores, incluso a los más expertos.

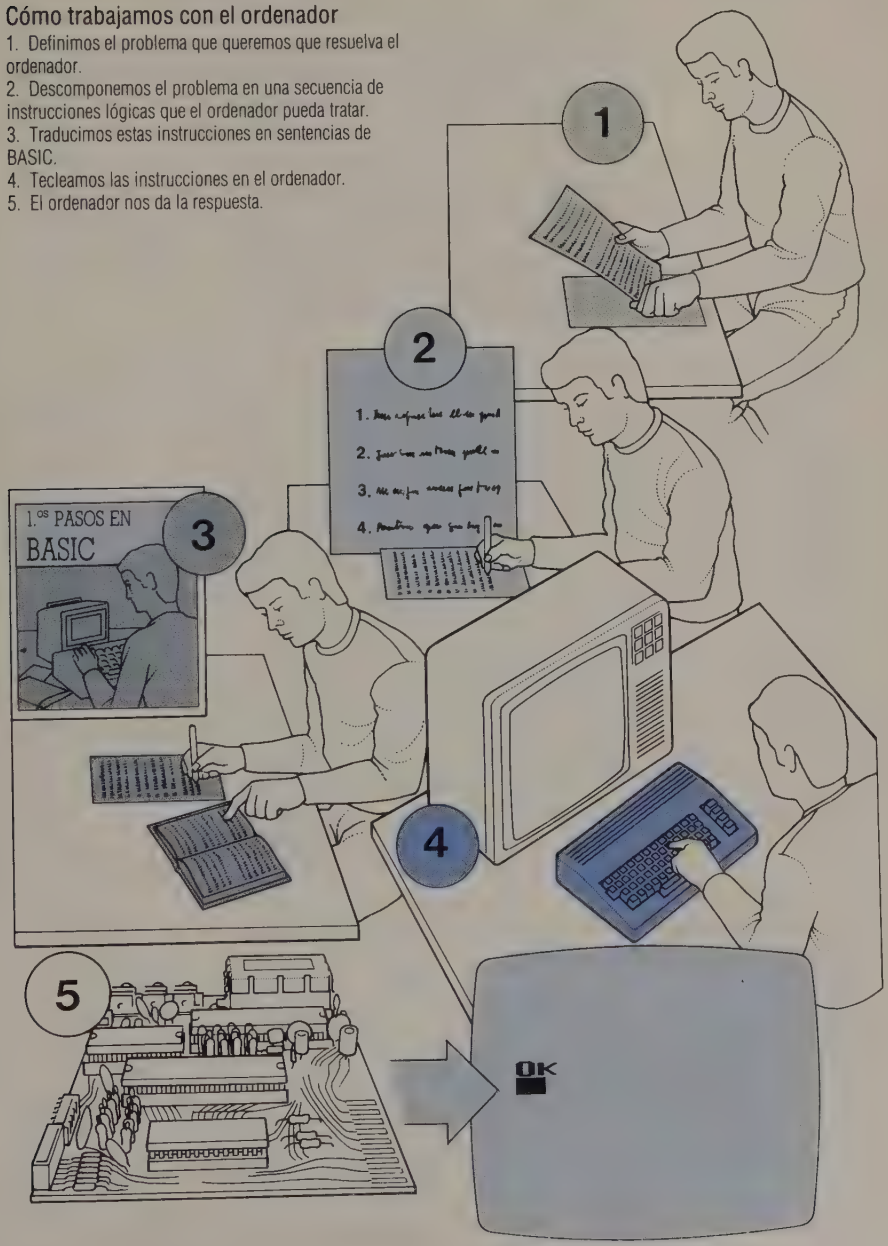
Probablemente se haya encontrado con una de las diferencias que existen entre nuestro BASIC y la versión de su ordenador. Si es así, el Apéndice 1 debe ayudarle a encontrarla.

Otra posibilidad es que usted haya cometido un pequeño desliz al planear o escribir su sentencia o programa. Si está usted haciendo uno de nuestros ejercicios, eche una mirada a nuestras respuestas de muestra para ver si le dan una pista sobre lo que ha ido mal. Si está intentando escribir su propio programa, busque consejo en el capítulo 5.

También es posible que se haya encontrado con una limitación en su ordenador respecto a la cantidad de información que puede almacenar, es decir, su ordenador puede haber agotado su capacidad de memoria.

Cómo trabajamos con el ordenador

1. Definimos el problema que queremos que resuelva el ordenador.
2. Descomponemos el problema en una secuencia de instrucciones lógicas que el ordenador pueda tratar.
3. Traducimos estas instrucciones en sentencias de BASIC.
4. Tecleamos las instrucciones en el ordenador.
5. El ordenador nos da la respuesta.



En la página 164 del capítulo 7 discutimos sobre tamaños de memoria y programas muy largos, pero usted no debe tener ningún problema de este tipo en estos primeros capítulos.

Ya sabemos que puede ser muy frustrante el encontrar que un programa tras otro no funcionan en absoluto como usted quisiera. Hemos intentado planear este libro de forma que esto no le suceda muy a menudo a medida que usted avanza en su lectura. Pero ciertamente le sucederá algunas veces. Tómeselo así: al menos está usted aprendiendo lo que son los mensajes de error.

También aprenderá otras cosas. Cuando haya terminado este libro estamos seguros de que será capaz de escribir sencillos programas de juegos, programas aritméticos y programas que almacenen y clasifiquen información. Entonces usted querrá seguir y aprender más sobre el BASIC e intentar programas más ambiciosos, por sí mismo. Para ayudarle, el último capítulo de este libro tiene una selección de programas para que usted los pruebe. No son únicamente ejercicios, sino que son programas reales que le enseñarán, sin más conocimiento del que habrá obtenido leyendo este libro, como puede usted hacer muchas cosas con su ordenador doméstico. Con algunos de ellos, hemos incluido muchas notas que describen su funcionamiento y el de los trucos de programación que hemos utilizado, de forma que al mismo tiempo que los utiliza, será capaz de aprender de ellos.

Ya es suficiente de introducción. Ahora vamos a empezar a convertirle en un programador de BASIC.

1 Cómo escribir en la pantalla

En este capítulo aprenderemos:
- **Las órdenes PRINT, PRINT@ y CLS.**
- **Cómo realizar aritmética sencilla con el ordenador.**
- **Cómo escribir mensajes en la pantalla.**
- **Cómo posicionar las salidas de la pantalla.**

Si va a seguir este libro, utilizando su propio ordenador para probar los ejemplos, ahora es el momento para empezar. Coloque y conecte su sistema, exactamente en la forma que le dice su manual. En algunas máquinas, hay que cargar las instrucciones que permiten al ordenador entender el BASIC, desde un cartucho, disco flexible o posiblemente un cassette. Hágalo, siguiendo las instrucciones de su manual. Quizá necesite también un papel y un lápiz, para poder tomar nota de lo que vaya descubriendo.

Repitiendo lo que ya dijimos en la introducción, no se preocupe si su ordenador no hace exactamente lo que nosotros decimos que hace el nuestro. Todos los ordenadores tienen sus diferencias. Si no está seguro de lo que está haciendo, o por qué, entonces léase otra vez la introducción, o mire en el Apéndice 1, donde se discuten las diferencias entre los dialectos del BASIC. Y deje cerca el manual de su ordenador. Le ayudará mucho.

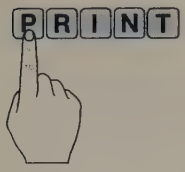
Entonces, empecemos a trabajar. Busque una tecla en el teclado (en su máquina o en el teclado de la figura de la página 198) que está marcada con RETURN, ENTER o NEWLINE.

Esta es una tecla muy importante. En nuestro ordenador está marcada con ENTER, y le indicaremos cuándo hay que apretarla mediante este símbolo:

{E}

Pruebe esto:

PRINT 1 {E}



Cada vez que encuentre «Pruebe esto», le estamos diciendo que teclee exactamente lo que viene a continuación, si es que tiene un ordenador donde hacerlo. En el caso de que no lo tenga, le diremos lo que sucede cuando utilizamos el nuestro.

«PRINT 1» es un ejemplo de una sentencia de BASIC. «PRINT» es una orden que le dice al ordenador que imprima algo en la pantalla de visualización. «1» es una expresión: en este caso, es lo que el ordenador tiene que imprimir.

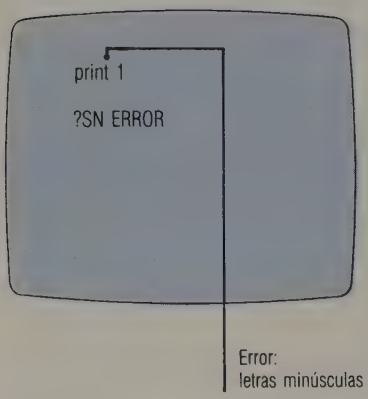
El ordenador imprime esto:

```
 1
OK
```

«OK» es el mensaje que nuestro ordenador nos da cuando ha terminado de hacer lo que le dijimos que hiciera. Otros ordenadores pueden dar el mensaje «READY». Nos avisa de que no hará nada más hasta que le demos más instrucciones, y por lo tanto es nuestro turno de hacer algo.

Las sentencias de BASIC empiezan con una orden. A continuación puede seguir una expresión, u otra cosa (que veremos muy pronto) o nada en absoluto. Probemos otro ejemplo:

```
print 1   {E}
```



Cuando cometemos errores

Si al ordenador no le gusta lo que escribimos, nos lo dirá mediante un mensaje de error. En este ejemplo, nuestro ordenador no acepta letras minúsculas, por lo que nos dice que hemos cometido un error. Si usted comete un error no se preocupe; no puede dañar al ordenador por el hecho de escribir algo incorrecto. Tan sólo tiene que intentar el seguir escribiendo sus instrucciones hasta que lo haga correctamente.

(Si su ordenador no puede generar minúsculas, entonces sáltese este ejemplo.)

Nuestro ordenador no aceptará órdenes de BASIC escritas en minúsculas. Nos dará un mensaje de error:

? SN ERROR

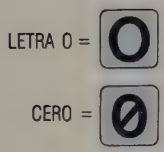
por lo tanto utilizaremos mayúsculas en lo que sigue de libro. Algunos ordenadores sí que aceptan letras minúsculas en las órdenes o en otros lugares. Si el suyo lo hace, entonces puede utilizar letras minúsculas donde nosotros utilizamos letras mayúsculas si así lo prefiere.

Si su ordenador no acepta ninguno de estos ejemplos, entonces es que tiene problemas —¡su ordenador no entiende el BASIC! Vuelva a su manual, o a su representante, si es necesario, y descubra el porqué. No puede utilizar su ordenador para ejecutar los programas de este libro a menos que acepte una de las dos sentencias anteriores.

Naturalmente, el ordenador también puede escribir otras cosas:

Pruebe esto:

PRINT 1234567890 {E}



Esto le tiene que haber enseñado cómo aparecen en la pantalla de su ordenador todos los números. Probablemente, han aparecido en un formato muy poco familiar, tal como

1.23456789E+09

lo cual significa que el número es demasiado grande para ser manejado por el ordenador de una forma normal, por lo que lo ha puesto en una notación científica. No se preocupe si no entiende esto: no es necesario verlo en detalle. Para descubrir la longitud máxima de un número que su ordenador puede manejar normalmente, puede probar de escribir:

PRINT 12345
PRINT 123456

y así sucesivamente, hasta que aparezca la notación científica.

Nuestra pantalla aparecerá de la siguiente forma:

```
PRINT 1
 1
OK
print 1
? SN ERROR
PRINT 1234567890
 1.23456789E+09
OK
PRINT 123456789
 123456789
OK
```

Nótese que los unos están debajo de la «R» de PRINT, no debajo de la «P». ¿Por qué? Nuestro ordenador suele dar un signo a todos los números (+ o –). De todas formas aquí no ha escrito el signo +, pero ha dejado un espacio para él, y por esto escribe el número, corrido un espacio a la derecha de donde esperábamos que estuviera.

Pruebe esto:

```
PRINT –12345   {E}
```

Esta vez nuestra entrada y salida aparecerá de la siguiente forma en la pantalla:

```
PRINT –12345
–12345
OK
```

con el signo – exactamente donde antes estaba el espacio en blanco.

Qué más puede hacerse con los números

Vamos a intentar algo un poco más sofisticado:

Pruebe esto:

```
PRINT 25+39   {E}
```

Obtenemos:

```
64
```

(No vamos a preocuparnos a partir de ahora de poner

el OK en los ejemplos. Suponga que cada vez aparece en la línea siguiente.)

«+» es una función. Hay muchas funciones en BASIC, y junto con los números (a veces) constituyen las expresiones. Así pues

1

es una expresión constituida tan sólo por un número, y

25+39

es también una expresión, pero esta vez constituida por dos números y la función «+». Cuando el ordenador encuentra una expresión, en una sentencia como ésta, la evalúa; en otras palabras, hace toda la aritmética y entonces imprime el resultado.

La siguiente expresión tiene una sutil diferencia: Pruebe esto:

PRINT 25 + 39 {E}

El resultado es el mismo que en la anterior expresión. Evidentemente, al ordenador no le importa si dejamos o no espacios entre los números y las funciones. Nosotros utilizamos espacios, sólo para poder hacer que las sentencias sean más fáciles de leer. Se pueden suprimir todos ellos, incluso el que está después del PRINT si así lo deseamos. Naturalmente no podremos poner espacios en medio de las palabras que constituyen una orden; PR INT no funcionará como orden.

Probemos algunas expresiones más. Esta vez, daremos la lista de las salidas del ordenador (en otras palabras lo que aparece en la pantalla) a la derecha de nuestra entrada (lo que escribimos en el teclado):

Nosotros	*El ordenador*
PRINT 3 + 4 + 5 {E}	*12*
PRINT –3 –4 –5 {E}	*–12*
PRINT 3 – 4 – 5 {E}	*–6*
PRINT 3.5 + 4.6 {E}	*8.1*
*PRINT 4*6 {E}*	*24*
PRINT 5/2 {E}	*2.5*
PRINT –5/2 {E}	*–2.5*

Estos ejemplos ilustran las funciones aritméticas básicas del BASIC. Como ya habrá visto, los símbolos que utilizamos para estas funciones son ligeramente diferentes de los que usamos cuando escribimos aritmética. Son los siguientes:

+ suma
– resta
* multiplicación
/ división

La mayoría de los ordenadores tienen también muchas otras funciones matemáticas. El nuestro acepta, por ejemplo, expresiones tales como:

Pruebe esto:

PRINT SQR(25) {E}

Pruebe esto:

PRINT COS(45) {E}

Nos da primeramente la raíz cuadrada de 25 (5, por si su aritmética está demasiado oxidada) después el coseno de 43 (.555113299). Algunos computadores tienen BASICs que no incluyen estas órdenes. No se preocupe: no las vamos a utilizar otra vez.

Hay muchas otras reglas especiales acerca de cómo utilizar las funciones aritméticas del BASIC, por ejemplo, el orden en que el ordenador ejecuta expresiones tales como:

*25*4+23/96*(2 + 3)*

No analizaremos esto con detalle, y probablemente tampoco querrá utilizarlas a menudo a menos que quiera hacer muchas matemáticas. Si quiere hacerlo, encontrará una explicación detallada en el manual de su ordenador o en cualquier libro más orientado a las matemáticas y que trate sobre el BASIC.

Ahora es su turno

Intente encontrar lo que hay que escribir para conseguir que el ordenador haga lo siguiente. Si tiene un ordenador, intente conseguir sus respuestas y ver si

son correctas. Si no lo tiene, en la página 200 le damos algunas respuestas dadas por el nuestro. Tenga presente que para estos y todos los otros ejemplos que siguen, no hay una única respuesta correcta. Por ejemplo, usted puede haber colocado espacios donde nosotros no, o viceversa, y aún así haber escrito sentencias que funcionan perfectamente. La única manera directa de encontrar si son correctas o si se ha equivocado es probarlas en un ordenador y ver si el ordenador las acepta. Sin embargo, nuestras respuestas pueden ayudarle a encontrar qué es lo que hay que hacer, o mostrarle qué es lo que ha olvidado.

1) Haga que el ordenador sume 56,4 y 76,3
2) Haga que el ordenador divida 74 por 13,1
3) Haga que el ordenador multiplique 45 por 7
4) Haga que el ordenador reste 96 de 1025.

El ordenador y las letras

Programar en BASIC no consiste únicamente en matemáticas. Veamos ahora cómo podemos conseguir que el ordenador utilice letras en lugar de números. Para hacer que el ordenador escriba una letra hay que ponerla entre comillas, así:

Pruebe esto:

```
PRINT "A"
```

El ordenador escribirá:

```
A
```

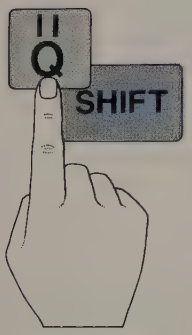
pero si usted saca las comillas, no lo hará.

Pruebe esto:

```
PRINT A
```

¿Qué ha escrito? Probablemente 0 (como ha hecho el nuestro) o bien un código de error. Debido a que el ordenador puede hacer tantas cosas con lo que usted escribe por el teclado, tiene reglas especiales que le dicen lo que hay que hacer con cada letra que usted escribe. En una sentencia tal como:

```
PRINT A
```



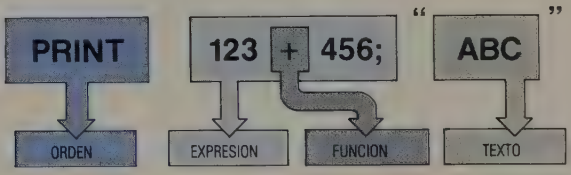
Escritura de las comillas
En la mayoría de los teclados, para escribir las comillas en la pantalla tendrá que apretar la tecla SHIFT, además de la tecla en cuestión.

sus reglas le dicen que A tiene un significado especial, y por lo tanto no se limita a escribirla como hizo con los números. (Veremos muy pronto cuál es este significado especial.) Utilizamos las comillas para decirle al ordenador que no queremos que la letra tenga esta vez su significado especial; lo mismo sucede con otras letras.

Pruebe esto:

Nosotros	*El ordenador*
PRINT "B" {E}	*B*
PRINT "BLANCO" {E}	*BLANCO*
PRINT "BLANCO Y NEGRO" {E}	*BLANCO Y NEGRO*
PRINT " MIRA A LA DERECHA" {E}	*MIRA A LA DERECHA*
PRINT "MIRA A LA IZQUIERDA" {E}	*MIRA A LA IZQUIERDA*
PRINT " ""DERECHA OTRA VEZ" {E}	*DERECHA OTRA VEZ*

Sentencias de BASIC



Una sentencia de BASIC consiste en una palabra de orden, generalmente seguida, ya sea por los caracteres que constituyen una expresión o por caracteres colocados entre comillas llamados textos, o ambas cosas. En la sentencia, la expresión está constituida por dos números y una función (el signo más).

Esto nos muestra una regla nueva. Cuando colocamos un espacio entre comillas, el ordenador lo tendrá en cuenta, mientras que cuando colocamos el espacio fuera de las comillas, el ordenador lo ignora.

Las expresiones colocadas entre comillas se conocen en el argot del ordenador como textos. Los textos pueden contener letras —como en los textos anteriores— o números, espacios, símbolos de puntuación o cualquier símbolo, que podamos escribir por el tecla-

do o describir al ordenador de otra forma. Pruebe todo esto:

Nosotros	*El ordenador*
PRINT "12345" {E}	*12345*
PRINT "12 + 345" {E}	*12 + 345*
PRINT "12 – 345" {E}	*12 – 345*
PRINT "12 – 345 =" 12 – 345 {E}	*12 – 345 = – 333*
PRINT "MANZANA,PERA,MELOCOTON" {E}	*MANZANA,PERA,MELOCOTON*
PRINT "..,,;;;;;;???" {E}	*..,,;;;;;;???*

¿Ha observado que el ordenador no ha evaluado las expresiones, sumando y restando, como lo hizo cuando utilizamos números fuera del texto? En el cuarto ejemplo, hemos utilizado esto para conseguir que el ordenador escriba las dos cosas, una expresión y su resultado. (En algunas máquinas quizá tendrá que teclear un «;» antes del segundo conjunto de caracteres entre comillas para conseguir que este ejemplo funcione. Le introduciremos en el uso del punto y coma fuera de los textos, en la página 28.) ¿Se ha dado cuenta también de que esta vez falta el espacio inicial que antes aparecía delante de los números positivos?

¿Cuál es la longitud de un texto?

Cada ordenador tiene un límite en cuanto a la longitud de los textos que puede manejar. A veces es un límite bastante bajo —generalmente 255 caracteres, lo que es una cantidad conveniente para ser almacenada por el ordenador—. A veces, un solo texto puede ocupar virtualmente todo el espacio de memoria del ordenador (miles de caracteres). Su manual le dirá qué es lo que acepta su ordenador. Sin embargo, textos muy largos son raramente útiles, y es más sencillo dividirlos en series de textos más cortos. No utilizaremos textos de más de 64 caracteres en este libro.

Generalmente, también hay un límite en cuanto a la longitud de las sentencias que puede usted escribir antes de apretar la tecla ENTER. Sorprendentemente quizás, este límite es a veces más corto (y nunca más largo) que la longitud máxima del texto. (Existen varias maneras peculiares de conseguir colocar largos textos en la memoria del ordenador, a partir de sentencias muy cortas que nosotros tecleamos.) Esto significa que no podemos, utilizando una única sentencia larga, in-

tentar ver cuál es la longitud máxima de un texto; lo que sí podemos ver es si la longitud máxima de la sentencia es lo suficientemente corta para preocuparnos, intentando escribir una sentencia que contenga un texto muy largo. Debe mostrarnos varias cosas interesantes.

Pruebe esto:

```
PRINT «ESTO ES UNA PRUEBA PARA
VER SI MI ORDENADOR PUEDE
MANEJAR LARGAS SENTENCIAS.
SEGUIRE TECLEANDO HASTA QUE EL
ORDENADOR ME DETENGA, O HASTA
QUE HAYA LLEGADO TAN LEJOS, QUE
LA LONGITUD DE LA SENTENCIA DEJE
DE PREOCUPARME"  {E}
```

¡Esperamos que no haya apretado la tecla ENTER hasta que se lo hayamos dicho! Como habrá visto, no hay necesidad de apretar esta tecla cuando el ordenador llega al final de la línea de la pantalla, de la misma forma que usted hubiera tenido que apretar el retorno de carro en una máquina de escribir, ya que el ordenador puede empezar una nueva línea por sí mismo. De todas maneras, al ordenador no le importa si está usted en medio de una palabra o expresión, no lleva (en argot del ordenador «wrap») la palabra a la línea siguiente; ya que no puede provocar un retorno de carro apretando ENTER sin causar que el ordenador empiece a ejecutar, tendrá que conformarse, con las palabras partidas que resultan, o escribir una serie de espacios cuando se acerque al final de la línea de la pantalla. Nuestra línea de pantalla contiene 32 caracteres. Si la suya es más corta o más larga, entonces las rupturas aparecerán, naturalmente, en puntos distintos.

¿Qué ha sucedido? Su ordenador quizá le haya permitido escribir todo el texto y lo ha sacado por la pantalla sin ninguna queja. O quizá le ha dejado escribir, digamos, las dos primeras líneas en la pantalla y entonces se ha quedado parado. También puede haber aceptado todo el texto, y haberle dado un mensaje de error en lugar de reproducirlo. En este caso, o si su ordenador ha aceptado alegremente esta sentencia, entonces su manual le dirá cuál es su máximo.

Evidentemente, es útil el que el ordenador le permita utilizar largas sentencias ocasionalmente, pero para nuestros propósitos no importa lo que haya sucedido en su ordenador. No utilizaremos sentencias más largas de 64 caracteres, lo que es un límite muy bajo y que casi todas las máquinas aceptarán.

Nótese que cuando el ordenador escriba su texto, las rupturas del final de línea de pantalla estarán en sitios distintos, ya que no escribe la palabra PRINT. Si no ha colocado espacios extras, habrá aparecido algo tal como:

ESTO ES UNA PRUEBA PARA VER SI MI ORDENADOR PUEDE MANEJAR LARGAS SENTENCIAS. SEGUIRE TECLEANDO HASTA QUE EL ORDENADOR ME DETENGA, O HASTA QUE HAYA LLEGADO, TAN LEJOS QUE LA LONGITUD DE LA SENTENCIA DEJE DE PREOCUPARME

Si quiere que la salida aparezca correctamente, entonces habrá que corregirlo, como ya veremos cuando empecemos a escribir programas.

Textos con aritmética

Un aspecto interesante de los ordenadores es que se puede utilizar la función de suma que hemos visto anteriormente, con textos en lugar de números. A menudo queremos convertir textos cortos en otros más largos, como en el caso en que estemos realizando, por ejemplo, proceso de palabras.

Pruebe esto:

PRINT "PALO"+"SANTO" {E}

¿Qué es lo que nos escribirá el ordenador? Directamente:

PALOSANTO

Sin embargo, hay una trampa para el que no vaya con cuidado:

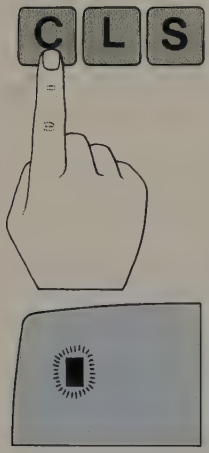
PRINT "22"+"33" {E}

Esto no da como resultado «55», sino el contenido de los dos textos unidos en uno solo:

2233

Por lo tanto, piénseselo dos veces antes de hacer la suma de dos textos que tienen números.

El cursor



Si todavía no lo ha hecho, eche una mirada ahora al cursor. Es el pequeño símbolo (quizás un cuadrado o línea que parpadea, quizás una letra que parpadea) que aparece en la pantalla cuando el ordenador está esperando que usted teclee algo. Puede moverlo alrededor de la pantalla (aunque no siempre por encima de donde está actualmente) mediante las teclas de cursor. Busque nuestras teclas en el teclado de la página 199. Tienen flechas para mostrar la dirección en la que se puede hacer mover al cursor.

El cursor es muy útil, porque le dice en qué lugar de la pantalla aparecerá lo que está usted tecleando. Hasta ahora, esto ha sido en la siguiente línea vacía, pero aprenderemos muy pronto cómo cambiar esto. Naturalmente, la posición del cursor se puede cambiar con facilidad apretando únicamente las teclas de cursor, pero a menos que vaya con cuidado, puede confundir al ordenador y, lo que estaba escribiendo, tenerlo en la pantalla en posiciones extrañas, a menudo exactamente donde no lo quería.

De todas maneras no se ve el cursor cuando le toca escribir al ordenador, pero usted puede seguir pensando que está allí. Algunas instrucciones, que puede darle al ordenador, pueden moverlo, sin que obliguen al computador a escribir algo en la pantalla:

Pruebe esto:

CLS {E}

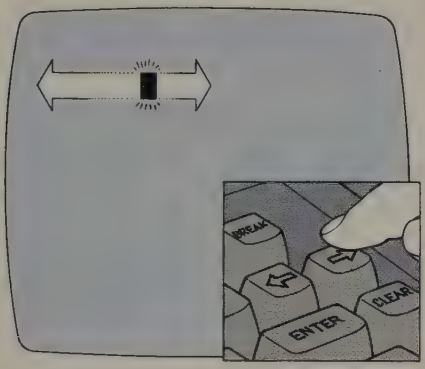
En la mayoría de los ordenadores (como el nuestro), esta orden le dice al computador que:

a) Limpie la pantalla.

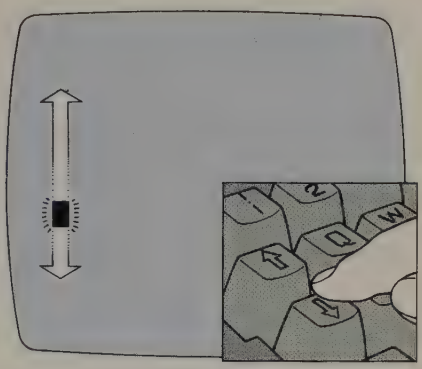
b) Ponga el cursor en el extremo superior izquierdo de la pantalla.

No hace falta decirle al ordenador el PRINT, cuando

Movimiento del cursor



El cursor es el símbolo parpadeante de la pantalla que le muestra dónde aparecerá el siguiente carácter que usted tecleará. Apretando las distintas teclas de cursor podrá



moverlo hacia arriba, hacia abajo, hacia la derecha o hacia la izquierda, es decir, podrá posicionarlo en cualquier lugar de la pantalla.

le decimos CLS.CLS es una orden, al igual que PRINT, y tan sólo hace falta una orden al principio de una sentencia.

Algunos ordenadores tienen distintas órdenes para limpiar la pantalla y para colocar el cursor en el extremo superior izquierdo. Otros tienen una tecla especial, que usted puede utilizar en lugar de la orden. Si CLS no funciona en su máquina, busque en su manual un método alternativo, o consulte el Apéndice 1 de este libro. Quizás encuentre ayuda mirando nuestras notas sobre distintas máquinas en las páginas 190, 191.

Puntuación

Ya hemos utilizado símbolos de puntuación, tal como comas o puntos, en los textos, pero todavía no hemos intentado usarlos fuera de las comillas. Vamos a probarlo ahora:

Pruebe esto:

```
PRINT 1,2,3,4,5,6  {E}
```

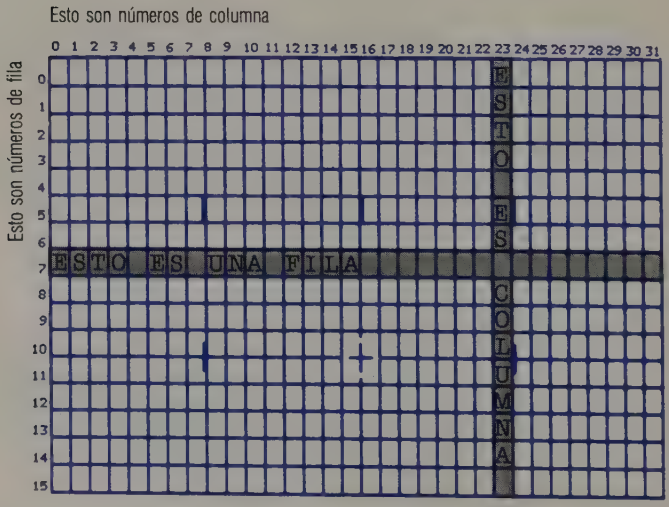
En la pantalla debe aparecer algo tal como el conjunto de cifras que hay a continuación.

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

¿Por qué? Existe un número determinado de «zonas de escritura» a lo largo de su pantalla: usted no puede verlas, pero el ordenador lo sabe todo sobre ellas. Cuando usted coloca una coma en una sentencia de PRINT (excepto en un texto, donde aparece como una simple coma), entonces el ordenador toma esto como su señal para hacer saltar el cursor hasta el principio de la siguiente zona. Si está ya en la última zona, entonces empieza una nueva línea. La coma no se escribe en la pantalla.

El número de zonas y el número de espacios en cada zona dependen del número de caracteres que su ordenador puede escribir en cada línea. Las zonas suelen tener ocho o diez espacios cada una. Nuestra línea de 32 caracteres tiene cuatro zonas con ocho espacios en cada una.

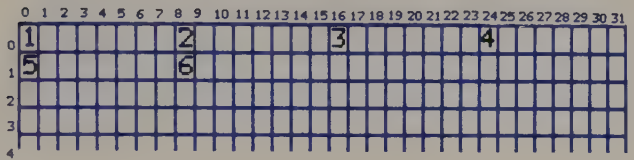
También puede interpretarse esto como posiciones de columna. Su pantalla puede dividirse en un reticulado de pequeños cuadrados o rectángulos en cada uno de los cuales el ordenador puede escribir un carácter, si así se lo indica, o que constituye cada una de las posiciones donde usted puede enviar el cursor. Una línea escrita a lo ancho de la pantalla constituye una fila y una línea escrita de arriba a abajo es una columna.



Mapas de pantalla
Este cuadriculado puede interpretarse como un tipo de «mapa» del área de pantalla del ordenador.

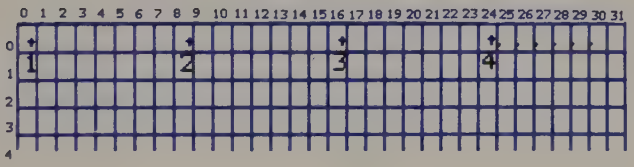
Podemos darle a cada fila y a cada columna un número. Por lo tanto en nuestra pantalla de 32 columnas, las zonas de escritura empiezan en las columnas 0, 8, 16 y 24, como se muestra en el dibujo:

Zonas de visualización
En nuestras pantallas hay cuatro zonas de visualización sobre las que se mueve el cursor cuando utilizamos una coma en las sentencias de PRINT.



Naturalmente, cuando le pedimos al ordenador que escriba 1, 2, 3, 4, empieza en la columna número 1 y no en la cero, debido al invisible signo +.

Números en las zonas de visualización
La coma colocará los números una columna a la derecha del principio de la zona de visualización.



Nótese que es normal para los números de columna —y los números de fila— empezar con cero, es decir, que la pantalla de 32 columnas termina en la columna 31 y una pantalla de 16 filas termina en la fila 15.

Naturalmente, podemos utilizar la misma técnica con los textos. Probemos una vez con textos cortos:

Pruebe esto:

```
PRINT "ROJO","VERDE","AZUL","ROSA
"  {E}
```

y una vez con otros más largos:

Pruebe esto:

```
PRINT "AGUAMARINA","ROJO CEREZA","VERDE
ESMERALDA"  {E}
```

En la pantalla deberá aparecer algo como esto, de-

pendiendo del tamaño de su pantalla y del de las zonas de escritura:

ROJO VERDE AZUL ROSA

AGUAMARINA ROJO CEREZA
VERDE ESMERALDA

Vamos a hacer algo ligeramente distinto:
Pruebe esto:

PRINT 1;2;3;4;5;6 {E}

En la pantalla aparecerá algo tal como:

1 2 3 4 5 6

El ordenador acaba de escribir todos los números con un único espacio entre ellos, pero ya que utiliza signos + invisibles, entonces, si se trata de números positivos habrá un espacio extra entre cada uno de ellos para permitir esto. Podemos demostrarlo comprobando la versión negativa:
Pruebe esto:

PRINT –1;–2;–3;–4;–5;–6 {E}

Esta vez, obtenemos:

–1 –2 –3 –4 –5 –6

También podemos usar puntos y comas en los textos.
Pruebe esto:

PRINT "ROJO";"NARANJA";"AZUL";"ROSA" {E}

Esta vez no hay espacios. Nuestra pantalla aparecerá así:

ROJONARANJAAZULROSA

Ahora le toca a usted
Vea si puede conseguir que el ordenador haga estas

cosas. De nuevo, algunas respuestas que funcionan en nuestro ordenador están en la página 200.

(5) Escriba su nombre, directamente junto al lado derecho de la pantalla. (Sugerencia: utilice espacios en el texto.)

(6) Escriba su nombre de arriba a abajo en una columna. (Sugerencia: utilice muchas comas.)

(7) Escriba «ROJO», «NARANJA», «AZUL» y «ROSA» con sólo un espacio entre cada palabra, utilizando puntos y comas.

Cómo utilizar la puntuación para separar los artículos que hay que imprimir

¿Hay que utilizar símbolos de puntuación entre los artículos que se van a imprimir, si hay más de uno después de la orden de PRINT? Bien, descúbralo:

Pruebe esto:

```
PRINT "ROJO""NARANJA""ROSA""AZUL"  {E}
```

Nuestro ordenador no lo acepta, aunque algunos sí.

Pruebe esto:

```
PRINT "ROJO"1"NARANJA"2"ROSA"3"AZU
L"4  {E}
```

De nuevo, nuestro ordenador no lo acepta, pero quizás el suyo sí.

Nuestro ordenador nos permitirá utilizar números y textos después de la misma orden de PRINT, aunque:

Pruebe esto:

```
PRINT "ROJO";1;"NARANJA";2;"ROSA";
3;"AZUL";4  {E}
```

En la pantalla aparecerá algo tal como:

```
ROJO  1  NARANJA  2  ROSA  3  AZUL  4
```

Es casi seguro que el suyo también lo aceptará. Si no, descubrirá en seguida como conseguir el mismo efecto utilizando distintas sentencias.

Dos puntos

Tan sólo hay otro símbolo de puntuación que tenga un significado especial después de la orden de PRINT.

Se trata de los dos puntos (:). Puede probar otros símbolos si así lo desea; pueden ampliar su conocimiento sobre los mensajes de error de su ordenador.

Pruebe esto:

```
PRINT "ROJO":"NARANJA":"VERDE":"PU
RPURA"  {E}
```

Conseguimos esta salida:

```
ROJO
?SN ERROR
```

(¿Recuerda? Esto es un mensaje de error.)

¿Dónde están el NARANJA, VERDE y PURPURA? El ordenador los ha ignorado. Esto se debe a los dos puntos al final de la sentencia de BASIC. Las órdenes sólo funcionan dentro de una única sentencia, por lo tanto el ordenador no utiliza la orden PRINT para imprimir cualquier cosa que esté después de los dos puntos. Nos ha dado un mensaje de error porque ha considerado que «NARANJA» era la sentencia siguiente.

En muchos ordenadores, utilizando los dos puntos, usted puede poner más de una sentencia en una línea de BASIC (hasta que usted no apriete la tecla ENTER; el texto puede ocupar más de una línea en la pantalla).

Pruebe esto:

```
PRINT "ROJO":PRINT "NARANJA":PRINT
"VERDE"  {E}
```

Esta vez, nuestro ordenador imprimirá los tres artículos, en una lista tal como ésta:

```
ROJO
NARANJA
VERDE
```

¿Lo ha hecho el suyo? Si no, significa que tan sólo puede utilizar una sentencia en cada línea. Esto no es un grave problema, ya que podrá hacerlo igualmente de esta forma cuando se trate de escribir programas.

¿Se acuerda de cuando analizamos la longitud máxima de una sentencia en la página 22? Aquello era la

longitud de línea máxima. Si se pone más de una sentencia en la línea, entonces tan sólo podrá llegar a la longitud máxima de una sentencia o a menos.

Ahora podemos descubrir otro uso del punto y coma, intentando colocar uno al final de una sentencia:

Pruebe esto:

PRINT "NO";:PRINT "BAJA" {E}

Después de la primera sentencia, el cursor no desciende al principio de la línea siguiente como lo haría normalmente, sino que aparece así:

NO BAJA

Cómo planear la impresión en la pantalla

Como ya hemos visto, pueden utilizarse los símbolos de puntuación para ayudarnos a decirle al ordenador dónde debe imprimir en la pantalla.

Sin embargo, esto puede ser engorroso cuando tratemos de dibujar una imagen elaborada, o colocar algunas figuras complejas en el ordenador. He aquí un método más fácil.

Primeramente, utilice CLS para limpiar la pantalla. Entonces:

Pruebe esto:

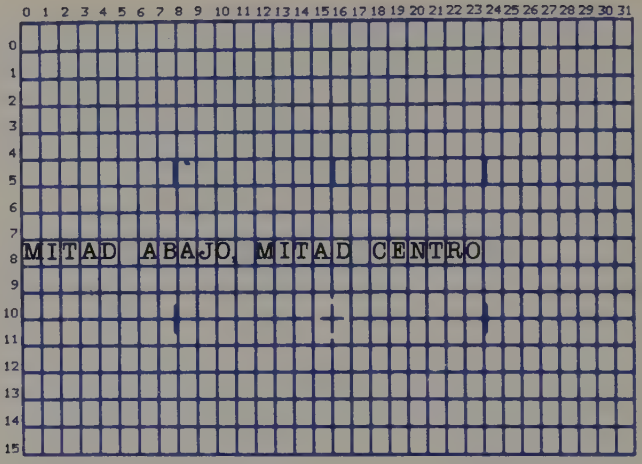
PRINT@ (8,0), "MITAD HACIA ABAJO Y MITAD HACIA EL MEDIO" {E}

Nuestro ordenador lo ha aceptado, y ha hecho exactamente lo que se le dijo. Lo que aparece en la pantalla es algo similar a lo que muestra el dibujo siguiente.

¿Lo ha hecho el suyo? Es bastante posible que no. Esta es una orden sobre la cual los ordenadores difieren en gran manera. Por eso, explicamos algunas variantes con detalle en las páginas 185 a 187. Recurra a ellas ahora, si necesita ayuda. La posibilidad de establecer dónde aparecerán los caracteres en la pantalla —que es lo que hace la orden PRINT@— es de tan gran ayuda que si PRINT@ no funciona en su ordenador, vale la pena gastar cierto tiempo para encontrar la manera alternativa de hacerlo.

Ya habrá supuesto que los números entre paréntesis

Visualización en una posición determinada de la pantalla
Así es como aparece nuestra pantalla después de haber entrado la sentencia PRINT@ de la página 31. (Para mayor claridad no le mostramos la propia sentencia, o el «OK».)



son los números que le dicen al ordenador dónde imprimir. En nuestra versión del BASIC, el primer número es el número de fila, el segundo es el número de columna. Son exactamente los mismos números que utilizamos al hablar de zonas de escritura. La orden «PRINT@» le indica al ordenador que debe mover su cursor hasta esta posición y entonces empezar a imprimir.

(En nuestro BASIC, la coma después del paréntesis de la derecha es una parte esencial de la estructura de la orden: no mueve al ordenador hasta la siguiente zona de impresión, como lo haría normalmente.)

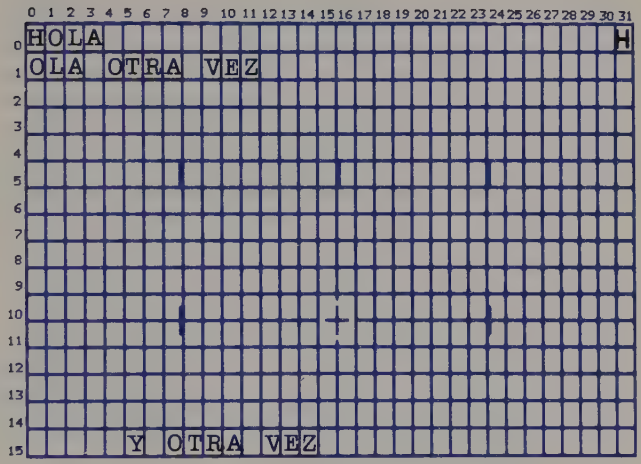
Veamos esto con un poco más de detalle. Sin mirar al dibujo siguiente, ¿dónde cree usted que las sentencias siguientes harán escribir al ordenador?

PRINT@ (0,0), "HOLA"
PRINT@ (0,31), "HOLA OTRA VEZ"
PRINT@ (15,5), "Y OTRA VEZ"

Para ayudarle a decidir, eche otro vistazo a la figura de la página 26, o examine la imagen anterior. Puede ver que cada dibujo sirve también como un tipo de mapa de la pantalla de nuestro ordenador, que le muestra las 16 filas y las 32 columnas. Eche una mirada

al manual de su ordenador y es casi seguro que encontrará un mapa similar. Quizá tenga un número diferente de filas y columnas en cuyo caso deberá utilizar su propio mapa, no nuestros dibujos, para planear dónde quiere que sea realizada la impresión.

Más posiciones de pantalla especificadas
Este dibujo le muestra dónde la orden PRINT@ ha situado nuestros tres textos de la página 32.

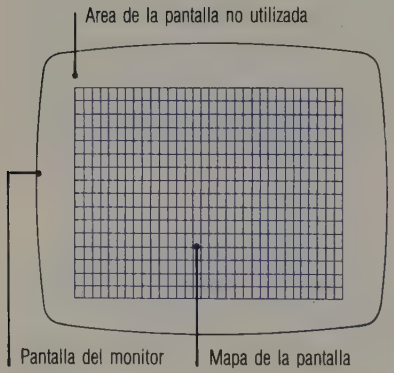


El dibujo le muestra dónde ha aparecido la impresión. ¿Acertó usted? Nótese que cuando le decimos al ordenador que ponga su cursor en la columna 31, hace exactamente lo que le decimos, y no escribe ninguna letra extra al principio de la línea siguiente. Si quisiéramos que apareciese «HOLA OTRA VEZ» completamente en la parte derecha de la pantalla, ¿dónde deberíamos haber puesto el cursor? Utilice su mapa para resolverlo.

Quizás encuentre varios mapas en su manual, con diferentes números de filas y columnas en cada uno. Esto le muestra diferentes maneras de utilizar la pantalla de su ordenador. Generalmente no puede colocar caracteres en la pantalla cuando utilice los mapas que tienen el mayor número de posiciones, por lo tanto asegúrese de que ha escogido el correcto, o al menos uno de los aceptables, si su ordenador le da la posibilidad de elegir. Asegúrese de que su ordenador está

trabajando actualmente con este número de posiciones. Rellene la pantalla de caracteres y cuente las filas y columnas. Ya que se trata de un libro primario de BASIC, no tratamos de cubrir todas las variaciones, utilizaremos únicamente la «resolución» de nuestro ordenador que es de 16 × 32.

Incidentalmente, usted descubrirá probablemente que ambos, lo que visualiza el ordenador y el mapa de la pantalla, no cubren por completo la pantalla del monitor o del televisor. Esto es debido a que la imagen aparece distorsionada en los bordes, por lo que el ordenador no los utiliza para texto. Si es un ordenador en color, quizá tenga un conjunto especial de órdenes que le permitan cambiar el color de los bordes, de forma que no quede feo cuando usted dibuje imágenes en la pantalla.



Llenado de la pantalla del televisor
La mayoría de los ordenadores utilizan únicamente la porción central de la pantalla, dejando un amplio margen en los bordes. No podrá escribir ningún carácter dentro de este margen, pero sí que podrá colorearlo.

Pueden comprarse cuadernos especiales en muchas tiendas de ordenadores que consisten en gran cantidad de mapas de pantalla, en blanco. Puede utilizarlos para planear cómo quiere que aparezca su pantalla cuando esté escribiendo un programa que utiliza gran cantidad de órdenes de posicionado. De hecho, podría utilizar uno ahora mismo, ya que vamos a realizar un dibujo.

Realización de un dibujo

Vamos a utilizar la orden de PRINT@ y las reglas de puntuación que acabamos de ver, e intentemos dibujar algunas imágenes muy sencillas, en la pantalla. Ya que

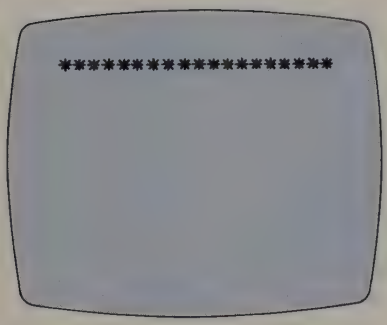
nuestro límite de caracteres por sentencia o de series de sentencias en una línea es 64, hemos planeado la mayoría de nuestros ejemplos con este número en mente. Usted, de hecho, puede construir una imagen con varias líneas de instrucciones, pero ya que no puede limpiar la pantalla después de cada una sin perder la imagen que estaba saliendo, terminará con ambas cosas, sus instrucciones y el dibujo; en la pantalla, afortunadamente, no estarán mezcladas.

He aquí algunas cosas que usted puede hacer con 64 caracteres, si es avaro con los espacios. Recuerde que el ordenador no está interesado en los espacios, a menos que estén encerrados entre comillas, por lo tanto puede suprimirlos. Todos ellos cuentan en este total de 64 caracteres.

a) Escriba una línea de asteriscos a través de la parte superior de la pantalla. Necesitará utilizar un texto muy largo que los contenga todos. No olvide las comillas.

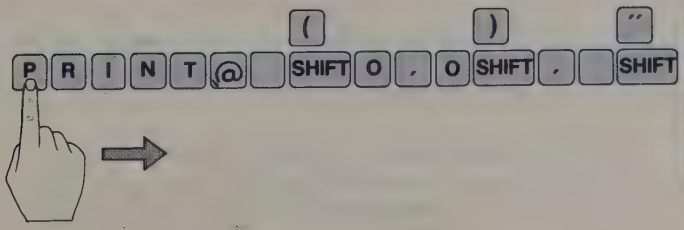
Pruebe esto:

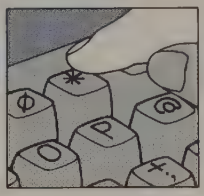
```
CLS: PRINT@ (0,0), "************
********************"  {E}
```



Realización de dibujos

Mediante la orden PRINT@, podrá posicionar un carácter en cualquier lugar de la pantalla. Por lo tanto, la orden PRINT@ puede utilizarse para crear sencillas formas o para decorar la pantalla. En este ejemplo, borramos primeramente la pantalla con la orden CLS y después posicionamos los asteriscos, contenidos en un texto, donde queremos que aparezcan.





Aprenderemos maneras más fáciles de hacer esto, más tarde. Nótese cómo hemos combinado aquí dos sentencias: una con la orden CLS y otra con la orden PRINT.

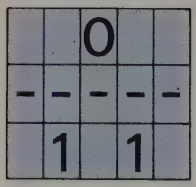
b) Divida la pantalla en dos con una línea horizontal. Pruebe esto:

```
CLS: PRINT@ (13,0), "----------
----------"  {E}
```

c) Utilice el número «1» para dibujar una línea hacia abajo de la pantalla. Esto es un poco más creativo. Pruebe esto:

```
CLS: PRINT ,,1,,,,1,,,,1,,,,1,,,
,1,,,,1,,,,1,,,,1,,,,1,,,,1   {E}
```

La letra «l» queda mejor, pero deberá encerrarse en un texto. Para dibujar una línea hasta el extremo inferior de la pantalla serían necesarios más de 64 caracteres.



d) Dibuje un hombre esquemáticamente, utilizando letras para sus brazos, piernas, etc., así:

```
 0
-----
1   1
```

Pruebe esto:

```
CLS:PRINT@(6,8),"0":PRINT@(7,6),
"-----":PRINT@(8,7),"1   1"  {E}
```

Ahora le toca a usted

Intente que su ordenador —o el nuestro— haga estas cosas:

(8) Dibuje un pequeño perro —tan parecido a un perro como pueda conseguir, lo que no será mucho— en la parte superior derecha de la pantalla.

(9) Limpie la pantalla y entonces escriba su nombre exactamente en el medio.

(10) Haga una línea diagonal (utilizando el símbolo «/») desde la parte superior derecha a la parte inferior izquierda. Esto necesitará más de 64 caracteres, por lo

tanto, si tan sólo puede utilizar una línea corta sin que el ordenador se queje, dibuje únicamente parte de la diagonal.

Comprobación

Al final de cada capítulo resumiremos lo que esperamos que haya aprendido. Si su ordenador difiere del nuestro, tendrá que haber conseguido parte de esta información del Apéndice 1 o del manual de su ordenador. Asegúrese de que entiende cómo son tratados los puntos siguientes utilizando el BASIC en su ordenador —o el nuestro, si no está utilizando uno— antes de pasar al capítulo siguiente.

Hasta el momento de llegar a este punto, usted debe saber:

Sobre sentencias de BASIC
- Cómo es una sentencia sencilla de BASIC.
- Cuál es la longitud máxima de las sentencias de BASIC en su ordenador.
- Si su ordenador aceptará más de una sentencia de BASIC en la misma línea.
- Cómo utilizar la tecla ENTER, o la tecla equivalente de su ordenador para terminar una sentencia.
- Cómo utilizar los dos puntos para separar sentencias.

Sobre funciones y expresiones
- Cuáles son los símbolos utilizados para la suma, multiplicación, división y resta.
- Muy por encima, qué otras funciones aritméticas puede realizar su ordenador.
- Qué aspecto tiene una sencilla expresión.

Sobre textos
- Qué es un texto y cuál es la longitud máxima que puede alcanzar en su ordenador.
- Qué caracteres pueden incluirse en los textos.
- Cómo utilizar la función «+» con los textos.

Sobre la pantalla de video
- Cuántas filas y columnas hay en la pantalla de su computador.

- Cómo utilizar la orden PRINT:
  - para imprimir números.
  - para imprimir otros caracteres.
  - con espacios en blanco, puntos y comas y comas.
  - para escribir varios fragmentos de texto de forma sucesiva.
- Cómo limpiar la pantalla con la orden CLS o cualquier otra equivalente.
- Cómo controlar el cursor de su computador.
- Cómo posicionar la salida en la pantalla utilizando la orden 'PRINT u otra equivalente.

2 Nuestros primeros programas

En este capítulo aprenderemos:
- **Qué significa escribir un programa.**
- **Las órdenes RUN, LIST, NEW, END, SAVE y LOAD.**
- **La orden REM.**
- **La orden GOTO.**
- **Caracteres de bloques gráficos.**

En el último capítulo jugamos con la orden PRINT, y vimos cómo utilizarla inmediatamente después de conectar el computador. Conseguimos hacer bastantes cosas, pero en realidad no escribimos ningún programa. Por lo tanto, ¿qué significa escribir un programa, y en qué se diferencia de lo que hemos estado haciendo?

Cuando utilizamos el ordenador en «modo inmediato» —qué es el nombre de lo que estuvimos haciendo en el último capítulo— el ordenador empieza a ejecutar nuestras órdenes inmediatamente después de apretar la tecla ENTER. En el «modo de programa» no hace esto. En su lugar nos permite construir una secuencia de sentencias que constituyen un programa, el cual podemos ejecutar posteriormente, o listar, o guardar en un cassette o incluso deshacernos de él.

¿Cómo procede el ordenador para diferenciar cuándo queremos escribir un programa y cuándo queremos que nos responda inmediatamente? Le decimos que queremos estar en el modo de programa dando un número —llamado número de línea— a cada línea de sentencia que pretendemos que forme parte de un programa.

Aclaremos esto escribiendo directamente un pequeño programa.

Pruebe esto:

```
10 PRINT "ESTO"  {E}
20 PRINT "ES"  {E}
30 PRINT "UN"  {E}
```

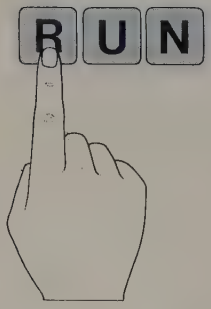
```
40 PRINT "PROGRAMA"  {E}
50 END  {E}
```

En este y en los otros programas que vamos a escribir, nos concentraremos en las sentencias que utilicen la orden PRINT. De esta forma, podemos poner más atención a la esctructura del programa que a su contenido.

Como puede usted ver, este sencillo programa tiene cinco líneas. Los números a la izquierda son los números de línea: hablaremos un poco más sobre ellos muy pronto, y explicaremos por qué utilizamos «10, 20, 30, etc. en lugar de 1, 2, 3». Las líneas de la 10 a la 40 únicamente contienen sencillas sentencias de PRINT, y la línea 50 tiene una orden nueva, END. El END le dice al ordenador —como es de suponer— cuándo ha llegado al final del programa. Usted puede pensar que esto es obvio para el ordenador y de hecho en muchos casos lo es —no hay más líneas de programa para que el ordenador las ejecute—. En algunos ordenadores sólo es necesario utilizar la orden END cuando no es obvio. (Más adelante escribiremos algunos programas donde no lo es.) Sin embargo, nuestro ordenador espera esta orden al final de cada programa.

¿Qué hacemos con nuestro programa? Lo tecleamos para entrarlo en el ordenador, tal como aparece aquí, apretando la tecla ENTER al final de cada línea como dijimos anteriormente. A medida que tecleamos cada línea irá apareciendo en la pantalla, de la misma manera que lo hacía cuando estábamos trabajando en modo inmediato. Pero no sucede nada más. ¿Cómo conseguiremos que el ordenador empiece a ejecutar el programa? Utilizaremos una orden especial.

Pruebe esto:




```
RUN  {E}
```

Entonces usted puede ver cómo el ordenador escribe lo que le dijimos que hiciera. Su pantalla debe aparecer así:

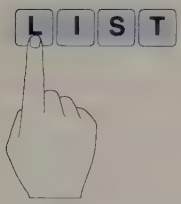
```
10  PRINT "ESTO"
20  PRINT "ES"
30  PRINT "UN"
```

```
40  PRINT "PROGRAMA"
50  END
RUN
ESTO
ES
UN
PROGRAMA
OK
```

Las palabras que le dijimos al ordenador que escribiera, han aparecido una debajo de otra porque nosotros le dijimos al ordenador que lo hiciera así. ¿Qué cree usted que hubiera sucedido si hubiésemos terminado cada línea con un punto y coma? Bien: el cursor no habría bajado a la línea siguiente y por lo tanto el ordenador hubiera escrito todas las palabras en la misma línea. (Naturalmente, hubiéramos tenido que acordarnos de colocar espacios entre los textos.) Lo haremos así en nuestros próximos programas. Pero antes de terminar con éste, hagamos con él algunas cosas más. Primeramente, otra orden:

Pruebe esto:

```
LIST  {E}
```



En la pantalla debe aparecer una lista completa de las sentencias del programa, incluyendo los números de línea. LIST es muy útil. Como ya se habrá dado cuenta, cuando usted entra una nueva línea en la pantalla, el ordenador mueve una línea hacia arriba, lo que se está visualizando en la pantalla, añadiendo la nueva línea al final; a esto se le llama *scrolling* (enrollar). Cuando la pantalla está llena, las líneas de la parte superior desaparecen a medida que se van añadiendo nuevas líneas. Puede utilizarse el LIST para comprobar lo que se ha puesto al principio de un largo programa, si es que ha desaparecido ya de la pantalla. También puede utilizarse el LIST para ver el programa en su totalidad, después de haberlo ejecutado, si es que la ejecución lo ha hecho desaparecer o si incluía una orden de CLS.

El LIST también tiene varias variantes. Por ejemplo:

Pruebe esto:

```
LIST 10  {E}
```

En este caso debe aparecer únicamente la línea 10, por lo que al principio de la pantalla aparecerá esta secuencia:

```
LIST 10   {E}
10   PRINT "ESTO"
OK
```

Pruebe esto:

```
LIST 20–50   {E}
```

Esta vez, en la pantalla aparecerán las líneas de la 20 a la 50 inclusive.

Pruebe esto:

```
LIST –30   {E}
```

Esto debe obligar al ordenador a relacionar las líneas desde el principio del programa hasta la línea 30. Y finalmente,

Pruebe esto:

```
LIST 40–   {E}
```

Esto hace que el ordenador relacione todas las líneas desde la línea 40 hasta el final del programa.

En la mayoría de los ordenadores, la orden LIST hace que el ordenador muestre la relación completa del programa o la parte de la misma que le haya indicado que muestre, de una forma muy rápida. Quizá demasiado rápido para que usted pueda ver las líneas iniciales de un programa largo, por lo tanto si hay más líneas de programa de las que puedan caber en la pantalla, tendrá que decirle que haga el listado de unas cuantas líneas cada vez, utilizando estas variantes.

Después de utilizar la orden LIST, todavía puede ejecutar el programa. También puede limpiar la pantalla, mediante CLS, y entonces ejecutar el programa. Aunque el CLS borra la información de la pantalla del ordenador, no afecta en absoluto la forma en que el programa está guardado en la memoria. ¿Cómo podemos pues deshacernos del programa?

Pruebe esto:

```
NEW   {E}
```

En nuestro ordenador, esto no limpia la pantalla. ¿Qué es lo que hace? Escriba RUN otra vez: el programa ha sido borrado de la memoria, por lo que no sucederá nada.

Generalmente tenemos que usar el NEW siempre que terminemos con un programa y empecemos a trabajar con otro. Si no, el ordenador se confundiría y no sabría las líneas pertenecientes a cada programa. Utilizando el NEW, nos aseguramos de que tenemos un solo programa a la vez en el ordenador. Hasta que no tenga más experiencia, esto es lo que tiene que hacer.

Ahora que ya hemos usado el NEW, podemos seguir adelante con otros programas:

Pruebe esto:

```
1 PRINT "ALGUNOS NUMEROS";  {E}
2 PRINT "DE LINEA DISTINTOS"  {E}
3 PRINT "EN ESTE PROGRAMA."  {E}
4 END  {E}
```

Este será el último ejemplo en el que le recordaremos que tiene que apretar la tecla ENTER al final de cada línea. A partir de ahora, esperamos que se acordará de hacerlo.

Se habrá dado cuenta de que, como ya prometimos, hemos utilizado una puntuación distinta en este programa. ¿Cómo aparecerá escrito en la pantalla cuando apretemos el RUN? De esta forma:

```
ALGUNOS NUMEROS DE LINEA DISTINTOS
EN ESTE PROGRAMA.
```

No obstante, nuestro propósito principal era introducir una discusión sobre los números de línea. La última vez utilizamos los números 10, 20, 30, etc. Esta vez hemos utilizado 1, 2, 3 y 4. Podemos, sin ninguna razón, utilizar cualquier número que nos guste, al ordenador no le importa qué números utilizamos, con tal que sean números enteros, inferiores al número máximo permitido (varios miles, generalmente), y que sigan la secuencia en la cual queremos que se ejecuten las líneas del programa. El uso de múltiplos de diez es convencional, ya que si queremos añadir una línea entre dos ya existentes (ya sea porque se ha olvidado algo, o porque

quiere mejorar o añadir algún detalle al programa), le queda un número libre que puede utilizar, o mejor dicho nueve números libres.

Ordenación de las líneas

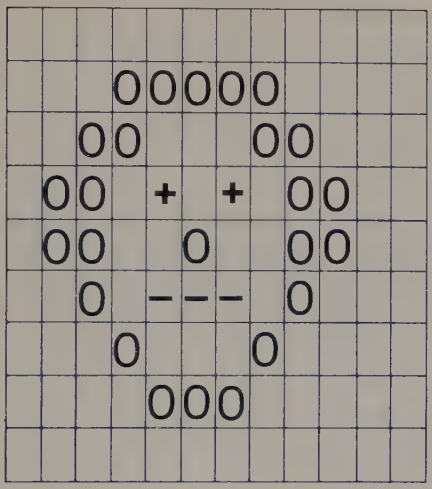
Otro aspecto interesante del uso de los números de líneas es que su ordenador clasificará su programa en el orden correcto: no hace falta que escriba las líneas en la secuencia correcta. Naturalmente, es un buen ejercicio de programación el planear su programa en un orden adecuado antes de teclearlo en el ordenador. Sin embargo, esta habilidad del ordenador es muy práctica, si usted quiere añadir una línea en algún lugar del programa. Tan sólo hay que teclear la nueva línea y el ordenador la colocará en su lugar correcto. Igualmente, si se da usted cuenta de que ha cometido un error después de haber entrado una línea, tan sólo hay que reescribir la línea correcta. El ordenador borrará automáticamente la versión previa y colocará la nueva en el lugar adecuado.

He aquí un programa enrevesado para ilustrar cómo el ordenador clasifica las líneas. Vamos a dibujar un sencillo dibujo en la pantalla, pero esta vez le será difícil ver lo que muestra, antes de ejecutar el programa. Si la imagen no aparece correctamente en la pantalla —o si no tiene un ordenador sobre el que probarlo— en la página siguiente le mostramos un dibujo que representa lo que intentábamos dibujar.

Pruebe esto:

```
50  PRINT "00   0   00"
70  PRINT "  0     0"
10  PRINT "    000"
90  END
40  PRINT "00  + + 00"
80  PRINT "    000"
30  PRINT " 00   00"
20  PRINT "  00000"
60  PRINT " 0  ---  0"
```

Como puede usted ver, incluso tenemos una sentencia de END, en el medio del listado. No importa, el ordenador no hará nada con él hasta después de haber clasificado el programa, y entonces aparecerá donde queríamos que estuviera, al final del programa.



Programa «cara»
Esta es la ejecución del programa, con las líneas entremezcladas, que le pedimos que intentara en la página anterior. ¿Le ha sorprendido el resultado? Si ha listado las líneas antes de ejecutar el programa, habrá podido ver la cara.

Ahora, pruebe de teclear un LIST. Esto le mostrará otro aspecto interesante de la orden LIST. Sin tener en cuenta el orden en que usted escriba las líneas del programa, el LIST las ordenará siempre en el orden numérico correcto.

Ahora le toca a usted

Vamos a darle cuatro programas cortos. Examínelos atentamente, e intente decidir si se ejecutarán correctamente. Si es así, ¿qué es lo que aparecerá en la pantalla? Y si no es así, ¿por qué no? (Las respuestas a estas preguntas están en la página 200.)

(1)
```
10 PRINT: "HOLA"
20 END
```

(2)
```
10 PRINT "NEGRA MARIA"
5  END
```

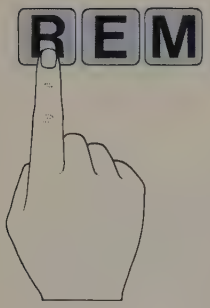
(3)
```
30  END
25  PRINT " 1 1 1"
10  PRINT "000 000 000"
20  PRINT "000 000 000"
29  PRINT " 1/ 1/ 1/"
```

```
(4) 1000  PRINT "ADIOS"
    800   PRINT
    700   PRINT
    200   PRINT
    300   PRINT "HOLA"
    1010  END
```

La orden REM

Ya es tiempo de introducir otra orden. Se trata de la orden REM, y que es una de las muchas órdenes que utilizamos únicamente en los programas, y no en el modo inmediato. Veamos qué es lo que hace:

Pruebe esto:




```
10  REM PROGRAMA PROGRAMA
20  REM ESTE PROGRAMA
30  REM NO PARECE
40  REM HACER DEMASIADO
50  PRINT "HOLA"
60  END
```

¿Qué ha sucedido? En su pantalla debe aparecer esto, cuando ejecute el programa:

```
HOLA
```

Ya sabe de dónde viene esto: de la línea 50. ¿Qué ha sucedido con el texto que escribió antes de la línea 50? Nada. Esto es lo correcto, nada. La orden REM se utiliza para decirle al ordenador que ignore totalmente todo lo que viene a continuación en la misma línea. (Como resultado, tan sólo debe introducir la última, o única, sentencia de la línea. Puede utilizar otra orden a continuación en la misma línea de programa.)

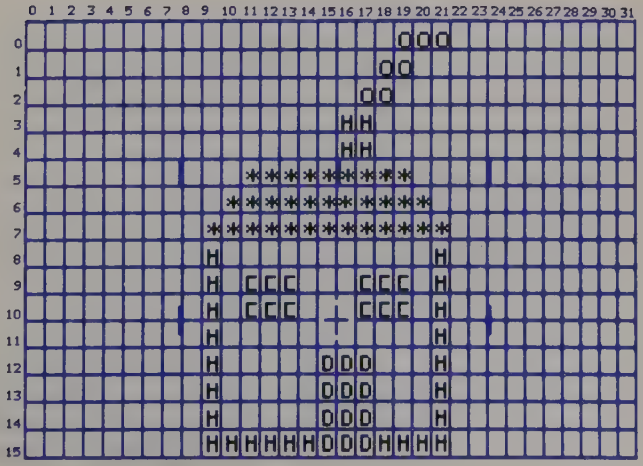
Por ahora, quizá debe parecerle que esto no es particularmente interesante. ¿Para qué teclear palabras si el ordenador no está interesado en ellas? La respuesta es que el que está interesado es usted. De hecho, los REMarks (observaciones) se utilizan para permitirle introducir observaciones y comentarios en el listado del programa, para usted o para otra gente que pueda utilizar su programa. Puede incluir, después de un REM por ejemplo:

– El nombre del programa. Hablaremos acerca de po-

nerle nombres a los programas, de una forma breve, cuando discutamos cómo guardarlos.
- Su propio nombre.
- Notas acerca de lo que hace cada línea del programa. Esto es de la mayor importancia como guía para usted o para otros usuarios, cuando se está escribiendo un largo y complicado programa.

Quizá para ver la utilidad de lo que acabamos de aprender, deberíamos intentar escribir un largo programa, si bien no particularmente complejo. Vamos a hacer dibujos con la sentencia PRINT@, para recordarle a usted cómo se utiliza. Vamos a intentar dibujar una pequeña casa, utilizando de nuevo letras para realizar las líneas de nuestro dibujo. Utilizaremos sentencias REM para describir las partes del programa que dibujen las distintas partes de la casa —las paredes, el techo, la puerta, etc.—. Y para hacer las distintas partes del programa más diferentes una de la otra, empezaremos cada sección con un número de línea fácil de recordar.

Hágase con un papel de gráficos o un cuaderno adecuado especial para utilizarlo con su ordenador, o si no lo tiene, dibuje un cuadriculado en una hoja de papel en blanco. Entonces síganos a medida que planeamos



Programa «casa»
He aquí el planteo de la casa que vamos a hacer que el ordenador dibuje en el próximo programa. Está dibujado sobre un cuadriculado que divide el área de pantalla de nuestro ordenador.

nuestra casa. Vamos a ceñirnos a la técnica de programación correcta y a planear la totalidad de nuestro dibujo antes de empezar a «codificar» el programa (esto es, convertir nuestras ideas en sentencias de BASIC).

La ilustración anterior muestra la casa que deseamos dibujar. No pretendemos ser los mejores artistas del mundo, por lo tanto no nos opondremos si desea mejorarlo. Hemos marcado también las posiciones de fila y columna correspondientes a la pantalla de nuestro ordenador para mostrar dónde planeamos colocarlo. De nuevo, usted puede adaptarlo para que coincida con el tamaño de su pantalla y con sus preferencias.

Sigamos un orden determinado, de forma que usted pueda continuar el desarrollo de la casa en la pantalla, paso a paso. Dibujaremos primero las paredes, por lo tanto teclee estas líneas ahora. No hemos terminado de escribir el programa todavía, pero puede ejecutar estas líneas si así lo desea. Quizás obtenga un mensaje de error que le diga que se ha olvidado de la sentencia END, pero podrá ver lo que aparece, y darse cuenta inmediatamente si ha cometido algún error en el tecleado.

```
100  REM DIBUJO DE LAS PAREDES
110  PRINT@ (8,9), "H"
120  PRINT@ (8,21), "H"
130  PRINT@ (9,9), "H"
140  PRINT@ (9,21), "H"
150  PRINT@ (10,9), "H"
160  PRINT@ (10,21), "H"
170  PRINT@ (11,9), "H"
180  PRINT@ (11,21), "H"
190  PRINT@ (12,9), "H"
200  PRINT@ (12,21), "H"
210  PRINT@ (13,9), "H"
220  PRINT@ (13,21), "H"
230  PRINT@ (14,9), "H"
240  PRINT@ (14,21), "H"
250  PRINT@ (15,9), "HHHHHH        HHHH";
```

La última línea tenía más de 32 caracteres de largo, como ya se habrá dado cuenta, si tiene una pantalla de 32 caracteres. Ya que las líneas en las páginas contienen más de 32 caracteres, en adelante escribiremos

nuestras sentencias de programa de acuerdo con ellas y no reflejarán la apariencia exacta de la pantalla de 32 caracteres. No se preocupe: no nos gusta utilizar sentencias largas que usted no pueda colocar en su ordenador, aunque cada sentencia ocupe más de una línea.

Si son demasiado largas para su ordenador, sencillamente descompóngalas en dos y utilice otro número de línea para la segunda mitad. Naturalmente, deberá asegurarse de que su segunda línea empieza a escribirse en el sitio correcto.

También deberá poner atención a la puntuación cuando haga dibujos de esta manera. Es importante señalar que sí necesita el punto y coma al final de la línea 250. De otra forma, el cursor, al volver a la línea siguiente, estropeará sus planes al correr su dibujo una línea hacia arriba.

Al entrar estas sentencias es un largo y lento trabajo, ya que todavía no conocemos muchas órdenes para acelerar este proceso. Sin embargo, el resto tampoco estará mal, ya que iremos escribiendo más caracteres con cada sentencia. Ahora vamos a por el techo:

```
300  REM DIBUJO DEL TECHO
310  PRINT@ (5,11), "##########"
320  PRINT@ (6,10), "############"
330  PRINT@ (7,9), "##############"
```

Ahora las ventanas:

```
400  REM DIBUJO DE LAS VENTANAS
410  PRINT@ (9,11), "CCC     CCC";
420  PRINT@ (10,11), "CCC     CCC";
```

De nuevo, los puntos y comas al final de cada línea son importantes, porque sin ellos el computador podría borrar la pared de la derecha cuando llevase el cursor hacia abajo, hasta la siguiente línea. Ahora la puerta:

```
500  REM DIBUJO DE LA PUERTA
510  PRINT@ (12,15), "DDD";
520  PRINT@ (13,15), "DDD";
530  PRINT@ (14,15), "DDD";
540  PRINT@ (15,15), "DDD";
```

Y finalmente, la chimenea y el humo:

```
600  REM DIBUJO DE LA CHIMENEA Y DEL HUMO
610  PRINT@ (4,16), "HH"
620  PRINT@ (3,16), "HH"
630  PRINT@ (2,17), "00"
640  PRINT@ (1,18), "00"
650  PRINT@ (0,19), "000"
```

Este es nuestro dibujo, pero todavía no hemos terminado nuestro programa. Añada estas líneas:

```
 10  REM ***CASA***
 20  REM POR SUSAN CURRAN
 30  CLS
700  END
```

(Si quiere puede poner su propio nombre en lugar del mío.) Los asteriscos en la línea 10 están sólo para destacar el título.

Si está usted utilizando un ordenador, haga un LIST del programa completo. ¿Ve usted ahora que las líneas de comentarios hacen más fácil el ver lo que sucede en cada punto? Utilizándolos, será más fácil cambiar una parte del dibujo que a usted no le guste lo suficiente, por ejemplo, utilizando la letra E o T en lugar de la C para las ventanas, o hacerlo más grande o más pequeño. Como dijimos anteriormente, la manera de realizar cambios es simplemente entrando las nuevas líneas. Automáticamente reemplazarán cualquier línea anterior que tenga el mismo número de línea. Si lo que quiere es, simplemente, deshacerse de una línea y no reemplazarla, entonces tan sólo hay que teclear el número de línea y un ENTER, así:

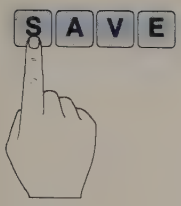
```
20  {E}
```

La línea en blanco 20 reemplazará a la anterior, que quedará borrada. Finalmente, intente ejecutar el programa.

Cómo guardar su programa

Este programa es un poco tonto comparado con los que va a escribir a continuación, pero para animarle un poco, y ayudarle a creer que ha hecho algo que vale la

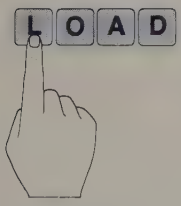
pena, ¿por qué no grabarlo de forma que pueda ejecutarlo otra vez? Si no lo graba en una cinta de cassette o en un disco flexible, entonces habrá perdido todo su trabajo tan pronto como desconecte su ordenador.



Nuestro sistema ordenador incluye un grabador de cassettes que utilizamos para grabar programas. Utilizamos la orden SAVE para decirle al ordenador que grabe, por lo tanto escribimos:

SAVE "CASA"

Ahora el ordenador sabe que nuestro programa se llama «CASA». Cuando escribimos SAVE y después una palabra entre comillas, el ordenador utiliza esta palabra —cualquiera que sea— para nombrar el programa que acabamos de escribir, y almacenarlo con este título. El ordenador no está interesado en absoluto en el nombre que hemos utilizado en los comentarios —de hecho, podríamos darle al computador un nombre bastante distinto del que le pusimos en las sentencias de REM, al principio del programa—. Esto no es una idea tan tonta como puede parecer, ya que (en nuestro ordenador) debemos limitar la longitud de los nombres que le damos al mismo a ocho caracteres, mientras que en las sentencias REM podemos hacerlo tan largo como queramos. Su manual debe decirle qué formato requiere su ordenador para los nombres, cuando trate de guardar un programa.



Sigamos ahora el proceso, según nuestro manual, para guardar programas. Esto varía bastante de una máquina a otra, por lo que dejaremos para usted el comprobarlo —y guardar este programa si así lo desea—. Una vez lo haya guardado, puede cargarlo (LOAD) de nuevo, siguiendo una vez más las instrucciones de su manual. Una vez que el programa está cargado, teclee RUN, y el ordenador volverá a ejecutarlo. Naturalmente, a menos que haya desconectado su ordenador, no es necesario cargar el programa otra vez para ejecutarlo después de guardarlo.

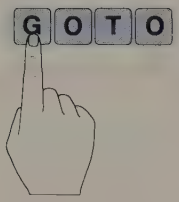
Ahora le toca a usted

Plantee algunos programas más de dibujo, utilizando muchas sentencias REM para recordarle cómo están construidas. Déle nombres y, si puede, pruébelos

sobre un ordenador. No le damos ninguna «solución» esta vez (iremos descubriendo formas más fáciles de realizar dibujos dentro de poco, y hay muchos ejemplos de programas de dibujo más interesantes en el capítulo 9), pero he aquí algunas sugerencias sobre cosas que puede dibujar en la pantalla:

5) Un coche.
6) Una nave espacial.
7) Un jardín lleno de flores.
8) Un elefante.

La orden GOTO



Como ya habrá descubierto, si ejecuta el programa «CASA», el final de la ejecución puede estropear el efecto. El computador insiste en escribir «OK» —o cualquiera que sea el mensaje que indica que ha terminado— y a menos que vaya con cuidado lo escribirá en el medio de su dibujo.

¿Cómo podemos evitar que el ordenador haga esto? Hay que asegurarse de que no llegue nunca al final del programa. La manera más sencilla de realizarlo es hacer que vaya dando vueltas una y otra vez haciendo algo que no afecte al resto del programa. De hecho podemos hacer que parezca que no haga nada en absoluto. El hecho de darle un número de línea fuera de la memoria y que vaya a mirar qué sentencia le sigue, toma tiempo al ordenador, por lo tanto el hacer que vaya repetidamente a mirar la misma línea servirá para hacer lo que queremos.

Para hacer esto, necesitamos una nueva orden, GOTO. Como ya hemos visto, normalmente el ordenador lleva a cabo una instrucción en una línea de programa e inmediatamente avanza hasta la línea siguiente de acuerdo con el orden numérico de las líneas. El GOTO cambia esto —le dice al ordenador que vaya a un número de línea en particular, que le damos, que puede ser cualquiera de los que hemos utilizado en nuestro programa—. Así es como lo utilizamos:

GOTO 10

le dice al ordenador que seguidamente ejecute las instrucciones que hay en la línea 10, y

GOTO 500

le dice al ordenador que ejecute las instrucciones de la línea 500. No debemos decirle al ordenador que vaya a un número de línea que no hemos puesto en el programa, ya que esto lo confundiría.

He aquí un programa corto que muestra cómo funciona el GOTO:

Pruebe esto:

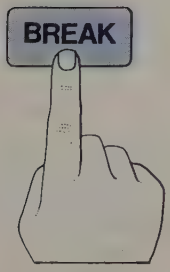
```
10  REM PROGRAMA GOTO
20  PRINT "ESTE": GOTO 70
30  PRINT "MUY": GOTO 60
40  PRINT "PROGRAMA": GOTO 30
50  PRINT "UN": GOTO 40
60  PRINT "TONTO": END
70  PRINT "ES": GOTO 50
```

El ordenador sigue nuestras instrucciones como un esclavo y escribe este mensaje en la pantalla:

```
ESTE
ES
UN
PROGRAMA
MUY
TONTO
```

Para conseguir que el ordenador no haga nada, escribimos un «bucle vacío» —una instrucción que le haga dar vueltas sobre sí mismo— que le dice que vaya a la misma línea que ya está en aquel momento. Por lo tanto en lugar de acabar nuestro programa «CASA» con un END en la línea 700, podemos reescribir esta línea como:

```
700  GOTO 700
```



Si su programa CASA está todavía cargado, pruebe esto ahora. Asegúrese de que numera la línea del GOTO con el mismo número que ha utilizado para la línea END, si ésta no era 700. Ejecute el programa de nuevo: ¿ha visto que el OK no aparece?

Naturalmente el programa no se ejecutará indefinidamente, y necesitamos encontrar una manera de pararlo para poder decirle al ordenador que queremos

hacer algo distinto. En nuestro ordenador apretamos la tecla BREAK para hacerlo; en otros esta tecla puede llamarse STOP.

Cómo hacer bucles con el GOTO

El GOTO es útil siempre que queramos hacer que el ordenador haga la misma cosa repetidamente, así como en otras circunstancias donde queremos cambiar el orden en que el ordenador ejecuta las instrucciones de las líneas de programa. Podemos utilizarlo para hacer un «bucle», que le dice al ordenador que vuelva atrás y ejecute una instrucción o una serie de instrucciones una y otra vez. He aquí algunos ejemplos:

Pruebe esto:

```
10 PRINT "*";
20 GOTO 10
```

La pequeña flecha en la parte izquierda del programa es tan sólo un dibujo de la estructura del bucle. Es para ayudarle: no hace falta que lo escriba de ninguna manera en el ordenador. Este escribirá línea tras línea de asteriscos a lo largo de la pantalla —suponiendo que ha advertido y tecleado el punto y coma. (Si no, habrá escrito una larga fila a lo largo de la columna de la izquierda.)—. El ordenador no tiene manera de saber cuándo debe pararse, por lo tanto una vez más tendrá que apretar la tecla BREAK cuando ya haya visto suficiente.

Pruebe esto:

```
10 CLS
20 PRINT@ (8,10), "EMERGENCIA"
30 GOTO 10
```

El ordenador limpia la pantalla y reescribe el mensaje cada vez que vuelve a iniciar el bucle, por lo que en la pantalla aparecerá el mensaje parpadeando.

Pruebe esto:

```
10 PRINT "/";
20 GOTO 10
```

Un bonito dibujo de líneas diagonales ¡debe ser útil para algo!

Todos estos programas son ejemplos de lo que llamamos «bucles sin fin». En los capítulos siguientes veremos diversas maneras de contar cuántas veces el ordenador debe ejecutar un bucle, y cómo decirle que termine el bucle cuando queremos que cese de ejecutarlo.

Caracteres gráficos

Hemos hecho muchos pequeños dibujos en la pantalla porque creemos que es una manera sencilla y divertida de acostumbrarse a la forma de trabajar de la pantalla del ordenador y a las órdenes que la controlan. Sin embargo, hasta ahora hemos utilizado caracteres alfanuméricos —letras y números— como nuestras herramientas de dibujo. La mayoría de los ordenadores ofrecen al menos una forma alterantiva de realizar dibujos, y ahora veremos una sencilla herramienta que casi todos los ordenadores domésticos tienen: los caracteres gráficos especiales.

Los caracteres gráficos de nuestro ordenador

Esta es la lista de todos los caracteres gráficos especiales que nuestro ordenador puede visualizar; debajo de cada uno de ellos está el código CHR$ que los ha producido. Cada símbolo gráfico individual ocupa el espacio de un cuadrado del mapa de la pantalla de nuestro ordenador; exactamente el mismo espacio que una letra o un número.

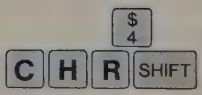
En nuestro ordenador, los caracteres gráficos no están marcados en el teclado, aunque en el suyo quizá lo estén. Sin embargo, hay una lista de ellos en nuestro manual. Hay un conjunto de 16, y se muestran en el dibujo anterior.

Para poderlos utilizar, debemos aprender un código especial que le dice al ordenador que los imprima en la pantalla. De hecho, cada carácter que puede visualizar, incluyendo letras y números, tiene un código numérico, así como las teclas de control como ENTER, BREAK y las teclas de cursor. Siempre que apretemos una tecla, el código de este carácter es enviado al ordenador y entonces éste deduce qué es lo que queremos ver en la pantalla. Nuestro ordenador, como casi todos, utiliza un conjunto de códigos basados en el Código Estándar Americano para el Intercambio de Información, o ASCII. (Los códigos ASCII estándar no incluyen los símbolos gráficos: para éstos cada fabricante de ordenador adjudica códigos numéricos individuales no utilizados en el ASCII.) Debajo de cada uno de los símbolos gráficos en la página 55, hay un número: éste es el código que utilizamos para describir el carácter al ordenador.

Cuando tecleamos un código numérico, debemos indicarle al ordenador que se trata de un código numérico, y no de un número ordinario. Si tan sólo tecleamos una sentencia tal como:

PRINT 137

entonces escribirá 137, y no dos pequeños cuadrados en diagonal. Utilizamos una función especial, CHR$, para decírselo al ordenador. ¿Se acuerda de las funciones que vimos en el capítulo anterior? CHR$ le dice al ordenador que haga algo con el número que le sigue, de la misma forma que SQR o TAN lo hacen —le dice al ordenador que trate el número que le sigue como un código ASCII—. Esta es la clase de sentencia que escribimos:



Pruebe esto:

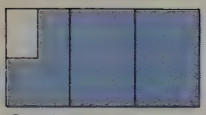
PRINT CHR$(137)

Los cuadrados en diagonal aparecen en la posición de escritura siguiente.

Podemos juntar funciones CHR$, de la misma forma que juntamos textos en el capítulo anterior, para hacer sentencias tal como:

Pruebe esto:

```
PRINT CHR$(136)+CHR$(128)+CHR$(128)
```



Camión

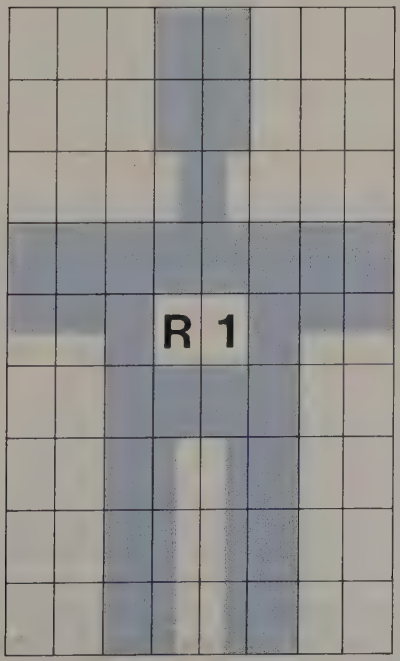
Esto produce algo parecido a un camión, como se muestra a la izquierda.

Ahora que ya conocemos algunos códigos de CHR$, podemos utilizarlos para realizar dibujos. Vamos a hacerlo ahora, y a realizar otro dibujo, utilizando los símbolos gráficos de nuestro ordenador. Ya que éste será el último ejemplo en este capítulo, vamos a llevar a cabo sin interrupción el proceso de realizar un planteo, escribir el programa, teclearlo y guardarlo.

Cómo dibujar un robot

Dibujemos una sencilla forma de robot, que haremos más interesante poniéndole un mensaje que parpadea

Forma de robot
He aquí el diseño de nuestro robot. Hemos dibujado una forma colocando caracteres gráficos en una cuadrícula que representa los cuadrados del mapa de pantalla de nuestro ordenador. El pecho del robot tendrá un mensaje parpadeante, «R1».



en el centro de su cuerpo. Primeramente cogemos nuestro cuaderno de gráficos y diseñamos la forma. El dibujo de la página anterior muestra nuestra versión, que utiliza una sencilla selección de formas tomadas de nuestro conjunto de caracteres gráficos.

Seguidamente, nos ponemos a planear cómo programarlo. Tomamos otra hoja de papel gráfico y tomamos nota, cuadro a cuadro, de los códigos CHR$ para cada carácter que vamos a utilizar. En este segundo esquema, marcaremos también las posiciones de fila y columna en las cuales hemos escogido poner el dibujo.

			128	128			
			128	128			
			138	133			
128	128	128	128	128	128	128	128
131	131	128	128/ R	128/ I	128	131	131
		128	128	128	128		
		128	133	138	128		
		128	133	138	128		
		128	133	138	128		

Plan de robot
En este segundo dibujo de la forma del robot, sustituimos los caracteres gráficos del cuadriculado por los números correspondientes a los códigos CHR$. El pecho parpadeante del robot tendrá dos caracteres gráficos alternativamente y dos códigos alfanuméricos.

Entonces describimos el programa, sobre papel. Usted puede hacerlo en la pantalla si así lo prefiere, pero a menos que su ordenador tenga unas características de listado y edición muy buenas —características para volver a ver y corregir líneas de programa que ha entrado en el ordenador—, lo cual no tiene el nuestro,

el papel es una opción mejor. Si estuviéramos escribiendo un programa muy largo, lo probaríamos sección a sección en el ordenador, a medida que fuéramos completando cada una. Ya que éste es relativamente corto, lo terminaremos primero, y entonces lo ejecutaremos.

Así, primero la preparación y explicación necesaria:

```
10  REM ***ROBOT PARPADEANTE***
20  REM POR SUSAN CURRAN
30  CLS
```

Y entonces el programa en sí:

```
100  REM DIBUJO DE LA FORMA DEL ROBOT
110  PRINT@ (5,15), CHR$(128)+CHR$(128)
120  PRINT@ (6,15), CHR$(128)+CHR$(128)
130  PRINT@ (7,15), CHR$(138)+CHR$(133)
140  PRINT@ (8,12), CHR$(128)+CHR$(128)+CHR$(128)
     +CHR$(128)
145  PRINT@ (8,16), CHR$(128)+CHR$(128)+CHR$(128)
     +CHR$(128)
150  PRINT@ (9,12), CHR$(131)+CHR$(131)+CHR$(128)
     +CHR$(128)
155  PRINT@ (9,16), CHR$(128)+CHR$(128)+CHR$(131)
     +CHR$(131)
160  PRINT@ (10,14), CHR$(128)+CHR$(128)+CHR$(128)
     +CHR$(128)
170  PRINT@ (11,14), CHR$(128)+CHR$(133)+CHR$(138)
     +CHR$(128)
180  PRINT@ (12,14), CHR$(128)+CHR$(133)+CHR$(138)
     +CHR$(128)
190  PRINT@ (13,14), CHR$(128)+CHR$(133)+CHR$(138)
     +CHR$(128)
```

Y ahora el fragmento que corresponde al parpadeo:

```
200  REM DIBUJO DEL PECHO QUE PARPADEA
210  PRINT@ (9,15), "R1";
220  PRINT@ (9,15), CHR$(128)+CHR$(128);
230  GOTO 210
```

¿Ha visto cómo hemos hecho lo que hicimos en el ejemplo anterior en que parpadeaba «EMERGEN-

CIA»? Sin embargo, esta vez, en lugar de limpiar la pantalla para hacer que parpadee el mensaje, hemos escrito alternativamente el mensaje (R1 de «robot 1») y los caracteres cuadrados que lo «tapan». Naturalmente, no necesitamos terminar el programa ni tampoco necesitamos un bucle vacío como en el programa de la casa, ya que el bucle que hace parpadear el pecho del robot asegura que nuestro programa no se detendrá.

Ahora conectamos el computador o tecleamos NEW si ya está conectado, y entramos el programa en el ordenador. Hacemos una lista, tomando unas 10 líneas cada vez, de forma que podamos comprobar que todo lo que hemos tecleado son los códigos correctos. Seguidamente lo ejecutamos y, si no coincide con lo que nosotros esperábamos, entonces lo modificamos y lo ejecutamos otra vez, y así sucesivamente si fuera necesario. Finalmente, cuando estemos satisfechos, le decimos al ordenador que, por ejemplo,

SAVE "ROBOT"

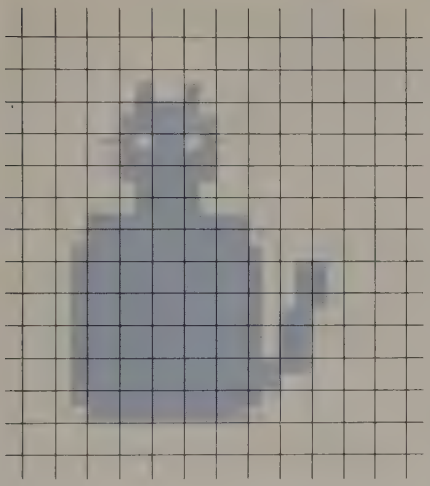
y realizamos el proceso, correspondiente a nuestro computador, para guardarlo en cinta.

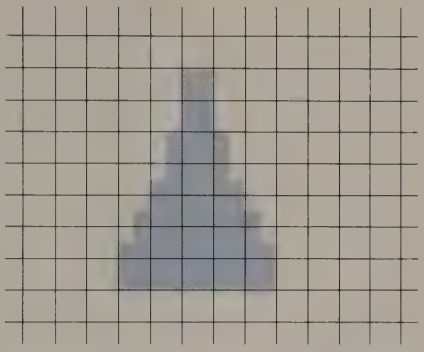
Ahora le toca a usted

Una vez que haya ejecutado con éxito el programa del «robot», intente escribir un par de programas similares por sí mismo. Haga este pequeño esfuerzo y coloque sentencias REM para explicar lo que está haciendo. Entonces guárdelo en una cinta, para añadirlo a su repertorio de efectos gráficos sencillos. No le daremos los programas. Lo que sí le daremos es un par de dibujos, ya realizados sobre papel gráfico. Deduzca por sí mismo qué códigos de caracteres gráficos necesitará, y cuál es la mejor manera de posicionarlos en la pantalla.

Comprobación

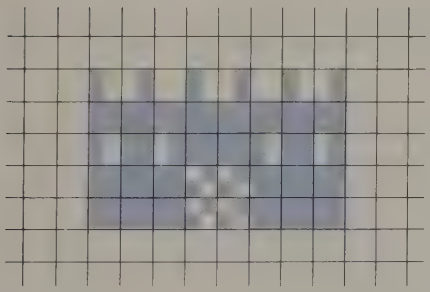
Esto es lo que tiene que haber aprendido en este capítulo. Si no está usted seguro sobre alguno de estos puntos, entonces vuelva atrás y revíselos y practíquelos —en su ordenador o escribiendo programas de muestra si no tiene ninguno— antes de seguir.

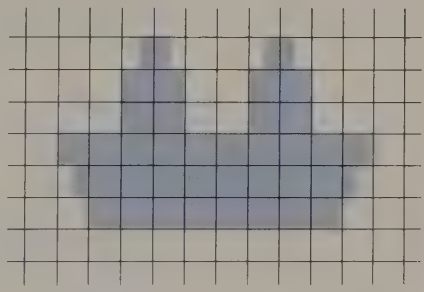




Formas que tiene que dibujar en la pantalla de su ordenador

Haga que el ordenador dibuje la forma de una torre, o la forma de gato que le mostramos aquí. Quizá tenga que improvisar un poco para dibujar la forma de un castillo y de un barco de guerra para que usted lo intente en su ordenador.





Actualmente debe saber:

Sobre estructura de un programa

- En qué se diferencian los programas de las sentencias utilizadas en el modo inmediato.
- Cómo se utilizan los números de línea.
- Qué números de línea pueden utilizarse.
- En qué orden pueden entrarse las líneas.
- Acerca de las siguientes sentencias utilizadas cuando se trabaja con programas:
  NEW
  LIST
  LIST (número de línea)

LIST (número de línea) - (número de línea)
LIST (número de línea)
LIST -(número de línea)
RUN
END
SAVE
LOAD

Sobre las sentencias REM
- Cómo se utilizan las sentencias REM, y qué puede colocarse después de ellas.
- En dónde hay que colocarla en una línea que contiene varias sentencias.
- Cómo utilizar el REM para aclarar la estructura de un programa.
- Cómo utilizar el REM para darle un nombre a un programa.

Sobre nombres de programas
- Qué formato requiere su ordenador para los nombres cuando se guarda un programa (SAVE).

Sobre la sentencia GOTO
- Cómo puede utilizarse el GOTO para crear un único bucle.
- Acerca del uso de bucles vacíos mediante el GOTO.

Sobre el BREAK, STOP o teclas similares
- Cómo se utilizan para terminar programas que están encallados en bucle sin fin.

Sobre caracteres gráficos
- Qué caracteres gráficos están disponibles en su/nuestro ordenador.
- Los códigos CHR$ para ellos.
- Cómo utilizar los códigos CHR$ en sentencias de PRINT.

3 Introducción de variables

En este capítulo aprenderemos:
- **Variables numéricas y de texto.**
- **La orden LET...=**
- **La orden INPUT.**
- **Las funciones RND e INT.**

El concepto de variable radica en los distintos métodos que el BASIC utiliza para permitirnos conseguir mucho con unas pocas líneas de programa. Es un concepto que no es fácil comprender, por lo que lo analizaremos con cierta profundidad en este capítulo.

¿Qué es una variable? ¿Se acuerda cuando, en el capítulo 1, le dijimos que probara:

PRINT A

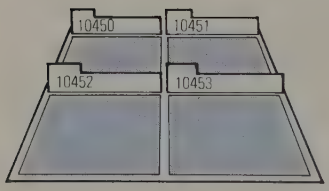
y el ordenador escribió un 0? «A», utilizado de esta manera, es una variable. Es el nombre de una posición de la memoria del ordenador. Ya que todavía no hemos puesto ninguna información en esta posición, el ordenador nos dice que su contenido era «0».

Veamos esto más despacio. Suponemos que ya sabe que el ordenador tiene mucho espacio de memoria —parte del cual está ocupado por los programas internos—. Sin embargo, gran parte de él está disponible para usted, como programador de BASIC, para utilizarlo cuando escriba programas. (A este tipo de memoria se le llama de lectura/escritura o memoria de acceso aleatorio —RAM de forma abreviada— que significa que la información que contiene puede ser ordenada, cambiada de lugar y borrada siempre que usted o el ordenador lo decida.) El ordenador utiliza esta memoria para guardar toda la información que usted le da cuando escribe su programa —instrucciones, números, series de datos, números de línea, etc.— y toda la información que él mismo produce cuando hace cálcu-

los o manipulaciones para usted. Cuando usted programa en BASIC, el ordenador decide por usted qué posiciones de memoria utilizará para guardar toda la información que necesita guardar.

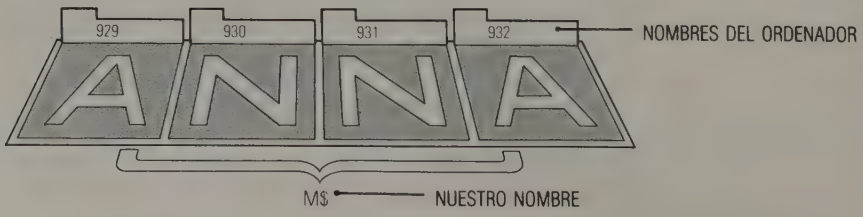
Cada posición de memoria tiene un número de dirección. Si su ordenador, como el nuestro, tiene una «RAM de 16K» (es decir, 16384 posiciones de memoria individuales en su memoria de lectura/escritura), entonces los números irán desde 0 hasta 16383. Ya que no necesitamos preocuparnos acerca de qué posiciones ha utilizado el ordenador para almacenar cada fragmento de información, de hecho no sabemos la dirección(es) de ningún fragmento. Sin embargo, a menudo será de ayuda el describir la información que está en el ordenador mediante su dirección. Veremos muy pronto cómo puede ayudarnos esto.

Para ayudarnos a hacerlo, el ordenador nos permite escoger un nombre que utilizaremos cuando hablemos de posición de memoria, o de un conjunto de posiciones de memoria. (Cada posición de memoria puede almacenar únicamente una letra o su equivalente, por lo tanto se necesita un conjunto de posiciones para



Posiciones de memoria

El ordenador almacena la información en posiciones numeradas y entonces nos permite escoger cualquier nombre que queramos ponerle a estas posiciones. Nosotros utilizaremos los nombres que hemos escogido cuando queramos poner alguna información en estas posiciones o queramos recuperar alguna información que esté ya almacenada en ellas.



guardar un texto, por ejemplo.) «A» es un ejemplo del tipo de nombre que podemos escoger. Algunos ordenadores (pero no el nuestro) le permiten escoger nombres mucho más largos. Hablaremos pronto en detalle acerca de los nombres que nuestro ordenador acepta.

Podemos elegir un nombre para una posición de memoria, en nuestro ordenador, simplemente mencionándolo en una línea de programa. Por lo tanto cuando entramos una línea tal como:

PRINT C

el ordenador responde colocando aparte una posición de memoria, o una serie de posiciones (dependiendo del tipo de nombre que le demos), y a la que podremos referirnos con el nombre «C». Siempre que mencionemos C, supondrá que estamos realmente interesados no en el nombre C, sino en la información que está en la posición que llamamos C. Por lo tanto, cuando le decimos que PRINT C, escribe no la letra C, sino la información que está en C.

La gran ventaja de esto es que, cuando escribimos un programa, no necesitamos saber exactamente lo que habrá en C cuando el ordenador escribe su contenido o lo utiliza de cualquier otra manera. ¿Por qué puede que no lo sepamos? Bien, «C» puede contener una cantidad que le hemos preguntado a usted, el usuario, para que la entrase cuando el programa se está ejecutando, en respuesta a una pregunta del ordenador tal como «¿A cuánto ascendía el último cheque que usted pagó de su cuenta bancaria?» (Veremos muy pronto cómo se hace esto.) O «C» puede contener la respuesta a un cálculo que el ordenador ha hecho. Además de, o en vez de, decirnos la respuesta directamente, puede ser de más utilidad el hacer que el ordenador la guarde para un uso posterior. O «C» puede contener muchos valores diferentes, uno después del otro, que queremos utilizar para un mismo propósito, por ejemplo para describir la posición donde queremos escribir en la pantalla, o para muliplicar por una misma cantidad, o sencillamente para sacarlos en una lista por la pantalla.

Si la diferencia entre el nombre de la variable C y la información que está en la posición llamada C no está

todavía clara, quizá lo esté cuando prosigamos y pongamos realmente algo en C. Hagamos esto ahora, utilizando una nueva orden, «LET...=»:

Pruebe esto:

```
10  REM PROGRAMA DE VARIABLES
20  LET C = 15
30  PRINT C
40  END
```

Un sencillo y bonito programa para ilustrar la forma de utilizar LET. ¿Qué es lo que sacará el ordenador? Lo ha adivinado:

```
15
```

LET es la orden que utilizamos para decirle al ordenador que coloque alguna información en la posición correspondiente a la variable (es decir, una posición en RAM). Hay otras maneras de poner información en las variables, pero por ahora nos ceñiremos únicamente a esta forma. Y utilizamos la función «=» después del LET, para decirle al ordenador qué información queremos colocar en la variable que acabamos de nombrar. Nótese que el «=» utilizado de esta manera no es el mismo «=» aritmético con el que usted está familiarizado. No se puede permutar el orden de la sentencia y decir:

```
LET 15 = C
```

Hay que dar primero el nombre de la posición, y después su contenido. Quizá pueda interpretarse el «=» con el significado «contener» en lugar de «igual», por lo tanto «LET C=15» significa «dejar que la posición C contenga el número 15».

Se puede cambiar el contenido de C, una vez le hemos dado un valor, sencillamente diciéndole al computador que LET C= a cualquier otra cosa. He aquí un ejemplo:

Pruebe esto:

```
10  REM CAMBIO DEL CONTENIDO DE LAS VARIABLES
20  LET C = 15
```

```
30  PRINT C
32  LET C = 25
33  PRINT C
40  END
```

Si no ha borrado todavía el programa anterior, no hace falta que entre todo éste. Tan sólo hay que añadir las líneas 32 y 33.

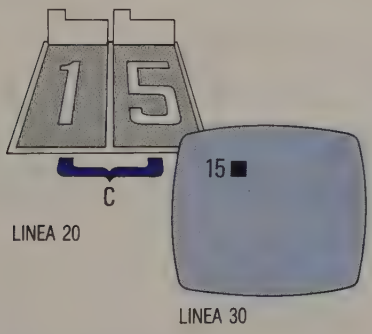
¿Qué es lo que sacará el computador? Esto:

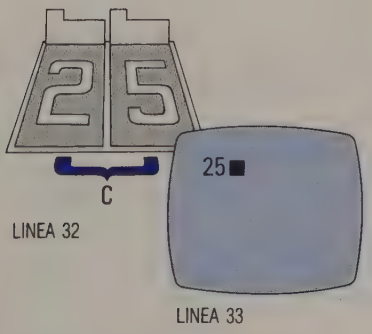
```
15
25
```

Nótese que lo que es variable en C no es su nombre sino su contenido. C continúa siendo el nombre de la posición: su contenido cambia de 15 a 25, o a cualquier otro valor que le demos a continuación.

Ya que el ordenador sabe que estamos únicamente interesados en el número que contiene C, podemos cambiar este contenido de muchas maneras diferentes. En el programa anterior hicimos que el ordenador escribiera 25 en lugar de 15, simplemente diciéndole que cambiase el contenido de C (el número 15) por el número 25. Podríamos haber conseguido el mismo re-

Utilización del LET





La orden LET no sólo nos permite poner datos en una posición de memoria del ordenador, sino que también nos permite cambiar el contenido de esta posición. En el programa de esta página, la línea 20 coloca en la posición C, el número 15 y la línea 30 escribe el contenido de C; la línea 32 cambia estos contenidos y la línea 33 escribe un 25, que es el nuevo contenido.

sultado mediante una pequeña suma aritmética en C, sustituyendo las líneas 32 y 33 por esto:

```
32  LET C = 12 + 13
33  PRINT C
```

Pruébelo. El ordenador realiza la aritmética y escribe 25.

Incluso podríamos decirle que hiciera un poco de aritmética con el contenido original, así: de nuevo, sustituya las líneas 32 y 33, pero por

```
32  LET C = C + 10
33  PRINT C
```

Esto puede parecer bastante extraño, pero lo único que estamos haciendo es decirle al ordenador que dé un nuevo valor a C añadiéndole 10 a su antiguo valor, dándonos 25. (Naturalmente esta vez nuestra sentencia LET parece menos una ecuación; puede usted ver a primera vista que no funcionaría en álgebra, porque ningún valor de C podría hacer que los dos lados de la ecuación fueran iguales.)

Cambiando el contenido de C de esta manera, podemos utilizar C en un bucle tal como los sencillos bucles de GOTO que vimos en el capítulo anterior.

Pruebe esto:

```
10  REM BUCLE DE VARIABLE
20  LET C = 1
30  PRINT C
40  LET C = C + 1
50  GOTO 30
```

Esta vez la sentencia LET únicamente añade 1 al valor almacenado en C cada vez que el ordenador da una vuelta sobre el bucle. Por lo tanto, en la pantalla aparecerá esto:

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

– apriete la tecla BREAK cuando ya haya visto suficiente.

Nombres para las variables

La forma en que el ordenador guarda la información difiere ligeramente, dependiendo de si se trata de un número o de un texto; cuando le damos un nombre de variable hay que especificarle al ordenador si vamos a utilizar la posición(es) para guardar un número o un texto. Hacemos esto mediante la utilización de distintos tipos de nombres de variables para los números y para los textos.

El utilizar nombres diferentes significa que una vez que le hemos dicho al ordenador, por ejemplo, que C es una posición que contiene una variable numérica, no podemos cambiar el contenido de C por un texto. Deberá continuar siendo un número, aunque siempre puede cambiarse a un número diferente.

Estos son los nombres que nuestro ordenador acepta:

Para variables numéricas: nombres de uno o dos dígitos, siendo el primer dígito una letra mayúscula, y el segundo una letra mayúscula o un número. Por lo tanto, todos los nombres de variables numéricas que siguen son aceptables:

AA
A1
Z9

y todos éstos no son aceptables:

2A	(el primero es un número: debería ser una letra).
ABC	(demasiado largo).
A?	(los símbolos que no sean letras o números no están permitidos).

Para variables de texto: nombres similares a los permitidos en las variables numéricas, pero esta vez seguidos por el signo «$». Por lo tanto, éstos son nombres de variables de texto aceptables:

A$
W4$
SS$

y éstos no:

B	(falta el signo $).
4C$	(el primero es un número: debería ser una letra).
X?	(símbolo erróneo: debería ser un $).

Si intenta poner un texto en una posición que tiene un nombre de variable numérica, el ordenador se dará cuenta, y le dirá que ha cometido un error. Lo que sí puede hacer siempre es poner un número en una posición de variable de texto, ya que los números pueden utilizarse en los textos sin ningún problema, pero no podrá realizar ninguna aritmética con la variable, como podría hacerlo si fuera una variable numérica. Y tiene que ponerlo entre comillas.

Variables aritméticas

Mientras se trate de variables numéricas, puede utilizarlas para hacer aritmética de la misma forma que si fueran números. Naturalmente, el ordenador está realmente haciendo la aritmética con los números que la posición de memoria correspondiente a la variable contiene, y no con sus nombres. He aquí un ejemplo:

Pruebe esto:

```
10  REM VARIABLES ARITMETICAS
20  LET X = 2: LET Y = 3
30  PRINT X + Y
40  PRINT X * Y
50  END
```

He aquí un ejemplo que muestra la aritmética con más detalle cuando se está ejecutando.

Pruebe esto:

```
10  REM MAS VARIABLES ARITMETICAS
20  LET X = 5: LET Y = 6
30  PRINT X;"+";Y;"=";X + Y
40  PRINT X;"*";Y;"=";X * Y
50  PRINT X;"/";Y;"="X / Y
60  PRINT X;"-";Y;"=";X - Y
70  PRINT "RAIZ CUADRADA DE";X;"="; SQR(X)
80  END
```

También pueden utilizarse las variables para describir posiciones en la pantalla.

Este es otro programa para llenar la pantalla, pero con una diferencia: ¡funciona hacia atrás!

Pruebe esto:

```
10  REM ENCANTADOR DE SERPIENTES
20  CLS
30  LET N = 15
40  PRINT@ (N,10), "SSSSS";
50  LET N = N – 1
60  GOTO 40
```

¡Obtendrá un mensaje de error cuando N se convierta en un número negativo!

Utilización de las variables de texto

¿Qué hay de las variables de texto? Son útiles en muchas maneras, como veremos más adelante. Primeramente, nos ahorran el teclear.

Pruebe esto:

```
10  REM DIBUJO
15  CLS
20  LET A$ = "** /** /"
30  LET B$ = "--------"
40  PRINT A$;A$;A$;A$;
50  PRINT B$;B$;B$;B$;
60  GOTO 40
```

Este programa está basado en una pantalla de 32 columnas; por lo tanto, cada vez que se escriban las líneas 40 y 50, cada una de ellas llenará una línea de la pantalla. Modifíquelo si su pantalla tiene un tamaño distinto.

Pruebe esto:

```
10  REM PANTALLA LLENA
15  CLS
20  LET A$ = "MUCHAS Y MUCHAS LETRAS"
30  PRINT A$;
40  GOTO 30
```

También podemos incluir códigos de caracteres

gráficos en las variables de texto, utilizando sentencias como:

```
LET A$ = CHR$(128)+CHR$(128)+CHR$(128)
```

que hacen que el dibujar algo como el robot que vimos en el último capítulo sea mucho más fácil. Y, naturalmente, podemos juntar variables de texto así:

```
10  LET A$ = "PALO"
20  LET B$ = "SANTO"
30  LET C$ = A$ + B$
40  PRINT C$
```

También pueden juntarse textos con variables de texto:

```
10  LET A$ = "POSTERIOR"
20  LET B$ = A$ + "MENTE"
30  PRINT B$
```

o puede cambiarse el contenido de una variable de texto, añadiéndole un texto a su contenido original.

Pruebe esto:

```
10  REM TEXTO CRECIENTE
20  LET A$ + "*"
30  PRINT A$
40  LET A$ = A$+"*"
50  GOTO 30
```

No hay que poner esta vez un punto y coma al final de la sentencia de PRINT, a diferencia del programa «pantalla ocupada», ya que queremos ver la línea de asteriscos crecer en una forma triangular, así:

```
*
**
***
****
```

Si deja que se ejecute el programa durante el tiempo suficiente, aparecerá un mensaje de error cuando A$ llegue a ser demasiado grande para que el ordenador pueda manejarlo.

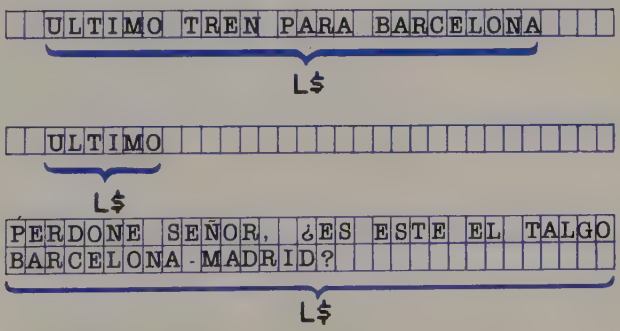
Además de hacer A$ cada vez más grande, siempre podemos hacerlo cada vez más corto, como en este programa:

Pruebe esto:

```
10  REM VARIABLE DE TEXTO MAS CORTA
20  LET A$ = "ULTIMO TREN A BARCELONA"
30  PRINT A$
40  LET A$ = "ULTIMO"
50  PRINT A$
60  END
```

El ordenador da como resultado:

```
ULTIMO TREN A BARCELONA
ULTIMO
```



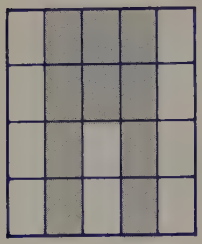
Cambio del contenido de una variable

Si colocamos un texto, tal como «último tren para Barcelona» en una posición de variable de texto llamada L$, entonces podemos decirle al ordenador que permita (LET L$) que contenga algo más pequeño, como en el programa anterior. Podemos también hacer que L$ contenga algo mucho más largo. En ambos casos, el ordenador cambia automáticamente el tamaño de L$ por nosotros.

Como ha podido ver, ha hecho desaparecer toda la información que estaba en A$ cuando le dijimos que pusiera otra cosa allí. No ha borrado únicamente el espacio necesario para colocar el nuevo texto.

Ahora le toca a usted

1) ¿Cuál de estas sentencias será aceptada por nuestro ordenador, y en cuáles señalará error?



(a) *LET A = "MANZANAS"*

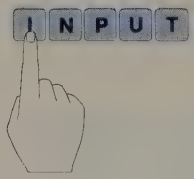
(b) *LET BC = 99*

(c) *LET X$ = 56*

2) Dibuje esta forma colocando textos que contengan los códigos de caracteres gráficos utilizados.

3) Ponga su nombre en una variable de texto, y dígale al ordenador que la escriba en la parte superior, en el medio y en la parte inferior de su pantalla.

La orden INPUT



Veamos ahora otra forma de colocar información en las variables: la orden INPUT. El INPUT le permite hacer exactamente esto: entrar información en el programa mientras se está ejecutando. Entra la información de la misma manera que cuando está escribiendo un programa o trabajando en modo inmediato, pero esta vez el programa lo utiliza directamente. Sin embargo, el programa tiene que colocar la información en algún lugar tan pronto como usted la entra, incluso aunque tenga que utilizarla en la siguiente línea del programa. Para identificar el lugar donde el ordenador coloca la información, se le da a esta posición —ya lo habrá adivinado— un nombre de variable.

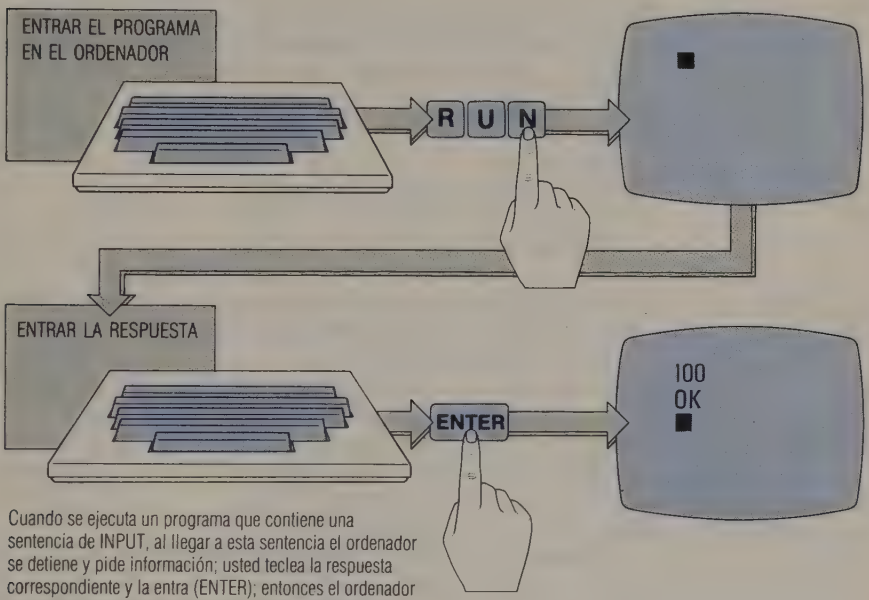
Así es como se utiliza el INPUT:

Pruebe esto:

```
10  REM PROGRAMA DE ENTRADA
20  PRINT "CUAL ES TU NOMBRE?"
30  INPUT N$
40  PRINT "HOLA, ";N$
50  END
```

¿Qué sucede cuando se ejecuta este programa? El ordenador imprime el mensaje de la línea 20, y entonces escribe un signo de interrogación (al menos esto es lo que hace el nuestro) para decirle que está esperando que usted le entre algo. No proseguirá hasta que usted lo haya hecho, por lo tanto escriba su nombre —o cualquier otra cosa que se le ocurra— y apriete ENTER para decirle al ordenador que ha terminado. El ordenador coloca su nombre en la variable que usted

Entrada de información



Cuando se ejecuta un programa que contiene una sentencia de INPUT, al llegar a esta sentencia el ordenador se detiene y pide información; usted teclea la respuesta correspondiente y la entra (ENTER); entonces el ordenador continúa la ejecución del programa, utilizando esta información.

ha llamado N$ (una variable de texto, naturalmente), y entonces lo escribe en la línea siguiente. Es decir, que la ejecución completa del programa es algo tal como:

CUAL ES TU NOMBRE?
? JAVIER MOLINA
HOLA, JAVIER MOLINA
OK

Puede utilizar esta técnica para hacer que el ordenador mantenga una sencilla conversación con usted. Ahora vamos a plantear una en la cual el ordenador nos pregunta:

- primero, nuestro nombre.
- segundo, acerca del tiempo.
- tercero, cómo estamos.

Llamaremos a estas tres piezas de información N$, T$, S$, nombres que nos ayudarán a recordar qué es cada cosa. Por lo tanto podemos empezar a plantear nuestro programa escribiendo una lista de variables, tal como esta:

Variables	
Entrada del nombre:	N$
Entrada del tiempo:	T$
Entrada del sentimiento:	S$

Ahora, planteamos el tipo de conversación que queremos que el programa produzca. Algo así:

El ordenador	*HOLA OTRA VEZ. PERDONA, ME HE OLVIDADO DE TU NOMBRE*
Usted	*RODOLFO*
El ordenador	*AH CLARO. BIENVENIDO, RODOLFO. HACE BUEN DIA?*
Usted	*ESTA LLOVIENDO*
El ordenador	*YA VEO, ESTA LLOVIENDO, NO ES ASI? COMO ESTAS?*
Usted	*MUY BIEN*
El ordenador	*MUY BIEN, EH? BIEN VAMOS A TRABAJAR, RODOLFO DAME TUS INSTRUCCIONES AHORA*
	OK

No es perfecto ya que muchas respuestas que podemos dar producirán que el ordenador nos dé contestaciones poco convincentes, pero sirve para empezar. Ahora intentemos programar esto. Primero la preparación:

```
10 REM ***CHARLA DE BIENVENIDA***
20  CLS
```

y después la propia conversación:

```
30  PRINT "HOLA OTRA VEZ. PERDONA ME HE OLVIDADO
    DE TU NOMBRE.."
40  INPUT N$
50  PRINT "AH CLARO. BIENVENIDO,";N$;". HACE BUEN
    DIA?"
```

```
 60  INPUT T$
 70  PRINT "YA VEO,.";T$;", NO ES ASI?"
 80  PRINT "COMO ESTAS?"
 90  INPUT S$
100  PRINT S$;", EH? BIEN VAMOS A TRABAJAR, ";N$;
110  PRINT "DAME TUS INSTRUCCIONES AHORA"
```

y para terminar:

```
120  END
```

¿Se ha fijado en dónde hemos puesto las comillas y dónde hemos dejado espacios en blanco entre ellas para que la salida aparezca correctamente? Si la suya no aparece en la forma que usted quería, quizá deberá colocar algunos espacios más, para que ninguna palabra aparezca partida al final de una línea de pantalla.

Pruebe el programa, en un ordenador o sobre papel, y vea qué tipos de conversación puede llegar a hacer. Los signos de interrogación no estarán en los lugares perfectos, pero si escoge su entrada cuidadosamente, el resultado puede ser bastante aceptable.

Cómo entrar dos o más datos

La orden INPUT puede utilizarse para entrar datos en más de una posición de variable, si así lo desea. Tiene que darle nombres a todas las variables que necesitará, así:

```
INPUT A, B, C
```

y entonces el ordenador esperará que usted entre cada una de ellas a su turno. En nuestro ordenador las entramos todas en una línea, separadas por comas.

Veamos un ejemplo que muestra las dos cosas, cómo entrar dos datos y cómo se utilizan las variables numéricas. He aquí un sencillo programa aritmético:

Pruebe esto:

```
10  REM ***CALCULADORA***
20  PRINT "MULTIPLICARE DOS NUMEROS"
30  PRINT "QUE TU ME DARAS"
40  PRINT "Y TE DARE EL RESULTADO"
50  PRINT "ENTRA LOS NUMEROS AHORA"
```

```
60  INPUT X,Y
70  PRINT X;"VECES";Y;"ES";X*Y
80  END
```

Nótese cómo hemos utilizado sentencias de PRINT para hacer que el ordenador nos diga lo que está haciendo. Es particularmente importante hacer esto en un programa que nos pide entradas. De otra forma, es fácil encontrarse con un signo de interrogación y no estar seguro de qué información nos está pidiendo el ordenador. Hemos utilizado muchas sentencias de PRINT cortas en lugar de pocas y más largas para que ninguna de las palabras en los textos quede partida al final de una línea de pantalla —es más fácil que descubrir dónde hay que colocar espacios en blanco.

De hecho es tan importante lograr que el computador nos diga lo que está haciendo y qué es lo que quiere, que en nuestro ordenador podemos utilizar el propio INPUT para hacer que nos escriba un texto. Funciona así:

Pruebe esto:

```
10  INPUT "DAME UN NUMERO";N
20  PRINT "TU NUMERO ES";N
30  INPUT "ESTA VEZ DOS NUMEROS";M,P
40  PRINT "HAS ELEGIDO";M;"Y";P
50  INPUT "Y AHORA UNA PALABRA O UNA FRASE";W$
60  PRINT "ESTA VEZ ME HAS DADO";W$
70  PRINT "LISTA COMPLETA:",N,,,,M,,,,P,,,,W$
80  END
```

¿Ha visto claramente lo que hemos hecho aquí? Para asegurarse, realice el programa y haga un listado para usted de todas las variables. Examine cuidadosamente la puntuación. ¿Ha visto cómo hemos dejado un espacio al final del texto a escribir cuando le sigue una variable de texto (no hay espacio al principio de los textos), pero que no hay espacio cuando le sigue un número (generalmente el ordenador ya colocará uno). Nótese también cómo hemos utilizado comas para conseguir los datos escritos en una lista, en la línea 70.

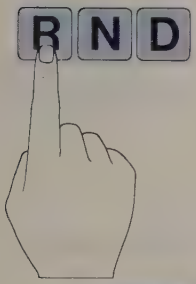
Ahora le toca a usted

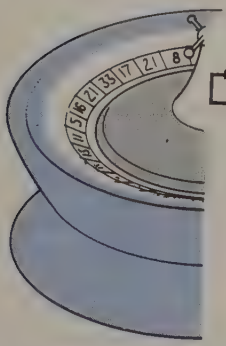
4) Escriba un programa que haga que el ordenador

le pida dos números x e y, y entonces le dé su suma (x + y), su diferencia (x – y), su producto (x veces y) y el resultado de dividir x por y.

5) Escriba un programa que haga que el computador le pida una serie de caracteres gráficos, los visualice y entonces vuelva atrás y le pida otra serie para visualizarla.

Números aleatorios





Elección de números aleatorios
Los números que aparecen en la rueda de una ruleta son generados de una forma completamente aleatoria. El ordenador puede hacer algo parecido a esto con la función RND.

Para terminar el capítulo, veremos otra forma de cambiar la información en un programa: la función de números aleatorios, RND. Existen versiones distintas de esto. Compruebe en su manual si su ordenador no acepta la nuestra.

El ordenador no puede realmente tirar un dado, pero puede aprovechar la enorme complejidad de sus circuitos para escoger números que aparezcan para nosotros tan aleatorios como los números que aparecen cuando se tira un dado. Nuestro ordenador produce números aleatorios entre 0 y 1, por lo que de hecho todos los números son decimales con una precisión máxima de 9 dígitos. Veamos qué apariencia tienen:

Pruebe esto:

```
10 PRINT RND
20 GOTO 10
```

Detenga el ordenador después de una página o algo así, para poder ver los números. Serán algo tal como:

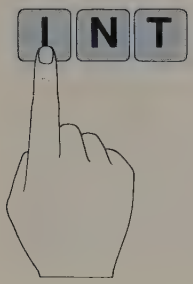
```
.753755291
.984257657
.528056959
.265991296
.0327720944
```

Nótese que RND es una función, no una orden. Como tal, no puede utilizarla al principio de una sentencia. Hay que empezar la sentencia con una orden tal como PRINT que le dice al ordenador qué es lo que hay que hacer con el resultado de la expresión RND.

Los números decimales aleatorios están bien para los matemáticos, pero para utilizar estos números en, por ejemplo, un programa de simulación de dados, debemos cambiarlos a números enteros. ¿Cómo hacemos esto? En dos etapas:

a) Multiplicamos los números aleatorios por un número entero que indica el rango de los números que queremos. Por ejemplo, 6 sería el número que nos daría un rango de seis números distintos tal como los que conseguiríamos al tirar un dado.

b) Entonces convertimos el resultado en un número entero, utilizando otra función, INT (que viene de INTeger, entero). El INT nos da el número entero que está por debajo del número decimal que le sigue; en otras palabras lo redondea por debajo. Así:



INT (5.5)

da como resultado 5; y

INT (79.23)

da 79.

La combinación de las etapas *a)* y *b)* nos da este tipo de sentencia:

```
PRINT INT(RND*6)
```

Necesitamos los paréntesis: le dicen al ordenador que haga primero la multiplicación, y entonces que coja el número entero por debajo del resultado. INT(RND)*6 nos daría el entero por debajo de los números aleatorios entre 0 y 1 (es decir, 0 en todos los casos) multiplicado por 6, que sigue siendo 0 y que por lo tanto no es muy útil.

Hay que tener en cuenta otra cosa. Ya que la función INT redondea el número por debajo, esta sentencia nos daría un rango de números aleatorios desde 0 a 5 y no desde 1 a 6 como los números de un dado. (Pruébelo y véalo, si así lo desea.) Para incrementar el punto inicial, tan sólo hay que añadir 1. Por lo tanto, para conseguir números enteros aleatorios en el rango de 1 a 6, nuestra sentencia completa es:

```
PRINT 1 + INT(RND*6)
```

Naturalmente, podemos conseguir números aleatorios dentro de cualquier otro rango, de la misma mane-

ra. Para conseguir números en el rango, por ejemplo, de 20 a 30, será necesaria esta sentencia:

```
PRINT 20 + INT(RND*11)
```

Liebre

Tortuga

¿Ha cogido esto? Esperamos que sí, ya que ahora vamos a escribir un sencillo programa que utiliza el RND. Se trata de una «carrera». Supongamos que tenemos una liebre y una tortuga haciendo una carrera a través de la pantalla, de un lado al otro. Cada vez los haremos avanzar por la pantalla un número aleatorio de columnas, desde 1 a 4. Puede apostar sobre quién llegará antes al lado derecho de la pantalla.

Ya que todavía no hemos aprendido cómo salir de un bucle sin apretar el «BREAK», realizaremos el movimiento repetido mediante un bucle sin fin con un GOTO.

Esto significa que nuestra carrera terminará no muy elegantemente, con un mensaje de error.

¿Cómo lo haremos? Bien, utilicemos dos variables de texto para contener la información sobre las formas de nuestra liebre y nuestra tortuga. No haremos las formas muy grandes, porque entonces serían difíciles de mover. Los dibujos le muestran las formas que hemos escogido. Llamaremos a las variables H$ y T$, que nos darán estas líneas:

```
100  LET H$ = CHR$(137)+CHR$(134)
110  LET T$ + CHR$(136)+CHR$(132)
```

Utilizamos números de líneas grandes, para ellas, para poder incluirlas más tarde en otras líneas de programa que deban ejecutarse antes.

¿Cómo decidiremos dónde colocar la liebre y la tortuga? Coloquémoslas en las líneas 6 y 8 de nuestra pantalla. Iremos avanzando sus posiciones de columna a medida que vayan corriendo, por lo que utilizaremos —¿cómo lo ha adivinado?— variables numéricas para describirlas. Digamos H para la liebre y T para la tortuga. El ordenador no las confundirá con H$ y T$ ya que sabe que H y T deben contener números y H$ y T$ deben contener textos o caracteres gráficos. Esto nos dará una sentencia de PRINT tal como:

```
PRINT@ (6,H), H$
```

¿Comprendido? Bien, digámosle al ordenador algo acerca de H y T. Aunque el ordenador las inicializará automáticamente a 0, es interesante el poner una línea en el programa que nos recuerde qué valores queremos que tengan al principio. Por lo tanto:

```
120  LET H = 0: LET T = 0
```

Puede utilizar dos líneas separadas para esto, si así lo prefiere.

Ahora el inicio de nuestro bucle, con un número de línea bien distintivo:

```
200  PRINT@ (6,H), H$
210  PRINT@ (8,T), T$
```

y un par de sentencias, en el bucle, que incrementen el valor de H y T en una cantidad aleatoria.

```
220  LET H = H + 1 + INT(RND*4)
230  LET T = T + 1 + INT(RND*4)
```

El ordenador tomará la totalidad de «H+1+ INT(RND*4)», por complicado que a nosotros nos parezca, como una expresión que evaluará. Entonces colocará el resultado en H, ya que esto es lo que le hemos dicho que haga. Por lo tanto, podemos volver atrás hasta la línea 200, y la liebre será visualizada en algún otro lugar.

Habrá que deshacerse de la liebre anterior o tendremos una serpiente de figuras de liebre a través de la pantalla. Hay varias maneras de hacer esto, pero nos ceñiremos a una muy sencilla, que nos deshará también de la anterior tortuga.

```
240  CLS
```

Esto va incluido también en el bucle, por lo que ahora podemos cerrarlo y volver atrás para mover la liebre y la tortuga a través de la pantalla:

```
250  GOTO 200
```

¿Contento con esto? Vamos a reescribir el programa

completo, junto con nuestra lista de variables, y algunos pasajes introductorios más profesionales:

Variables

Forma de la liebre:	H$
Forma de la tortuga:	T$
Posición de la columna de la liebre:	H
Posición de la columna de la tortuga:	T

```
10  REM ***LA CARRERA DE LA LIEBRE Y LA TORTUGA***
20  REM POR SUSAN CURRAN
30  CLS
100 LET H$ = CHR$(137)+CHR$(134)
110 LET T$ = CHR$(136)+CHR$(132)
120 LET H = 0: LET T = 0
190 REM EMPIEZA LA CARRERA
200 PRINT@ (6,H), H$
210 PRINT@ (8,H), T$
220 LET H = H + 1 + INT(RND*4)
230 LET T = T + 1 + INT(RND*4)
240 CLS
250 GOTO 200
```

Como recuerdo para usted, ¿por qué no guardar (SAVE) este programa?

Ahora le toca a usted

6) Escriba un programa que genere una pantalla llena de números enteros aleatorios dentro del rango de 17 a 26.

7) Escriba un programa que envíe una forma de camión a través de la pantalla, desde la izquierda hacia la derecha, a una velocidad aleatoria que usted escoge antes de que empieze a moverse (y que entonces permanece constante).

Comprobación

He aquí nuestra comprobación de los puntos que hemos examinado en este capítulo. Asegúrese de que está satisfecho de todos ellos, antes de pasar al próximo capítulo.

Sobre variables
- Qué es un nombre de variable.
- Qué nombres acepta su ordenador para textos y variables numéricas.
- Cómo puede cambiar la información en una variable.
- Cómo se utiliza la orden LET...= para colocar información en una posición de variable.
- Cómo utilizar nombres de variables en lugar de los datos,en las sentencias de PRINT, en expresiones aritméticas, etc.

Sobre la orden INPUT
- Cómo utilizar el INPUT para hacer que el ordenador se detenga y espere una entrada.
- Cómo asignar un nombre de variable a una entrada.
- Cómo utilizar el INPUT para hacer salir un signo de «preparado» en la pantalla.
- Cómo entrar más de un dato al mismo tiempo.

Sobre las funciones RND y INT
- Cómo se utiliza el RND para generar números aleatorios entre 0 y 1.
- Cómo regenerar números enteros aleatorios dentro de distintos rangos.
- Cómo se utiliza generalmente el INT para convertir números decimales en números enteros.

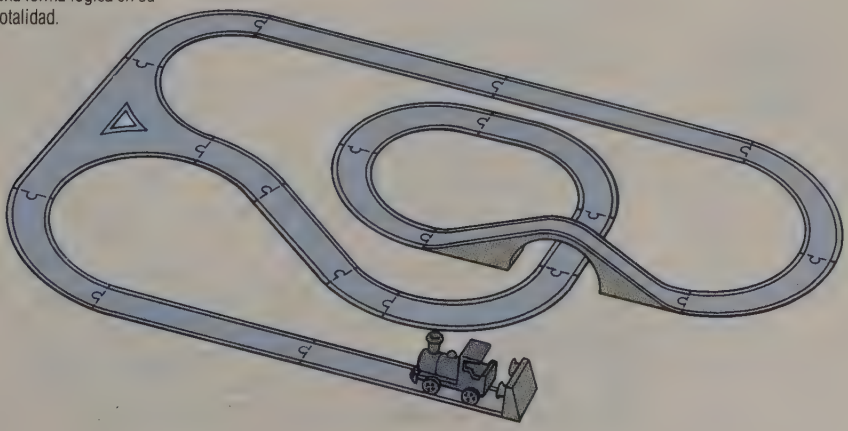
4 Bucles y ramificaciones

En este capítulo aprenderemos acerca de:
- **Bucles abiertos y cerrados.**
- **Las órdenes IF...THEN.**
- **Las órdenes FOR...NEXT y STEP.**
- **Operadores lógicos.**

Nuestro hijo pequeño tiene un tren de madera. Consta de muchas pequeñas y grandes piezas de vía, desvíos, puentes y pasos a nivel, que hay que juntar correctamente para conseguir una vía por la que pueda circular el tren. Esto puede ser un trabajo no tan sencillo como parece, ya que hay que planearlo de forma que los pequeños bucles estén dentro de los grandes, que la vía del fondo dé la vuelta y pase exactamente por debajo del puente, etc.

En muchos aspectos, el planear un programa es como el montar la vía de este tren. Las piezas de vía son como las líneas de programa, que hay que juntar en un orden lógico. De cuando en cuando hay un des-

En la vía correcta
Empalmando correctamente las vías de un tren eléctrico se realiza algo parecido a asegurarse que las distintas partes de un programa encajan de una forma lógica en su totalidad.



vío —un punto de decisión— donde hay que elegir un cambio u otro. Esto corresponde a un aspecto de la programación que estamos a punto de abordar —cómo hacer que el ordenador realice varias acciones alternativas que podemos colocar en un programa—. Hay grandes bucles y a veces pequeños bucles en su interior e incluso bucles todavía más pequeños dentro de éstos. Todo debe ser planeado cuidadosamente para que no haya un lío de finales de vía perdidos, de forma que pueda hacerse el máximo posible con un número limitado de piezas. Si se ha hecho bien, entonces el tren circulará a lo largo de la vía —o el ordenador ejecutará el programa— de una forma fácil y efectiva.

Ya hemos visto algunas de las sentencias que constituyen las vías de nuestro programa. En este capítulo veremos algunas de las técnicas utilizadas para proveer de bucles y ramificaciones a nuestro sistema.

Bucles cerrados y abiertos

Ya hemos utilizado la orden GOTO, para hacer bucles cerrados muy sencillos en nuestros programas. Les llamamos bucles cerrados, ya que una vez el ordenador ha entrado en uno de estos bucles, no tiene manera de salirse. Tendremos que apretar la tecla BREAK para pararlo. Esto es como tirar de la palanca de alarma en un tren; efectivo, pero no la forma ideal para hacer paradas rutinarias programadas.

¿Cómo podemos entonces abrir un bucle? Hay básicamente dos maneras. La primera, podemos decirle al ordenador que cuente cuántas veces realiza el bucle y, cuando lo ha realizado un número determinado de veces, que ejecute el resto del programa. (Esto es como el tener a un guardavías que cuente cuántas veces el tren pasa por un bucle de vía. Cuando ya ha dado suficientes vueltas, el guardavías cambia el desvío, y la próxima vez el tren seguirá por la vía principal.) O, la segunda, podemos colocar un desvío o ramificación —un punto de decisión— dentro del bucle. Cada vez que el ordenador da una vuelta por el bucle, le preguntamos si ha sucedido algo; por ejemplo, si una variable que estamos incrementando continuamente ha alcanzado un valor determinado, o si una parte de una entrada en particular es igual a un cierto valor. Si la respuesta es sí y ésta es la respuesta que

queremos, enviamos al ordenador por un camino diferente. Si es no, lo enviamos de nuevo al bucle.

Veremos primero una secuencia de órdenes que hace la segunda alternativa. Después, más adelante en el capítulo, volveremos atrás para ver la primera alternativa.

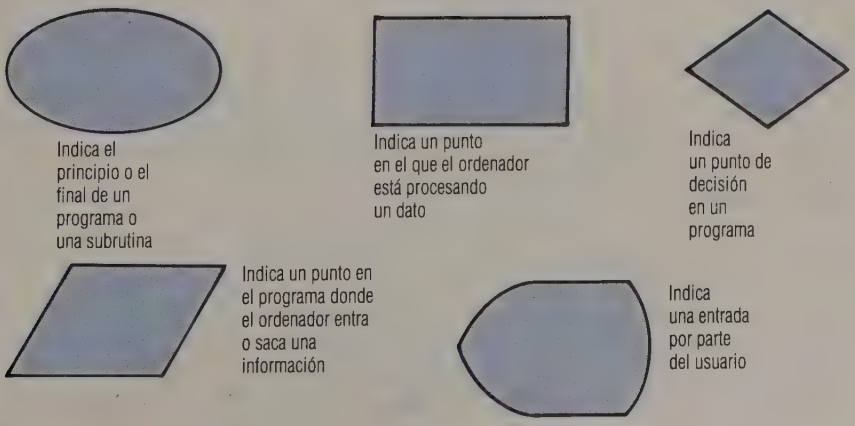
Diagramas de flujo

Quizás haya usted oído hablar antes de diagramas de flujo. Son el equivalente para el programador del trazado de las vías. Son sencillamente diagramas que indican qué caminos están disponibles para el ordenador, y cómo decide qué camino tomar. Es decir, consisten en vías —líneas con flechas que muestran el camino que sigue el computador en el programa— separadas por zonas de acción —puntos donde el computador ejecuta alguna orden, por ejemplo, imprimiendo algo—, puntos de decisión —donde el programa le da al ordenador alternativas que tiene que evaluar y escoger—, puntos donde se recibe una entrada, y así sucesivamente.

Utilizamos un conjunto de símbolos convencional para marcar los distintos sucesos que aparecen en un diagrama de flujo. La figura le muestra los símbolos que vamos a utilizar en este libro.

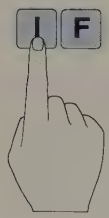
Depende de usted el que quiera utilizar los diagramas de flujo en los programas que usted mismo escri-

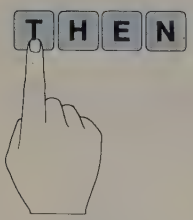
Símbolos del diagrama de flujo

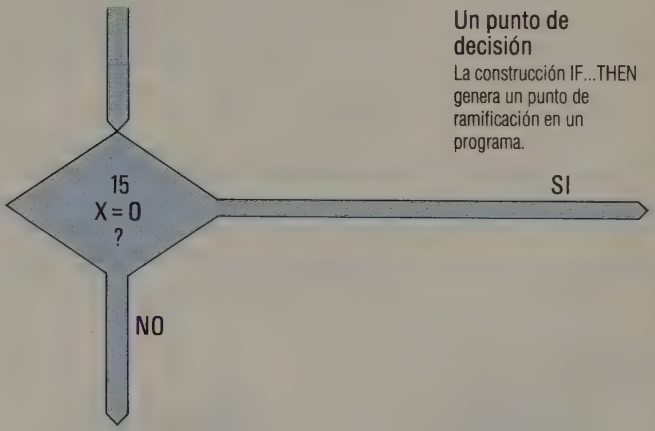


be. De hecho necesita planear sus programas, pero quizás encuentre más fácil hacerlo con palabras que con símbolos. Sin embargo, nosotros preferimos los diagramas de flujo, y utilizaremos unos cuantos en este libro, a medida que vayamos escribiendo programas un poco más complicados.

La siguiente orden que veremos es la construcción IF...THEN. Señala —como ya prometimos— un punto de ramificación, o punto de decisión, en el programa. Este es el punto representado en un diagrama de flujo por el símbolo del rombo. En el rombo hacemos una pregunta. Si la respuesta es SI, entonces nos desviamos por el camino marcado con el SI. Si es NO, entonces nos desviamos por el camino del NO. En un diagrama de flujo esto es algo tal como:







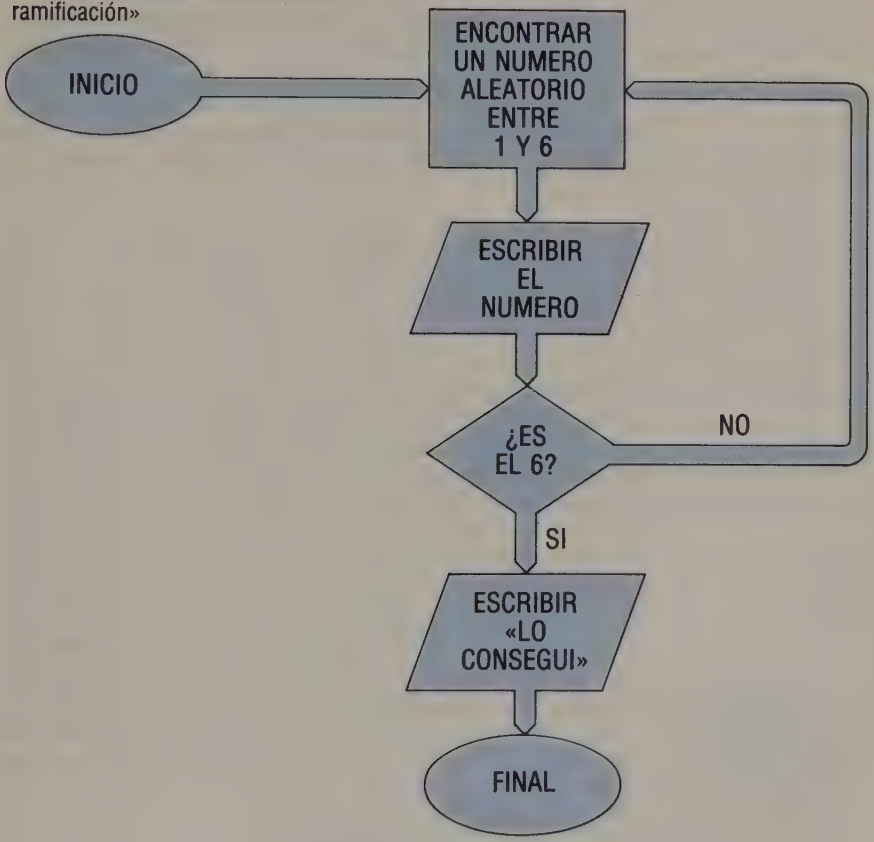
Un punto de decisión
La construcción IF...THEN genera un punto de ramificación en un programa.

Así es como aparece en un programa. Aquí lo utilizamos para romper un bucle abierto, pero naturalmente puede utilizarse también en cualquier otro sitio.

Pruebe esto:

```
10  REM BUCLE CON RAMIFICACION
20  LET N = 1 + INT(RND*6)
30  PRINT N
40  IF N = 6 THEN GOTO 60
50  GOTO 20
60  PRINT "LO CONSEGUI!"
70  END
```

Diagrama de flujo de un «bucle de ramificación»



¿Ve usted lo que sucede aquí? A medida que el programa da vueltas alrededor del bucle, va dando valores distintos a N, como si estuviera tirando un dado una y otra vez. Si el número que está en N no es 6, el programa vuelve al principio del bucle y lo intenta otra vez. Si es 6, la sentencia THEN entra en acción, y desvía el programa fuera del bucle.

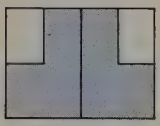
Nótese que, a diferencia de la mayoría de las órdenes del BASIC, el THEN no aparece al principio de la sentencia. Por el contrario, donde aparece es en la mitad de la sentencia que empieza por IF. Y generalmente va seguida por una segunda orden, que le dice al ordenador qué es lo que debe hacer si se cumple la

condición IF. (Algunos BASICs permiten suprimir el THEN o la segunda palabra de la orden, como puede ser GOTO, pero en nuestra versión hay que poner ambos dentro de la sentencia.)

Veamos algunas maneras en que podemos utilizar esto en los programas reales. Este es un ejemplo que se encontrará muchas veces en listados de programas de juegos:

```
100  REM BUCLE CON REPETICION
110  INPUT "QUIERES JUGAR DE NUEVO? S/N";R$
120  IF R$ = "S" THEN GOTO 30
130  END
```

Esto es realmente el final de un bucle mayor, que consiste en un nuevo intento en un juego. Iría al final de la parte principal del programa del juego, después de los mensajes de introducción que no se quieren repetir cada vez. Utilizamos el valor de una variable de texto que el jugador entra (R$ de «respuesta») para realizar nuestro punto de decisión. Si R$ es «S», entonces el jugador es que quiere jugar otra vez y se repite el bucle del juego. (Estamos suponiendo que la línea 30 es el punto donde empieza de nuevo el juego.) Si R$ es cualquier cosa distinta de «S», entonces el programa se desvía fuera del bucle del juego, y se acaba.



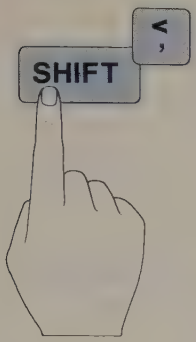
Coche

He aquí una manera de simplificar los programas de movimiento a través de la pantalla, que aprendimos a manejar en el capítulo anterior:

Pruebe esto:

```
 10  REM COCHE EN MOVIMIENTO
 15  CLS
 20  LET P = 0
 30  LET C$ = CHR$(136)+CHR$(132)
 40  PRINT@ (10,P), C$
 50  LET P = P + 1
 60  IF P = 30 THEN GOTO 90
 70  CLS
 80  GOTO 40
 90  GOTO 90
```

Esta vez, utilizaremos la variable P (posición del coche) como nuestra herramienta de desvío. Si P es 30,



lo que significa que los dos caracteres del coche han llegado a las dos últimas columnas de nuestra pantalla de 32 columnas (recuerde que empezamos a visualizar el coche en P: hay que determinar también dónde terminamos de visualizarlo), entonces detenemos el programa. Si P es menor que 30, entonces tenemos que incrementar P y visualizar el coche una columna más a la derecha.

Si estuviéramos utilizando un movimiento aleatorio para el coche entonces tendríamos que modificar esto ligeramente. P quizás nunca llegue a ser exactamente 30. El coche puede ser visualizado en un bucle en la columna 29(/30), y entonces P ser incrementado a, digamos, 33. Para manejar esto, le decimos al ordenador que compruebe si P es mayor o igual que 30. Existen una serie de «operadores lógicos» (de hecho son tipos de función al igual que los «operadores aritméticos» +, –, etc.) que el ordenador utiliza para comprobar este tipo de condiciones. He aquí la totalidad de ellos:

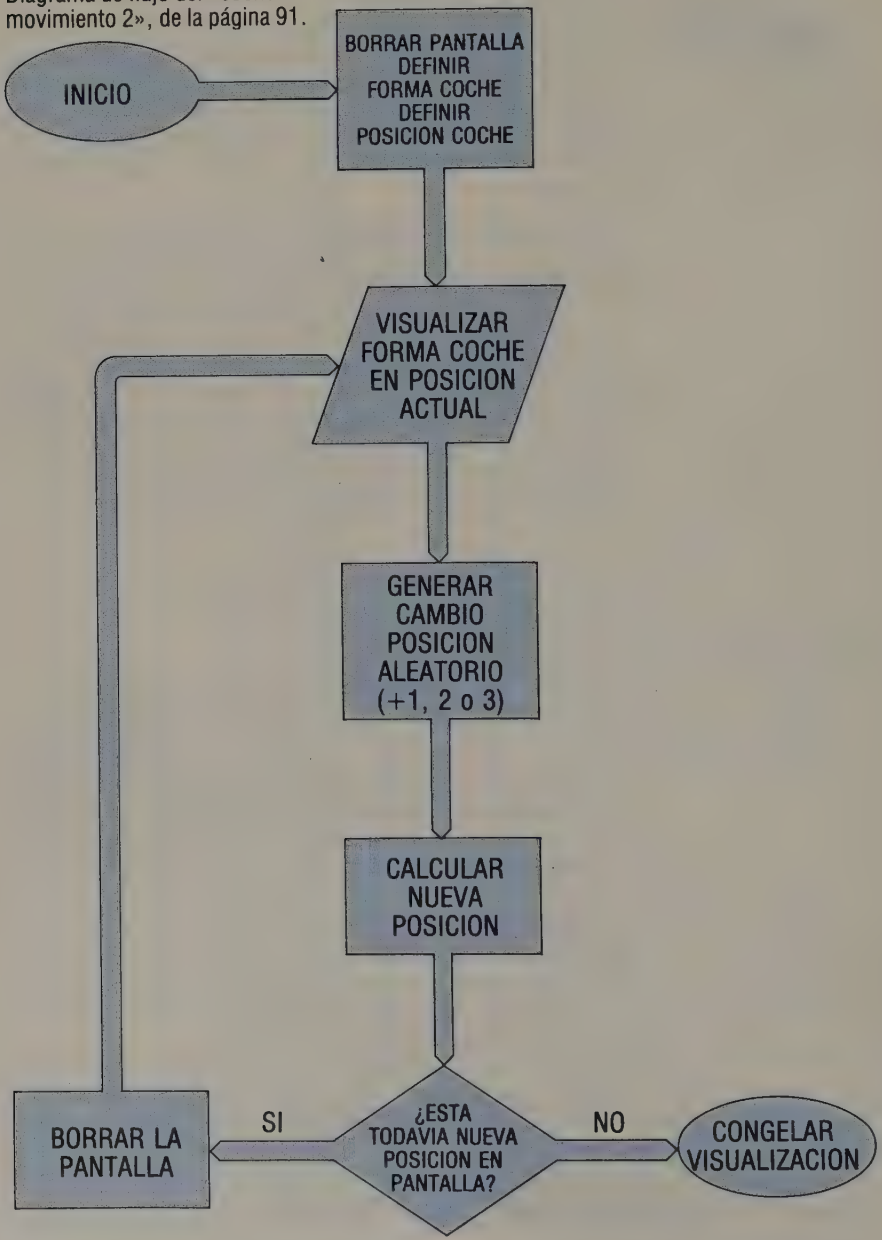
=	igual a
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
<>	distinto que (o mayor o menor que).

He aquí el anterior programa de «coche», modificado para darle al coche un movimiento aleatorio correcto, y para mostrar cómo cambiamos la sentencia IF:

Pruebe esto:

```
10  REM COCHE-EN-MOVIMIENTO 2
15  CLS
20  LET P = 0
30  LET C$ = CHR$(136)+CHR$(132)
40  PRINT@ (10,P), C$
45  LET R = 1 + INT(RND*3)
50  LET P = P + R
60  IF P >= 30 THEN GOTO 90
70  CLS
80  GOTO 40
90  GOTO 90
```

Diagrama de flujo del «coche en movimiento 2», de la página 91.



Usted puede ver que hemos añadido la línea 45, para hacer que el coche avance un número de columnas aleatorio distinto (de 1 a 3) cada vez que el programa da una vuelta sobre el bucle, y que hemos modificado las líneas 50 y 60.

En la página siguiente mostramos un diagrama de flujo para este programa, para ilustrar cómo es en la forma de diagrama de flujo.

La pelota que rebota

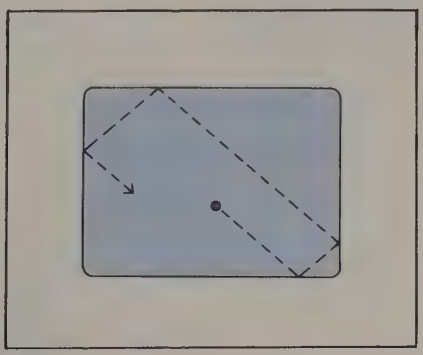
Hasta ahora, hemos escrito programas en los que la orden THEN va seguida por una orden GOTO. Esta es la forma en que podemos salir de un bucle —dejamos de dar vueltas sobre el bucle y vamos a algún otro lugar—. Pero también podemos utilizar el IF... THEN para hacer otra clase de cambios en nuestros programas. Por ejemplo, podemos poner a continuación del THEN una orden LET que cambie el valor de la variable.

Para ilustrar esto escribamos un programa de la «pelota que rebota». Utilizaremos una «O» como pelota. Pero si lo prefiere puede utilizar un pequeño cuadrado negro, el CHR$ (128). La idea del programa es muy sencilla. Nuestra pequeña pelota se mueve a través de la pantalla, arriba y abajo en direcciones diagonales. Cada vez que golpea el extremo de la pantalla, rebota hacia otra dirección. Exactamente la misma idea, con algunos adornos más, es utilizada en programas de juego como el «Frontenis».

Esto no es tan fácil de programar como parece, por

La pelota que rebota

Podemos utilizar puntos de decisión para cambiar el valor de una variable —en este caso, el cambio de las coordenadas de la pelota producirá un cambio de dirección—. Las líneas punteadas muestran el camino que seguirá la pelota en su movimiento a través de la pantalla, rebotando en dirección opuesta cuando alcanza el extremo del área de la pantalla.



lo tanto vamos a planear nuestro programa dibujando un diagrama de flujo (que está en la página siguiente). Necesitaremos cuatro variables en este programa, y no dos como quizás usted había pensado. Una para la columna en la cual visualizaremos la pelota en el movimiento siguiente, y una para su fila, pero también una para el cambio de la posición horizontal (movimiento a izquierda o derecha), y una para el cambio de la posición vertical (movimiento hacia arriba o hacia abajo). Llamaremos a las dos primeras R y C, y a las dos segundas DR y DC; todas variables numéricas, naturalmente.

Sigamos nuestro proceso habitual, y pongámoslas en una lista de variables:

Variables	
Posición de fila:	R
Posición de columna:	C
Cambio horizontal:	DR
Cambio vertical:	DC

No necesitamos tener seis variables: variables distintas para el movimiento arriba, abajo, izquierda y derecha. Arriba es la misma que «menos abajo» y derecha es lo mismo que «menos izquierda». En caso de que no esté usted seguro sobre esto, escribiremos unas líneas del programa que se lo aclararán.

Primeramente le daremos a la pelota una posición inicial en el centro de la pantalla:

```
100  LET R = 8: LET C = 15
```

y entonces cambiaremos la posición de la bola en una fila y una columna cada vez que pase por el bucle:

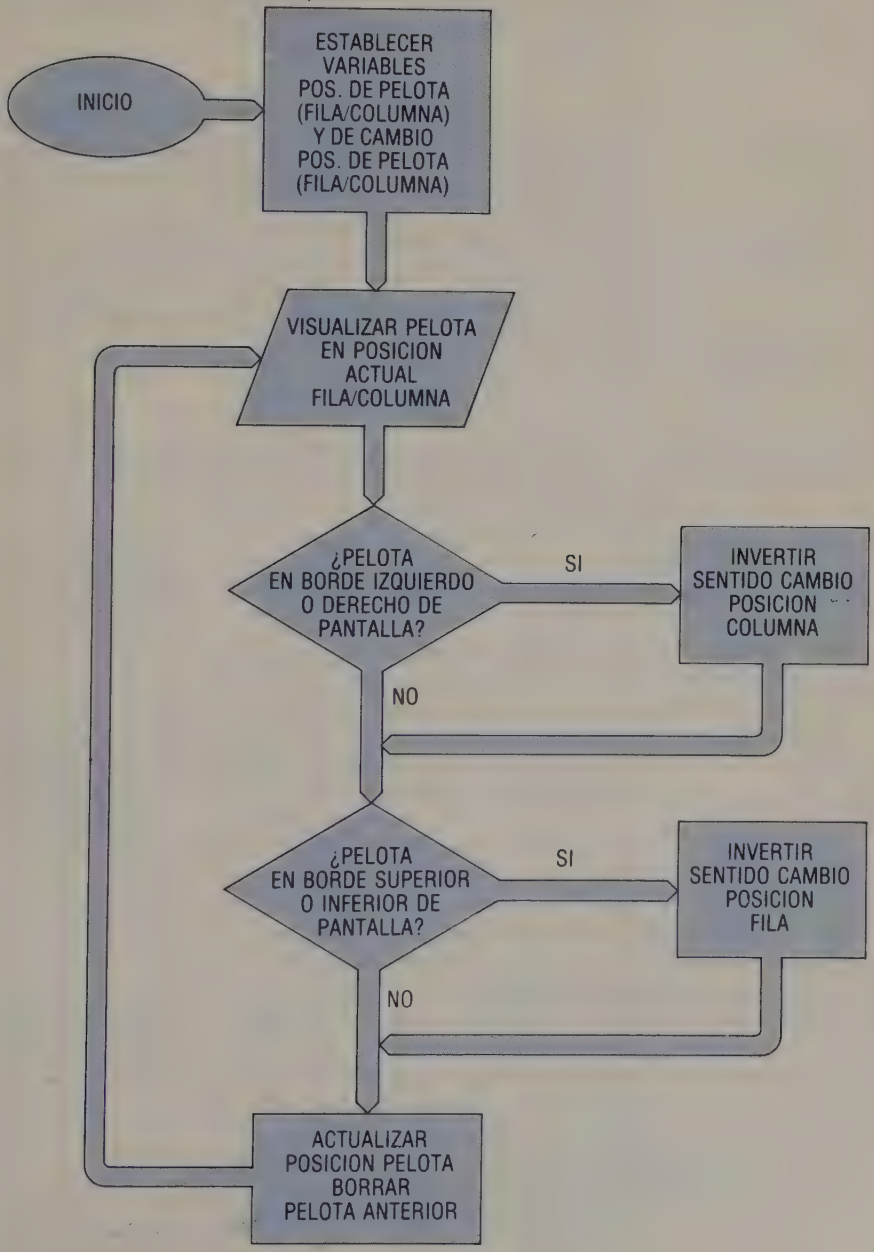
```
110  LET DR = 1: LET DC = 1
```

Para ir hacia abajo, por ejemplo, incluimos una línea como ésta:

```
200  LET R = R + DR
```

de forma que cuando R era 8, se convierte en 8 + 1, o

Diagrama de flujo de la «pelota que rebota»



9. Para ir hacia arriba sencillamente cambiamos el valor de DR a –1:

```
LET DR = –1
```

(no es todavía una línea, sino únicamente una sentencia que tendremos que colocar después) y entonces cuando se llegue de nuevo a la línea 200, si R era 8, se convierte en 8+(–1) o 7.

¿Cuándo cambiamos DR a –1? Cuando la bola golpee la parte inferior de la pantalla y no pueda seguir moviéndose hacia abajo, naturalmente. En este caso R será 15. Por lo tanto podemos cambiar el valor de DR en una línea tal como:

```
300  IF R = 15 THEN LET DR = –1
```

Cuando la bola golpee la parte superior, naturalmente tendremos que hacer lo contrario:

```
310  IF R = 0 THEN LET DR = 1
```

Y tendremos que hacer lo mismo con las posiciones de columna:

```
320  IF C = 0 THEN LET DC = 1
330  IF C = 31 THEN LET DC = –1
```

Ya está escrita la mayor parte de nuestro programa. Coloquemos estas líneas en una versión definitiva y veamos si puede usted seguirla:

```
10   REM ***LA PELOTA QUE REBOTA***
20   REM POR SUSAN CURRAN
30   CLS
100  LET R = 8: LET C = 15
110  LET DR = 1: LET DC = 1
120  PRINT@ (R,C), "0";
200  LET R = R + DR
210  LET C = C + DC
300  IF R = 15 THEN LET DR = –1
310  IF R = 0 THEN LET DR = 1
320  IF C = 0 THEN LET DC = 1
330  IF C = 31 THEN LET DC = –1
340  CLS
350  GOTO 120
```

Sorprendentemente corto cuando está acabado, ¿no es así? Utilice el diagrama de flujo para recordar lo que se está haciendo en cada línea.

Ahora le toca a usted

1) Vea si puede programar un sencillo juego de adivinanzas, utilizando el IF...THEN. Haga que el computador escoja un número, digamos desde el 1 al 6, utilizando la función RND, y entonces entre su suposición. Si su suposición es cierta, entonces haga que el ordenador se lo diga. Si es incorrecta haga que el ordenador le dé otra oportunidad.

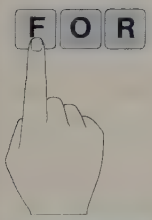
2) Haga que el ordenador le invite a escoger un número del 1 al 6. En función de cuál es el número escogido por usted cada vez, haga que el ordenador escriba un mensaje distinto.

3) Reescriba el programa del «encantador de serpientes» de la página 71, de forma que la serpiente se detenga en la parte superior de la pantalla.

Cómo contar los bucles

Veamos ahora una forma alternativa de abrir un bucle, el contador de bucle. Utilizamos la orden FOR-...NEXT para este caso.

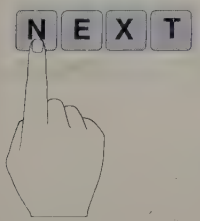
Para poder contar cuántas veces pasamos por un bucle, necesitamos —como ya se habrá imaginado— la ayuda de una variable numérica, que actúa como contador. La orden FOR es la que nos permite hacer esto. Colocamos una sentencia tal como:



FOR N = 1 TO 10

que le dice al ordenador que ponga un «contador de bucle», que llamaremos N. Le da a N los valores 1, 2, 3, ..., hasta 10, cada vez que pasamos por la línea donde está el FOR del bucle.

Para volver al principio del bucle utilizamos la parte NEXT de la construcción. Por lo tanto el bucle terminará con una sentencia tal como:



NEXT N

Una vez más, esto estará probablemente más claro una vez que lo hayamos hecho. Por lo tanto probemos un programa:

Pruebe esto:

```
10  REM CONTADOR DE BUCLE
20  FOR N = 1 TO 10
30  PRINT N
40  NEXT N
50  END
```

Cuando se ejecute esto, deberá obtenerse esta salida:

```
1
2
3
4
5
6
7
8
9
10
OK
```

Está claro, ¿no es así?

Veamos esto de nuevo. La línea 20 instruye al ordenador para que establezca una secuencia de números —y los deja preparados para ir cogiéndolos uno a uno cada vez— y le da a N el valor del primero de la secuencia. La línea 40 hace volver atrás, hasta la línea 20, y hace que N tenga el valor del siguiente número de la secuencia.

Nótese que aunque la línea que contiene el FOR y la línea que contiene el NEXT están separadas —y pueden estar separadas por muchas líneas de programa— son en realidad una única construcción como lo es IF...THEN. La orden FOR estaría incompleta si no estuviera el NEXT en algún lugar del programa.

Naturalmente, se pueden colocar muchas cosas más elaboradas, dentro del bucle FOR...NEXT, si así lo desea. El programa que le mostramos a continuación es un ejemplo ligeramente más exótico:

Pruebe esto:

```
10  REM PIRAMIDE
20  CLS
```

```
30  LET C = 15
40  LET A$ = "*"
50  FOR R = 0 TO 15
60  PRINT@ (R,C), A$;
70  LET C = C - 1
80  LET A$ = A$ + "**"
90  NEXT R
100 GOTO 100
```

¡No hay premio por acertar el cómo será la salida esta vez!

Puede usted ver lo útil que la construcción FOR-...NEXT es para los bucles con contador, y generalmente utilizaremos esto cuando queramos que un programa lleve a cabo una operación un número determinado de veces antes de pasar a hacer otra cosa.

Sin embargo, algunas veces quizá queramos empezar a contar una secuencia, pero no necesariamente que continúe hasta que el contador quede exhausto. Por ejemplo, podemos dar a un jugador en un juego, seis intentos para encontrar una respuesta, o para disparar a un blanco. Si el jugador realiza la tarea en menos de seis intentos, entonces querremos salir directamente del bucle y entrar en una secuencia de «has ganado». En este caso, la orden FOR...NEXT no es realmente apropiada.

La mayoría de los ordenadores no le avisan inmediatamente de que ha cometido un error si sale de un bucle FOR...NEXT antes de hora, pero es una técnica de programación pobre el hacerlo. El ordenador mantiene un registro de los FOR y de los NEXT correspondientes, en su memoria, y si no tiene una oportunidad de «tacharlos» a medida que ejecuta las órdenes, entonces su «stack» de memoria queda lleno de valores inservibles. Si se desea salir de un bucle con contador, entonces será necesario utilizar en su lugar una construcción IF...THEN, de la forma que hicimos en el programa de «coche en movimiento» de la página 90. Es ligeramente más largo, pero ahorra problemas a la larga.

He aquí un ejemplo de este tipo de bucles «semicontados». Es un juego de adivinanza y similar al que le pedimos que programase en la página 96. Verá que colocamos un contador, T, para el número de inten-

tos que el jugador tiene para adivinar la respuesta, pero que utilizamos T con un LET y no con FOR, en este tipo de construcción.

Pruebe esto:

```
10   REM ADIVINA EL NUMERO
20   LET N = 1 + INT (RND*10)
30   PRINT "ADIVINA EL NUMERO"
40   PRINT "QUE ESTOY PENSANDO"
50   PRINT "SI NO LO ADIVINAS EN CINCO INTENTOS"
60   PRINT "TE DIRE LA RESPUESTA"
70   LET T = 1
80   INPUT G
90   IF G = N THEN GOTO 170
100  LET T = T + 1
110  IF T = 6 THEN GOTO 140
120  PRINT "NO, PRUEBA OTRA VEZ"
130  GOTO 80
140  PRINT "LO SIENTO, YA NO TIENES MAS INTENTOS"
150  PRINT "LA RESPUESTA ERA";N
160  END
170  PRINT "MUY BIEN, ERA";N
180  END
```

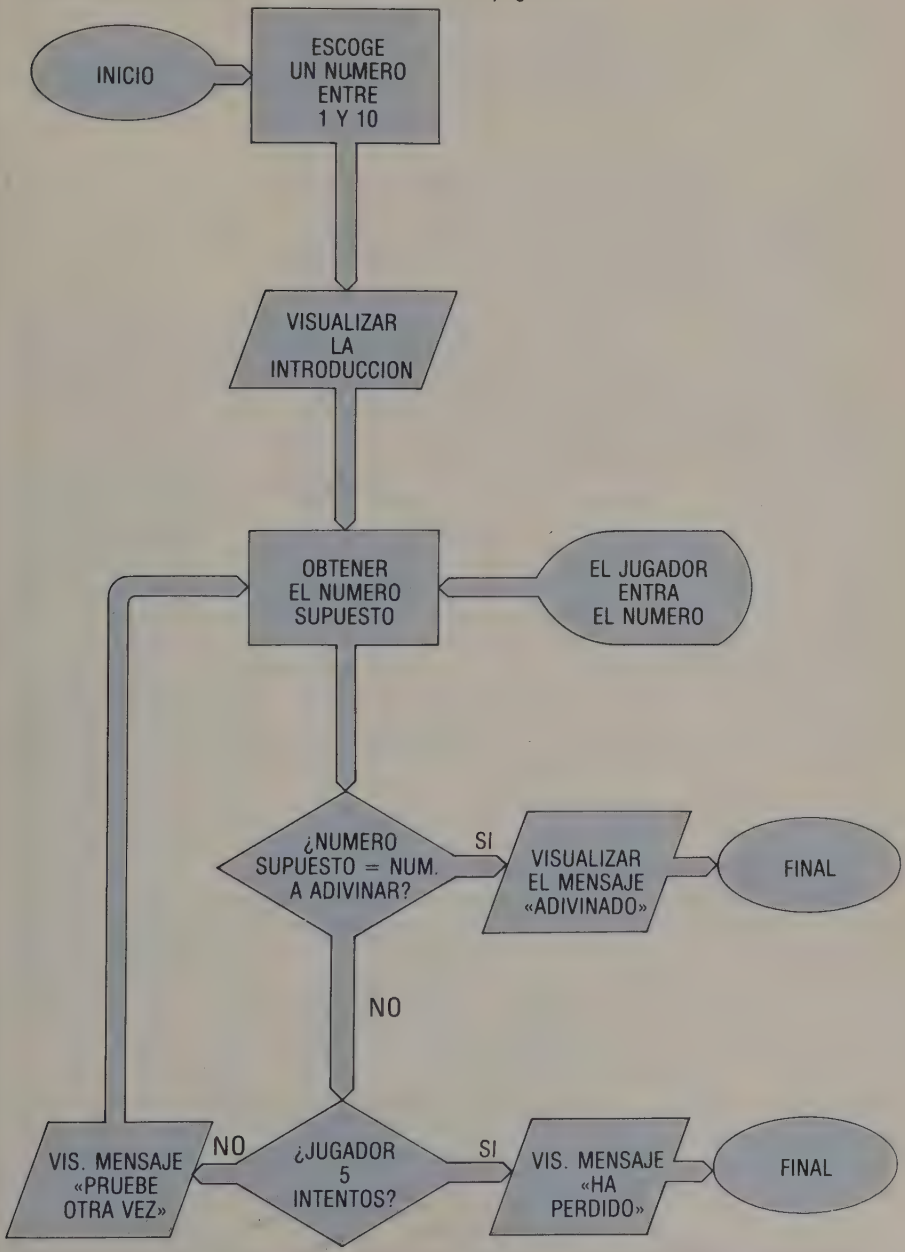
Este tiene una estructura bastante compleja, por lo que dibujaremos un diagrama de flujo para él. Nótese que hay dos finales distintos. Si el jugador no acierta la respuesta, entonces el programa sigue, y cuando sus cinco intentos se han terminado, ejecutará las líneas 140 a 160. Se detiene aquí, tal como se le dice. Si el jugador lo acierta correctamente, entonces las líneas 170 y 180 ofrecen un final alternativo.

Bucles encadenados

El uso de bucles llega a ser una herramienta todavía más potente cuando los encadenamos. Es decir, cuando colocamos bucles más pequeños dentro de otros más grandes. Sin embargo, no hay que mezclar los bucles. Hay que asegurarse de que abrimos *y* cerramos el pequeño dentro del grande.

Naturalmente, si estamos utilizando bucles con contador, entonces será necesario colocar dos contadores de bucle, y dar nombres de variable distintos a cada uno de ellos. He aquí un ejemplo:

Diagrama de flujo para «adivina el número», de la página 100.

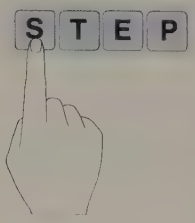


Pruebe esto:

```
10  REM RELOJ
20  PRINT "CONTARE DEL 1 AL 10"
30  PRINT "A INTERVALOS DE APROXIMADAMENTE 1 SE-
    GUNDO"
40  FOR N = 1 TO 10
50  PRINT N
60  FOR D = 0 TO 460
70  NEXT D
80  NEXT N
90  END
```

Puede usted ver cómo hemos utilizado nuestras indicaciones de bucle, en la parte izquierda del programa, como una comprobación para asegurar que nuestros dos bucles están encadenados adecuadamente.

El bucle interior, el «D», puede quizá confundirle. ¿En qué consiste? No, no se engaña usted. No hay nada en su interior. El ordenador tan sólo da vueltas sobre el bucle repetidamente desde la línea 60 hasta la línea 70 y de allí de nuevo hacia atrás, 460 veces. ¿Se acuerda de nuestro robot del capítulo 2? Allí utilizamos el tiempo que tarda el ordenador para ejecutar las órdenes como un «temporizador» para los parpadeos del nombre del robot. Esta vez utilizamos un «bucle vacío» como temporizador. En nuestro ordenador, el tiempo que tarda éste para ir a través del bucle D es alrededor de un segundo, —de ahí el título del programa—. Normalmente utilizaremos «D» como nuestra variable de contador del bucle en un «bucle de espera». De esta forma, reconocerá los bucles de espera siempre que los utilicemos.



Vamos a introducir un refinamiento más, y después utilizaremos el bucle «D» en otro programa. Se trata de la orden STEP. El STEP es una parte de la construcción FOR...NEXT. En los ejemplos que hemos dado hasta ahora, hemos utilizado el FOR para que el ordenador cuente hacia adelante de uno en uno, 1, 2, 3, etc. De hecho, estaba contando en escalones de 1.

Pero de hecho podemos decirle al ordenador que cuente en escalones de cualquier tamaño —números enteros o decimales, más o menos— añadiéndole la orden STEP. Funciona así:

Pruebe esto:

```
10  REM RELOJ 2
20  FOR N = 0 TO 60 STEP 5
30  PRINT N
40  FOR D = 0 TO 460*5: NEXT D
50  NEXT N
60  END
```

Esta vez el «reloj» funciona a la misma velocidad, pero tan sólo escribe un número cada cinco segundos. Y tendremos que modificar el bucle de espera de acuerdo con ello. No hay que preocuparse para calcular cuánto es 5 veces 460, ya que el ordenador puede hacer esto por nosotros.

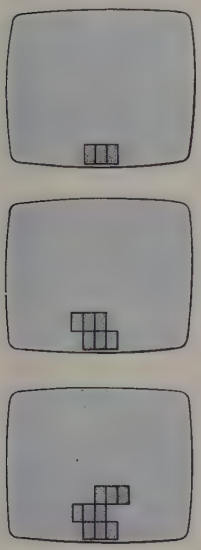
Esta vez, hemos escrito el bucle de espera todo en una línea. Así lo haremos normalmente, para enfatizar el hecho de que no hay nada en su interior.

He aquí un par de ejemplos más que utilizan el contador del bucle para otras funciones dentro del bucle:

Pruebe esto:

Serpenteo

Esta serpiente serpentea a través de la pantalla, avanzando una línea cada vez, en una posición aleatoria.



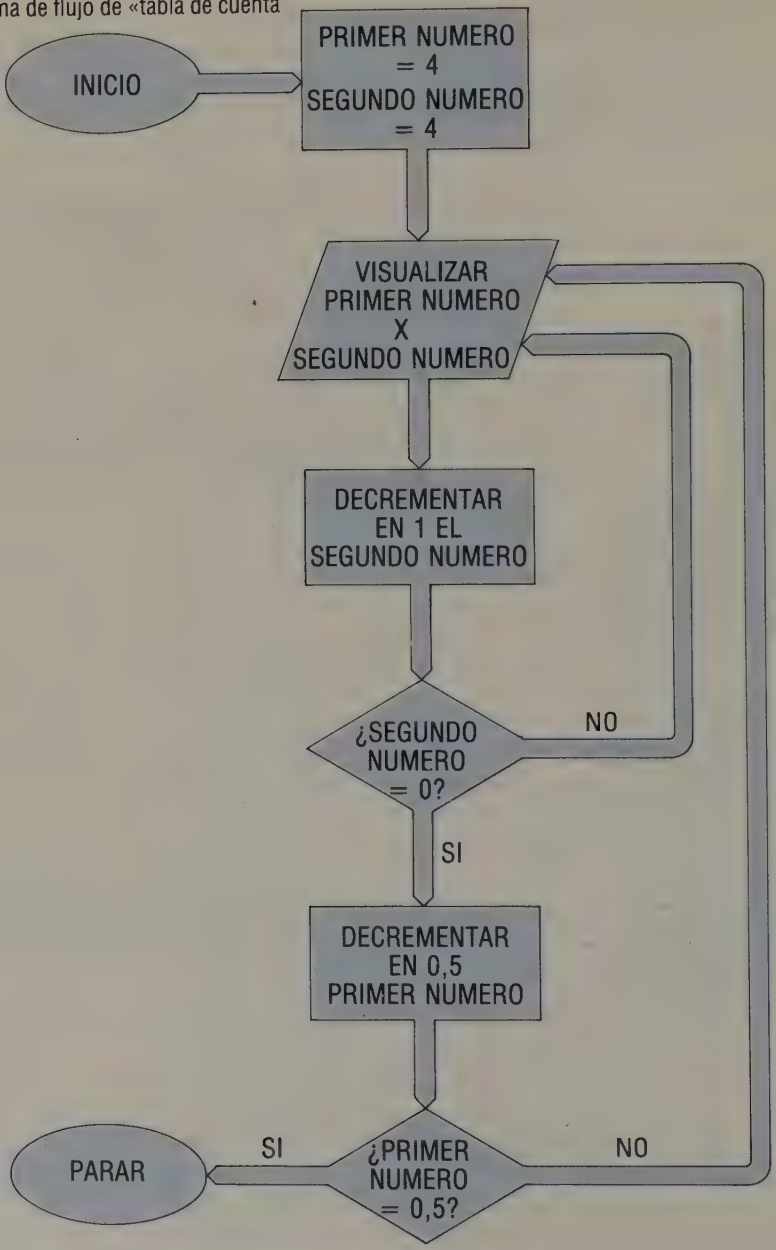
```
10   REM SERPENTEO
15   CLS
20   LET C = 15
30   FOR R = 15 TO 0 STEP -1
40   PRINT@ (R,C), CHR$(128)+CHR$(128)+CHR$(128);
50   LET C = C -2 + INT(RND*5)
60   IF C < 0 THEN LET C = 0
70   IF C > 29 THEN LET C = 29
80   FOR D = 0 TO 500: NEXT D
90   NEXT R
100  GOTO 100
```

Esta es una serpiente muy superior. Adviértase que si se quiere contar hacia atrás en una secuencia, hay que utilizar la orden STEP con un signo menos.

Pruebe esto:

```
10  REM TABLA DE CUENTA ATRAS
15  CLS
20  FOR N = 4 TO 1 STEP -0.5
30  FOR M = 4 TO 1 STEP -1
40  PRINT N;"X";M;"="; N*M,
50  NEXT M
60  NEXT N
70  END
```

Diagrama de flujo de «tabla de cuenta atrás»



Ahora le toca a usted

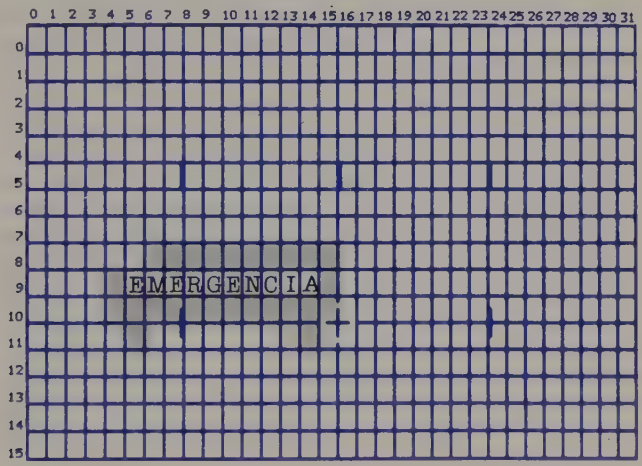
4) Dibuje una ambulancia en la pantalla con la palabra «emergencia» parpadeando lentamente a su lado.

5) Dígale al ordenador que escriba todos los códigos CHR$ que constituyen el conjunto de caracteres gráficos de su computador.

6) Programa de «parada de reloj». Haga que el ordenador le pregunte el número de segundos que tiene que contar, y entonces cuente los segundos uno a uno hasta que llegue al número que usted le ha dado. Después haga que le dé un mensaje de «se ha agotado el tiempo».

Ambulancia

He aquí la disposición sugerida para la forma de una ambulancia, con un signo que parpadea.



Comprobación

He aquí la lista de las materias que hemos cubierto en este capítulo. Ahora ya debe saber:

Sobre bucles

- Qué diferencia hay entre bucle cerrado y bucle abierto.
- Cómo puede salirse de un bucle mediante la cuenta de las repeticiones o mediante un punto de ramificación.

Sobre los diagramas de flujo
- Cuáles son los símbolos gráficos del diagrama de flujo, y cómo se utilizan.

Sobre la orden IF...THEN
- Cómo se utiliza para proporcionar un punto de decisión.
- Cómo se utiliza para abrir un bucle.

Sobre operadores lógicos
- Cuáles son los operadores lógidos del ordenador, y cómo se utilizan.

Sobre las órdenes FOR...NEXT
- Cómo se utilizan para temporizar un bucle.
- Cómo pueden encadenarse varios bucles.
- Cómo programar un bucle de espera.
- Cómo utilizar la orden STEP con la FOR...NEXT.

5 Edición y corrección de errores

En este capítulo veremos:
– Bugs (errores) en el programa y cómo arreglarlos.
– Posibilidades de edición en los ordenadores domésticos.
– Procesos de test.

Vamos a hacer una pausa en el proceso de escribir programas, en este capítulo, y a analizar otro tópico muy importante: cómo hacer que sus programas funcionen.

Si ha pasado a través de los capítulos 1 a 4, probando nuestros ejemplos y escribiendo programas en respuesta a nuestras sugerencias del apartado «Ahora le toca a usted», entonces casi seguro que ha tenido algunos problemas en hacer que sus programas funcionen; ¡a todos los programadores les sucede! En este capítulo le sugeriremos qué es lo que debe hacer para resolver estos problemas, e intentaremos explicarle lo importante que es el aprender a editar y depurar adecuadamente.

El editar un programa significa únicamente el corregirlo, cambiar, o añadir algo una vez que se ha entrado en el ordenador. Un «bug» es la palabra generalmente utilizada por la gente para indicar algo que impide al programa el hacer algo que debería hacer. Muy pocos «bugs» son culpa de su ordenador. Generalmente, es usted el que se ha equivocado; lo que sucede es que no ha escrito el programa que debería haber escrito. Cualquiera que sea la causa del error, deberá usted depurar el programa y hacer que funcione adecuadamente.

Si no tiene usted un ordenador, este capítulo puede parecerle sin importancia, pero de todas maneras léaselo entero, ya que los puntos que se discuten en él son importantes y le ayudarán a tener una idea más clara de lo que es realmente la programación.

La parte positiva de los problemas

En muchas actividades, los errores son algo que intenta olvidarse en seguida. Se suele deshacer uno de la evidencia, y se empieza de nuevo con una hoja nueva y limpia. En la programación no es así. No se puede pretender el ser un buen programador si sencillamente se pulsa el NEW cada vez que un programa no funciona. Hay que aprender de los errores, estudiando las listas de los programas, probando los cambios y desarrollando algunas estrategias para encontrar qué es lo que no funciona y corregirlo.

¿Ha cometido hasta ahora muchos errores, o los va a cometer en el futuro? Depende mucho de su estilo de trabajo. A algunos les gusta perfeccionar el programa antes de entrarlo en el ordenador. Después lo comprueban cuidadosamente, línea a línea, antes de intentar ejecutarlo. Todavía encontrarán que algunas veces el programa no hace lo que ellos pretendían que hiciera, pero esto probablemente no sucede muy a menudo. Otros prefieren intentarlo de una forma más rápida. Prueban el programa así que han escrito un par de líneas, y dejan que el ordenador les diga dónde están los errores.

¿Qué es mejor? Los dos sistemas tienen ventajas y desventajas. La mayoría de los ordenadores, incluso los más pequeños y baratos, tienen buenas características para señalar errores y para ayudarle a editar las líneas del programa de una forma cómoda. Se trata tan sólo de aprender a utilizarlas. No se puede dañar a un ordenador por el hecho de teclear una línea incorrecta de BASIC. En el peor de los casos se quedará sencillamente encallado en un bucle sin fin, que usted siempre puede romper, aunque sea desconectando el ordenador. También, en el aspecto positivo de esto, suele ser fácil el ver los errores cuando se ejecuta el programa; por ejemplo, una parte de una imagen en la pantalla puede aparecer equivocada o puede aparecer un mensaje de error. Sobre el papel, es difícil darse cuenta de si falta un punto y coma. Es una pérdida de tiempo el pasarse una hora comprobando el programa para encontrar algún signo de puntuación que falte, antes de ejecutarlo. El ordenador está allí para ayudarle, si usted le deja.

Por otra parte, es igualmente importante el asegurarse de que se ha planteado el programa de forma

adecuada. Mírelo de esta forma: si tratara usted de comprar un programa, ¿compraría uno en el que estuviera anunciado el que ha funcionado después de corregir 500 errores, o uno donde se le dice que ha funcionado a la primera? Probablemente sacará la conclusión de que si había 500 errores en el programa, hay todavía más que no han sido detectados. Sin embargo, si el programa ha funcionado a la primera, es lógico pensar que ha sido planteado de una forma lógica y cuidadosa, y la probabilidad de que hayan problemas cuando se ejecute es muy baja. Usted no querrá terminar con un programa que funciona y se aguante con el equivalente en programación de la cinta adhesiva, lo que usted quiere es un programa que esté bien diseñado, bien construido y que se aguante incluso con utilizaciones no demasiado cuidadosas.

Un aspecto importante para un planteo cuidadoso es el proceso de comprobación y depuración de su programa. Esto significa el suponer que su programa no funcionará, y que hay que hacer previsiones para poder arreglarlo. Esto nó es un consejo tan negativo como pueda parecer. Los mismos planteos serán de ayuda siempre que intente adaptar o mejorar un programa ya existente.

Estas son algunas indicaciones a grosso modo:

1) Divida el programa en pequeñas secciones. Generalmente, le será más fácil si busca puntos adecuados para dividir el programa. Estos serán los lugares en los que se empieza a hacer algo marcadamente distinto, como el empezar un cálculo complejo, dibujar una nueva forma, comprobar si un jugador ha conseguido un punto, etc. ¿Se acuerda de cómo dividimos el programa «casa» en el capítulo 2? Empiece cada nueva sección con un comentario aclaratorio (REM) y quizá con un número de línea bien distintivo.

2) Deje muchos números de línea libres para poder utilizarlos caso de insertarse líneas nuevas entre las ya existentes si se necesita. Esto es particularmente importante si su ordenador no tiene una orden de RENUM (renumeración); una orden muy útil que separa las líneas en un programa renumerándolas (por ejemplo, en un programa muy apretado, las líneas 10, 11, 12, 13, ... pueden ser RENUMeradas a 10, 20, 30, ... etcétera).

3) Compruebe el programa sección a sección, tanto como le sea posible. De esta manera podrá detectar posibles errores antes de ejecutar el programa. Por ejemplo, pruebe los dibujos en la pantalla antes de moverlos a través de ella, mediante procesos más complicados. Compruebe el programa básico del juego, cálculo o lo que sea, antes de colocar la introducción y las secciones finales. Se pueden mantener varias secciones del programa en la memoria al mismo tiempo, y probarlas una a una, utilizando sencillas técnicas para separarlas. Para probar las líneas 50 a 100 en un programa con números de línea que van desde la 10 a la 200, por ejemplo, se pueden colocar líneas tal como:

```
2    GOTO 50
101  END
```

Después se pueden sacar estas líneas fácil y rápidamente tecleando tan sólo los números de línea y dejando un espacio en blanco después de ellos, así:

```
2    {E}
101  {E}
```

Hemos tratado de darle algunas indicaciones generales sobre cómo planear un programa antes de ponerse a codificarlo en BASIC, tal como hacer una lista de variables, dibujar un diagrama de flujo, etc. Naturalmente, usted querrá experimentar hasta que encuentre un método de trabajo que le convenga. Pero cuando lo tenga, y todavía se encuentre con algunos problemas, ¿qué hará?

Esto depende en parte de la cantidad de ayuda que el computador le dé. Veamos tres posibilidades.

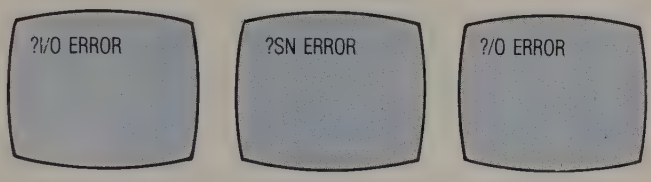
1. Intenta ejecutar un programa, y aparece un mensaje de error

Esto al menos no es un gran problema. El ordenador le indica que hay algo equivocado, y si tiene suerte, le dirá exactamente lo que es. Todo lo que tiene que hacer es mirar a donde le dice el ordenador y poner las cosas correctamente.

Los mensajes de error del ordenador varían de una forma muy amplia. Algunos ordenadores (como el

Mensajes de error

Los distintos ordenadores tienen varias maneras de indicar qué tipo de error se ha cometido. Estos son algunos de los mensajes de error más comunes, que aparecen en la pantalla de nuestro ordenador.



Error de entrada/salida | Error de sintaxis | División por cero

nuestro) le dan breves mensajes escritos como «SN Error» o «TM Error» que muy pronto conocerá y comprenderá. Algunos le dan únicamente códigos numéricos, tal como «Error 42». Tendrá que mirar en una tabla de su manual para ver lo que significa exactamente «Error 42» y allí habrá una nota que diga qué significa «Error de Sintaxis» o lo que sea. Algunos ordenadores tienen un amplio rango de mensajes distintos y otros tienen tan sólo unos pocos generales. La mayoría de los ordenadores suelen indicar la línea de su programa donde se han atascado. Algunos le dicen incluso qué sentencia es la que no les gusta, si hay varios errores en la línea.

Si el error está en el medio del programa, entonces deberá —por lo menos hasta que no tenga usted más experiencia— arreglarlo antes de poder utilizar el programa. Si no, el ordenador sencillamente se detendrá cada vez que llegue a la sentencia que no entiende. Si el error aparece al final, entonces puede, si lo desea, dejarlo, tal como hicimos en algunos de nuestros primeros programas. El programa se terminará torpemente, pero podrá ejecutarse perfectamente hasta que no llegue al final.

Estos son algunos de los mensajes de error más frecuentes que puede encontrarse, si no se los ha encontrado ya.

Utilización de una función ilegal. Esto puede aparecer, por ejemplo, si ha utilizado números fuera del rango permisible para describir una posición PRINT. El ordenador no puede escribir en la fila 17 de una pantalla de 16 filas. Solución: asegúrese de que su programa incluye una secuencia de comprobación, para ver si los números que van a una variable son aceptables, y haga la acción pertinente si no lo son. (Vea el

programa de «la pelota que rebota» de la páginas 94-96 como ejemplo de ideas para hacer esto.)

Error de Entrada/Salida. Esto le indica si un programa que estaba intentando cargar no se ha cargado correctamente, o si existe algún otro problema con un aparato de entrada (por ejemplo, el teclado) o un aparato de salida (por ejemplo, la pantalla). Muchos interfases de cassette (la circuitería que permite al ordenador utilizar el reproductor de cassettes) son extremadamente temperamentales. Quizá tenga que intentar varias veces, colocando el control de volumen en posiciones ligeramente distintas, antes de que el programa se cargue correctamente. Naturalmente, compruebe que está utilizando el lado correcto, de la cinta correcta. Y haga dos copias de todos los programas importantes, de forma que si una copia está en mal estado —esto significa que nunca funcionará porque se han perdido los datos— pueda utilizar la otra.

Un Next sin el For. Ha cruzado sus bucles encadenados. Compruebe que todos los NEXT y FOR coinciden, utilizando marcas tales como las que hemos utilizado en el margen izquierdo de nuestros programas del capítulo cuarto.

Error de Sintaxis. Un aviso muy general de que hay algo incorrecto en la forma en que ha escrito sus sentencias de BASIC. Por ejemplo:

- Puede haberse equivocado en una orden y poner por ejemplo «PRITN» en lugar de «PRINT».
- Puede haberse olvidado de una pieza vital de la puntuación, como una coma que separe dos sentencias en la misma línea.

Error de tipo de variable. Ha intentado colocar un texto en una variable numérica, o viceversa, ya sea directamente, o en una sentencia que modificase el contenido de una variable. Por ejemplo esto sería un error:

```
LET C$ = T + 1
```

(C$ es un nombre de variable de texto, y T es nombre de variable numérica). Debe arreglar esto cambiando el nombre de su variable, o su sentencia, de forma que las variables numéricas y de texto se usen correctamente. A veces su equivocación habrá sido al teclear. Otras veces, más seriamente, habrá planeado hacer

algo que el ordenador no le permite hacer. En este caso, deberá encontrar que una de las funciones de manipulación de textos de su versión del BASIC (éstas pueden utilizarse para cambiar variables numéricas por variables de texto, etc.) le ayudará a encontrar la solución de su problema. Para estas funciones, consulte su manual.

Línea no-definida. Esto aparece cuando se intenta ir (GOTO) a un número de línea que de hecho no ha sido utilizada en su programa. Solución: cambie la sentencia GOTO.

/O (División por cero). Ha intentado dividir un número por cero, quizá cuando estaba utilizando una secuencia de números que incluían el cero en una variable. Solución: cambie su secuencia de, digamos 0 a 10, por una secuencia de 1 a 11.

2. Intenta ejecutar el programa, y su salida es incorrecta

El ordenador no comprueba si usted le está diciendo que haga algo determinado, tan sólo comprueba para asegurarse de que le está diciendo que haga algo que sabe hacer. Por lo tanto, en muchas ocasiones el programa se ejecutará sin ninguna protesta por parte del ordenador, pero no hará lo que usted pretendía que hiciera. Estos son el tipo de problemas que pueden aparecer:

– No ha borrado las posiciones anteriores cuando estaba dibujando un objeto en movimiento, con lo que obtiene una fila, por ejemplo, de pequeños trozos de coche, todas a través de la pantalla, en lugar de tener un coche que se mueve.
– Ha confundido los números de fila y de columna, o ha escrito los códigos de CHR$ incorrectos, por lo que la imagen en la pantalla aparece equivocada.
– Se ha olvidado de un paréntesis vital en una línea de complejas expresiones matemáticas, por lo que el ordenador no ha hecho los cálculos que usted pretendía que hiciera.
– Se ha equivocado al escribir el nombre de una variable, por lo que el ordenador ha supuesto que se trataba de una nueva variable y le ha dado el valor 0.

En estos casos, le corresponde a usted el repasar las sentencias que han causado el problema. Le ayudará

el hecho de haber documentado bien su programa, colocando muchas sentencias de REM. Analizaremos esto con más profundidad en el capítulo 7.

3. Intenta ejecutar el programa y no sucede nada

Esto es lo más duro de todo, ya que el ordenador le puede ayudar muy poco a descubrir el problema. Apriete la tecla BREAK, y mire dónde se detiene el programa. Obtendrá probablemente un mensaje tal como:

BREAK IN LINE 60

que le dice lo que el ordenador estaba haciendo cuando se le ordenó parar. Esto puede darle una pista. Si la tecla BREAK no funciona, busque en su manual las instrucciones para reiniciar su sistema. En muchos ordenadores puede hacer esto sin perder el listado de su programa. Si todo esto también falla, puede tirar del enchufe, pero entonces se perderá toda la información que había en la máquina.

¿Por qué es posible que no suceda nada? Es posible que, sencillamente, se haya olvidado de decirle al ordenador que imprima los resultados. (En este caso, habrá obtenido el mensaje de «OK», que le dice que el ordenador ha hecho todo lo que se le dijo que hiciera; lo que sucede es que se le dijo que hiciera las cosas mal.) Lo más frecuente es que el ordenador se haya perdido en un bucle sin fin, haciendo algo repetidamente sin producir ninguna salida.

¿Cómo puede descubrirse este problema? Una manera es el colocar algunas líneas de traza. Si pretendía que el ordenador visualizara un pequeño hombre en las líneas 500 a 600, por ejemplo, entonces coloque algunas líneas de programa extras, como:

```
501  PRINT "LINEA 501"
521  PRINT "LINEA 521"
```

De esta forma, sabrá si el ordenador va realmente a la línea 500, a la línea 520, o donde sea. Quizá se haya olvidado un GOTO, o ha estructurado su programa pobremente o de una forma equivocada, por ejemplo, no encadenando los bucles adecuadamente o confun-

diendo sus estructuras de bucle, por lo que el ordenador no consigue llegar allí.

Descubrirá que las técnicas de que hablamos anteriormente para dividir su programa en secciones, son de utilidad aquí. Utilice GOTOs y ENDs u otras órdenes similares de su versión de BASIC para comprobar las secciones una a una, o para saltar una sección que usted sospecha que pueda causar problemas, y ver si el resto del programa funciona.

A muchos programadores les gusta utilizar números de línea distintivos en el programa para estas sentencias de depuración. Si las colocan todas en números tales como 29, 39, 49, etc., después podrán sacarlas otra vez con un mínimo de confusiones, una vez se haya resuelto el problema.

Técnicas de edición

Los ordenadores varían en sus características de edición incluso más ampliamente de lo que varían en sus mensajes de error. Algunos, por ejemplo, cuando han encontrado un error en una línea, automáticamente colocan el cursor en la línea, dejándola preparada para la edición y corrección. Unos tienen editores que controlan la totalidad de la pantalla y en los cuales se puede mover el cursor a través de la totalidad del listado hasta llegar al lugar adecuado, y entonces hacer las correcciones. Otros tienen tan solo editores de línea en los cuales hay que entrar una orden tal como:

EDIT 50

antes de poder alterar la línea 50. Si su ordenador tiene una orden de EDIT, busque en su manual una explicación de cómo puede utilizarse para hacer correcciones.

En algunos ordenadores es muy fácil cambiar el texto de una línea que ha sido ya entrada mediante la utilización de órdenes especiales de edición y de movimientos del cursor. En otros, no hay otra alternativa que el reescribir la línea completa. Siempre se puede reescribir la línea en cualquier momento, naturalmente; la nueva versión sustituye sencillamente a la antigua.

Si su ordenador tiene características de edición de línea pobres, intente mantener las líneas del programa

tan cortas como sea posible, con tan sólo una sentencia por línea, sin demasiados caracteres. De esta forma, la acción de reescribirla se restringe a un mínimo. Si quiere deshacerse de una única línea, puede entrar una «línea en blanco» reescribiendo tan sólo un número de línea y después apretando el ENTER.

Algunos ordenadores tienen órdenes de DELETE (borrar) que le permiten suprimir una sección completa de un programa con una sola acción.

Si tiene mucha edición que hacer, entonces tenga esto presente; aunque su programa no funcione, podrá siempre guardarse en una cinta. De esta manera, puede tomarse un descanso a mitad de la edición y reemprenderla más tarde. Si está haciendo muchas cosas sobre un programa largo, querrá guardar el listado regularmente, para prevenir el que algo vaya mal y se pierda la versión que está actualmente en el ordenador.

Hay que saber cuándo se está vencido

Se puede cambiar un programa hasta que sea irreconocible a base de editarlo después de entrarlo en el ordenador. No obstante, por más que lo edite no podrá salvarse cuando se ha hecho un error básico en el planteo del programa.

Cuando aparezca un mensaje de error deténgase y piense, no se limite a hacer cambio tras cambio para ver si acierta con uno que funcione. A veces sabrá exactamente qué hay que hacer para que funcione el programa. Otras veces se dará cuenta de que realmente no hay ninguna manera de hacer que funcione y de que hasta ahora estaba siguiendo un camino equivocado y que debe prescindir de lo que ha hecho hasta ahora y empezar de nuevo.

No todos los ordenadores pueden hacer todos los trabajos. A veces se encontrará con que el ordenador no puede hacer lo que usted quería que hiciera. Posiblemente sea debido a sus limitaciones de hardware, por ejemplo, por no tener suficiente memoria o no tener los caracteres que usted quería usar. Si es debido a las limitaciones de su dialecto de BASIC, entonces la solución puede ser utilizar un lenguaje de programación distinto (si es que esto es posible en su máquina). Veremos esta cuestión en el capítulo 8.

Probarlo también es importante

Si ha tenido dificultad en conseguir que funcione un programa, entonces es un gran alivio cuando finalmente hace lo que usted quería que hiciese. Sin embargo, no ha terminado todavía. Debe usted asegurarse de que funciona, no sólo esta vez, sino siempre.

¿Puede un programa funcionar una vez y no a la siguiente? Ciertamente. Tiene que comprobar estos puntos, por ejemplo:

1. ¿Funcionará cuando entre usted datos diferentes?

Si da usted a los usuarios la oportunidad de usar los números desde el 0 al 10, por ejemplo, ¿funcionará el programa cuando ellos entren un 0? ¿Y un 10? ¿Y cualquier otro número? Si da usted la oportunidad a los usuarios de entrar su nombre, ¿funcionará en el caso de que tengan nombres muy largos? Pruebe varias entradas posibles, o todas las posibilidades si ha fijado usted una lista de ellas. Si está usted moviendo objetos en la pantalla, ¿sucederá algo raro cuando estos objetos lleguen a los extremos de la pantalla, o dejará de funcionar el programa? Si no ha prevenido a los usuarios de que utilicen números enteros y no decimales, ¿funcionarán entonces los decimales?

2. ¿Qué sucederá cuando se entre un dato equivocado?

Hasta ahora no hemos hecho mucho para incluir comprobaciones dentro de nuestros programas. Si hemos planteado el programa de forma que el usuario entre un número del 1 al 10, entonces hemos supuesto que el usuario hará exactamente esto. En la vida real es aconsejable poner algún tipo de rutina que compruebe si el usuario escribe la cosa correcta y eliminar la situación caso de que no lo haga.

Hay muchas técnicas para construir comprobaciones, pero muchas de ellas son demasiado complicadas para nosotros para que las veamos ahora. He aquí un par de sencillos ejemplos, para que se haga una idea:

```
40  PRINT "ESCOGE UN NUMERO ENTRE 5 y 50"
50  INPUT N
60  IF N<5 THEN GOTO 40
70  IF N>50 THEN GOTO 40
```

Este es suficientemente obvio. Se le pide al usuario de nuevo que entre el dato, si él o ella da un número que es demasiado grande o demasiado pequeño.

```
100  INPUT "QUIERES JUGAR OTRA VEZ? S/N";R$
110  IF R$="S" THEN GOTO 30
120  IF R$="SI" THEN GOTO 30
130  END
```

Estamos pidiendo una sencilla respuesta, es decir, «S», pero el programa está escrito de forma que el «SI», entrado completamente, también producirá una respuesta. Técnicas más sofisticadas pueden, por ejemplo, aceptar cualquier palabra que empiece por «S» o «s» como significado de «si».

Otras técnicas de comprobación pueden asegurar, por ejemplo, que el usuario da exactamente el número correcto de letras o de números; una fecha que sea posible (digamos, en el año o siglo correcto, en función de lo que haga el programa); un número entero en lugar de un número decimal; etc.

3. ¿Funcionan todos los bucles distintos?

De nuevo, tomemos un programa de juegos como ejemplo. Sólo por el hecho de que la secuencia «ha perdido» funcione, no significa que la secuencia «ha ganado» también lo haga. ¿Y qué sucede con la secuencia de «volver a jugar»? ¿Vuelve atrás hasta el lugar adecuado, dentro del juego? ¿Todas las variables han sido reinicializadas a sus valores iniciales correctos, listas para volver a jugar? ¿Mantiene el programa la puntuación anterior correctamente, al iniciarse de nuevo el juego?

Naturalmente estos procesos de comprobación no son tan divertidos como el empezar a escribir un programa. Son un «hueso», pero un «hueso» necesario. Si usted quiere escribir programas que quiere seguir utilizando, y que su familia y amigos, y posiblemente otra gente, puedan también utilizar, entonces debe acostumbrarse a estas complicaciones. Una vez se familiarice con ellas, habrá también otra ventaja, sabrá cómo ponerse a utilizar, y a modificar, los listados de programas que se encuentran en libros y revistas. Desgraciadamente, algunos de ellos tienen errores producidos

ya sea por las pocas comprobaciones del autor o por deslices a la hora de mecanografiarlos. La mayoría de los más largos, cuando los haya entrado en su computador, tendrán también errores causados por su mecanografiado. Si va usted a utilizarlos necesitará aprender cómo localizar estos errores y cómo corregirlos.

En el capítulo 7 veremos en detalle cómo escribir dos programas más largos. Volveremos otra vez sobre algunas de estas cuestiones, y a analizar el tipo de errores que pueda cometer y algunas de las posibilidades para corregirlos.

Comprobación

Cuando haya terminado este capítulo deberá saber:

Sobre debugging (corrección)
- Cómo son los mensajes de error de su ordenador, y cómo son las explicaciones sobre ellos en su manual.
- Cómo utilizar sencillas técnicas para detectar y eliminar los errores de su programa.

Sobre edición
- Qué tipo de facilidades de edición le ofrece su ordenador, y cómo utilizarlas.
- Sobre las órdenes EDIT y RENUM, si es que existen en su ordenador.

Sobre comprobaciones
- Por qué la comprobación completa de un programa es importante y cómo llevar a cabo comprobaciones distintas.

6 Cómo manejar los datos

En este capítulo veremos:
- **Conjunto de datos de una dimensión.**
- **La orden DIM.**
- **Las órdenes READ...DATA.**

Ya hemos visto cómo se utilizan las variables en el BASIC para el manejo de datos. Ahora vamos a ver algunas maneras por las que podemos manejar muchos más datos de los que hemos manejado hasta ahora. Esto nos permitirá colocar en un programa, o entrar mientras el programa se está ejecutando, un pequeño archivo de información que el programa pueda utilizar.

Si queremos utilizar un fragmento de información cada vez, entonces podemos —como ya sabe ahora— utilizar un único nombre de variable, y darle un contenido tras otro. Como recordatorio, veamos un pequeño programa que hace tan sólo esto:

Pruebe esto:

```
10  REM FAHRENHEIT A CENTIGRADO
20  PRINT "DAME UNA TEMPERATURA EN FAHRENHEIT"
30  PRINT "Y YO TE DARE SU"
40  PRINT "EQUIVALENTE EN CENTIGRADO"
50  PRINT "ENTRA UN 0 PARA TERMINAR"
60  INPUT "FAHRENHEIT";F
70  IF F = 0 THEN GOTO 110
80  PRINT "EL EQUIVALENTE EN CENTIGRADO DE";F;
90  PRINT "ES";(F - 32)*5/9
100 GOTO 60
110 END
```

Tan sólo utilizamos una variable, F, para tantas temperaturas Fahrenheit como el usuario quiera entrar.

Nótese el que esta vez hemos hecho que el programa termine de una forma correcta. Hemos colocado un «valor muerto» —un valor que el usuario normalmente no daría— y esto proporciona un punto de decisión, que nos permite romper el bucle de repetición y terminar el programa.

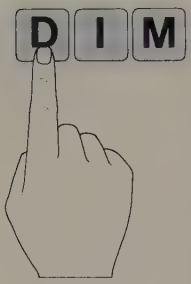
Este método con las variables es correcto si queremos que el programa nos permita entrar uno o dos datos, entonces hacer algo con ellos, y después que nos permita entrar alguno más. Sin embargo, no funciona si queremos entrar muchos datos y después que el programa los utilice todos a la vez, que es lo que solemos querer que se haga en un programa, por ejemplo para clasificar una lista. Naturalmente, podemos colocar una larga lista de posibles nombres de variables en un programa —A, B, C, D, etc.— y hacer que el ordenador las utilice una a una. Sin embargo, de esta manera no es fácil el programar un bucle. Por lo tanto, ¿cómo podemos hacer un bucle? Utilizando nombres de variables que estén enlazadas lógicamente y diseñadas para usarlas conjuntamente.

Tomemos un ejemplo cotidiano: los días de la semana. Podemos colocarlos en una lista, que llamaremos «día», y entonces describir cada artículo de la lista mediante «un número de día» —es decir, que el día número uno es el lunes, y el día número cuatro es el jueves, y así sucesivamente—. Podemos utilizar «D$» para guardar el «día» en nuestro programa, y después seguir utilizando «N» para cambiar el valor de «número».

He aquí un programa de ejemplo:

Pruebe esto:

```
10  REM ***INTRODUCCION DE VARIABLES DE LISTA***
20  DIM A$ (5)
30  PRINT "DAME CINCO ARTICULOS"
40  PRINT "Y YO LOS PONDRE EN UNA LISTA"
50  FOR N = 1 TO 5
60  INPUT "ARTICULO:";A$(N)
70  NEXT N
80  REM HE AQUI LA LISTA
90  FOR N = 1 TO 5
100 PRINT "ARTICULO";N,A$(N)
110 NEXT N
120 END
```



Este programa ha introducido una orden nueva. ¿Se ha dado cuenta de ella, en la línea 20? Se trata de la orden DIM. El «DIM» viene de DIMensión. Le estamos diciendo al ordenador la DIMensión de un conjunto de posiciones de variables —en otras palabras, cuántas posiciones de variables hay en este conjunto—. Llamaremos al conjunto de estas posiciones variable de lista.

Observará usted que estas posiciones de variables tienen nombres muy parecidos a los que hemos utilizado para las variables. Ponemos el signo del dólar después del nombre, para las variables de texto, exactamente de la misma forma que hemos venido haciendo. Pero esta vez, después del nombre de la variable irá a continuación un número entre paréntesis.

Las variables de la lista tienen todas números distintivos, de forma que tienen nombres individuales como:

A$(1)
A$(2)
A$(3)

y así sucesivamente, hasta el valor máximo que escojamos utilizar — A$(5), en nuestro ejemplo.

La orden DIM le dice al ordenador cuál es el valor máximo, para que el computador pueda reservar el espacio suficiente en su memoria para guardar toda la información que contendrá la lista. A veces, en algunos BASIC el ordenador hará esto automáticamente. Sin embargo, nuestro ordenador espera encontrar una orden DIM cada vez que colocamos una lista.

Nuestro ordenador empieza a numerar las listas desde 1 (es decir, que los números son 1, 2, 3, etc.). Otros ordenadores empiezan desde 0, pero no hay necesidad de cambiar nuestros programa si el suyo así lo hace, ya que puede dejar la variable «0» vacía.

Utilizamos listas siempre que tenemos un conjunto de datos que vienen juntos de la misma forma que los sucesivos contenidos de una variable sencilla, y que queremos utilizar juntos en un programa. Es decir, que podemos, como ya hemos visto, colocar una lista que contenga como datos los meses del año, así:

```
LET D$(1) = "ENERO"
LET D$(2) = "FEBRERO"
```

Sin embargo, el verdadero punto interesante de listas como éstas es que se puede utilizar otra variable que contenga el número de orden de la lista. En nuestro ejemplo hemos utilizado N. Por lo tanto «N» es realmente una variable completamente separada, que toma —como puede hacerlo en cualquier circunstancia— los valores sucesivos de 1, 2, 3, etc., cada vez que damos una vuelta sobre el bucle y entramos un valor diferente en la lista.

Naturalmente, es más económico hacerlo como hemos hecho y utilizar la misma variable para ambas cosas, para contar el bucle, utilizando órdenes FOR...NEXT, y para contener el número de la variable de nuestra lista. Y una vez hemos utilizado un bucle para entrar todos los datos en nuestra lista, utilizaremos otro para escribirlos todos de nuevo. Para mantener el programa en una forma sencilla, escribirlos es todo lo que haremos, pero naturalmente podíamos haber hecho cosas mucho más elaboradas con ellos, una vez que estuvieran todos guardados de una forma correcta en la memoria del ordenador.

Probemos otro ejemplo. Esta vez utilizaremos un bucle para entrar en el computador una serie de posiciones en las que queremos visualizar un bloque en la pantalla —o mejor dicho, dos series de posiciones, una para las filas y otra para las columnas—. Estableceremos dos listas, R y C, que contendrán los datos que entraremos.

Hay que decirle al ordenador cuántos artículos queremos colocar en cada lista, pero esta vez el número de artículos puede variar en función del número de bloques que queramos en nuestro dibujo. Por lo tanto, vamos a permitir un máximo de 50 conjuntos de posiciones de visualización. Si no queremos utilizar tantas, siempre podemos dejar algunas de las variables de la lista vacías.

Esto nos dará una línea tal como:

DIM R(50): DIM C(50)

(No hay que poner el signo $ esta vez, ya que entramos números.)

Ya que no sabemos por adelantado cuántos artículos habrá, y por lo tanto cuántas veces el programa ejecutará el bucle, no utilizaremos el FOR...NEXT, sino el IF...THEN. De hecho, existirán dos condiciones que pueden hacer que queramos romper el bucle. Primero, podemos haber terminado de entrar los datos, lo que indicaremos dando un «valor muerto», tal como hicimos en el ejemplo anterior de «Fahrenheit a centígrado»; o segundo, podemos haber llenado completamente nuestra lista. Podemos comprobar fácilmente ambas condiciones.

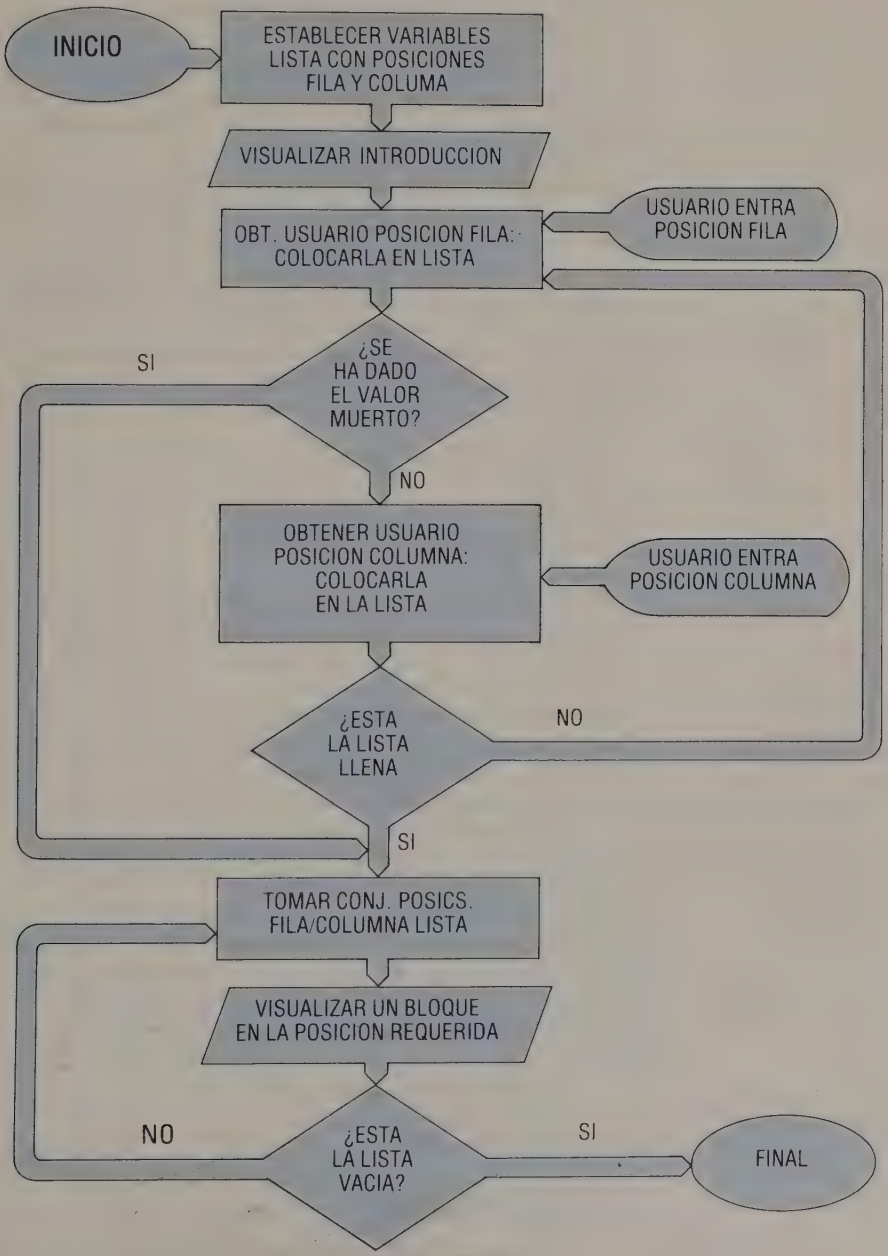
Ya que no tenemos un contador de bucle, esta vez tendremos que colocar una variable —N— que nos diga el número del elemento de la lista que estamos llenando cada vez que entramos los números de fila y columna y utilizaremos esta variable para decirle al ordenador cuándo debe detenerse. Tendremos que acordarnos de incrementar esta variable en una unidad cada vez que ejecutemos el bucle.

He aquí el programa; el diagrama de flujo correspondiente está en la página siguiente.

Pruebe esto:

```
10   REM ***HACER UN DIBUJO***
20   DIM R(50): DIM C(50): LET N = 1
30   PRINT "DIME DONDE PONGO BLOQUES"
40   PRINT "EN LA PANTALLA Y CONSTRUIRE"
45   PRINT "UN DIBUJO PARA TI"
50   PRINT "(MAXIMO 50 BLOQUES)"
60   PRINT "ENTRA UN 1000 CUANDO HAYAS TERMI-
     NADO"
70   REM ENSAMBLAJE DE LOS DATOS ENTRADOS
80   INPUT "POSICION DE FILA";R(N)
90   IF R(N) = 1000 THEN GOTO 200
100  INPUT "POSICION DE COLUMNA";C(N)
110  LET N = N + 1
120  IF N = 51 THEN GOTO 200
130  GOTO 80
200  REM REALIZACION DEL DIBUJO
210  CLS
220  FOR M = 1 TO (N-1)
230  PRINT@ (R(M),C(M)),CHR$(128);
240  NEXT M
250  GOTO 250
```

Diagrama de flujo de «relación de un dibujo», de la página 124



Hay dos secciones principales en el programa. La primera sección construye un archivo de posiciones de visualización que queremos utilizar, y la segunda visualiza los bloques en cada una de ellas. Siguiendo las reglas que discutimos en el capítulo anterior, los hemos separado claramente con sentencias de REM y un número de línea bien distintivo.

Nótese que esta vez no podíamos utilizar el «0» como valor muerto, porque quizá queramos visualizar un bloque en la posición (0,0). Por lo tanto hemos utilizado el 1000, que está fuera del rango permisible de posiciones de visualización. Naturalmente, también podíamos haberlo comprobado —como vimos en el capítulo anterior— para asegurarnos de que el usuario entra posiciones de visualización aceptables cada vez. Para hacerlo, podríamos colocar líneas tal como:

```
92   IF R(N) <0 THEN GOTO 80
94   IF R(N) >15 THEN GOTO 80
102  IF C(N) <0 THEN GOTO 100
104  IF C(N) >31 THEN GOTO 100
```

Al volver atrás hasta la línea de INPUT, reescribiríamos sobre la información incorrecta en la posición de la variable, por lo que no es necesario el eliminarlo de cualquier otra forma.

Ya que la idea de variables de lista puede ser nueva y confusa para usted, veamos cómo se ejecuta este programa, y qué es lo que hace. Quizás encuentre también de ayuda el seguir las líneas del programa en conjunción con el diagrama de flujo, lo que puede ayudarle a ver lo que estas líneas pretenden estar haciendo.

Naturalmente, primero, el ordenador escribirá el mensaje de introducción, diciéndonos qué es lo que espera que le digamos. Entonces nos pedirá alguna entrada. Vamos a decirle que visualice tres bloques en el medio de la pantalla, en las posiciones (8, 10), (8, 11) y (8, 12). Esto nos dará un intercambio con el ordenador tal como:

El ordenador	*El usuario*
POSICION DE FILA?	*8*
POSICION DE COLUMNA?	*10*

POSICION DE FILA? *8*
POSICION DE COLUMNA? *11*
POSICION DE FILA? *8*
POSICION DE COLUMNA? *12*
POSICION DE FILA? *1000*

A la cuarta vez que se ejecute el bucle, entramos el «valor muerto» y el ordenador irá a ejecutar la segunda parte del programa.

En la memoria del ordenador, habrá una lista de datos tal como:

	Posición	Contenido
Variable de lista R:	R(1)	8
	R(2)	8
	R(3)	8
	R(4)	1000
	R(5 a 50)	0
Variable de lista C:	C(1)	10
	C(2)	11
	C(3)	12
	C(4 a 50)	0
Variable N:		4

Nótese que N es ahora *mayor* en una unidad que el número de bloques que queremos imprimir, ya que la hemos utilizado también para el «valor muerto». Así, en la segunda parte del programa —en la línea 220— le decimos al ordenador que utilice los datos de la lista que están en las posiciones con valores inferiores a N–1. Ya que necesitamos guardar el valor de N —no queremos cambiarlo— utilizaremos una variable distinta, M, como contador de bucle en esta segunda mitad del programa. El ordenador ejecuta el bucle tres veces, y cada vez toma las coordenadas de fila y columna de una posición distinta en las dos listas R y C.

¿Ha cogido esto? Esperamos que sí, porque:

Ahora le toca a usted

1) Escriba un programa que le permita entrar cuatro números, y después escríbalos en columna en el medio de la pantalla.

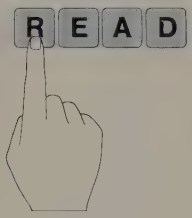
2) Escriba un programa que le permita entrar hasta

veinte códigos de CHR$, y después los visualice en una fila en el medio de la pantalla.

3) Escriba un programa que le permita entrar cinco nombres, y después los escriba en orden invertido.

Si todavía se siente ligeramente confuso con las variables de lista, consulte nuestras respuestas en la página 204 y examine cuidadosamente los programas que allí sugerimos, analizando qué es lo que sucede en cada línea. Esto debe acostumbrarle al uso de las listas.

Lectura de datos



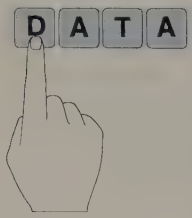
Acabamos de ver cómo se utiliza la sentencia de INPUT, en bucles repetitivos, para poner datos en las variables de lista.

Existe otro método que utilizamos cuando queremos poner datos permanentemente en un programa, en lugar de tener que entrar datos distintos cada vez que se ejecuta el programa. Esto puede interpretarse como el equivalente a una lista de órdenes LET.

Naturalmente, las órdenes LET pueden usarse para poner datos dentro de la lista. Pero ya que tan sólo se puede poner un solo dato, en una posición cada vez, se tendrían que poner una gran cantidad de LETs dentro del programa, así:

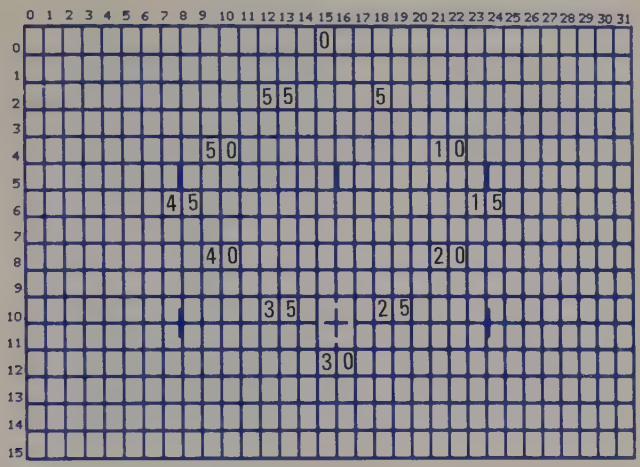
```
LET A(1) = 20
LET A(2) = 30
LET A(3) = 40
```

para leer datos y colocarlos en las sucesivas posiciones de una lista.



El método que utilizamos tiene una estructura de orden nueva: READ...DATA. Utilizamos el READ en lugar del LET para decirle al ordenador en qué posiciones de la variable queremos colocar el dato, y después utilizamos el DATA en lugar del =, para decirle qué es lo que va a ir en estas posiciones. El ejemplo siguiente le muestra cómo funciona. Vamos a hacer que el ordenador actúe como un reloj, visualizando los números del 0 al 60, siguiendo la forma de un círculo que se parezca a la esfera de un reloj. Tardará un minuto en completar la figura. Los datos que entraremos en el programa son los números de fila y columna que especifican las 13 posiciones (12 distintas y una que se

Reloj
Esta es la disposición del programa de las páginas 129-131. Los números irán apareciendo uno a uno, a intervalos de 5 segundos.



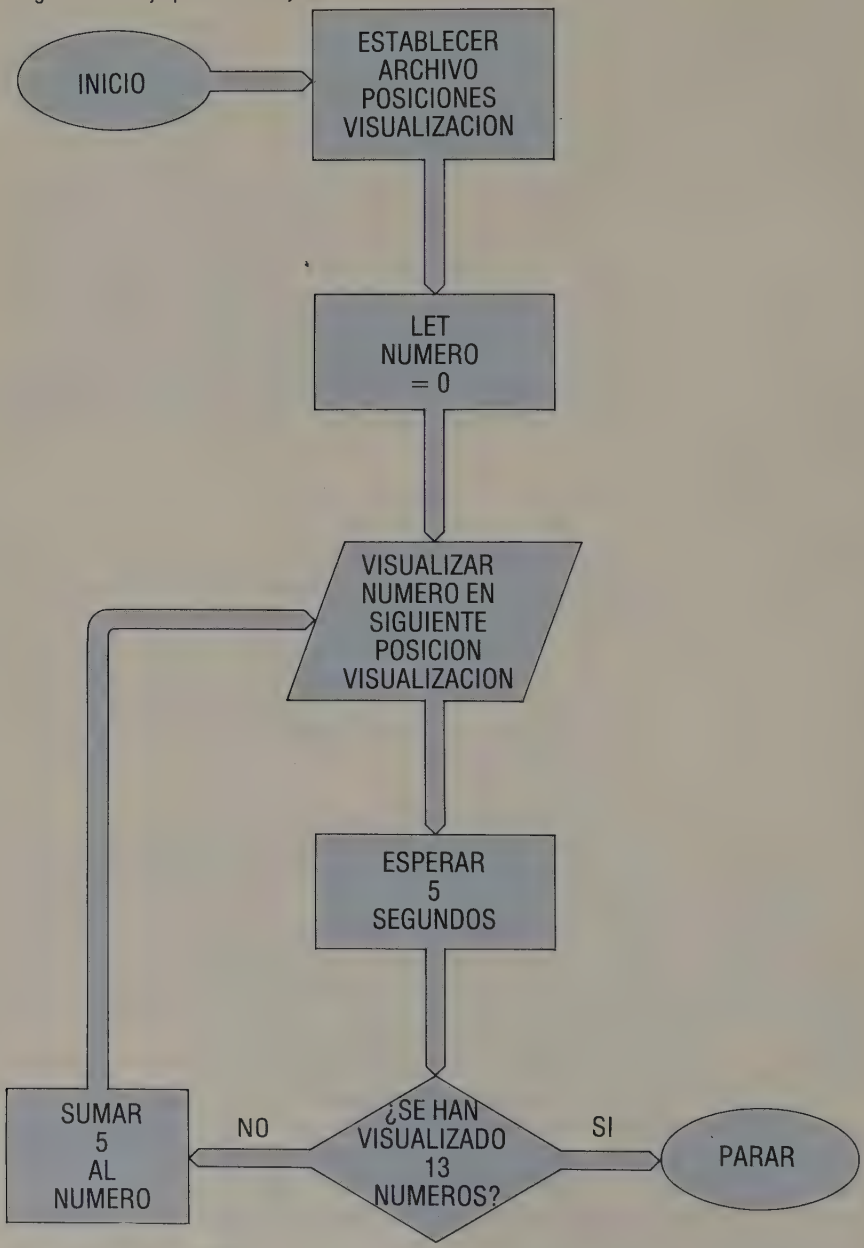
utiliza dos veces) que el ordenador irá leyendo uno a uno. Observe el diagrama de flujo y la disposición del reloj para que le ayude a ver lo que está haciendo.

El ordenador necesitará que se le dé una posición de fila seguida por una posición de columna y después otra posición de fila, de la misma forma que el programa de «hacer un dibujo» que vimos anteriormente, así:

Fila 0, columna 15,
Fila 2, columna 18,
Fila 4, columna 21,
Fila 6, columna 23... y así sucesivamente.
Así es como funciona:
Pruebe esto:

```
10  REM ***RELOJ***
20  REM ENTRA LAS POSICIONES DE VISUALIZACION EN
    LA LISTA
30  DIM R(13): DIM C(13)
40  FOR N = 1 TO 13
50  READ R(N): READ C(N)
60  NEXT N
70  REM VISUALIZA LOS NUMEROS DEL RELOJ A INTER-
    VALOS DE 5 SEG.
80  CLS
```

Diagrama de flujo para el «reloj»



```
90   FOR N = 1 TO 13
100  PRINT@ (R(N),C(N)),(N-1)*5;
120  FOR D = 0 TO 460*5: NEXT D
130  NEXT N
140  GOTO 140
200  DATA 0,15,2,18,4,21,6,23,8,21
210  DATA 10,18,12,15,10,12,8,9,6,7
220  DATA 4,9,2,12,0,15
```

Volvamos de nuevo sobre esto. En realidad el programa esta vez tiene tres partes. La primera parte lee los datos y los coloca en dos variables de lista, una para las posiciones de fila y otra para las posiciones de columna. Estas son las posiciones en las que queremos que los números de la esfera del reloj sean visualizados. Después, la segunda parte visualiza el reloj, en intervalos de 5 segundos. (Se acordará del bucle de espera del capítulo 4.) Y la tercera parte suministra los datos.

Debe ser capaz de seguir cómo la sentencia READ coge cada posición de la lista, una a una, y le dice al ordenador que ponga el dato en cada una de ellas. Pero ¿cómo funciona la sentencia DATA? La mejor manera de verlo es considerar que los datos que van después de las órdenes DATA son una larga lista de artículos, que el ordenador va tomando secuencialmente. Cada vez que el ordenador toma un dato de la lista y lo coloca en una posición de la variable, la tacha. La siguiente vez que se le diga que lea (READ) tomará el artículo siguiente de la lista, y así sucesivamente hasta que se termine la lista. Por lo tanto, no hay que hacer nada más para decirle al ordenador cuál es el dato que va en cada posición de la variable. Queda determinado por el orden de los datos en la lista.

Si hay demasiados datos en nuestra lista, el ordenador ignorará simplemente los datos extra. Si no hay datos suficientes en la lista, nos dará un mensaje de error del tipo «OD» o «OUT OF DATA» (faltan datos).

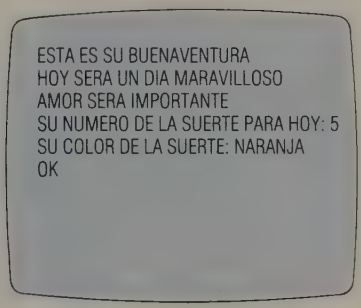
Nótese que hemos puesto los datos exactamente al final del programa, después del bucle cerrado que determina el final de la ejecución. De todas maneras el ordenador los encontrará. De hecho, podemos poner los datos en el sitio que nos parezca más conveniente

de la lista, ya sea antes o después de las sentencias de READ que los utiliza . Y podemos utilizar exactamente la misma forma de sentencias de DATA para entrar datos en cualquier cantidad de variables y de variables de lista.

He aquí otro ejemplo:

El ordenador le dice la buenaventura

Esta es la ejecución de un típico programa de «buenaventura» tal como se describe a continuación.



Pruebe esto:

```
10   REM ***BUENAVENTURA***
15   REM ENTRA LOS DATOS EN LAS VARIABLES DE LA
     BUENAVENTURA
20   DIM A$(3): DIM B$(4): DIM C$(6)
30   FOR X = 1 TO 3: READ A$(X): NEXT X
40   DATA "MARAVILLOSO", "ABURRIDO", "HORRIBLE"
50   FOR X = 1 TO 4: READ B$(X): NEXT X
60   DATA "DINERO", "TRABAJO", "FAMILIA", "AMOR"
70   FOR X = 1 TO 6: READ C$(X): NEXT X
80   DATA "ROJO", "AMARILLO", "VERDE", "AZUL"
90   DATA "NARANJA", "PURPURA"
100  REM ESCRIBE LA BUENAVENTURA
110  CLS
120  PRINT@ (1,1), "ESTA ES TU BUENAVENTURA"
130  PRINT@ (3,1), "HOY SERA UN"; A$(1+INT(RND*3));"
     DIA"
140  PRINT@ (5,1) B$(1+INT(RND*4));" SERA
     IMPORTANTE"
```

```
150 PRINT@ (7,1), "TU NUMERO DE LA SUERTE PARA
    HOY" 1+INT(RND*9)
160 PRINT@ (9,1), "TU COLOR DE LA SUERTE: ";
    C$(1+INT(RND*6))
170 END
```

Parece que tengamos tres listas de datos en este programa: una para cada una de las variables de lista A$, B$ y C$. Pero de hecho, tenemos únicamente una larga lista —y la línea 130 es el punto donde se elige entre el 1.°, 2.° o 3.° artículo de la lista, en la línea 140 se escoge entre el 4.° al 7.° y en la línea 160 entre el 8.° al 13.°—. Podrá ver que se trata únicamente de una lista si intenta cambiar la línea 40 por:

```
40 DATA "MARAVILLOSO", "ABURRIDO", "HORRIBLE",
   "DIVERTIDO"
```

y después prueba algunas ejecuciones aleatorias, o le pide al ordenador que visualice el contenido de cada variable de lista, para poder ver cómo el último artículo de la primera sentencia de DATA ha sido colocado en la segunda variable de lista, y el último artículo de la segunda sentencia en la tercera variable. Obviamente, tendrá que comprobar dos veces sus sentencias de DATA, sobre todo cuando contengan listas de números, para asegurarse que coinciden adecuadamente con la estructura de sus variables de lista.

Una base de datos muy sencilla

Quizás haya oído hablar de la idea de una base de datos. Se trata tan sólo de una colección ordenada de información a la que usted puede acceder mediante programas de ordenador apropiados y hacer que el ordenador, por ejemplo, encuentre y visualice un dato o conjunto de datos en particular que usted desee. La mayoría de las instalaciones de ordenadores comerciales utilizan bases de datos para ordenar sus archivos. Podemos escribir un programa que genere una estructura de base de datos muy sencilla, utilizando las sentencias READ y DATA para colocar nuestros datos en una serie de variables de lista. Y podemos hacerle preguntas, diciéndole al programa qué parte de los datos queremos ver. Vamos a hacer esto ahora, utili-

zando un sencillo «catálogo de bibilioteca» como nuestro ejemplo.

Vamos a colocar tres listas, cada una de las cuales contendrá los autores, los títulos, y los números de referencia respectivamente. Entremos la siguiente información:

n.º de ref.	*autor*	*título*
0001	C Dickens	David Copperfield
0002	J Austen	Pride and prejudice
0003	H Melville	Moby Dick
0004	C Dickens	Hard times
0005	E Bronte	Wuthering Heights
0006	H James	Portrait of a Lady

Llamaremos a estas variables de lista R, A$ y T$. Esto nos dará en nuestro programa una sección tal como:

```
100 DIM R(6): DIM A$(6): DIM T$(6)
110 FOR X = 1 TO 6
120 READ R(X): READ A$(X): READ T$(X)
130 NEXT X
140 DATA 0001, "C DICKENS", "DAVID COPPERFIELD"
150 DATA 0002, "J AUSTEN", "PRIDE AND PREJUDICE"
160 DATA 0003, "H MELVILLE", "MOBY DICK"
170 DATA 0004, "C DICKENS", "HARD TIMES"
180 DATA 0005, "E BRONTE", "WUTHERING HEIGHTS"
190 DATA 0006, "H JAMES", "PORTRAIT OF A LADY"
```

Nótese cómo hemos ordenado cuidadosamente nuestra sentencia de datos de forma que la pieza de información correcta vaya a su correspondiente posición en la variable de lista.

¿Qué es lo que queremos hacer con los datos? Supongamos que queremos que el usuario nos dé un número de referencia, el nombre del autor o un título, y después que el ordenador nos visualice el resto del conjunto de información; es decir, el autor y el título del libro que tiene este número de referencia; todos los libros que tiene este autor con su número de referencia y así sucesivamente. He aquí un diagrama de flujo que muestra cómo manejaremos esto. De nuevo,

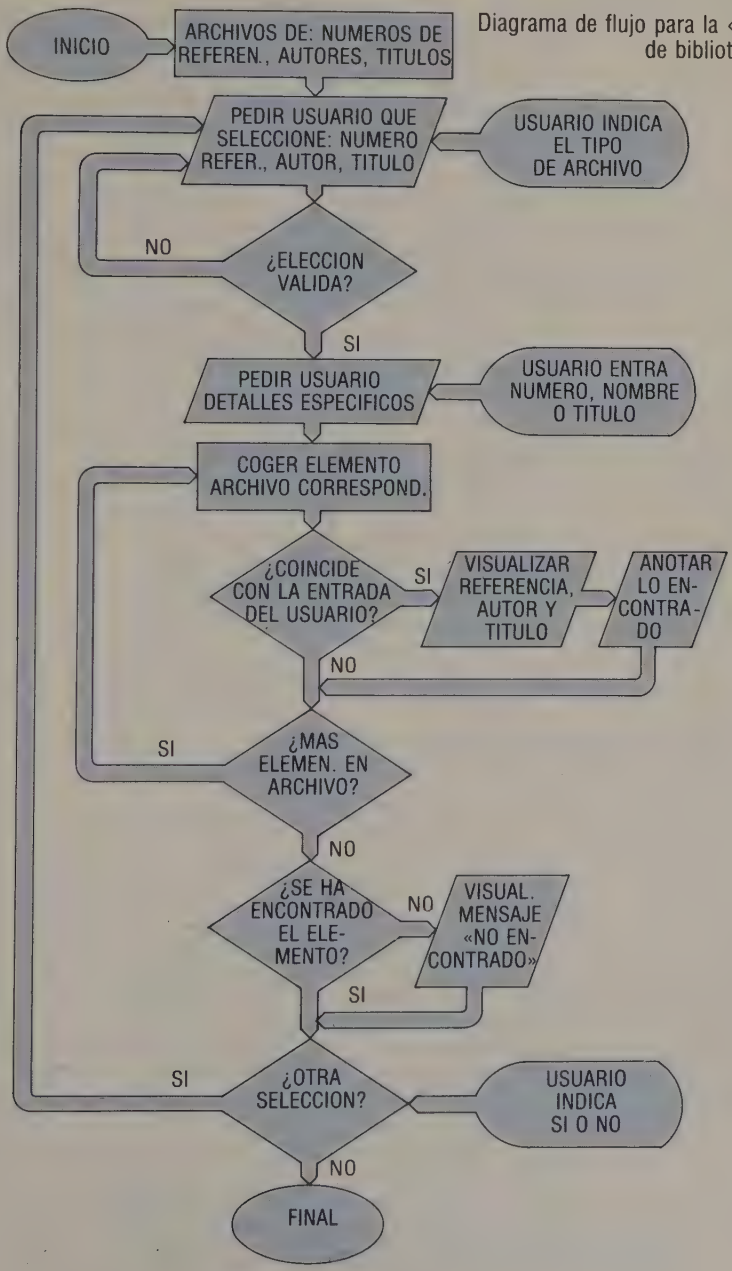


Diagrama de flujo para la «lista de biblioteca»

quizás encontrará de ayuda el analizar este diagrama a medida que sigue el programa.

Habrá que hacerle al usuario dos preguntas. Primero, ¿qué categoría de información nos está dando? Es decir, ¿se trata del número de referencia del autor o del título? Segundo, ¿cuál es la información que quiere comprobar? Pongamos lo primero en una variable llamada B$, y lo segundo en variable numérica llamada M, o en una variable de texto llamada M$. Utilizaremos también otra variable para comprobar si hemos encontrado la entrada o no: la llamaremos C, y otra para comprobar si se quiere otra entrada: Y$. Por lo tanto he aquí nuestra lista de variables:

Variables

Variable de lista para los números de referencia:	R(1 a 6)
Variable de lista para los nombres de los autores:	A$(1 a 6)
Variable de lista para los títulos:	T$(1 a 6)
Variable de texto para la pregunta:	B$
Variable para el dato numérico que hay que encontrar:	M
Variable para el texto que hay que encontrar:	M$
Variable de comprobación:	C
Variable para la reejecución:	Y$

Y he aquí nuestro programa:

```
10   REM ***LISTA DE LA BIBLIOTECA***
20   REM COLOCACION DE LA BASE DE DATOS
100  DIM R(6): DIM A$(6): DIM T$(6): LET C = 0
110  FOR X = 1 TO 6
120  READ R(X): READ A$(X): READ T$(X)
130  NEXT X
140  DATA 0001, "C DICKENS", "DAVID COPPERFIELD
150  DATA 0002, "J AUSTEN", "PRIDE AND PREJUDICE"
160  DATA 0003, "H MELVILLE" "MOBY DICK"
170  DATA 0004, "C DICKENS", "HARD TIMES"
180  DATA 0005, "E BRONTE", "WUTHERING HEIGHTS"
190  DATA 0006, "H JAMES", "PORTRAIT OF A LADY"
200  REM OBTENCION DE LA PREGUNTA
210  PRINT "QUIERE DAR UN NO. REF,"
```

```
220  PRINT "AUTOR O TITULO (R,A O T),"
230  INPUT B$
240  IF B$ = "R" THEN GOTO 500 ——→
250  IF B$ = "A" THEN GOTO 400 ——→
260  IF B$ < > "T" THEN GOTO 210 ——→
300  REM RUTINA SI SE HA PEDIDO EL TITULO
310  INPUT "TITULO";M$
320  FOR X = 1 TO 6
330  IF M$ = T$(X) THEN PRINT T$(X): PRINT
     A$(X): PRINT "REF."; R(X): LET C = 1
340  NEXT X
350  GOTO 550 ——→
400  REM RUTINA SI SE HA PEDIDO EL AUTOR
410  INPUT "AUTOR (INICIAL Y APELLIDO)";M$
420  FOR X = 1 TO 6
430  IF M$ = A$(X) THEN PRINT A$(X): PRINT T$(X)
     PRINT "REF.";R(X): LET C = 1
440  NEXT X
450  GOTO 550 ——→
500  REM RUTINA SI SE HÁ PEDIDO NO. REF.
510  INPUT "NO.REF. (4 DIGITOS)";M
520  FOR X = 1 TO 6
530  IF M = R(X): PRINT A$(X):
     PRINT T$(X): LET C = 1
540  NEXT X
550  REM SUMARIO Y SECUENCIA DE REEJECUCION
560  IF C = 0 THEN PRINT "DATO NO ENCONTRADO"
570  INPUT "OTRA PREGUNTA? S/N"; Y$
580  IF Y$ = "S" THEN LET C = 0: GOTO 200 ——→
590  END
```

Aunque es un programa bastante largo, no es uno realmente muy elaborado. Las bases de datos en la realidad hacen muchas más cosas, por ejemplo, el encontrar entradas que se parezcan, aunque no sean exactamente iguales. Nuestra versión encontrará únicamente los datos en el archivo que coincidan exactamente con la entrada del usuario. De todas maneras, es suficiente para darle una idea.

Ahora le toca a usted

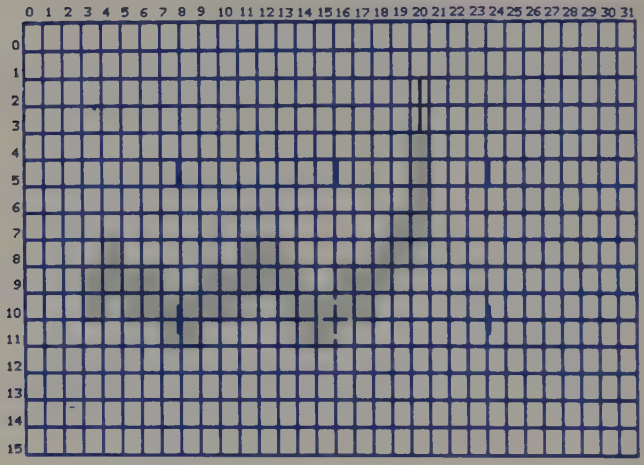
4) Escriba un programa que dibuje una «serpiente» que serpentee a lo largo de la pantalla, colocando los

datos en posiciones de visualización mediante sentencias de DATA.

5) Escriba un programa que contenga nombres y direcciones de diez personas, y que visualice sus direcciones cuando usted entra su nombre.

Serpiente

Intente dibujar una serpiente como ésta. La muestra crece bloque a bloque y después saca su lengua venenosa.



Comprobación

He aquí el resumen del material que hemos analizado en este capítulo. Ahora debe usted saber:

Sobre variables de lista

- Cómo dimensionar una variable de lista unidimensional mediante la orden DIM.
- Cómo son los nombres de los elementos de una variable de lista.
- Cómo entrar datos en una variable de lista mediante la orden INPUT.
- Cómo colocar datos en una variable de lista utilizando las órdenes READ...DATA.

Sobre bases de datos

- Cómo un conjunto ordenado de datos pueden actuar como una «base de datos» en el ordenador.
- Cómo construir y hacer preguntas en una sencilla base de datos mediante órdenes DATA y variables de lista.

7 Cómo escribir programas más largos

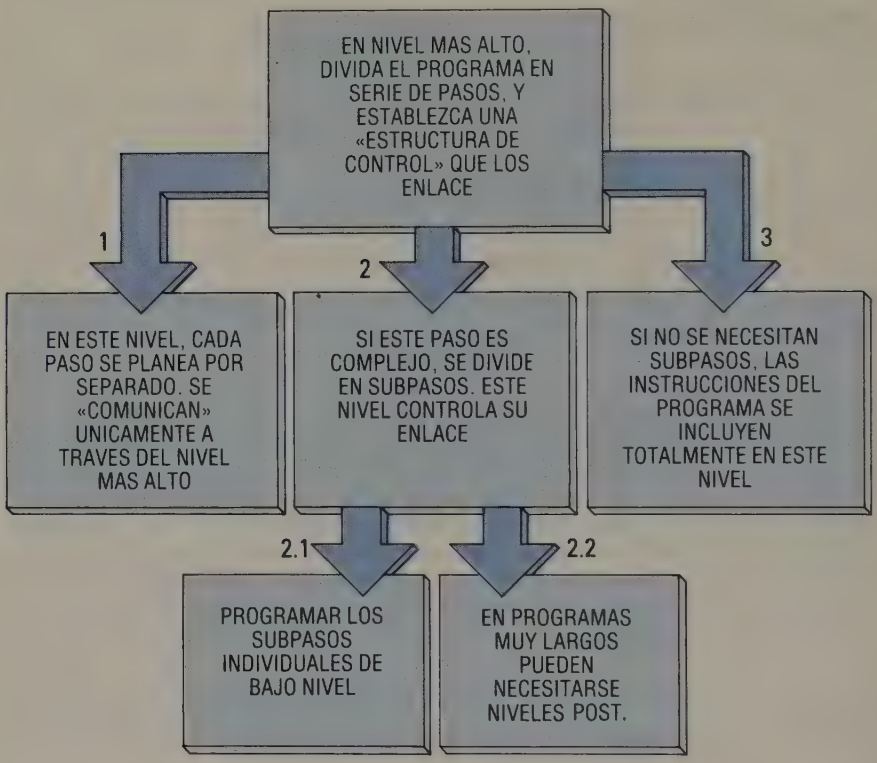
En este capítulo veremos:
- **Técnicas para la estructuración de programas más largos.**
- **Las órdenes GOSUB y RETURN.**

Habrá advertido que a medida que avanza a través de este libro, nuestros ejemplos de programas son cada vez más largos. De hecho es imposible introducir algunos conceptos, tales como el manejo de datos, en programas extremadamente cortos. Sin embargo, hemos intentado mantener nuestros programas lo más cortos posibles y poner muchos ejemplos distintos.

En este capítulo haremos exactamente lo contrario. Tomaremos dos programas más largos —aunque no son excepcionalmente largos en comparación con los que puede encontrar en cualquier otro sitio— y avanzaremos paso a paso en el planteo y en la escritura de los mismos.

Programación estructurada

Quizás haya oído hablar del término «programación estructurada». Se refiere a una manera de escribir programas que se caracteriza por el planteo de una estructura bien clara en la que encajarán las sentencias individuales. El programa se plantea primero en líneas generales —a un «alto nivel», si se le quiere llamar así— mediante una serie de secciones, o módulos, interrelacionados entre sí únicamente por formas claramente definidas. Cada módulo, si es necesario, se divide en módulos cada vez más pequeños, en niveles cada vez más bajos, hasta que cada uno de ellos tiene un tamaño fácilmente manejable. Entonces los módulos individuales se programan por separado y se comprueban hasta donde sea posible antes de incluirlos en el programa total. La programación estructurada es un método muy adecuado para plantear y escribir largos



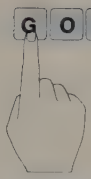
Programación estructurada
Esta es, en líneas generales, la forma en que funciona la programación estructurada. Naturalmente la estructura de un programa en concreto puede ser mucho más compleja que esto.

programas para que sean fáciles de comprobar y de depurar; los programas que están bien estructurados son más fáciles de entender y de modificar. Sin embargo, aprender a programar sobre el papel es un método bastante difícil.

A los programadores estructurados, por ejemplo, no les gusta en absoluto la orden GOTO. Dicen que muchos programadores caen en la costumbre de colocar, en una larga lista, gran cantidad de sentencias GOTO, aparentemente de una forma aleatoria. Prefieren secuencias de órdenes más formales para los bucles y las ramificaciones. Algunas de estas órdenes no se encuentran en absoluto en el BASIC, sino únicamente en lenguajes como el Pascal y COMAL. Algunas (por ejemplo, el IF...THEN...ELSE y las órdenes de manejo de subrutinas del BBC BASIC) se encuentran en algu-

nos BASIC mejores y más grandes, pero no en muchas de las versiones de ordenadores domésticos.

En este libro, nos hemos concentrado en conseguir que usted se familiarice con las maneras más sencillas de escribir programas. Hemos utilizado, por ejemplo, el GOTO ya que creemos que es más fácil de entender y utilizar que muchas otras órdenes de estructuración, y porque todos los BASIC lo tienen.



Nuestros métodos son correctos para escribir programas cortos como los que hemos visto hasta ahora, pero hay que admitir que quedan un poco cortos cuando se trata de planear programas más largos. En este capítulo veremos con más profundidad el planteo de programas, utilizando formas más estructuradas. Sin embargo, introduciremos únicamente una nueva secuencia de órdenes, la ampliamente utilizada secuencia de GOSUB...RETURN. Si pretende escribir muchos programas largos, descubrirá muy pronto lo fácil que es perder el hilo de su lógica, y lo interesante que es tener herramientas de estructuración flexibles. Cuando llegue a este punto estará preparado para un libro de BASIC más avanzado.

Resumiendo, lo que le enseñamos —incluso en este capítulo— no es programación estructurada en la manera que mucha gente entiende este concepto, aunque toma prestadas muchas ideas de la programación estructurada. Creemos que nuestro planteo en esta dirección le hará más fácil su introducción en la programación estructurada. No obstante, es importante el tener en mente que a medida que la longitud de sus programas aumenta, también lo hace la dificultad de escribirlos ¡más que proporcionalmente! Cuando plantee programas más largos, tendrá que concentrarse más y más en la lógica del conjunto y en la forma en que las secciones de su programa se conectan entre sí.

Un programa para «empapelar»

Nuestro primer programa tratará de resolver un problema de la vida real que es menos sencillo de lo que pueda parecer a primera vista: calcular cuánto papel se necesita para empapelar una habitación.

Antes de pensar acerca de qué sentencias de BASIC vamos a utilizar —y antes de conectar el ordenador— necesitaremos sentarnos y pensar sobre nuestro programa en su totalidad. ¿Qué es lo que que-

remos hacer y cómo, muy generalmente, vamos a hacerlo?

La gente empapela todo tipo de áreas extrañas de la pared y del techo, y, primero de todo, necesitaremos calcular el área que tiene que ser empapelada. Por lo tanto hagamos que sea éste el primer paso de nuestro plan. También tendremos que calcular cuánto papel hay en un rollo: llamaremos a esto el paso dos. Y después querremos descubrir cuántos rollos necesitamos para cubrir el área: éste será el paso tres.

He aquí un planteo muy general, muy a alto nivel, si se quiere llamar así, de nuestro programa:

Paso 1: Calcular el área a empapelar.

Paso 2: Calcular el área que puede cubrirse con un rollo de papel.

Paso 3: Calcular el número de rollos necesarios.

Estos pasos constituyen tres secciones separadas de nuestro programa. Las programaremos una a una, pero primero necesitaremos hablar de cómo las enlazaremos.

El hacer que las secciones sean consistentes es muy importante. Para tomar un simple ejemplo, tendremos que asegurarnos de que utilizamos la misma unidad de medida (pulgadas, centímetros o lo que sea) en todas las secciones, o habrá que incluir una secuencia de conversión. Tendremos que asegurarnos también de que utilizaremos nombres de variables que se correspondan de una sección a otra. Por lo tanto, está claro que la consistencia a lo largo del programa es algo que tiene que ser construido al principio, es decir, en el nivel más alto del planteo del programa.

Hagamos una lista de los puntos que afectan a este nivel «más alto»:

Unidad de medida. Al ordenador no le importa qué unidad es, con tal de que permanezca siempre la misma. Pongamos una sentencia de PRINT en la introducción que recuerde al usuario que debe mantener sus medidas consistentes.

Variables del programa. Se trata de variables a alto nivel. Quizá necesitemos otras, variables de bajo nivel, en las secciones separadas. Utilizamos variables de programa a lo largo de todo programa; las variables

de bajo nivel afectan únicamente a una única sección del mismo.

Area que debe ser empapelada: A
Area cubierta por un rollo: R
Número de rollos necesarios: N

Ahora podemos diseñar un plan más completo:

Introducción: Decir al usuario lo que hace el programa.
Recordarle que mantenga la unidad de medida constante
Paso 1: Obtener el área a empapelar: A
Paso 2: Obtener el área cubierta por un rollo: R
Paso 3: Obtener el número de rollos necesarios: N

Hemos escrito esto en forma de notas, pero naturalmente podríamos dibujar un diagrama de flujo a alto nivel para ilustrarlo. Un ejemplo de esto se muestra en la página siguiente.

Ahora vamos a convertir este alto nivel en un programa de BASIC. ¿Cómo podemos hacerlo sin saber lo que irá dentro de estas secciones individuales? Mediante la utilización de subrutinas.

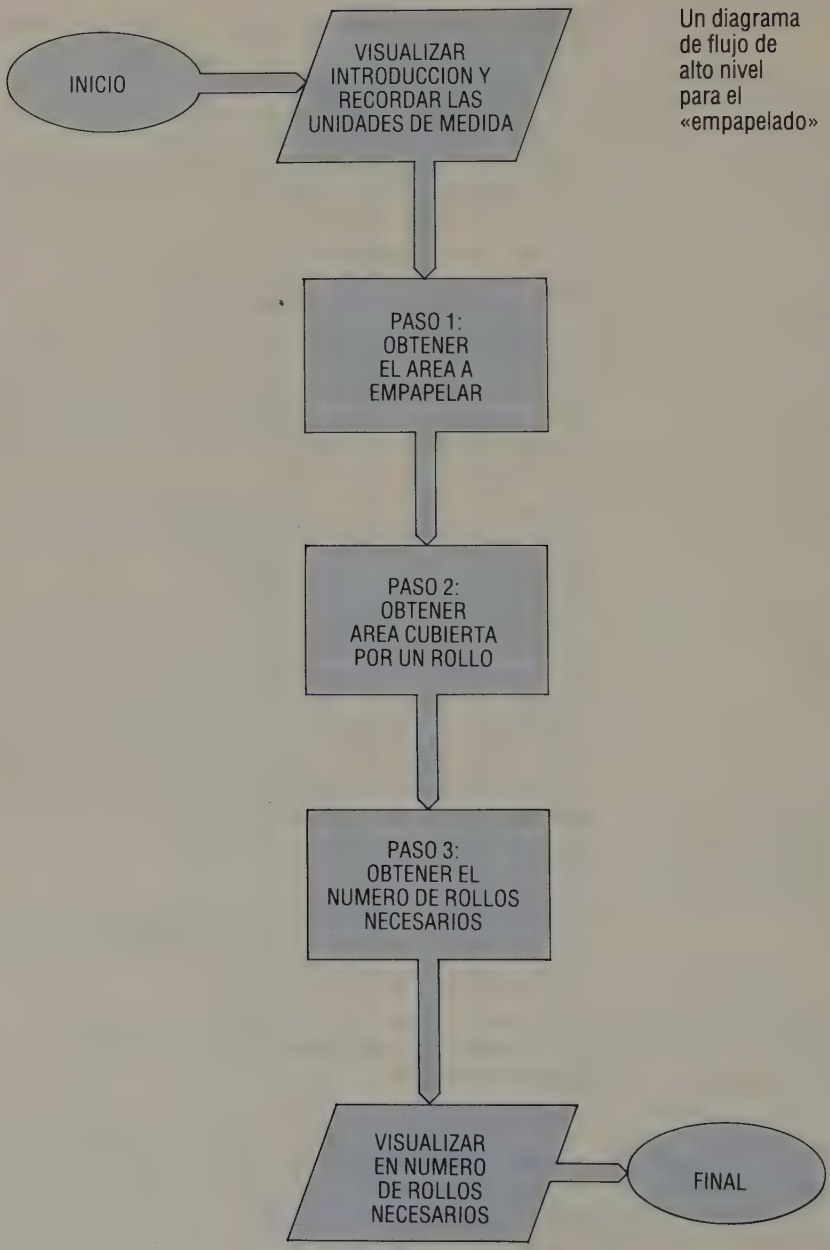
Una subrutina es una rutina completa utilizada por el programa. El programa puede utilizarla una vez, o puede utilizarla muchas veces; no importa. La subrutina tiene un conjunto separado de números de línea (normalmente más altos que los utilizados en el programa principal) y el programa la «llama» (es decir, le dice al ordenador que ejecute las instrucciones que hay en ella) mediante la orden GOSUB. Se utiliza así:

```
GOSUB 1000
```

Esto significa «vete a la línea 1000 y ejecuta la subrutina que empieza allí». En la línea 1000 habrá una secuencia de sentencias tal como:

```
1000 REM DESCRIPCION DE UNA SUBRUTINA
1010 (sentencias que constituyen la subrutina)
1020 RETURN
```

La orden RETURN le dice al ordenador que cuando haya alcanzado el final de la subrutina, tiene que volver atrás hasta el punto donde había dejado anteriormente al programa principal. Funciona como una orden GOTO, pero al final el ordenador recuerda por



Un diagrama de flujo de alto nivel para el «empapelado»

sí mismo dónde ir seguidamente, por lo que no hay que poner un número de línea después del RETURN.

De hecho, las subrutinas son especialmente útiles cuando se necesita repetir una pieza del programa en particular —por ejemplo, hacer ciertos cálculos o un conjunto de cálculos— una y otra vez a través del programa. Tan sólo hay que teclear la subrutina una vez, lo que ahorra espacio y trabajo, y, sin importar cuántas veces o dónde en un programa se llame a la subrutina, el ordenador retornará automáticamente al lugar correcto del programa principal después de ejecutar las órdenes contenidas en la subrutina.

Sin embargo, en el programa siguiente no necesitaremos repetir ninguna subrutina. Esta es la línea general de nuestro programa, utilizando subrutinas:

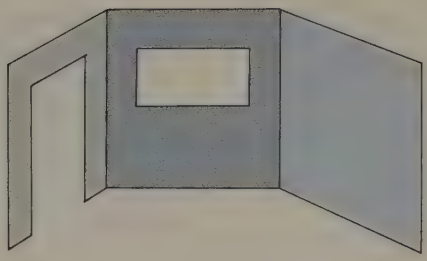
```
10  REM ***EMPAPELADO***
20  REM POR SUSAN CURRAN
30  PRINT "ESTE PROGRAMA CALCULA LA"
40  PRINT "CANTIDAD DE PAPEL NECESARIO"
50  PRINT "PARA EMPAPELAR UNA HABITACION"
60  PRINT "DEME TODAS LAS MEDIDAS"
70  PRINT "EN LA MISMA UNIDAD"
80  REM EL AREA A EMPAPELAR ES A
90  REM EL AREA DE UN ROLLO DE PAPEL ES R
100 REM EL NUMERO DE ROLLOS NECESARIOS ES N
110 GOSUB 1000: REM OBTENER A
120 GOSUB 2000: REM OBTENER R
130 GOSUB 3000: REM OBTENER N
140 PRINT "NUMERO DE ROLLOS NECESARIOS ES";N
150 END
```

¿Qué puede ser más fácil de entender que esto? Y esto es lo que aparecerá al principio de nuestro programa completo. Todo lo que ahora hace falta hacer es realizar nuestras tres subrutinas igualmente claras.

Empecemos con la subrutina 1, la que calcula el área a empapelar. Una vez más debemos detenernos y pensar qué es lo que queremos hacer. ¿Qué tipo de áreas queremos empapelar? Generalmente paredes y techos. Suelen ser rectangulares, con huecos para las puertas, ventanas, armarios, etc. Para no hacer nuestra tarea demasiado difícil, vamos a ceñirnos a rectángulos, con huecos rectangulares.

Area de empapelar

Para calcular la cantidad de papel necesaria, podemos considerar que paredes y techos son rectángulos, algunos de los cuales tienen huecos rectangulares, donde están puertas y ventanas.



Desde luego solemos querer recubrir más de una pared con el mismo papel. Por lo tanto, tendremos que hacer las medidas de varias áreas individuales y entonces hacer la suma de todas ellas. Planteemos nuestra subrutina así:

a) obtener las dimensiones de un área.
b) calcular el área y añadirla al total.
c) volver a a) si todavía hay más áreas para incluir.
d) obtener las dimensiones de un hueco.
e) calcular el área del hueco y restarla del total.
f) volver a d) si hay más huecos que deban incluirse.

Necesitaremos estas variables:
Primera dimensión del área (llamémosle altura): H
Segunda dimensión del área (llamémosle anchura): W
Area total: A

A, naturalmente, es la variable del programa resultado de nuestra subrutina. Las otras son de bajo nivel, o variables de la subrutina.

No sabemos cuántas áreas o huecos habrá, por lo tanto utilizaremos bucles abiertos con sentencias IF...THEN...GOTO. He aquí la subrutina:

```
1000 REM SUBRUTINA PARA CALCULAR EL AREA
1010 REM QUE TIENE QUE EMPAPELARSE
1020 PRINT "LAS AREAS A EMPAPELARSE TIENEN QUE"
1030 PRINT "SER DESCRITAS COMO RECTANGULOS
     CON"
1040 PRINT "HUECOS RECTANGULARES. DEME PRI-
     MERO"
1050 PRINT "TODAS LAS AREAS Y DESPUES TODOS LOS
     HUECOS"
```

```
1060  LET A = 0
1070  INPUT "PRIMERA DIMENSION DEL AREA";H
1080  INPUT "SEGUNDA DIMENSION DEL AREA";W
1090  LET A = A + (H*W)
1100  INPUT "MAS AREAS? S/N";R$
1110  IF R$ = "S" THEN GOTO 1070
1120  INPUT "PRIMERA DIMENSION DEL HUECO";H
1130  INPUT "SEGUNDA DIMENSION DEL HUECO";W
1140  LET A = A - (H*W)
1150  INPUT "MAS HUECOS? S/N";R$
1160  IF R$ = "S" THEN GOTO 1120
1170  RETURN
```

¿Está todo suficientemente claro? Está bastante claro, pero como subrutina es bastante larga. Quizá sería mejor convertirlo en una subrutina y en dos subsubrutinas: La primera como subrutina general y las otras dos para calcular las áreas y los huecos. No hay ningún problema en hacer esto ya que pueden encadenarse subrutinas de la misma forma que se encadenan los bucles. Le daremos la subrutina principal, y dejaremos el trabajo de programar las otras dos subrutinas para usted (una versión está incluida en el programa completo en las páginas 165-167).

```
1000  REM SUBRUTINA PARA CALCULAR EL AREA
1010  REM A EMPAPELAR
1020  PRINT "LAS AREAS A EMPAPELAR DEBEN SER"
1030  PRINT "DESCRITAS COMO RECTANGULOS CON"
1040  PRINT "HUECOS RECTANGULARES. DEME PRIMERO"
1050  PRINT "TODAS LAS AREAS Y TODOS LOS HUECOS"
1060  LET A = 0
1070  GOSUB 1500: REM CALCULO TOTAL DE TODAS LAS
      AREAS
1120  GOSUB 1700: REM CALCULO DEL AREA TOTAL
      MENOS LOS HUECOS
1170  RETURN
```

Si esto ya está claro vamos a nuestra segunda subrutina. Esta será más corta. Tan sólo hay que pedir las dimensiones de un único rollo de papel y calcular su área. No se necesita ningún bucle. Nuestras variables serán las mismas:

Diagrama de flujo de la subrutina 1 de «empapelado»

Anchura del rollo:	W	variables de subrutina
Longitud del rollo:	L	
Area del rollo:	R	variable de programa

He aquí la subrutina:

```
2000  REM SUBRUTINA PARA CALCULAR EL AREA
2010  REM DE UN ROLLO DE PAPEL
2020  INPUT "LONGITUD DE UN ROLLO DE PAPEL"; L
2030  INPUT "ANCHURA DEL ROLLO";W
2040  LET R = L*W
2050  RETURN
```

Esto ha sido bastante sencillo, por lo tanto vayamos con la tercera subrutina. Hay un par de puntos a tener en cuenta esta vez. Primero, hay que considerar los trozos no aprovechables. Hay finales de rollo que no pueden utilizarse y existen formas que hay que hacer coincidir. Si fuéramos realmente científicos, tendríamos que calcular cuántas «longitudes» de papel podríamos sacar de cada rollo. Ya que se trata de un programa de ejemplo y no queremos hacerlo demasiado complicado, añadiremos un «margen de desaprovechamiento». Invitaremos al usuario a dar una cifra en tanto por ciento de desaprovechamiento.

Segundo, tan sólo podemos comprar rollos de papel completos por lo que habrá que redondear nuestra respuesta. La función INT redondea por debajo, por lo tanto a menos que el resultado de nuestro cálculo sea un número entero tendremos que añadirle una unidad.

Veamos cómo puede hacerse todo esto. Primero, nuestras variables:

Margen de error:	M	variable de subrutina
Area que debe empapelarse:	A	todas las variables del programa
Area cubierta por un rollo:	R	
Número de rollos:	N	

Ahora vamos a detenernos para considerar el margen de error. El usuario nos dará un número como «10» que representa «el 10 %». Será necesario multiplicar el área no por diez sino por 110/100 para añadir el mar-

Diagrama de flujo de la subrutina 2 de «empapelado»

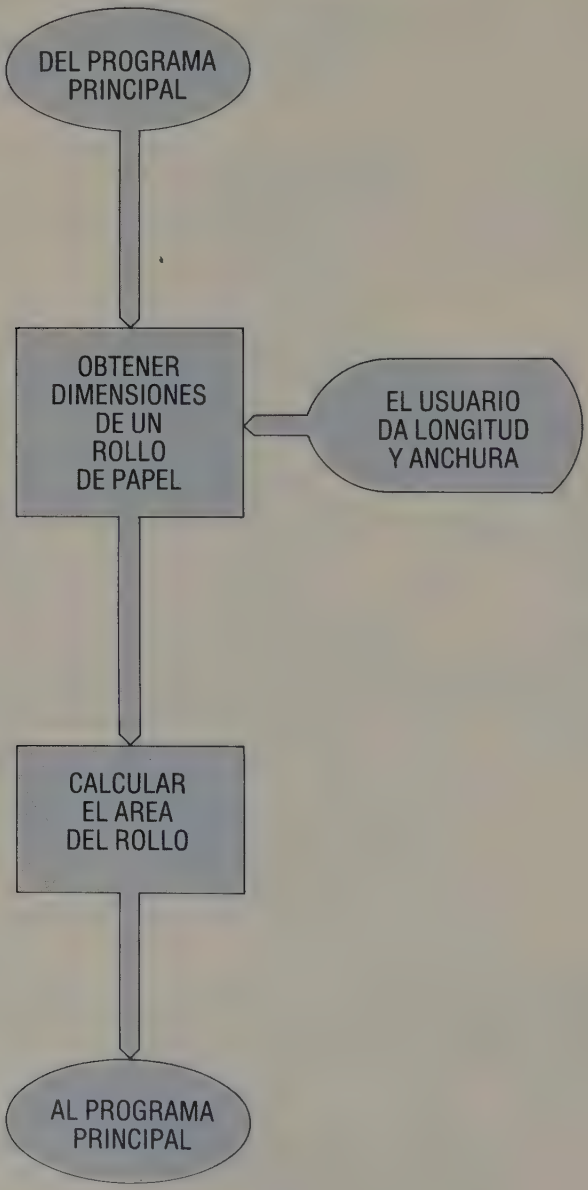
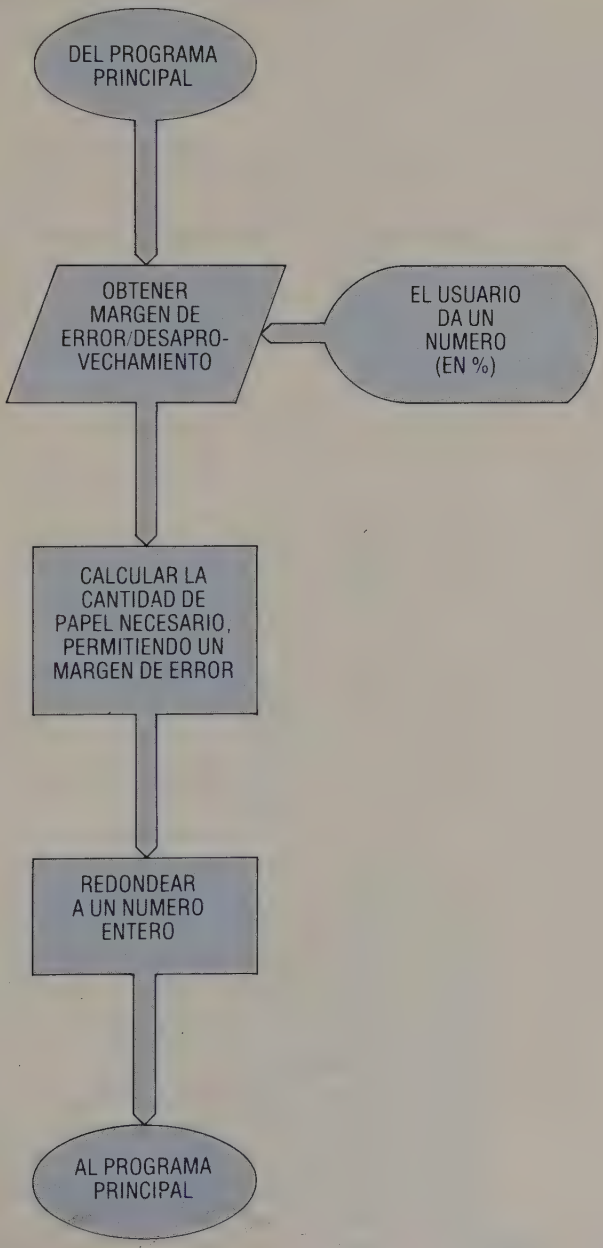


Diagrama de flujo de la subrutina 3 de «empapelado»



gen. Por lo tanto necesitaremos una sentencia tal como ésta para conseguir una respuesta que contenga el porcentaje de error:

```
LET N = (A*(M+100))/(R*100)
```

No hemos hablado con detalle acerca del orden en el que el ordenador lleva a cabo las funciones aritméticas, por lo tanto pondremos cantidad de paréntesis para asegurarnos que lo hace correctamente. Como comprobación extra, ¿por qué no probarlo? Si ha entrado ya parte del programa principal, puede mantener su comprobación separada de él utilizando números de línea muy bajos. Esto sería así:

```
1  LET A = 100: LET M = 10: LET R = 10
2  LET N = (A*(M+100))/(R*100)
3  PRINT N
4  END
```

Compruebe que el ordenador obtiene el resultado correcto haciendo la suma usted mismo o quizá con una calculadora.

He aquí la subrutina completa:

```
3000  REM SUBRUTINA PARA CALCULAR
3010  REM EL NUMERO DE ROLLOS NECESARIOS
3020  INPUT "DEME UN MARGEN DE ERROR EN %";M
3030  LET N = (A*(M+100))/(R*100)
3040  IF N > INT(N) THEN LET N = 1 + INT(N)
3050  RETURN
```

Y éste es nuestro programa completo. Finalmente ha sido muy fácil ¿no es así? Eche un vistazo al listado completo en las páginas 165-167, y verá que aunque éste es un programa bastante largo, con 51 líneas de BASIC, está bastante claro y es sencillo de seguir. El uso de subrutinas realmente ayuda a hacer que el programa sea más comprensible.

Probablemente encontrará que los diagramas de flujo no son de tanta utilidad cuando se adopta este planteo estructurado. Pero si quiere usarlos, deberá diseñar un conjunto, uno que muestre la línea general del programa y otros que representen todas las subrutinas, como hemos hecho hasta ahora.

Comprobación y depuración del programa

No hemos terminado todavía; debemos asegurarnos de que el programa funciona adecuadamente. Hablemos brevemente sobre ello. ¿Podemos comprobar el programa subrutina a subrutina? Sí, si hacemos algunas modificaciones para aislar cada sección, igual que hicimos en nuestra pequeña comprobación anteriormente. Necesitaremos comprobar también que cada sección visualiza el resultado correcto, añadiendo muchas sentencias de PRINT (ejemplo, PRINT A).

Es una buena idea el acostumbrarse a visualizar los resultados intermedios siempre que se tenga algo que no sean operaciones aritméticas muy triviales. De esta forma, puede verse si los resultados intermedios son adecuados, así como el resultado final.

¿Es correcto el resultado final? Ejecute el programa utilizando algunas cifras de prueba que sean suficientemente sencillas para poderlas calcular usted mismo y ver si el ordenador obtiene el mismo resultado que usted.

Ahora pruebe algunos números decimales, y estime aproximadamente el resultado antes de intentarlo en el ordenador. Es importante el habituarse a hacer este tipo de comprobaciones en un programa, de forma que pueda darse cuenta en seguida si algo va mal.

El programa «carrera de caballos»

Nuestro segundo programa es más divertido. Vamos a escribir un programa de juego —muy sencillo, pero bien diseñado, con secuencias de títulos y una puntuación que debe guardarse.

Vamos a programar una carrera de caballos. Vamos a tener cuatro caballos haciendo carreras a lo largo de la pantalla, de forma parecida a como la liebre y la tortuga lo hicieron en la página 82, pero con una «pista de carreras» dibujada alrededor de ellos. Invitaremos al jugador a apostar sobre qué caballo ganará la carrera, y a darle a él o a ella una bolsa, por ejemplo de 10 000 ptas. para apostar. Después iremos siguiendo lo que hace el jugador.

A alto nivel, nuestro planteo sería tal como:

Paso 1: Visualización de los títulos y explicación de las reglas.

Paso 2: Establecer una bolsa para apostar y pedir la apuesta.

Paso 3: Visualizar la pista de carreras.

Paso 4: Mover los caballos a lo largo de la pista hasta que un caballo alcance la línea de llegada.
Paso 5: Comprobar el resultado de la apuesta y poner al día la bolsa del jugador.
Paso 6: Realizar otra carrera si el jugador lo desea.
Paso 7: Terminar el programa con un comentario sobre lo bien que lo haya hecho el jugador.

¿Qué variables queremos a nivel de programas? Primero, variables que manejen el dinero. No le daremos ninguna ventaja a ningún caballo. Todos tendrán igual oportunidad de ganar, por lo tanto mantendremos las ventajas a un valor constante de 4 a 1. Nuestra lista de variables será algo así:

Bolsa del jugador: P
Apuesta del jugador en la carrera: S
Caballo sobre el que apuesta el jugador: H

«H» es una variable numérica como las otras, ya que numeraremos a los caballos del 1 al 4. Vamos a darles nombres también, y colocaremos los nombres en una lista de variables de texto:

H$ (1) FURIA
H$ (2) RELAMPAGO
H$ (3) BABIECA
H$ (4) ROCINANTE

He aquí la línea general del programa, utilizando subrutinas para cada paso importante:

```
10  REM ***CARRERA DE CABALLOS***
20  LET P = 10000: REM BOLSA DEL JUGADOR
30  DIM H$(4): REM NOMBRES DE LOS CABALLOS
40  FOR X = 1 TO 4: READ H$(X): NEXT X
50  DATA "FURIA", "RELAMPAGO"
60  DATA "BABIECA", "ROCINANTE"
70  GOSUB 1000: REM TITULO
80  GOSUB 1500: REM APUESTA
90  GOSUB 2000: REM DIBUJO DE LA PISTA DE CARRE-
    RAS
100 GOSUB 2500: REM CARRERA
110 GOSUB 3000: REM COMPROBACION DEL RESULTA-
    DO
120 IF P = 0 THEN GOTO 150
130 INPUT "QUIERE JUGAR OTRA VEZ? S/N";R$
140 IF R$ = "S" THEN CLS: GOTO 80
150 GOSUB 3500: REM SECUENCIA FINAL
```

Quizás esto le parezca demasiado abstracto, pero nosotros no pensamos así. Considérelo como un índice de la estructura del programa. Nos servirá como disciplina cuando empecemos a escribir las subrutinas. Obsérvese que la secuencia final no será estrictamente una subrutina ya que no retornaremos de ella como hacemos con todas las demás.

Empecemos con nuestra primera subrutina, que escribe los títulos y explica las reglas. ¿Qué le parece un título que parpadee? Limpiaremos la pantalla, y dejaremos las palabras «CARRERAS POR ORDENADOR» parpadeando durante algunos segundos. ¿Se acuerda de cómo hacer esto? Limpiaremos alternativamente la pantalla y escribiremos el mensaje, utilizando repetidamente el bucle FOR...NEXT. Cuando termine el bucle, nos aseguraremos de que la pantalla está limpia y proseguiremos con la explicación de las reglas, lo que nos llevará hasta el punto en el que queremos reinicializar el programa, cuando se ejecute de nuevo la carrera. Aquí nos detendremos unos instantes, antes de limpiar la pantalla y volver al programa principal.

Esta subrutina parece dividirse en dos de una forma natural, por lo tanto vamos a programarla así:

```
1000  REM SECUENCIA DE TITULO
1010  CLS
1020  GOSUB 1100: REM PARPADEO DEL TITULO
1030  GOSUB 1200: REM EXPLICACIONES DE LAS
      REGLAS
1040  RETURN
1100  REM PARPADEO DEL TITULO
1110  FOR N=0 TO 200
1120  PRINT@ (8,9), "CARRERAS POR ORDENADOR"
1130  CLS
1140  NEXT N
1150  RETURN
1200  REM REGLAS
1210  PRINT "TIENE 10 000 PTS PARA APOSTAR"
1220  PRINT "APUESTE UNA CANTIDAD DE PTS"
1230  PRINT "AL CABALLO ELEGIDO"
1240  PRINT "APUESTE EN CUANTAS CARRERAS"
1250  PRINT "QUIERA, PERO TIENE QUE APOSTAR"
1260  PRINT "ALGUN DINERO EN CADA APUESTA"
1270  FOR D = 0 TO 5000: NEXT D
```

```
1280  CLS
1290  RETURN ⟶
```

Puede cambiar los comentarios si así lo desea. Pruebe varios valores distintos en el bucle de espera, si fuera necesario, hasta que encuentre uno que le parezca correcto.

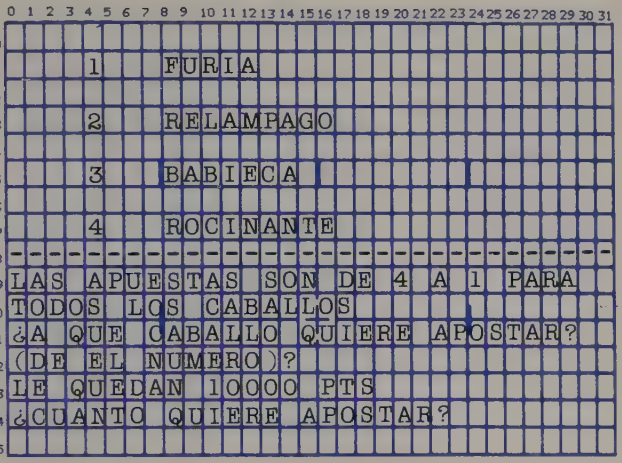
Hemos realizado la «limpieza» en esta subrutina, y nos hemos quedado con una pantalla en blanco, por lo tanto prosigamos con el planteo de la subrutina de apuesta. En ésta, presentaremos los cuatro caballos y preguntaremos al jugador sobre a cuál quiere apostar y a cuánto asciende la apuesta. Tendremos bastantes cosas que colocar en la pantalla, por lo tanto decidamos primero cómo queremos que aparezcan. El dibujo de la siguiente página muestra la disposición que hemos escogido y ésta es la subrutina que lo crea:

```
1500  REM SUBRUTINA DE APUESTA
1510  REM VISUALIZACION DE LOS NOMBRES DE LOS CABALLOS
1520  FOR X = 1 TO 4
1530  PRINT: PRINT "   ";X;"   ";H$(X)
1540  NEXT X
1550  FOR X = 0 TO 31: PRINT"–";:NEXT X
1560  PRINT "LAS APUESTAS SON DE 4 a 1 PARA TODOS LOS CABALLOS"
1570  PRINT "A QUE CABALLO QUIERE APOSTAR?"
1580  INPUT "(DEME UN NUMERO)";H
1590  PRINT "LE QUEDAN";P;"PTS"
1600  INPUT "CUANDO QUIERE APOSTAR";S
1610  IF S>P THEN GOTO 1600
1620  LET P = P – S
1630  CLS
1640  RETURN ⟶
```

Obsérvese que hemos utilizado sentencias de PRINT en blanco para separar los nombres, y puede verse cómo hemos utilizado las variables del programa H, P y S que decidimos anteriormente.

Ahora la siguiente sección: dibujar la pista de carreras. De nuevo, tomemos papel y lápiz para plantear la distribución de la pantalla como hicimos anteriormente. Tendremos que decidir a lo largo de qué filas correrán los cuatro caballos, pero no dibujaremos los ca-

Apuestas en la carrera de caballos
Esta es la distribución de la sección de apuesta. Al jugador se le dice cuánto dinero tiene, y entra el número del caballo que ha escogido y cuánto quiere apostar sobre él.



ballos todavía. En la sección de «carrera», la pista permanecerá constante mientras movemos los caballos. Nuestra versión no es muy elaborada, pero usted puede mejorarla colocando espectadores, árboles o lo que sea. Nuestro dibujo está en la página siguiente.

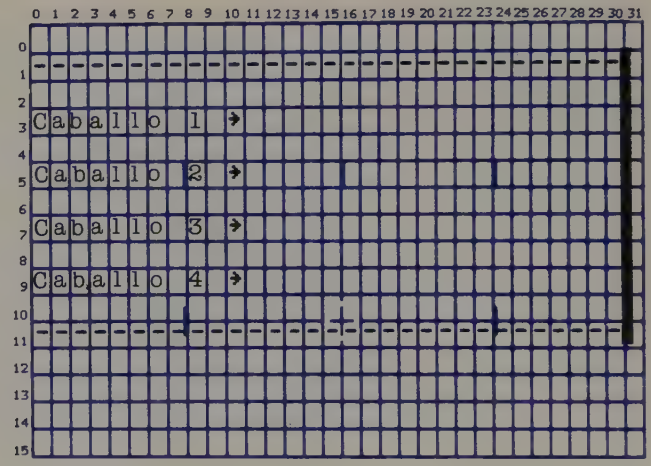
He aquí la subrutina:

```
2000  REM DIBUJO DE LA PISTA DE CARRERAS
2010  FOR X = 0 TO 31: PRINT@ (1,X),"-";:NEXT X
2020  FOR X = 0 TO 31: PRINT@ (11,X),"-";:NEXT X
2030  FOR F = 1 TO 11: PRINT@ (F,31),CHR$(133);:
      NEXT F
2040  RETURN
```

Y ahora la parte realmente importante, ¡la propia carrera! Hay varias cosas que conviene ponderar aquí. Primero, no podemos limpiar sencillamente la pantalla a medida que los caballos se mueven a través de ella, a menos que queramos volver a dibujar la pista de carreras cada vez. En la práctica, esto haría que la visualización fuese demasiado intermitente. Por lo tanto, esperaremos brevemente después de cada conjunto de movimientos y entonces borraremos cada caballo, visualizando un blanco en su lugar, calcularemos su nueva posición y después lo dibujaremos. Segundo, deberemos comprobar si algún caballo ha ganado la carrera. Haremos esto comprobando sencillamente el número de la columna en la cual haremos la próxima

La pista de carreras

Este es nuestro planteo para la distribución en la pantalla de la pista de carreras, mostrando dónde está colocado cada caballo. La línea de llegada es la línea situada en la parte derecha de la pantalla, constituida por 11 cuadrados gráficos.



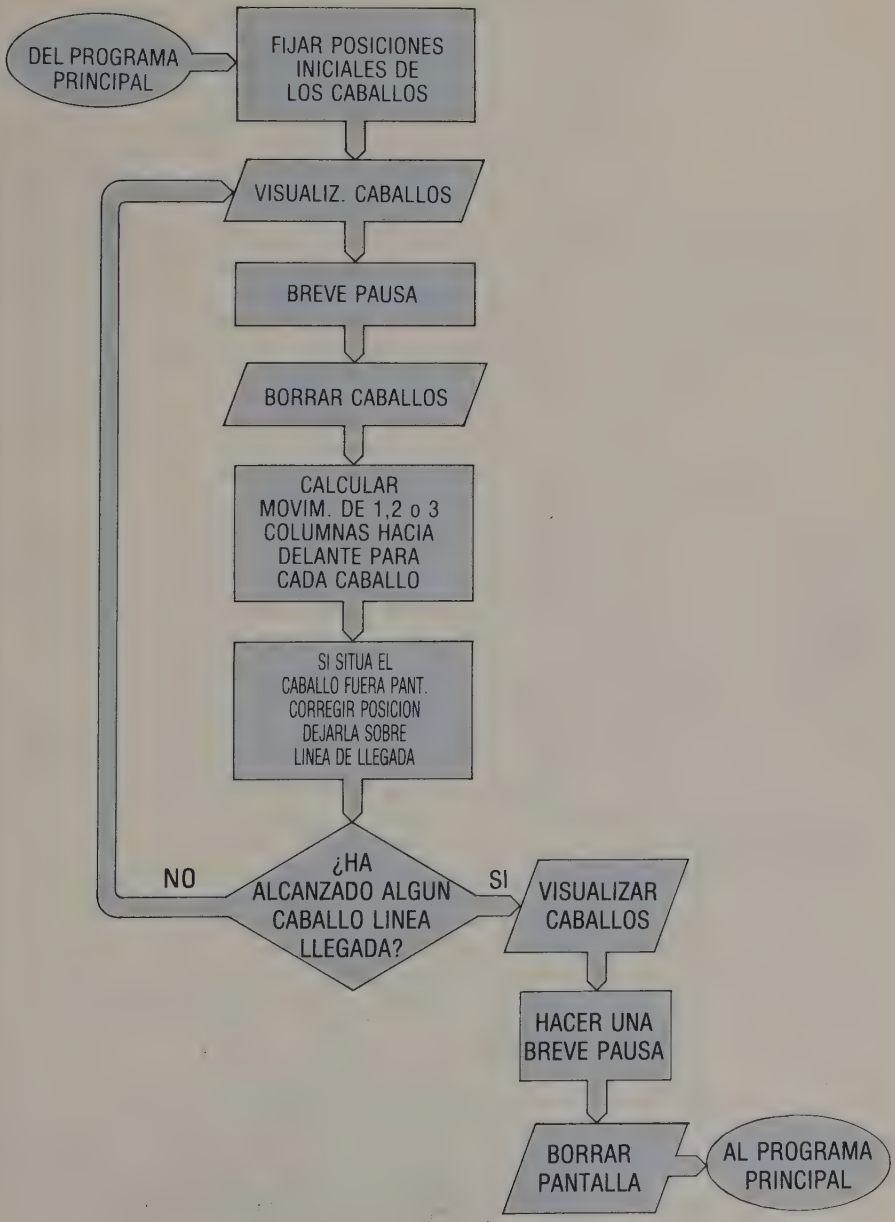
visualización. Si éste es mayor que 31 (suponiendo que nuestros caballos avanzan únicamente una columna) lo dejaremos en 31 de forma que no nos salgamos de la pantalla, y cambiaremos el contenido de la variable «ganador» (W), que habremos colocado a 0 al principio de cada carrera, a 1. Después nos desviaremos hacia otra sub-subrutina, que visualiza el último movimiento de la carrera y retorna de la subrutina. Naturalmente, deberemos comprobar todos los caballos para ver si ha habido algún empate. Si se siente un poco confuso, quizá le ayude el analizar el diagrama de flujo que hemos dibujado para esta sección.

Ahora las variables. Las filas en las cuales visualizamos cada caballo no cambiarán, pero de todas maneras las pondremos en una variable de lista. Las posiciones de columna sí que cambiarán, a medida que los caballos se mueven a través de la pantalla, y necesitaremos otra variable de lista para ellas. Llamemos a estas dos variables de lista, R para las filas, y C para las columnas. Por lo tanto ésta será la lista de variables para la subrutina de la carrera:

Posiciones de fila para cada caballo:	R(1 a 4)
Posiciones de columna para cada caballo:	C (1 a 4)
Variable de ganador (inicialmente a 0):	W

Aquí nos encontramos con un ligero problema, que no hemos discutido anteriormente. No podemos colocar la sentencia DIM para dimensionar nuestras varia-

Diagrama de flujo para la subrutina de la sección de la carrera de «carrera de caballos»



bles de lista, en una subrutina que el programa llamará varias veces. El ordenador dimensionará una lista sólo una vez. Si intentamos repetir la operación, señalará esto como un error. Por lo tanto volvamos atrás, y utilicemos algunos de los números de línea vacíos en nuestra estructura del programa para dimensionar nuestras dos variables de lista, y para entrar (READ) los datos en la variable de lista R. No queremos, en este caso, entrar datos en la variable de lista C, ya que queremos poner a 0 el contenido de C cada vez que volvamos a realizar la carrera, por lo tanto esto debe estar incluido en la subrutina.

Este sería un lugar adecuado para las sentencias:

```
65  DIM R(4): DIM C(4): REM FILAS/COLUMNAS PARA LOS
    CABALLOS
66  REM ELECCION DE LA FILA DE CADA CABALLO
67  FOR X = 1 TO 4: READ R(X): NEXT X
68  DATA 3,5,7,9
```

Un caballo

Permitiremos que cada caballo avance 1,2 o 3 columnas cada vez, utilizando la función RND para seleccionar el número.

Un sencillo carácter gráfico actuará como caballo (véase el dibujo). Ahora probaremos la subrutina:

```
2500  REM CARRERA
2510  LET W = 0: REM VARIABLE DE GANADOR
2520  REM POSICIONES DE LA COLUMNA DE INICIO DE LA
      CARRERA
2530  FOR X = 1 TO 4: LET C(X) = 0: NEXT X
2540  REM BUCLE DE LA CARRERA
2550  FOR X = 1 TO 4
2560  PRINT@ (R(X),C(X)),CHR$(137);
2570  NEXT X
2580  FOR D = 0 TO 50: NEXT D
2590  FOR X = 1 TO 4
2600  PRINT@(R(X),C(X)),"  ";
2610  NEXT X
2620  FOR X = 1 TO 4
2630  LET C(X) = C(X)+1+INT(RND*3)
2640  IF C(X) > 31 THEN LET C(X) = 31: LET W = 1
2650  NEXT X
2660  IF W = 1 THEN GOSUB 2800: RETURN ──►
2670  GOTO 2540
```

```
2800  REM MOVIMIENTO DEL GANADOR
2810  FOR X = 1 TO 4
2820  PRINT@ (R(X),C(X)),CHR$(137);
2830  NEXT X
2840  FOR D = 0 TO 500: NEXT D
2050  CLS
2860  RETURN
```

Esto es suficientemente complicado para ser por sí mismo un programa, ¿no es así? Por lo tanto, será mejor analizarlo separadamente, y no preocuparse por el resto del programa mientras lo realizamos. Si tiene dificultad en seguir esta sección, quizá le sea de ayuda el volver atrás hasta el diagrama de flujo de la página 159, para ver qué función realiza cada parte.

Vayamos ahora a la subrutina «resultado». En esta subrutina haremos:

- Analizar qué caballo o caballos han ganado
- Ver si el jugador ha apostado por el caballo ganador
- Decirle al jugador cuánto ha ganado o perdido
- Poner al día la bolsa del jugador

Necesitaremos otra variable de «ganador» aquí. Ya que la variable de ganador de la última subrutina era una variable de subrutina, que no afecta al resto del programa, por lo tanto, podemos utilizar otra vez W. Debemos acordarnos de ponerla de nuevo a 0. Ya que podemos utilizar distintos mensajes, en función de si el jugador gana o pierde, pongamos dos subrutinas que manejen estas posibilidades.

Vea si puede seguir esta sección. De nuevo, puede utilizar el diagrama de flujo de la «sección de resultado», que está en la página 163, si necesita ayuda.

```
3000  REM SUBRUTINA DEL RESULTADO
3010  REM VER SI EL JUGADOR HA GANADO
3020  LET W = 0: REM VARIABLE DEL GANADOR
3030  FOR X = 1 TO 4
3040  IF C(X) = 31 THEN IF H = X THEN LET W = 1
3050  NEXT X
3060  IF W = 1 THEN GOSUB 3300
3070  IF W = 0 THEN GOSUB 3400
3080  LET P = P + S
3090  RETURN
3300  REM RUTINA SI EL JUGADOR HA GANADO
```

```
3310  LET S = S*4
3320  PRINT@ (6,10), "FELICITACIONES!"
3330  PRINT H$(H); "HA GANADO LA CARRERA"
3340  PRINT "HA GANADO"; S; "PTS"
3350  FOR D = 0 TO 1000: NEXT D
3360  CLS
3370  RETURN ——→
→3400 REM RUTINA SI EL JUGADOR HA PERDIDO
3410  LET S = 0
3420  PRINT@ (6,10), "MALA SUERTE"
3430  PRINT H$(H); "NO HA GANADO"
3440  FOR D = 0 TO 1000: NEXT D
3450  CLS
3460  RETURN ——→
```

Ya casi está terminado. Tan sólo nos queda realizar la secuencia final. Veamos qué es lo que ha hecho el jugador, y visualicemos un mensaje adecuado para el final del juego.

```
→3500 REM SECUENCIA FINAL
3510  IF P = 0 THEN GOTO 3600 ——→
3520  IF P < 10000 THEN GOTO 3700 ——→
3530  IF P > 10000 THEN GOTO 3800 ——→
→3600 REM BANCARROTA
3610  PRINT@ (6,10), "BANCARROTA!"
3620  PRINT "MEJOR SUERTE LA PROXIMA VEZ"
3630  END
→3700 REM EL JUGADOR HA PERDIDO, NO HAY BANCA-
      RROTA
3710  PRINT@ (6,10), "NO DEMASIADO BIEN"
3720  PRINT "SU BOLSA HA BAJADO A";P;"PTS"
3730  PRINT "MEJOR SUERTE LA PROXIMA VEZ"
3740  END
→3800 REM EL JUGADOR HA GANADO
3810  PRINT@ (6,10), "MUY BIEN"
3820  PRINT "DEJELO AHORA QUE ESTA GANANDO"
3830  PRINT "AHORA TIENE";P;"PTS"
3840  END
```

Una vez más, el programa completo está descrito en las páginas 167-170. Analice cuidadosamente el listado completo, para ver cómo es, una vez ha sido ensamblado. Y no olvide el comprobarlo completamente, después que lo haya entrado en su ordenador.

Diagrama de flujo de la subrutina de resultado de «carrera de caballos»

Cómo colocar programas largos en su ordenador

Vamos a terminar el capítulo con una pregunta que será particularmente importante para usted, cuando escriba programas largos. ¿Cómo es de grande su ordenador? El aspecto realmente importante es la cantidad de memoria de acceso aleatorio, o RAM, que tenga. Este determina la cantidad de espacio disponible para guardar su programa y para manejar la ejecución del mismo. Los ordenadores domésticos más pequeños tienen únicamente 1K (un K es 1024 bytes) de RAM. Uno grande tiene 64K, y los ordenadores más grandes naturalmente pueden llegar a tener memorias mucho más grandes. ¿El tamaño de la memoria es realmente algo que deba preocuparle?

Si su ordenador tiene una memoria muy pequeña, realmente sí que debe preocuparle, ya que se encontrará con que la llenará muy rápidamente. Por ejemplo, los programas de este capítulo no cabrán en 1K; ¿qué tamaño de memoria necesitan? Esto depende en gran medida de cómo su ordenador organice su memoria, y también de la forma exacta en que usted escriba el programa. Nosotros lo hemos probado limitando nuestro ordenador de 16K a tan sólo 3,5K de memoria útil y se han ejecutado sin ningún problema.

Si se queda sin espacio de memoria cuando esté entrando o ejecutando su programa, el ordenador se lo hará saber dándole un mensaje de error de «Out of Memory» (memoria llena). Si sucede esto ¿qué puede hacerse?

Bueno, pueden eliminarse las partes del programa que utilizan espacio de memoria innecesariamente. He aquí alguna sugerencia:

1) En la mayoría de los ordenadores, cada espacio que se teclea ocupa un espacio de memoria, incluso aunque el ordenador ignore los espacios cuando traduce la sentencia de BASIC. Elimine cuantos espacios le permita el ordenador. (Estos serán muchos, quizás todos los espacios de la lista.)

2) Los comentarios (REM) ocupan espacio. Hemos utilizado los REM como nuestro método principal para documentar programas ya que es sencillo y fácil de comprender y, porque utilizándolos, no puede nunca perderse sus explicaciones. Sin embargo, si se queda corto de espacio, elimine las sentencias REM en la versión que entrará en el ordenador. Manténgalas —u

otras explicaciones detalladas de cómo funciona su programa— en forma escrita, como referencia.

3) Los números de línea ocupan espacio. Si su ordenador se lo permite, coloque varias sentencias en cada línea, en cualquier lugar adecuado donde pueda hacerlo sin cambiar la estructura del programa. No aparece de una forma tan clara cuando se hace el listado del programa, pero el programa se ejecutará igualmente.

Pronto tendrá una idea del tamaño de los programas que su ordenador puede ejecutar. Si su ordenador tiene más de 16K de RAM, probablemente no tendrá nunca que preocuparse acerca del espacio de memoria cuando ejecute sus propios programas, tan sólo lo hará cuando compre programas ya hechos. Si el programa no cabe, incluso «apretándole el cinturón», entonces tan sólo le queda reducir la escala de sus ambiciones, comprar alguna memoria extra para añadir a su ordenador, ¡o comprar un ordenador más grande!

Listados de los programas

A continuación están los listados completos de ambos programas que hemos analizado en este capítulo. A continuación de ellos está nuestro sumario usual de comprobación.

Programa de empapelar

Variables utilizadas

Area que tiene que ser empapelada:	A
Area de un rollo de papel:	R
Número de rollos necesarios:	N
Variable de subrutina:	
Dimensiones de las áreas/y huecos:	H,W
Dimensiones del rollo de papel:	L,W
Margen de error:	M
Respuesta sí/no a las preguntas:	R$

```
10  REM ***EMPAPELAR***
20  REM POR SUSAN CURRAN
30  PRINT "ESTE PROGRAMA CALCULA LA"
40  PRINT "CANTIDAD DE PAPEL NECESARIO"
50  PRINT "PARA EMPAPELAR UNA HABITACION"
60  PRINT "DEME TODAS LAS MEDIDAS"
```

```
70   PRINT "EN LA MISMA UNIDAD"
80   REM EL AREA A EMPAPELAR ES A
90   REM EL AREA DE UN ROLLO DE PAPEL ES R
100  REM EL NUMERO DE ROLLOS NECESARIOS ES N
110  GOSUB 1000: REM OBTENER A
120  GOSUB 2000: REM OBTENER R
130  GOSUB 3000: REM OBTENER N
140  PRINT "NUMERO DE ROLLOS NECESARIOS ES";N
150  END
1000 REM SUBRUTINA PARA CALCULAR EL AREA
1010 REM QUE TIENE QUE EMPAPELARSE
1020 PRINT "LAS AREAS A EMPAPELAR TIENEN QUE"
1030 PRINT "SER DESCRITAS COMO RECTANGULOS CON"
1040 PRINT "HUECOS RECTANGULARES. DEME PRIMERO"
1050 PRINT "TODAS LAS AREAS Y DESPUES TODOS LOS
     HUECOS"
1060 LET A = 0
1070 GOSUB 1500: REM CALCULO TOTAL DE TODAS LAS
     AREAS
1080 GOSUB 1700: REM CALCULO DEL AREA TOTAL
     MENOS LOS HUECOS
1090 RETURN
1500 REM SUBRUTINA PARA CALCULAR EL TOTAL DE
     TODAS LAS AREAS
1510 INPUT "PRIMERA DIMENSION DEL AREA";H
1520 INPUT "SEGUNDA DIMENSION DEL AREA";W
1530 LET A = A+(H*W)
1540 INPUT "MAS AREAS? S/N";R$
1550 IF R$ = "S" THEN GOTO 1510
1560 RETURN
1700 REM SUBRUTINA PARA CALCULAR EL AREA TOTAL
     MENOS LOS HUECOS
1710 INPUT "PRIMERA DIMENSION DEL HUECO";H
1720 INPUT "SEGUNDA DIMENSION DEL HUECO";W
1730 LET A = A–(H*W)
1740 INPUT "MAS HUECOS? S/N";R$
1750 IF R$ = "S" THEN GOTO 1710
1760 RETURN
2000 REM SUBRUTINA PARA CALCULAR EL AREA
2010 REM DE UN ROLLO DE PAPEL
2020 INPUT "LONGITUD DE UN ROLLO DE PAPEL";L
2030 INPUT "ANCHURA DEL ROLLO";W
2040 LET R = L*W
2050 RETURN
```

```
3000 REM SUBRUTINA PARA CALCULAR
3010 REM EL NUMERO DE ROLLOS NECESARIOS
3020 INPUT "DEME UN MARGEN DE ERROR EN %";M
3030 LET N = (A*(M+100))/(R*100)
3040 IF N > INT(N) THEN LET N = 1+INT(N)
3050 RETURN
```

Programa de carrera de caballos

Variables utilizadas

Bolsa del jugador:	P
Apuesta del jugador en la carrera:	S
Caballo sobre el que apuesta el jugador:	H
Nombres de los caballos:	H$(1 a 4)
Filas en las que los caballos corren:	R(1 a 4)
Posiciones de columna para los caballos:	C(1 a 4)
Variable de ganador (subrutina):	W
Contadores de bucle:	X, N, D (bucle de espera)
Elección de la repetición del juego:	R$

```
10   REM ***CARRERA DE CABALLOS***
20   LET P = 10000: REM BOLSA DEL JUGADOR
30   DIM H$(4): REM NOMBRES DE LOS CABALLOS
40   FOR X = 1 TO 4: READ H$ (X):NEXT X
50   DATA "FURIA", "RELAMPAGO"
60   DATA "BABIECA", "ROCINANTE"
65   DIM R(4): DIM C(4): REM FILAS/COLUMNAS PARA LOS
     CABALLOS
66   REM ELECCION DE LA FILA DE CADA CABALLO
67   FOR X = 1 TO 4: READ R(X): NEXT X
68   DATA 3,5,7,9
70   GOSUB 1000: REM TITULO ——→
80   GOSUB 1500: REM APUESTA
90   GOSUB 2000: REM DIBUJO DE LA PISTA DE CA-
     RRERAS
100  GOSUB 2500: REM CARRERA
110  GOSUB 3000: REM COMPROBACION DEL
     RESULTADO
120  IF P = 0 THEN GOTO 150
130  INPUT "QUIERE JUGAR OTRA VEZ? S/N";R$
```

```
140 IF R$ = "S" THEN CLS: GOTO 80
150  GOTO 3500: REM SECUENCIA FINAL
1000 REM SECUENCIA DE TITULO
1010 CLS
1020 GOSUB 1100: REM PARPADEO DEL TITULO
1030 GOSUB 1200: REM EXPLICACIONES DE LAS REGLAS
1040 RETURN
1100 REM PARPADEO DEL TITULO
1110 FOR N = 0 TO 200
1120 PRINT@ (8,9), "CARRERAS POR ORDENADOR"
1130 CLS
1140 NEXT N
1150 RETURN
1200 REM REGLAS
1210 PRINT "TIENE 10 000 PTAS PARA APOSTAR"
1220 PRINT "APUESTE UNA CANTIDAD DE PTS"
1230 PRINT "AL CABALLO ELEGIDO"
1240 PRINT "APUESTE EN CUANTAS CARRERAS"
1250 PRINT "QUIERA, PERO TIENE QUE APOSTAR"
1260 PRINT "ALGUN DINERO EN CADA APUESTA"
1270 FOR D = 0 TO 5000: NEXT D
1280 CLS
1290 RETURN
1500 REM SUBRUTINA DE APUESTA
1510 REM VISUALIZACION DE LOS NOMBRES DE LOS CA-
     BALLOS
1520 FOR X = 1 TO 4
1530 PRINT: PRINT "   ";X;"   ";H$(X)
1540 NEXT X
1550 FOR X = 0 TO 31: PRINT"–";:NEXT X
1560 PRINT "LAS APUESTAS SON DE 4 A 1 PARA TODOS
     LOS CABALLOS"
1570 PRINT "A QUE CABALLO QUIERE APOSTAR?"
1580 INPUT "(DEME UN NUMERO)";H
1590 PRINT "LE QUEDAN";P;"PTS"
1600 INPUT "CUANTO QUIERE APOSTAR?";S
1610 IF S>P THEN GOTO 1600
1620 LET P = P – S
1630 CLS
1640 RETURN
2000 REM DIBUJO DE LA PISTA DE CARRERAS
2010 FOR X = 0 TO 31: PRINT@ (1,X),"–";:NEXT X
2020 FOR X = 0 TO 31: PRINT@ (11,X),"–";:NEXT X
2030 FOR F =1 TO 11: PRINT@ (F,31),CHR$(133);: NEXT F
```

```
2040 RETURN
2500 REM CARRERA
2510 LET W = 0: REM VARIABLE DE GANADOR
2520 REM POSICIONES DE LA COLUMNA DE INICIO DE LA
     CARRERA
2530 FOR X = 1 TO 4: LET C(X) = 0: NEXT X
2540 REM BUCLE DE LA CARRERA
2550 FOR X = 1 TO 4
2560 PRINT@ (R(X),C(X)),CHR$(137);
2570   NEXT X
2580   FOR D = 0 TO 50: NEXT D
2590   FOR X = 1 TO 4
2600   PRINT@ (R(X),C(X))," ";
2610 NEXT X
2620 FOR X = 1 TO 4
2630 LET C(X) = C(X)+1+INT(RND*3)
2640 IF C(X) > 31 THEN LET C(X) = 31: LET W = 1
2650 NEXT X
2660 IF W = 1 THEN GOSUB 2800: RETURN
2670 GOTO 2540
2800 REM MOVIMIENTO DEL GANADOR
2810 FOR X = 1 TO 4
2820 PRINT@ (R(X),C(X)),CHR$(137);
2830 NEXT X
2840 FOR D = 0 TO 500: NEXT D
2850 CLS
2860 RETURN
3000 REM SUBRUTINA DEL RESULTADO
3010 REM VER SI EL JUGADOR HA GANADO
3020 LET W = 0: REM VARIABLE DEL GANADOR
3030 FOR X = 1 TO 4
3040 IF C(X) = 31 THEN IF H = X THEN LET W + 1
3050 NEXT X
3060 IF W = 1 THEN GOSUB 3300
3070 IF W = 0 THEN GOSUB 3400
3080 LET P = P + S
3090 RETURN
3300 REM RUTINA SI EL JUGADOR HA GANADO
3310 LET S = S*4
3320 PRINT@ (6,10), "FELICITACIONES!"
3330 PRINT H$(H); "HA GANADO LA CARRERA"
3340 PRINT "HA GANADO";S;"PTS"
3350 FOR D = 0 TO 1000: NEXT D
3360 CLS
```

```
3370 RETURN
3400 REM RUTINA SI EL JUGADOR HA PERDIDO
3410 LET S = 0
3420 PRINT@ (6,10), "MALA SUERTE"
3430 PRINT H$(H); "NO HA GANADO"
3440 FOR D = 0 TO 1000: NEXT D
3450 CLS
3460 RETURN
3500 REM SECUENCIA FINAL
3510 IF P = 0 THEN GOTO 3600
3520 IF P < 10000 THEN GOTO 3700
3530 IF P > 10000 THEN GOTO 3800
3600 REM BANCARROTA
3610 PRINT@ (6,10), "BANCARROTA!"
3620 PRINT "MEJOR SUERTE LA PROXIMA VEZ"
3630 END
3700 REM EL JUGADOR HA PERDIDO, NO HAY BANCA-
     RROTA
3710 PRINT@ (6,10), "NO DEMASIADO BIEN"
3720 PRINT "SU BOLSA HA BAJADO A" ;P;"PTS"
3730 PRINT "MEJOR SUERTE LA PROXIMA VEZ"
3740 END
3800 REM EL JUGADOR HA GANADO
3810 PRINT@ (6,10), "MUY BIEN"
3820 PRINT "DEJELO AHORA QUE ESTA GANANDO"
3820 PRINT "AHORA TIENE";P;"PTS"
3840 END
```

Comprobación

En este capítulo, tan sólo hemos podido darle una visión muy breve de un aspecto muy complicado; por lo tanto, no será sorprendente si usted no está todavía absolutamente seguro sobre el hecho de escribir programas largos por usted mismo. Quizá quiera concentrarse en observar programas hechos por otra gente, incluyendo los que están en el próximo capítulo, antes de escribir muchos programas propios. Estos son los aspectos que hemos cubierto.

Sobre programas estructurados

– El concepto de programación estructurada, dividiendo un programa en módulos manejables.

- Métodos sencillos para enlazar módulos entre sí, utilizando la estructura de subrutina.
- Documentación de un programa estructurado.
- Distinción entre variables de programa y variables de subrutina.

Sobre las órdenes GOSUB...RETURN
- Cómo se utilizan estas órdenes para evitar el control de programa a una subrutina y retornar desde ella.

Sobre diagramas de flujo
- Cómo diseñar un conjunto de diagramas de flujo que ilustren el planteo de un programa estructurado.

Sobre comprobación
- La importancia de mostrar resultados intermedios en los programas aritméticos.
- La técnica de utilizar ejemplos sencillos o de hacer estimaciones aproximadas parecidas a lo que el programa dará como resultado.

Sobre el manejo de la memoria
- Cómo reacciona el ordenador cuando su memoria está llena.
- Técnicas sencillas para reducir el tamaño de un programa.

8 Los siguientes pasos

En este capítulo veremos:
- **Cómo el ordenador maneja el BASIC.**
- **Algunos lenguajes alternativos.**

Ahora, que ya ha aprendido suficiente sobre el BASIC para poder escribir programas sencillos, ¿qué debe hacer a continuación? En este capítulo final, intentaremos explicarle las opciones que tiene.

En este libro no hemos cubierto totalmente el lenguaje BASIC. Nos hemos concentrado en un subconjunto de él que es razonablemente común para todos los ordenadores. Todas las versiones tienen más órdenes de las que hemos mencionado, y el primer lugar donde buscar acerca del resto de las mismas es en el manual de su ordenador.

¿Tiene ya un ordenador? Si no, y si cree que le gustaría hacer algo de programación, entonces necesitará hacerse con uno antes de ir mucho más lejos. No puede pretender el llegar a ser un buen programador (o incluso uno pasable) sin un ordenador sobre el que trabajar.

Hay muchos libros especializados, escritos para describir la programación en BASIC, en los ordenadores domésticos más populares. Le sugerimos que si puede encontrar un libro que trate sobre su propia máquina, éste sería el mejor paso que puede realizar a continuación. Intentando permanecer en un terreno común, hemos tenido que prescindir de muchas características y órdenes —por ejemplo las que tratan con el sonido y color— que hacen tan divertidos a los ordenadores domésticos.

Si tiene una máquina muy nueva, o una no muy conocida, entonces quizá tenga dificultad en encontrar un libro que trate sobre ella. Le sugerimos algunos libros generales que traten sobre Microsoft BASIC en la página 199. Si su máquina no puede ejecutar el Microsoft, entonces un libro que trate sobre una máquina distinta con la misma, o similar, forma de BASIC (PET BASIC

o Atari BASIC, por ejemplo) sería un camino mejor.

Puede aprender mucho leyendo e intentando programas hechos por otra gente, incluso aunque encuentre que no le gusta su estilo. Muchas revistas de ordenadores personales publican listados. Algunos especializados en ordenadores en particular. Déles un vistazo y vea para qué máquinas están diseñados. Si su ordenador no es exactamente el mismo que el modelo de la revista, prepárese para una larga sesión de depuración. Léase primero el programa entero, y analícelo para ver dónde puede encontrarse con problemas —por ejemplo, el programa puede estar diseñado para una pantalla de distinto tamaño o puede utilizar una orden especial que su BASIC no tiene—. Asegúrese de que sabe cómo resolver las dificultades, antes de pasarse una hora o más delante de la pantalla.

El comprar programas ya codificados en cassette o en disco, ciertamente le ayuda a aprender lo que su ordenador es capaz de hacer. No sea orgulloso e intente hacer todos los programas por usted mismo —invierta en algunos programas profesionales realmente buenos—. Pero no espere ser capaz de obtener un listado y aprender los trucos que ellos utilizan. La mayoría de los profesionales emplean trucos de programación que detienen el listado o copia de sus programas.

A medida que vaya aprendiendo el BASIC, empezará a descubrir las limitaciones del mismo, así como su fuerza. ¿Qué limitaciones? He aquí algunas de ellas:

–El BASIC es lento. Está bien para juegos muy sencillos y para rutinas gráficas tal como las que hemos escrito hasta ahora, pero su lentitud es un verdadero inconveniente para programas que contengan muchos cálculos matemáticos, o programas que construyan complejas visualizaciones animadas en la pantalla.

– El BASIC tiene herramientas de estructuración relativamente pobres. Como resultado, los programas en BASIC muy largos pueden ser realmente difíciles de escribir y de entender una vez han sido escritos.

– El BASIC tiene capacidades de manejo de archivos muy limitada. Tan sólo se puede guardar un archivo de datos muy sencillo en un disco o cassette mediante instrucciones BASIC.

– El BASIC no tiene posibilidades para programar aplicaciones de control, es decir, aplicaciones tales

como controlar la temperatura de su invernadero mediante su ordenador doméstico, etc.

Cuando quiera hacer algo que el BASIC no hace bien, entonces la solución es utilizar otro lenguaje. ¿Qué otro lenguaje? Antes de mencionar alguno, veamos brevemente en qué forma el ordenador maneja el BASIC.

Cómo trata el ordenador un programa de BASIC

Las habilidades fundamentales de su ordenador son muy limitadas. Puede tan sólo utilizar un pequeño conjunto de operaciones de movimiento de símbolos, tales como datos, sumar números, restarlos y comparar los conjuntos de símbolos. Los circuitos que permiten al ordenador hacer estas funciones están construidos dentro de la unidad de proceso central (en los microordenadores, su chip procesador).

Los fabricantes de ordenadores utilizan este conjunto básico de operaciones para hacer los programas, dando al ordenador la posibilidad de hacer otras funciones, tal como codificar las teclas que usted aprieta en el teclado y visualizar la información en la pantalla de televisión. Las instrucciones que permiten al ordenador hacer estas funciones constituyen su sistema operativo. Si su ordenador puede guardar y cargar datos desde y hacia un cassette, entonces su sistema operativo será un «sistema operativo de cassette». Si contiene discos duros o flexibles, entonces necesitará un «sistema operativo de disco». Generalmente las instrucciones del sistema operativo están contenidas dentro del ordenador, en la memoria ROM (Memoria de Sólo-Lectura). A veces los sistemas operativos de disco, al menos parcialmente, se cargan desde un disco flexible.

Quizá sin darse cuenta, usted ya ha utilizado órdenes que forman parte del sistema operativo de su ordenador. «LOAD» y «SAVE» están entre ellas; no forman parte estrictamente de lenguaje BASIC.

El sistema operativo constituye una gran parte de lo que se conoce como el software de sistema del ordenador. También en este software, ya que usted tiene un ordenador que entiende BASIC, habrá un interpretador de BASIC.

Puede tomarse el interpretador de BASIC como algo parecido a un intérprete humano. Usted no habla el

lenguaje propio de su ordenador, el lenguaje máquina, cuando programa usted en BASIC. En su lugar, le pasa las sentencias en BASIC a un programa interpretador, que tiene la tarea de traducirlas al código máquina —estas series de ceros y unos que mencionamos en la introducción—. Por lo tanto, cada sentencia de BASIC que usted entra es transformada en series de instrucciones para el procesador central, direcciones de memoria y datos con los que el ordenador puede trabajar directamente.

Eliminación del interpretador

Aunque el interpretador hace un buen trabajo, su utilización no es tan rápida o eficiente como el hablar directamente al ordenador en su propio lenguaje. El problema estriba en que el lenguaje del ordenador es menos fácil de aprender, y mucho menos fácil el programar con él. Sin embargo, si usted empieza a hacer mucha programación, querrá intentar el trabajar en código máquina, quizás utilizando un programa «ensamblador», que le permite escribir palabras nemotécnicas como ADD o LDA para representar instrucciones como «sumar el número que hay en el acumulador con uno que le voy a dar» o «guardar este número en el acumulador», en lugar de series de ceros y unos.

Otros lenguajes de alto nivel

Incluso cuando llegue a ser un experto, la programación en código máquina es laboriosa. Una forma alternativa de esquivar las limitaciones del BASIC es el utilizar otros lenguajes de alto nivel. Se trata de lenguajes desarrollados como el BASIC, para pasar instrucciones al ordenador a través de sus propios interpretadores, o a través de programas similares llamados compiladores. (Hablaremos de los compiladores más adelante.)

Los siguientes son los lenguajes alternativos más comunes que pueden ser soportados por los ordenadores domésticos.

Forth, un lenguaje nuevo que es actualmente muy interesante. Sus funciones aritméticas, que utilizan la extraña notación Polaca —un tipo de notación invertida—, pueden ser un inconveniente. Su gran ventaja es su gran velocidad.

Pascal, un lenguaje estructurado, muy adecuado para escribir programas largos.

LOGO, un lenguaje sencillo (al menos en algunas

versiones), particularmente útil para ayudar a los niños a aprender sobre ordenadores.

Si usted desea escribir tipos particulares de programas para su ordenador, es ciertamente interesante el buscar un lenguaje que sea adecuado al tipo de programa que tiene en mente. Hable con su proveedor, o lea revistas de ordenadores para descubrir qué lenguajes están disponibles para su ordenador, y para escoger uno que sea adecuado para usted y para su aplicación.

Utilización de un compilador

Una última nota sobre compiladores. Compiladores e interpretadores ofrecen caminos alternativos para traducir las instrucciones de alto nivel, que usted escribe en el código máquina del ordenador. El interpretador traduce sentencia a sentencia, mientras se está ejecutando el programa. El compilador traduce la totalidad del programa de una vez, antes de que cualquiera de ellas sea ejecutada.

El BASIC es primariamente un lenguaje interpretado. La ventaja principal es que usted puede editar las sentencias que escribe sin ningún problema, si encuentra que su programa no funciona o si tan sólo quiere mejorarlo. El compilar un programa puede hacer que se ejecute mucho más rápido, pero también lo transforma en un formato que no es editable directamente. Por ello los lenguajes que son primariamente compilados, como el FORTRAN, son más difíciles de utilizar.

Sin embargo, usted puede comprar compiladores que funcionen con programas BASIC. Para utilizarlos, primero depura y corrige su programa utilizando el interpretador, y después lo pasa a través del compilador. Posteriormente puede ejecutar la versión compilada, guardando el código fuente —es decir, lo que escribió al principio— en un lugar separado por si necesita consultarlo o corregirlo más adelante. Si la velocidad es su problema principal, entonces el comprar un compilador de BASIC puede ser la solución para usted.

Para resumir lo que estamos tratando de decirle... ¡no ha llegado todavía al final de la carretera! De hecho, apenas ha empezado, y la parte más interesante del viaje está todavía por llegar, ya que ahora usted

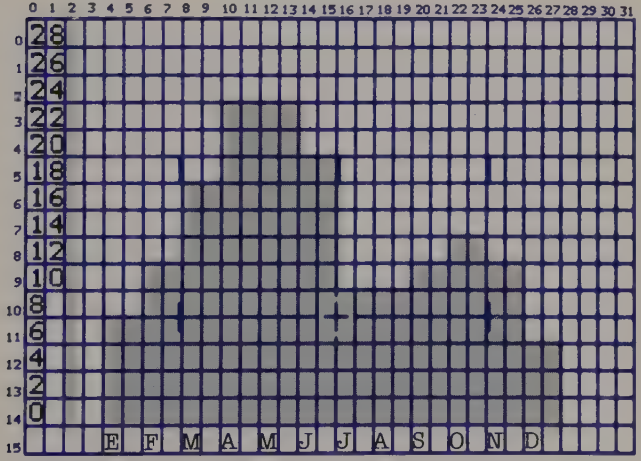
está listo para empezar a planear y escribir programas por sí mismo; para hacer las cosas que usted quiere hacer en su ordenador.

No vamos a realizar ningún programa más con usted. Pero a continuación de este capítulo, encontrará los listados de unos cuantos programas cortos más. Se trata de unos muy sencillos que siguen utilizando únicamente las pocas órdenes de BASIC que hemos analizado en detalle. Pretenden darle ideas, y no es necesario que se utilicen exactamente en la forma que están, sin ningún cambio. Construya con ellos, en otros listados de programa que usted se encontrará y mediante sus propias ideas, y pronto escribirá programas con los que pueda trabajar y jugar en el futuro...

Gráfico de barras

Este programa genera un sencillo gráfico de barras utilizando los datos de cada mes del año. Genera un escalado adecuado automáticamente (más bien un poco tosco) mediante el escalado del más alto de los valores dados. Encontrará que funciona mejor con valores enteros por debajo de 100.

Disposición típica de la pantalla para un «gráfico de barras»



Variables utilizadas

Variable de lista para los datos mensuales: M(1 a 12)

El artículo más grande: L

Contadores de bucle:	X,N
Escala:	S
Eje horizontal:	H
Eje vertical:	V
Número de la escala: (vertical)	T
Columna para visualizar cada artículo:	P

```
10   REM ***GRAFICO DE BARRAS***
20   REM POR SUSAN CURRAN
30   DIM M(12): REM DATOS MENSUALES
40   PRINT "ENTRAR LOS DATOS DE UN MES"
50   PRINT "EMPEZANDO POR ENERO"
60   FOR X = 1 TO 12
70   INPUT M(X)
80   NEXT X
90   REM ENCONTRAR EL VALOR MAS GRANDE
100  LET L = 0
110  FOR X = 1 TO 12
120  IF M(X)>L THEN LET L = M(X)
130  NEXT X
140  LET S = INT(L/14)+1: REM ESCALA
150  REM DIBUJO DE LOS EJES
155  CLS
160  FOR H = 4 TO 27
170  PRINT@ (14,H),CHR$(128);:REM ROTULOS DE LOS
     EJES
180  NEXT H
190  PRINT@ (15,3), "E F M A M J J A S O N D"
200  LET T = S*14:REM MARCADO DE LA ESCALA
210  FOR V = 0 TO 14
220  PRINT@ (V,3),CHR$(133);: REM EJE VERTICAL
230  PRINT@ (V,0),T;
240  LET T = T - S
250  NEXT V
300  REM DIBUJO DE LOS DATOS
310  LET P = 4: REM DIBUJO DE LA COLUMNA
320  FOR N = 1 TO 12
330  FOR X = 13 TO 14-(INT(M(N)/S)) STEP -1
340  PRINT@(X,P), CHR$(128)+CHR$(128)
350  NEXT X
360  LET P = P + 2
370  NEXT N
380  GOTO 380
```

Balance del talonario de cheque

Una versión muy sencilla, pero no debe tener problemas en elaborar otras a partir de la idea.

Variables utilizadas

Balance:	B
Reintegros:	D
Créditos:	C

```
10  REM ***BALANCE DEL TALONARIO DE CHEQUES***
20  INPUT "ENTRA EL ULTIMO BALANCE";B
30  PRINT "ENTRA LOS REINTEGROS"
40  PRINT "DESPUES DE LA ULTIMA CANTIDAD, ENTRA
    UN 0"
50  INPUT D
60  LET B = B - D
70  IF D = 0 THEN GOTO 90
80  GOTO 50
90  PRINT "ENTRA LOS CREDITOS"
100 PRINT "DESPUES DE LA ULTIMA CANTIDAD, ENTRA
    UN 0"
110 INPUT C
120 LET B = B + C
130 IF C = 0 THEN GOTO 150
140 GOTO 110
150 PRINT "BALANCE ACTUAL: ";B
```

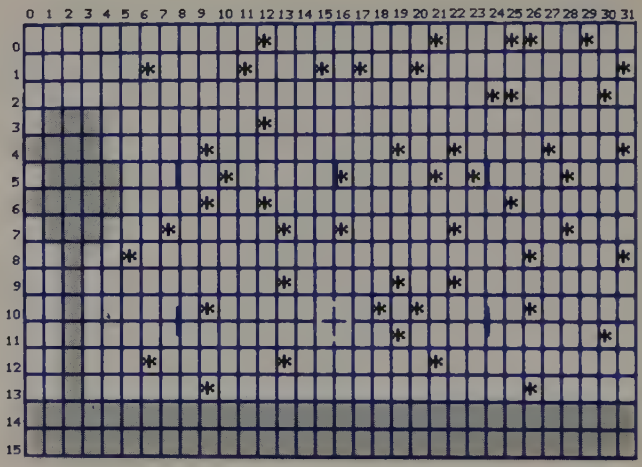
Estrellas

Una rutina gráfica muy sencilla para que usted la pruebe.

Variables utilizadas

Números de fila:	R
Números de columna:	C
Bloques de texto:	
x2:	A$
x3:	B$
Contador de bucle:	N

Aspecto de la pantalla para el programa «estrellas»



```
10   REM ***ESTRELLAS***
20   CLS
30   LET A$ = CHR$(128)+CHR$(128)
40   LET B$ = CHR$(128)+CHR$(128)+CHR$(128)
50   REM DIBUJO DEL SUELO
60   FOR R = 14 TO 15
70   FOR C = 0 TO 31
80   PRINT@ (R,C),CHR$(128);
90   NEXT C
100  NEXT R
110  REM DIBUJO DEL ARBOL
120  FOR R = 8 TO 13
130  PRINT@ (R,2), CHR$(128);
140  NEXT R
150  PRINT@ (3,1),B$;
160  PRINT@ (4,0),A$+A$;
170  PRINT@ (5,1),A$+A$
180  PRINT@ (6,0),A$+B$;
190  PRINT@ (7,1),B$;
200  REM SALEN LAS ESTRELLAS
210  FOR N = 0 TO 50
220  PRINT@ (INT(RND*14),INT(RND*27)+5),"*";
230  NEXT N
240  GOTO 240
```

Cohete

Obsérvese cómo el cohete va borrando su propia cola, dejando un sendero de blancos detrás de él.

Variables utilizadas

Números de fila:	R
Números de columna:	C
Bloques de texto:	
x1:	A$
x2:	B$

```
10   REM ***COHETE***
20   LET A$ = CHR$(128)
30   LET B$ = CHR$(128)+CHR$(128)
40   LET R = 12: LET C = 1
50   CLS
60   FOR C = 10 TO 0 STEP -1
70   PRINT@ (5,15),C;
80   FOR D = 0 TO 460: NEXT D
90   NEXT C
100  PRINT@ (5,10), "DESPEGUE!!"
110  PRINT@ (R,C+1),B$;
120  PRINT@ (R+1,C),A$+B$;
130  PRINT@ (R+2,C-1),"   ";A$;
140  PRINT@ (R+3,C)," ";
150  LET R = R-1: LET C = C+1
160  IF R = -1 THEN GOTO 180
170  GOTO 110
180  CLS: END
```

Diario

Una indicación de cómo pueden utilizarse las variables de lista para crear un sencillo programa «diario». Ya que los datos se guardan en el programa mediante sentencias DATA, no son destruidos cuando se vuelve a ejecutar el programa. Para cambiar los datos, tan sólo hay que volver a teclear la sentencia de DATA. Obsérvese la necesidad de rellenar las sentencias de datos con 0.

No hemos utilizado la expresión OR en el libro (línea 320): debe ser autoexplicatoria en la forma en que se utiliza aquí.

Variables utilizadas

Variables de lista para datos almacenados en el programa:

Fechas (día, mes, año)	D1$(6)
Ocasiones a encontrar:	01$(6)
Fechas (día, y solo mes):	D2$(6)
Ocasiones a encontrar:	02$(6)
Fechas (únicamente día):	D3$(6)
Ocasiones a encontrar:	03$(6)
Fecha dada por el usuario:	
Día:	C1$
Mes:	C2$
Año:	C3$
Contadores de bucle:	X

```
10   REM ***DIARIO***
20   REM POR SUSAN CURRAN
30   CLS
40   REM ESTABLECIMIENTO DE LAS LISTAS DE DATOS
50   DIM D1$(6): DIM 01$(6)
60   FOR X = 1 TO 6
70   READ D1$(X): READ 01$(X)
80   NEXT X
90   DIM D2$(6): DIM 02$(6)
100  FOR X = 1 TO 6
110  READ D2$(X): READ 02$(X)
120  NEXT X
130  DIM D3$(6): DIM 03$(6)
140  FOR X = 1 TO 6
150  READ D3$(X): READ 03$(X)
160  NEXT X
170  REM OBTENCION DE LOS DATOS A COMPROBAR
180  INPUT "ENTRAR LOS DATOS EN EL FORMATO
     DD,MM,YY";C1$, C2$,C3$
190  PRINT "SU DIARIO DE HOY:"
200  REM COMPROBAR LA LISTA D1
210  FOR X = 1 TO 6
220  IF C1$+C2$+C3$ = D1$(X) THEN PRINT 01$(X)
230  NEXT X
240  REM COMPROBAR LA LISTA D2
250  FOR X = 1 TO 6
260  IF C1$+C2$ = D2$(X) THEN PRINT 02$(X)
270  NEXT X
280  REM COMPROBAR LA LISTA D3 INCLUYENDO
```

```
281  REM LA COMPROBACION DE FINAL DE MES
290  FOR X = 1 TO 6
300  IF C1$ = D3$(X) THEN PRINT 03$(X): GOTO 340
310  IF C1$ = "28" THEN IF C2$ = "02" THEN IF
     D3$(X) = "FIN" THEN PRINT 03$(X) GOTO 340
320  IF C1$ = "30" THEN IF C2$ = "04" OR IF C2$
     = "06" OR IF C2$ = "09" OR IF C2$ = "11" THEN
     IF D3$(X) = "FIN" THEN PRINT 03$(X): GOTO 340
330  IF C1$ = "31" THEN IF D3$(X) = "FIN" THEN
     PRINT 03$(X): GOTO 340
340  NEXT X
350  PRINT "LISTA COMPLETA": END
400  REM DATOS PARA LA LISTA ANUAL ESPECIFICA
410  DATA "020285", "DENTISTA LIDIA 10.30"
     "030285", "CLUB ORDENADORES 7.30", "040285",
     "EL QUIJOTE CAPITULO 7"
420  DATA "110285", "CARLOS HABLAR CON J.A 7.30",
     "0", "0", "0", "0"
430  REM DATOS PARA LA LISTA ANUAL
440  DATA "2603", "CUMPLEAÑOS DE CARLOS", "1405",
     "CUMPLEAÑOS DE LIDIA"
450  DATA "0711", "CUMPLEAÑOS DE RODOLFO", "1412",
     "CUMPLEAÑOS DE JUANA"
460  DATA "3112", "RENOVAR CARNET CONDUCIR",
     "3006"
     "COMPROBAR LA CALEFACCION"
470  REM DATOS PARA LA LISTA MENSUAL
```

Apéndice 1: Ordenadores distintos, BASICs distintos

Cómo aparecen las líneas en la página y en la pantalla

En primer lugar, unas breves palabras acerca de cómo nuestros programas de ordenador y otras líneas tecleadas aparecen en este libro. Generalmente hablando, cuando colocamos espacios en las líneas, no importa si usted los pone o no. En los capítulos 1 y 2, sí que utilizamos grupos de espacios entre los textos para poder colocar las cosas de una forma determinada en la pantalla. No especificamos cuántos hemos utilizado, pero ya que un espacio en blanco utiliza el mismo espacio que un carácter (número o letra), puede fácilmente contar cuántos utilizamos en cada caso.

La manera en que las líneas a teclear aparecen en la página se diferenciará en dos aspectos de la forma en que aparecerán en su pantalla. Primeramente, el número 0 aparece como «0» en el libro, pero en su pantalla aparecerá como una O con una línea a través de ella, Ø. Segundo, para hacer los programas más fáciles de leer, hemos colocado los números de línea separados, en la parte izquierda, y hemos cortado las líneas de programa demasiado largas. En su pantalla las líneas largas continuarán en la parte izquierda de la siguiente línea de la pantalla.

Diferencias en BASIC

Los ordenadores domésticos varían en gran manera. Difieren en sus dialectos de BASIC y en su hardware y sistema operativo: sus teclados, por ejemplo, y en las órdenes usadas para hacer cosas no cubiertas por el BASIC.

Hemos intentado hacer este libro lo más generalmente aplicable que sea posible y para ayudarnos a conseguirlo hemos utilizado seis ordenadores distintos. En este apéndice haremos dos cosas. Primero, discutiremos de una forma general algunas diferencias entre los BASICs, tomando los puntos, capítulo a capítulo, tal como los encontrará en el libro. Y segundo, veremos brevemente cada uno de los ordenadores que hemos

utilizado y la forma como manejan los distintos aspectos de la programación en BASIC.

Debemos insistir en el hecho de que este libro no es un sustituto del manual de su propio ordenador. Este le dirá en detalle exactamente cómo funciona su ordenador y qué órdenes acepta. Lo que intentaremos decirle es dónde deberá encontrar exactamente en su manual la información que necesita.

Capítulo 1

Mayúsculas y minúsculas. Todos los ordenadores domésticos que hemos probado aceptan las órdenes en BASIC en mayúsculas. En algunos, pueden apretarse teclas especiales de orden, sin necesidad de teclear la totalidad de la orden. Unos pocos aceptan órdenes en minúsculas. La mayoría de los que pueden generar letras minúsculas en la pantalla aceptarán nombres de variables, textos, datos, etc. en minúsculas.

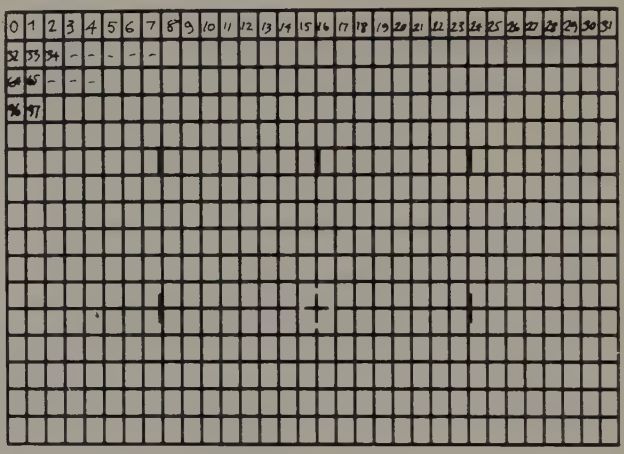
Manejo de números. La precisión con que son manejados los números es importante si se quiere hacer mucha aritmética en el ordenador. No es lo mismo que el tamaño de los números que puede manejar. Por ejemplo, 0,0000234552334 es un número muy pequeño, manejado comparativamente con una gran precisión —hay nueve dígitos significativos—. Una cifra pobre es la que tiene seis dígitos de precisión, una cifra más útil tiene una precisión de nueve o diez dígitos. Todos los ordenadores adoptan la notación científica para visualizar números grandes y pequeños. El punto a partir del cual aparece esta notación depende del espacio que utilizan para almacenar un número, que está relacionado —aunque no es lo mismo— con la precisión.

No todos los ordenadores utilizan el «+ invisible». En algunos, los números separados por punto y coma aparecen de lado.

Funciones aritméticas. Algunos ordenadores ofrecen un rango bastante amplio de funciones aritméticas; otros, pocas o ninguna. Nosotros utilizamos muy pocas en este libro.

CLS. Todos los ordenadores tienen una orden que limpia la pantalla. No es siempre CLS. Consulte su manual, y vea las notas sobre las órdenes PRINT que vienen a continuación.

Un mapa de pantalla alternativo
Algunas pantallas de ordenador están numeradas consecutivamente, fila a fila, así



Sintaxis general. Nuestra versión es aceptable generalmente, pero consulte su manual —quizás en el apartado de PRINT— si tiene algún problema.

PRINT@. Esta es quizá la orden con más trucos que manejamos, desde el punto de vista de las diferencias del BASIC. He aquí algunas orientaciones detalladas.

La sintaxis requerida (paréntesis, comas, puntos y comas, etc.) varía considerablemente. Consulte su manual buscando en el índice el «PRINT@». Algunos ordenadores sustituyen el símbolo «@» por «AT».

Muchos ordenadores utilizan únicamente un número para describir la posición de visualización. Numeran la pantalla de izquierda a derecha, línea a línea, como en el dibujo. No es difícil el convertir los números de fila y columna, que utilizamos nosotros, en este único número, de esta forma:

Tome el número de fila y multiplíquelo por el número de posiciones que tiene cada fila de su pantalla. Por ejemplo, para visualizar en la línea 11 de una pantalla de 32 columnas, será:

*11*32*

Después añada el número de la columna. Por lo tanto, para visualizar en la columna 10 de la fila 11, necesitará:

*11*32+10*

Puede colocar una expresión tal como ésta en su sentencia de PRINT@ y dejar que el ordenador haga la aritmética por usted, así:

*PRINT@ (11*32+10), "COLUMNA 10", "FILA 11"*

Cuando utilice variables en las sentencias de PRINT@, probablemente encontrará que es más sencillo utilizar dos variables como lo hacemos nosotros.

En algunos ordenadores no existe la orden de PRINT@. Habrá una manera de describir las posiciones de la pantalla, pero quizá sea más complicada. Si su ordenador es uno de los que no tiene el PRINT@, lea las secciones sobre máquinas individuales que hay a continuación. Incluso aunque no tenga una de las máquinas que describimos, viendo cómo manejan las posiciones en el PRINT@, puede mostrarle cómo lo hace su ordenador.

En muchas de esas máquinas, puede ayudar el pensarlo en términos de la posición del cursor. En lugar de utilizar el PRINT@, se utiliza una secuencia de órdenes que mueven el cursor hasta donde se quiere realizar la visualización.

Algunos ordenadores tienen sistemas operativos de pantalla más bien «abstractos», pero las órdenes que pueda parecer intuitivamente que no son muy claras, como las series de PUT del NewBrain y las órdenes de VDU del micro de BBC, pueden ser utilizadas para varios otros aspectos del manejo de la pantalla, incluyendo el limpiar la pantalla y el colocar el cursor en la parte superior izquierda de la pantalla.

Capítulo 2

END. Algunos pocos BASIC no incluyen el END. La mayoría no insisten sobre él, por lo que normalmente se pasa por alto. Cuando sea necesario, puede utilizar en su lugar el STOP (que es ligeramente distinto).

LIST. La sintaxis utilizada para describir las líneas a listar puede variar. Consulte su manual si nuestras versiones no funcionan.

NEW. En algunos ordenadores esta orden borrará la pantalla así como la memoria principal.

SAVE y *LOAD.* En realidad, éstas pertenecen al sistema operativo de cassette de su ordenador, no a su

BASIC. Hay muchas variantes. Algunas alternativas incluyen CSAVE, CLOAD y CHAIN. También puede encontrar otras órdenes útiles como SKIPF y VERIFY. Vea la sección de su manual que trata sobre cómo utilizar el grabador de cassette.

BREAK. No todos los ordenadores tienen una tecla de «break». Si no la tiene, debe haber alguna otra manera de interrumpir un programa. Pruebe mirando en el índice de su manual la palabra «interrupción».

Caracteres gráficos. La mayoría de los ordenadores domésticos, pero no todos, tienen estos caracteres o similares. Si el suyo no los tiene, utilice caracteres normales de teclado (letras, números, etc.). Más tarde aprenderá a definirse sus propios caracteres. En algunos micros pueden describirse caracteres gráficos únicamente pulsando la tecla «gráfico» y después una tecla normal QWERTY, en lugar de (o además de) utilizar los códigos CHR$.

Capítulo 3

LET. En la mayoría de los ordenadores, el LET es opcional y puede suprimirlo, si así lo prefiere. Tan sólo hay que utilizar, por ejemplo, «C = 10». En unos pocos, hay que suprimirlo.

Nombres de variable. Muchos ordenadores le permiten utilizar nombres más largos y más descriptivos tales como «DIRECCION» o «CONTADOR». Esto le ayudará a hacer sus programas más fáciles de leer. En algunos, el ordenador utiliza únicamente las dos primeras letras, por lo tanto MIPUNTUACION y MITOTAL serían tratados como la misma variable. Muchos tienen varios tipos diferentes de variable numérica, para los números enteros, decimales y algunas veces para los números con doble precisión. Los nombres sencillos que utilizamos son generalmente aceptables.

INPUT. La sintaxis puede variar, particularmente cuando se entran dos o más artículos a la vez. Consulte su manual.

Números aleatorios. De nuevo, la sintaxis varía. Algunos ordenadores tienen una versión alternativa que genera directamente números enteros aleatorios. Esta es más sencilla de utilizar, por lo que no debe usted tener ningún problema. A veces puede necesitar la

orden RANDOMIZE, que afecta la secuencia en que los números aleatorios son generados.

Capítulo 4

Velocidad. Observe que no todos los ordenadores funcionan a la misma velocidad. Un bucle de espera de 0 a 460 representa un retardo de un segundo en, por ejemplo, un Tandy Colour Computer. Quizá deba utilizar un número distinto en el suyo.

Capítulo 5

Muchos ordenadores ofrecen órdenes especiales para ayudar en la edición y depuración. No las hemos utilizado en este libro.

Capítulo 6

Variables de lista. Hemos utilizado únicamente listas unidimensionales. Muchos ordenadores le permiten utilizar dos o más dimensiones, lo que es muy interesante, aunque no tengamos espacio para describirlo en detalle. Si se utilizan grandes listas, quizá necesite reservar espacio de memoria para ellas. La orden CLEAR puede hacer esto.

Obsérvese que los ordenadores Sinclair/Timex manejan las listas de una forma ligeramente distinta. Lo analizamos brevemente.

READ...DATA. Estas dos órdenes son casi universales, pero los primeros ordenadores Sinclair/Timex las suprimieron; un fallo muy criticado. Utilice el LET, y modifique nuestros programas de acuerdo con ello.

Otros aspectos generales

Hemos suprimido el color y el sonido, no porque sean difíciles, sino porque su manejo varía mucho. Si su máquina tiene estas características, compruebe para ver cómo funcionan. Tiene que ser capaz de añadir color y/o sonido a muchos de nuestros programas con un esfuerzo muy pequeño.

Naturalmente, hay otras varias órdenes de BASIC que no hemos analizado. De nuevo, hay que tener en cuenta que muchas de éstas varían de una máquina a otra.

Muchos ordenadores le permiten abreviar sus programas, mediante la supresión de algunas palabras,

entrando abreviaciones en lugar de teclear las órdenes completas, etc. Consulte su manual para detalles.

Las máquinas que hemos utilizado

Hemos intentado utilizar una variedad de máquinas que representen a los distintos fabricantes y a las distintas versiones del BASIC utilizadas más comúnmente. Las que hemos escogido son ejemplo de muchas de las diferencias que se encuentran en un amplio rango de los ordenadores domésticos más populares.

Commodore 64. Más potente que el VIC-20 y menos que el ordenador personal, orientado a la gestión de empresas de Commodore. Tiene buen sonido y color, pero su BASIC es francamente pobre. El manejo de los colores y de los gráficos de alta resolución es particularmente difícil a menos que posea una versión ampliada del BASIC, que no estaba disponible cuando escribimos este libro.

En el Commodore 64 y en el VIC-20, pueden incluirse órdenes de tecla en los textos; órdenes que, en modo inmediato, se generan pulsando una tecla especial. Para borrar la pantalla en un programa, hay que poner la orden «CLEAR» dentro de un texto, mediante una sentencia tal como:

```
PRINT "(tecla CLEAR)"
```

No existe la orden PRINT@. En su lugar hay que utilizar las órdenes de cursor en los textos, combinadas con el TAB, lo que permite saltar columnas. Hay ejemplos en el manual del usuario, pero esta orden tiene muchos trucos y quizás usted prefiera adaptar nuestros programas y utilizar tan sólo el PRINT más a menudo.

El Commodore 64 tiene una longitud de línea corta. Si se teclean más de 80 caracteres, le dará un código de error. Una palabra final a su favor: tiene realmente un buen editor de pantalla.

Dragon 32. Este utiliza una versión del «Microsoft» BASIC muy similar a la que utilizamos nosotros. Tendrá que convertir nuestros números de fila/columna en una única cifra y modificar las órdenes C y R.

Hay que reservar espacio para todos, incluso las pequeñas listas, utilizando el CLEAR.

Comparativamente es fácil el utilizar el color y la orden SOUND: pruébelo.

Grundy NewBrain. Una máquina en blanco y negro con buenas características para los negocios, con un sistema operativo de cassette bastante flexible y buenos gráficos para la gestión empresarial. Sin embargo, el manual puede decepcionarle.

El NewBrain utiliza ANSI (el Instituto de Estándars de Estados Unidos), el Estándar BASIC. En realidad todavía no es muy estándar entre los pequeños ordenadores, pero puede llegar a serlo. Tiene un sistema operativo muy «abstracto». Sin embargo, puede utilizar la mayoría de nuestros programas sin luchar con sus complejidades. No existen las órdenes CLS o PRINT@ —hay que utilizar la orden PUT, seguido de algunos códigos—. «PUT 31» borra la pantalla, «PUT 22» es el código del cursor, por lo tanto:

```
PUT 22, 5, 15
```

por ejemplo, sería el equivalente de PRINT@ (15,5). Obsérvese que los números de fila y columna están invertidos. Nótese también la sintaxis distinta de las sentencias de INPUT. Los textos de INPUT deben ir entre paréntesis, en lugar de entre comillas.

Spectrum de Sinclair/Timex 2000. Este es más potente y versátil que el ZX80 y el ZX81 (Timex 1000), pero utiliza un BASIC similar. Todas las palabras de orden deben entrarse pulsando una única tecla. Si se teclean en su totalidad no funcionan.

El Sinclair/Timex maneja las cadenas de listas de una manera no usual: todos los artículos deben tener una longitud fija. Esto conduce a ciertas dificultades en el programa de «lista de biblioteca» de la página 134, pero puede hacerse funcionar mediante un cuidadoso uso de los espacios. Manejan la pantalla de una forma inhabitual, reservando las líneas inferiores para que usted pueda entrar el texto. Se pueden generar buenos dibujos en modo inmediato.

En su totalidad, nos gusta y recomendaremos el BASIC utilizado por Sinclair/Timex. La máquina es muy buena en el hecho de visualizar o señalar errores y los programas son fáciles de relacionar y editar.

También, el color y sonido tienen órdenes directas, y usted puede mejorar muchos de nuestros programas mediante su utilización.

Tandy/Radio Shack Colour Computer. Es casi un Dragon con otro nombre, aunque el Tandy/Radio Shack apareció primero. Es menos potente, y su BASIC estándar es más limitado. El BASIC ampliado, que no hemos utilizado nosotros, es idéntico al BASIC del Dragon. Es una máquina fácil de utilizar para un principiante, en la que todos nuestros programas funcionan con un mínimo de modificaciones. Véanse los comentarios sobre el Dragon que hemos hecho anteriormente. Hay que suprimir el LET.

Torch (y BBC) Computer. Hemos utilizado el procesador de palabras WordStar en el Torch, un ordenador personal mayor y más caro, para escribir este libro. El Torch ejecuta BBC BASIC así como el Microsoft.

El BBC BASIC no tiene la orden PRINT@. El PRINT TAB —a diferencia de las órdenes de PRINT TAB en muchos otros ordenadores— es efectivamente la misma que PRINT@. No hay que poner un espacio después del TAB y hay que invertir el orden de los números de fila y columna respecto a lo que nosotros damos.

Los distintos «modos» en los que la máquina de BBC funciona pueden confundir un poco al principiante. Para ejecutar nuestros programas, manténgase en el modo 7; éste es el modo en que está el ordenador cuando se conecta. El uso de los símbolos gráficos de teletexto mediante el PRINT TAB tiene su truco, ya que los códigos de control le mantienen en movimiento. Si se encuentra con problemas, puede utilizar únicamente el CHR$(255), un cuadrado en blanco, que no necesita códigos de control. Deje el color hasta que se haya familiarizado con la máquina; también los códigos son más bien confusos.

Apéndice 2: La versión del BASIC utilizada en este libro

Comentarios generales y sintaxis

Generalmente, todas las sentencias de BASIC están escritas en forma completa, sin simplificaciones u omisiones de palabras de orden.

Las líneas de programa están precedidas por números de línea enteros dentro del rango de 1 a 9999.

Se permite más de una sentencia por línea. Las sentencias están separadas por(:).

Se utilizan dos clases de nombre de variable:

Variables numéricas:	letra mayúscula/letra mayúscula o número opcional, ejemplo A1
Variables de texto:	igual que anteriormente, pero seguidos por el signo $, ejemplo BB$

Las listas unidimensionales tienen nombres similares, pero con subíndices numéricos, ejemplo BB$(5),A(5).

Se utilizan dieciséis caracteres de bloques gráficos. Se obtienen mediante las expresiones CHR$. Hay una lista de estos códigos y caracteres en la página 55.

Los espacios fuera de los textos generalmente se supone que son opcionales.

Ordenes

CLS	Borra la pantalla de video.
DATA	Introduce una lista de datos en un programa. Los artículos van separados por comas, y los textos van entre comillas, ejemplo DATA 1, 2, 3 DATA «MARIA», «JUANA», «ANA»
DIM	Dimensiones de una lista de datos, ejemplo DIM A(4) asigna cuatro posi-

	ciones para la lista A. Las listas se supone que tienen base 1 (es decir, no tienen el elemento cero). Tan sólo se utilizan listas unidimensionales. El DIM hay que colocarlo antes de todas las listas que se utilicen.
END	Produce la detención de la ejecución del programa. Es necesario.
FOR...TO ...NEXT.	Construcción utilizada para un bucle con contador, ejemplo FOR X = 1 TO 4.....NEXT X. Véase también STEP.
GOSUB	Envía el control del programa a una subrutina que empieza en el número de línea dado, ejemplo GOSUB 1000.
GOTO	Envía el control del programa a un número de línea determinado, ejemplo GOTO 50.
IF...THEN	Si la sentencia que sigue al IF es verdad, entonces la sentencia(s) que sigue al THEN (hasta el final de la línea) es ejecutada. Si no, el control del programa pasa a la siguiente línea del programa.
INPUT	El programa espera que uno o más datos sean entrados en las variables dadas. Puede estar seguido por un mensaje entre comillas, ejemplo INPUT «DEME DOS NUMEROS»; A, B. Visualiza DEME DOS NUMEROS en la pantalla, y asigna los dos números entrados a las variables A y B.
LET	Asigna un valor a una variable. La construcción utilizada es LET (variable)=(valor), ejemplo LET A = 50. Hay que asignar un valor a todas las variables antes de poder utilizarlas (aunque sea el 0); el LET no se omite nunca.
LIST	Lista la totalidad, o una parte especificada, de un programa. Opciones: LIST: el programa completo.

	LIST (línea): ejemplo (LIST 10): sólo la línea especificada. LIST (línea)–(línea):(ejemplo LIST 10-50): la parte entre las dos líneas. LIST (LINEA)–:(ejemplo LIST 50–): desde la línea especificada hasta el final del programa. LIST-(línea):(ejemplo LIST-50): desde el principio del programa hasta la línea especificada.
LOAD	Carga un programa desde el cassette. No se discute en detalle.
NEW	Borra el programa y las variables que están actualmente en memoria.
NEXT	Utilizado con el FOR...TO.
PRINT	Visualiza los datos que le siguen (números, textos o variables) en la pantalla de visualización, ejemplo PRINT A (visualiza el contenido de la variable A). PRINT «HOLA» (visualiza el texto exactamente como está). Las comas hacen que se visualice en la siguiente zona de visualización (se supone que la pantalla está dividida en zonas de 4×8 columnas). Los puntos y comas separan los artículos que deben ser visualizados; al final de la línea, hay que tener en cuenta que el cursor desciende a la línea siguiente. No hay espacios entre los elementos de un texto; hay un espacio a cada lado de los elementos numéricos.
PRINT@	Visualiza los datos en una posición de la pantalla, que se da a continuación. Deben especificarse dos coordenadas, fila y columna, ejemplo: PRINT@ (4,5), «HOLA» produce el que aparezca un HOLA al principio de la columna 5 en la fila 4. El tamaño de la pantalla se supone que es de 16 filas por 32 columnas, empezando por la fila 0, columna 0.

READ	Asigna valores a las variables, utilizando sucesivos elementos de lista de DATA.
REM	Se coloca delante de los comentarios del programador/operario. Todo lo que hay a continuación de un REM en la misma línea es ignorado por el ordenador.
RETURN	Vuelve el control al programa principal después de que la subrutina haya sido ejecutada.
RUN	Produce la ejecución, por el ordenador, del programa que está actualmente en la memoria.
SAVE	Guarda un programa en el cassette (no se ha analizado en detalle en el libro).
STEP	Determina el paso utilizado por el contador en los bucles FOR...NEXT, ejemplo, FOR X = 1 TO 10 STEP 0,5. Si se omite se supone un paso de 1.

Funciones y operadores

Operadores aritméticos

+	suma, ejemplo 1 + 2
–	resta, ejemplo 3 – 4
/	división, ejemplo 3/4
*	multiplicación, ejemplo 3 * 4

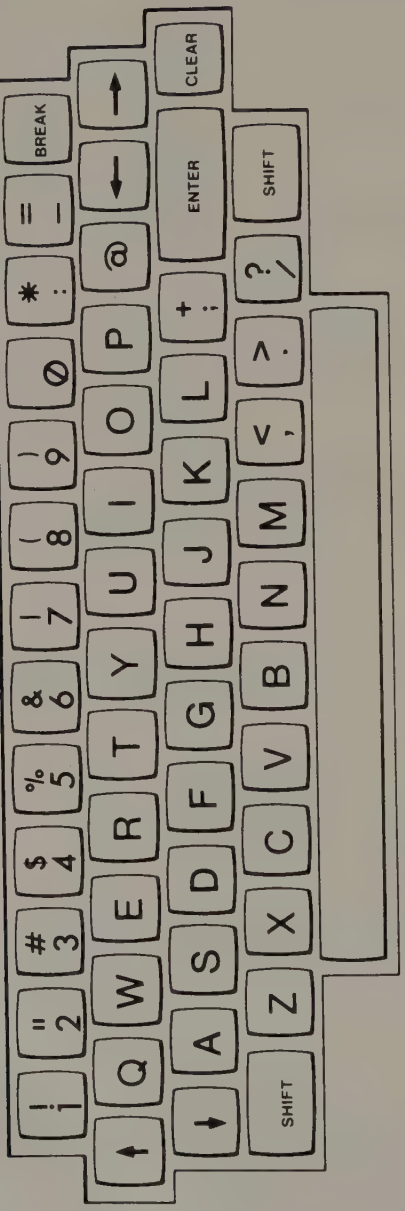
El orden de ejecución no se analiza en el texto. En expresiones más complejas, los paréntesis se utilizan libremente para hacerlas más claras. Operadores lógicos

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

Funciones

CHR$	Da el carácter cuyo código está a continuación, ejemplo CHR$(135). Se utiliza en el texto únicamente para generar caracteres gráficos. Los códigos están en la lista de la página 55.
COS	Da el coseno del número que le sigue, ejemplo COS(55). Se menciona tan sólo brevemente en el texto.
INT	Redondea por debajo al siguiente número entero, ejemplo INT(5,5) es igual a 5.
RND	Genera un número aleatorio entre 0 y 1. Para generar un número entero aleatorio, se utiliza la siguiente expresión: INT (RND (rango)) ejemplo INT(RND 5).
SQR	Produce la raíz cuadrada, ejemplo SQR(25). Se menciona brevemente en el texto.

El teclado de nuestro ordenador



Apéndice 3: lecturas recomendadas

Las revistas sobre ordenadores domésticos aparecen y desaparecen cada mes. He aquí algunas de las ya establecidas que recomendamos. Se trata de revistas generales. Muchas otras tratan tan sólo sobre determinados ordenadores.

Byte (revista USA): puede ser altamente técnica, pero siempre interesante.
Creative Computing (revista USA): una revista de amplio espectro centrada en aplicaciones.
Personal Computer World: una buena revista general mensual.
Personal Computer News: una revista semanal del mismo tipo.
Practical Computing: es también una buena revista de carácter general.

Los siguientes libros tratan sobre aspectos del BASIC con más profundidad de la que hemos podido tratar nosotros:

Henry Ledgard & Andrew Singer, *Elementary BASIC,* Vintage (USA), Fontana (UK). Principalmente trata de la solución de problemas mediante el BASIC.

Bob Albrecht, LeRoy Finkel & Jerald R. Brown, *BASIC, for Home Computers,* Wiley. Un buen libro de carácter general sobre Microsoft.

Ian Stewart & Robin Jones, *Machine Code & Better BASIC,* Shiva. Se centra en el ZX81, pero es una buena introducción al BASIC y a la estructuración de datos y programas, aplicable a todas las máquinas, y una introducción al código máquina para todas las máquinas que utilicen el microprocesador Z80 o Z80A.

Garry Marsshall, *Programming with Graphics*, Granada. Una breve y adecuada introducción orientada al BASIC, en un campo que hemos tocado ligeramente.

Respuestas a las preguntas

Capítulo 1

(1) *PRINT 56.4 + 76.3 {E}*
(2) *PRINT 74/13.1 {E}*
(3) *PRINT 45*7 {E}*
(4) *PRINT 1025 – 96 {E}*
(5) *PRINT "*
LIDIA"

(Hay 27 espacios antes del LIDIA, por lo tanto habrá llegado a una nueva línea en la pantalla del ordenador.)

(6) *PRINT "L",,,,"U",,,,"I",,,,"S"*
(7) *PRINT "ROJO ";"NARANJA ";"AZUL ";"ROSA"*
(8) éste es nuestro perro: *00-----/----",*
00----/
1 1

Y éste es nuestro programa:

```
CLS:PRINT@(0,16),"00-----/----
,,"00----/",,," 1 1"
```

(9) *CLS:PRINT@ (8,12), "CAROLINA"*

Hemos utilizado una palabra de 8 letras, por lo tanto hemos dejado 12 espacios (en nuestra línea de 32 espacios) a cada lado de él.

(10) *CLS:PRINT@(0,31),"/":PRINT@(1,30),"/":PRINT@(2,29),"/".....ETCETERA*

...etc.

Capítulo 2

(1) No, éste no funcionará. ¿Por qué? Ya que los dos puntos después del PRINT terminan la sentencia.
La segunda sentencia de la línea 10 («HOLA») es gramaticalmente incorrecta.
(2) Este tampoco funcionará. La sentencia END está

antes de la de PRINT, cuando el programa se coloca por orden de líneas.

(3) Este funcionará. La pantalla aparecerá así:

```
000 000 000
000 000 000
 1 1 1        (los mismos tres flujos estilizados)
 1/ 1/ 1/
```

(4) Y éste funcionará. Las sentencias de PRINT en blanco moverán únicamente el cursor una línea hacia abajo, por lo que aparecerá así:

```
HOLA

ADIOS
```

Capítulo 3

(1) (a) No aceptable: «A» es un nombre de variable numérica, y «MANZANAS» es un texto.
(b) Sí, éste es aceptable.
(c) No aceptable. Puede colocarse un 56 en un texto pero hay que ponerlo entre comillas («56»).

(2)
```
10  REM PUENTE
15  CLS
20  LET A$ = CHR$(128)+CHR$(128)+CHR$(128)
30  LET B$ = CHR$(128)+CHR$(143)+CHR$(128)
40  PRINT A$:PRINT A$:PRINT B$:PRINT B$
```

(3)
```
10  REM NOMBRE
15  CLS
20  LET A$ = "MARY PEREZ"
30  PRINT@ (0,5), A$
40  PRINT@ (8,5), A$
50  PRINT@ (15,5), A$
60  END
```

(4)
```
10  REM PROGRAMA DE X,Y
20  INPUT "DEME DOS NUMEROS";X,Y
30  PRINT X;"+";Y;"=";X+Y
40  PRINT X;"-";Y;"=";X-Y
50  PRINT X;"*";Y;"=";X*Y
```

```
60  PRINT X;"/";Y;"=";X/Y
70  END
```

(5)
```
10  REM ***DIBUJELO USTED MISMO***
20  PRINT "DEME UNA LINEA PARA DIBUJAR"
30  PRINT "EN LA FORMA DE TRES CODIGOS CHR$"
40  PRINT "EJEMPLO 128, 143, 137"
50  PRINT "Y DESPUES CUANDO LO HAYA DI-
    BUJADO"
60  PRINT "DEME OTRA"
70  INPUT X, Y, Z
80  PRINT CHR$(X)+CHR$(Y)+CHR$(Z)
90  GOTO 70
```

No hay sentencia END esta vez ya que el programa da vueltas sobre un bucle hasta que se aprieta la tecla BREAK.

Recuerde que estos programas son respuestas de ejemplo únicamente. Su respuesta puede ser correcta, pero no coincidir exactamente con la nuestra.

(6)
```
10  PRINT 17 + INT(RND*10)
20  GOTO 10
```

(7) He aquí nuestra lista de variables:

Forma de camión:	L$
Posición de columna:	N
Paso de cambio:	P

...y nuestro programa:

```
10  REM CAMION
20  LET L$ = CHR$(128)=(CHR$(128)+CHR$(132)
30  LET N = 0
40  LET P = 1 + INT(RND*3)
50  PRINT@ (8,N), L$
60  LET N = N + P
70  CLS
80  GOTO 50
```

Capítulo 4

(1) Primero, la lista de variables:

Número aleatorio del ordenador:	N
Su número supuesto:	G

Ahora el programa:

```
10   REM JUEGO DE ADIVINANZAS
20   LET N = 1 + INT(RND*6)
30   PRINT "ESTOY PENSANDO UN NUMERO"
40   PRINT "ENTRE 1 Y 6"
50   PRINT "QUIERE ADIVINAR CUAL ES?"
60   INPUT G
70   IF G = N THEN GOTO 100
80   PRINT "NO, NO ES ESTE. PRUEBE OTRA VEZ"
90   GOTO 60
100  PRINT "MUY BIEN. SI, ERA ESTE"
110  END
```

(2) Una sola variable esta vez:
Su elección: N
Ahora el programa:

```
10   REM ***GENERADOR DE MENSAJES***
20   PRINT "ESCOJA UN NUMERO DEL 1 AL 6"
30   PRINT "Y LE DARE UN MENSAJE"
40   INPUT N
50   IF N = 1 THEN PRINT "BUENOS DIAS. QUE LO
     PASE BIEN"
60   IF N = 2 THEN PRINT "HOLA. QUE PUEDO
     HACER  HOY?"
70   IF N = 3 THEN PRINT "QUE HAY DE NUEVO.
     OTRO  DURO DIA DE TRABAJO"
80   IF N = 4 THEN PRINT "QUE TAL SI HOY
     HACEMOS  FIESTA"
90   IF N = 5 THEN PRINT "BUENOS DIAS. BIENVENIDO
     A  CASA"
100  IF N = 6 THEN PRINT "QUE TAL COMO ESTAS"
200  END
```

(3)
```
10   REM ENCANTADOR DE SERPIENTES 2
20   CLS
30   LET N = 15
40   PRINT@ (N,10), "SSSSS";
50   LET N = N – 1
60   IF N = –1 THEN GOTO 80
70   GOTO 40
80   GOTO 80
```

(4)
```
10   REM ***AMBULANCIA***
15   CLS
```

```
    20  LET A$ = CHR$(128)+CHR$(128)+CHR$(128)+
        CHR$(128)
    30  LET B$ = CHR$ (128)
    40  LET C$ = "EMERGENCIA"
    50  PRINT@ (8,6), A$+A$+B$+B$;
    60  PRINT@ (9,4), A$+A$+A$;
    70  PRINT@ (10,4), A$+A$+A$;
    80  PRINT@ (11,5), B$:PRINT@ (11,14), B$;
    90  PRINT@ (9,5), C$;
   100  FOR D = 0 TO 50: NEXT D
   110  PRINT@ (9,6), A$+A$+B$;
   120  FOR D = 0 TO 25: NEXT D
   130  GOTO 90
```

(5)

```
    10  REM CARACTERES GRAFICOS
    20  FOR G = 128 TO 143
    30  PRINT CHR$(G);" ";
    40  NEXT G
    50  END
```

(6)

```
    10  REM ***CRONOMETRO***
    20  PRINT "DEME UN NUMERO DE SEGUNDOS"
    30  PRINT "PARA CONTAR, Y YO"
    40  PRINT "LOS CONTARE PARA USTED"
    50  INPUT S
    60  FOR N = 0 TO S – 1
    70  PRINT N
    80  FOR D = 0 TO 460: NEXT D
    90  NEXT N
   100  PRINT S;" SE ACABO EL TIEMPO!"
   110  END
```

Capítulo 6

(1)

```
    10  REM PROGRAMA DE CUATRO NUMEROS
    20  DIM A(4)
    30  PRINT "DEME CUATRO NUMEROS"
    40  PRINT "Y YO LOS ESCRIBIRE PARA USTED"
    50  FOR N = 1 TO 4
    60  INPUT "NUMERO";A(N)
    70  NEXT N
    80  FOR N = 1 TO 4
    90  PRINT,,A(N)
   100  NEXT N
   110  END
```

(2)
```
10  REM VISUALIZACION DE UN NUMERO
20  DIM C(2): LET N = 1
30  PRINT "DEME HASTA 20 CODIGOS CHR$"
40  PRINT "Y DIBUJARE LOS CARACTERES PARA
    USTED"
50  PRINT "ENTER O STOP"
60  INPUT "CODIGO":C(N)
70  IF C(N) = 0 THEN GOTO 200
80  LET N = N + 1
90  IF N = 21 THEN GOTO 200
100 GOTO 60
200 REM DIBUJO DE LOS CODIGOS
210 CLS
220 FOR M = 1 TO (N-1)
230 PRINT@ (8,6+M),CHR$(C(M));
240 NEXT M
250 END
```

(3)
```
10  REM LISTA DE NOMBRES AL REVES
20  DIM N$(5)
30  PRINT "DEME CINCO NOMBRES"
40  PRINT "Y LOS ESCRIBIRE EN ORDEN INVERTIDO"
50  FOR X = 1 TO 5
60  INPUT "NOMBRE";N$(X)
70  NEXT X
80  FOR X = 5 TO 1 STEP -1
90  PRINT N$(X)
100 NEXT X
110 END
```

(4)
```
10  REM SERPIENTE
20  CLS
30  REM DATOS EN EL CUERPO
40  DIM R(19): DIM C(19)
50  FOR X = 1 TO 19
60  READ R(X): READ C(X)
70  NEXT X
80  DATA 9,3,8,4,9,5,9,6,10,7,10,8,9,9,9,10
90  DATA 8,11,8,12,9,13,10,14,10,15,9,16,9,17
100 DATA 8,18,7,19,6,20,4,20
110 REM DIBUJO DEL CUERPO
120 FOR P = 1 TO 19
130 PRINT@ (R(P),C(P)),CHR$(128);
140 PRINT@ (R(P)+1,C(P)),CHR$(128);
150 NEXT P
```

```
160 REM DIBUJO DE LA LENGUA
170 PRINT@ (3,20),"1";:PRINT@ (2,20),"1";
180 GOTO 180
```

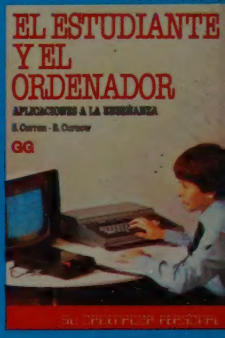
(5) Utilizaremos la variable de lista N$ para los nombres, la variable de lista A$ para las direcciones y la variable de lista M$ para las coincidencias. Para mantener la respuesta corta, pondremos únicamente tres nombres y direcciones en el archivo y no diez como usted habrá hecho.

```
10  REM ARCHIVO DE NOMBRES Y DIRECCIONES
20  DIM N$(3): DIM A$(3): LET C = 0
30  REM ENTRADA DE DATOS EN EL ARCHIVO
40  FOR X = 1 TO 3
50  READ N$(X): READ A$(X)
60  NEXT X
70  DATA "JUAN PELEGRIN", "PUERTAFERRISA 22"
80  DATA "BENJAMIN TORCAL", "BALMES 1"
90  DATA "JAVIER MOLINA", "VALENCIA 348"
100 REM SECUENCIA DE PREGUNTAS
110 PRINT "NOMBRE DE LA PERSONA CUYA
    DIRECCION  QUIERE"
120 INPUT "(NOMBRE, APELLIDO)";M$
130 FOR X = 1 TO 3
140 IF M$ = N$(X) THEN PRINT A$(X): LET C = 1
150 NEXT X
160 IF C = 0 THEN PRINT "LO SIENTO, NO ESTA
    EN EL  ARCHIVO"
170 INPUT "OTRO INTENTO? S/N";R$
180 IF R$ = "S" THEN LET C = 0: GOTO 100
190 END
```

Indice analítico

Los números en *cursiva* corresponden a la ilustración.

Colección
SU ORDENADOR PERSONAL



Una guía clara y sencilla para principiantes, aplicable a cualquier ordenador personal

En este volumen PRIMEROS PASOS EN BASIC encontrará

- ☐ diferentes maneras de combinar las pocas y sencillas habilidades del ordenador
- ☐ un análisis profundo de la planificación y compilación de una selección de útiles programas en BASIC
- ☐ visualización de textos y gráficos de una forma atractiva en la pantalla
- ☐ programas de debug
- ☐ ideas para escribir programas
- ☐ compra de programas pregrabados
- ☐ más de 60 ilustraciones a dos colores

Editorial Gustavo Gili, S.A.
Rosellón, 87-89
08029 Barcelona