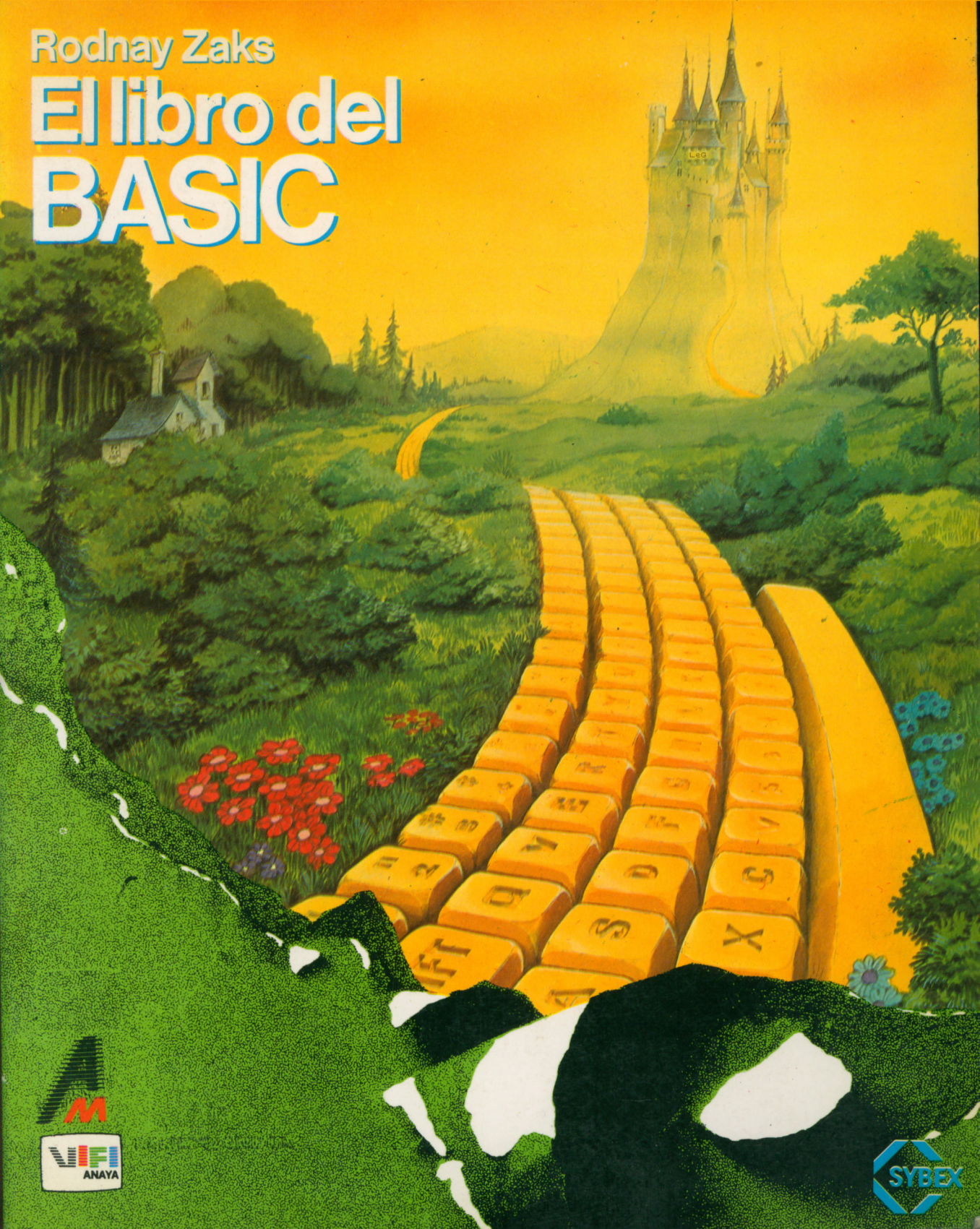
Rodnay Zaks

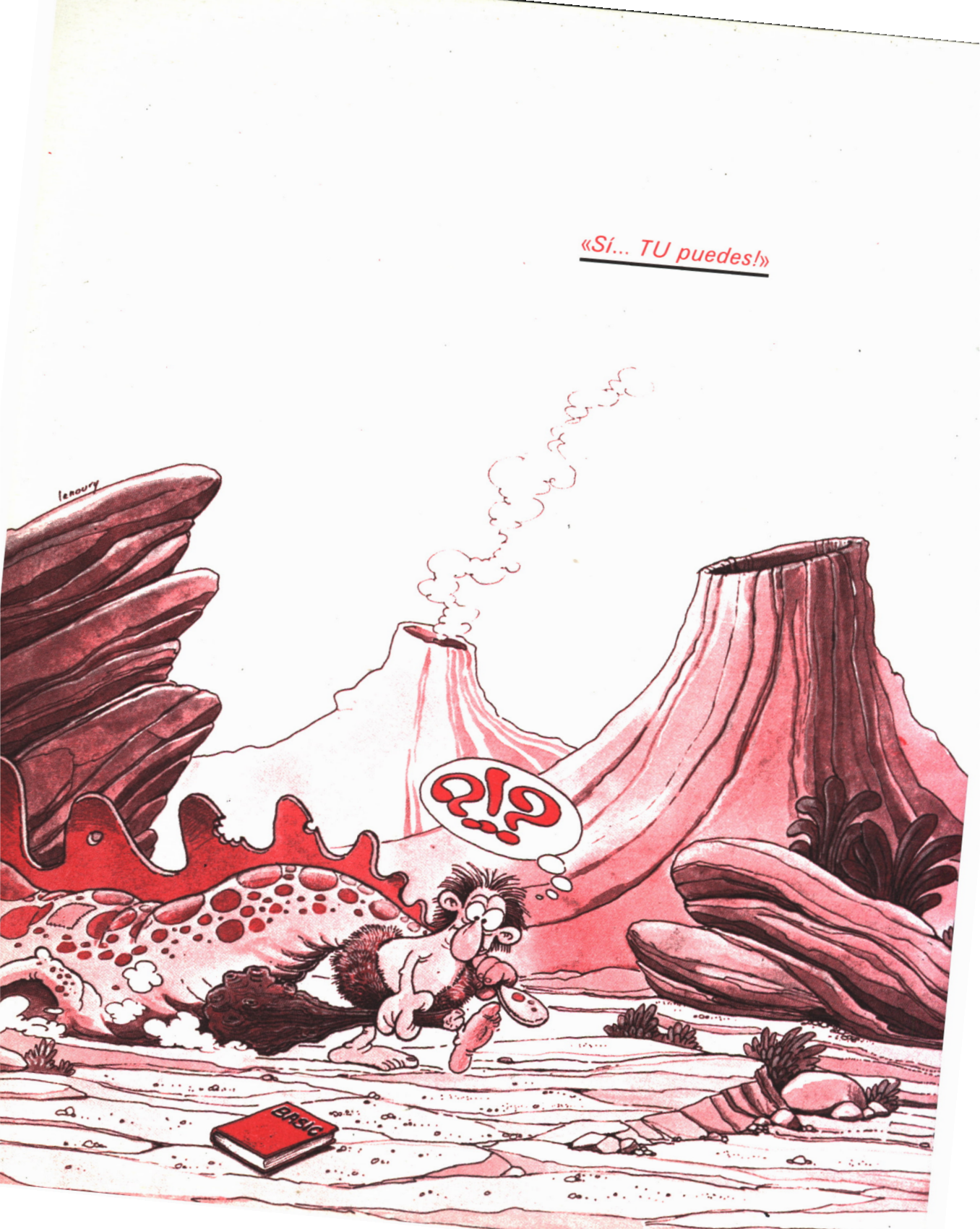
El libro del BASIC



VIFI ANAYA

SYBEX

«Sí... TU puedes!»



EL LIBRO DEL BASIC

EL

DEL

LIBRO BASIC

Rodnay Zaks

ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
«YOUR FIRST BASIC PROGRAM»

Traducción de: William J. Wallace
Diseño de colección: Antonio Lax

Primera edición
Segunda reimpresión, marzo 1985

Versión castellana:

© EDICIONES ANAYA MULTIMEDIA, S. A., 1985
Villafranca, 22. 28028 Madrid
Depósito legal: M. 8.370-1985
ISBN: 84-7614-005-3
Printed in Spain
Imprime: JOSMAR, S. A.
Artesanía, 17. Políg. Ind. de COSLADA (MADRID)

1

Hablando BASIC

2

Comunicándote con tu ordenador

9

Estudio de un caso real: La conversión al sistema métrico

10

El siguiente paso

Apéndices

Introducción

Se han escrito ya centenares, quizá miles de libros sobre el lenguaje BASIC.

¿Para qué escribir otro? Pues porque los que trabajan en BASIC han cambiado mucho. Hace algunos años, el uso de los lenguajes de programación, como el BASIC, era privilegio de las pocas personas que tenían acceso a los ordenadores. Los programadores constituían un grupo pequeño y selecto. Pero las cosas han cambiado, y mucho. Los ordenadores personales han hecho que el BASIC sea el lenguaje de programación de más uso y extensión, y la mayoría de los usuarios actuales no son especialistas en Informática. Estos nuevos usuarios utilizan los ordenadores para la diversión, la educación, los negocios, o para su profesión.

Este libro se dirige a este nuevo grupo de usuarios. No sólo posee un aspecto muy diferente, sino que tiene un planteamiento nuevo. Se dirige al principiante, por lo que no presupone ningún conocimiento previo por parte del lector. Es para todo el mundo —desde los 8 a los 88 años— que quiera iniciarse rápidamente en el BASIC.

El autor es de la opinión que todo nuevo usuario de los ordenadores pequeños que además quiere aprender a programar sus propias aplicaciones en BASIC, es realmente una persona caracterizada por un gran entusiasmo y por su juventud (¡o manera de pensar joven!). Querrán acercarse al BASIC de una manera sencilla, directa y didáctica. Disfrutarán, pues, de la filosofía de este libro: está pensado para que el aprendizaje del BASIC resulte cómodo y divertido.

Es más, al seguir este libro conocerás en muy poco tiempo las características esenciales del BASIC, y te encontrarás escribiendo tu primer programa en BASIC. Poco tiempo después poseerás ya los conocimientos necesarios para escribir programas útiles y prácticos.

El tiempo pasa... ¡comencemos cuanto antes!

El autor te desea un feliz viaje a lo largo del sendero mágico hacia los nuevos conocimientos.

Rodnay Zaks

Berkeley, enero de 1984

Cómo seguir este libro

Este es un libro totalmente didáctico. Has de leerte cada capítulo correlativamente, procurando entenderlo antes de continuar. He colocado algunos ejercicios prácticos al final de cada capítulo para que compruebes tus conocimientos nuevos. Haz todos los ejercicios que puedas. Las respuestas para algunos de los ejercicios —los más importantes— se encuentran al final del libro, en el apéndice A.

Si posees un ordenador, prueba todos los programas. Para aprender bien y recordar lo aprendido, has de practicar y experimentar. Este libro te dará los conocimientos que necesitas para arrancar, pero, recuerda, no hay nada mejor que la experiencia.

El objetivo principal del libro es el de iniciarte rápida y eficazmente en la programación en BASIC. Para lograr este objetivo y asegurarme de que no te confundes en ningún momento, he tenido que seleccionar el contenido cuidadosamente, por lo que podrás comprobar que el libro no describe absolutamente todas las características y conceptos relacionados con el BASIC, sino sólo los más fundamentales.

Tengo la esperanza de que con la ayuda de este libro entenderás todo sin problemas, y que pronto estarás escribiendo tus primeros programas en BASIC, maravillándote y disfrutando de lo que te permiten hacer tus nuevos y excitantes conocimientos.

Lo que vas a estudiar

El *capítulo 1* explica el lenguaje de los ordenadores y te presenta a los héroes de este libro: el Ordenador, el Intérprete, el Programa, las Instrucciones y los demás personajes de la obra.

El *capítulo 2* te muestra cómo te puedes comunicar con tu equipo, utilizando los recursos del teclado y la pantalla. Aprenderás a escribir tus primeros programas en BASIC y cómo ponerlos en marcha.

El *capítulo 3* te enseña a hacer cálculos con el BASIC.

El *capítulo 4* te ayuda a escribir programas que podrán ser utilizados una y otra vez. Es más, aprenderás a manipular las variables de modo correcto y eficaz.

El *capítulo 5* te enseña los procedimientos a seguir para que tus programas queden claros, bien organizados y fáciles de leer.

El *capítulo 6* te enseña a tomar decisiones complejas basadas en la lógica y en los valores.

El *capítulo 7* explica la manera de automatizar las tareas repetitivas, mediante el uso de bucles.

El *capítulo 8* muestra el método correcto para realizar el diseño de un programa, desde el algoritmo inicial hasta el programa completo y documentado, incluyendo el diseño del diagrama de flujo.

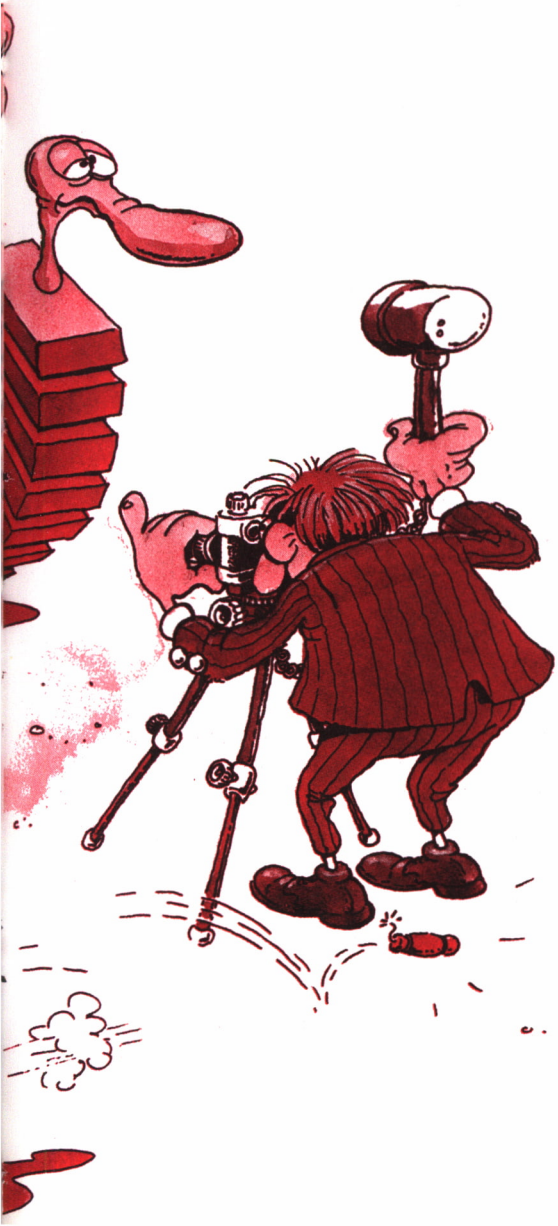
El *capítulo 9* te ayuda a aplicar a un caso práctico todos los conceptos aprendidos.

El *capítulo 10* te ayuda a examinar el siguiente paso en tu camino a la programación sofisticada.

Los *apéndices A, B,* y *C* te dan las respuestas a algunos ejercicios seleccionados, una lista de las palabras reservadas del BASIC y un glosario de los términos empleados en el libro.

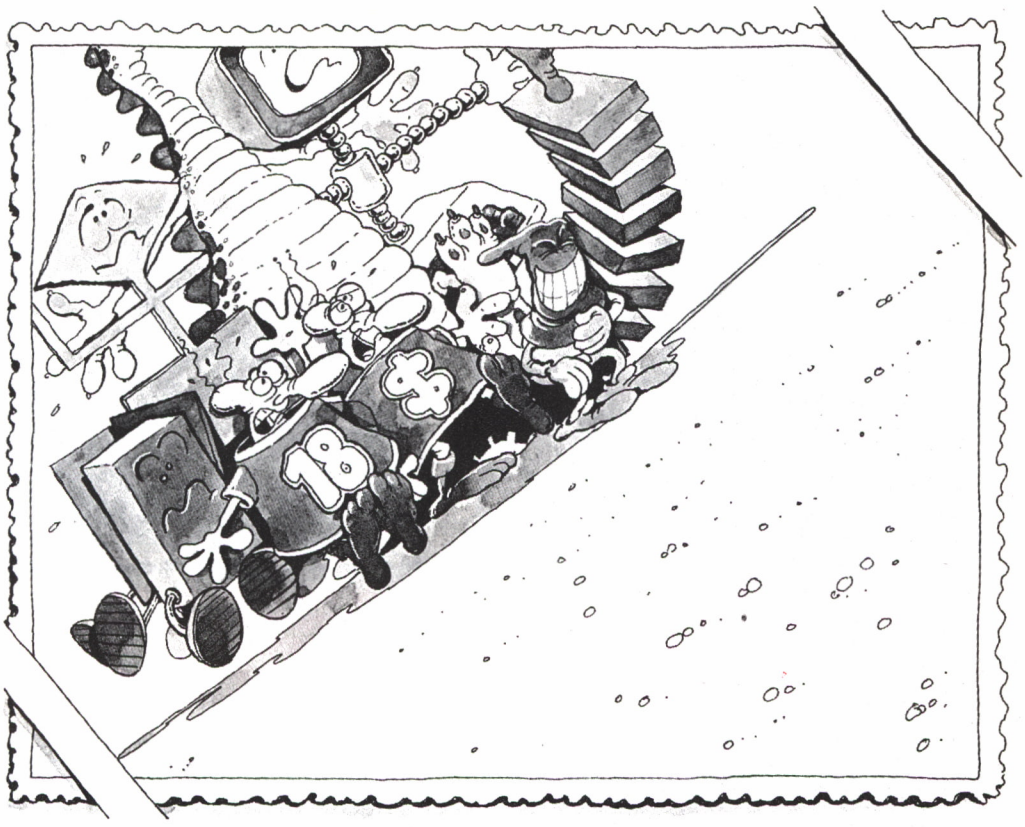
Si estás preparado, vamos a abrir nuestro álbum familiar para que te presente a todods los héroes de este libro...


18
$



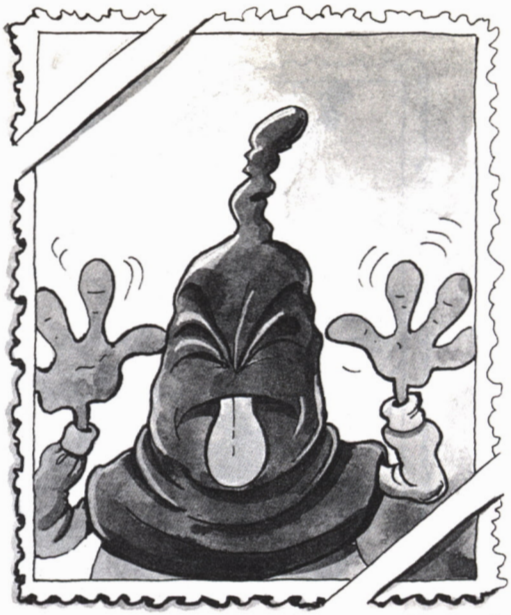
Conoce a nuestros héroes

Siguiendo las agujas de un reloj, tenemos a: Dino el Programador, el Intérprete de BASIC apoyado sobre su amigo el Ordenador, la Serpiente-Programa, el travieso Armafallos, dos Variables, Instrucciones de Programa listas para colocarse en sus respectivos sitios, y el indispensable Diagrama de Flujo sobre el que se apoya Dino... ¿Cómo...? ¡Parece que nuestro Armafallos ya está tramando algo!

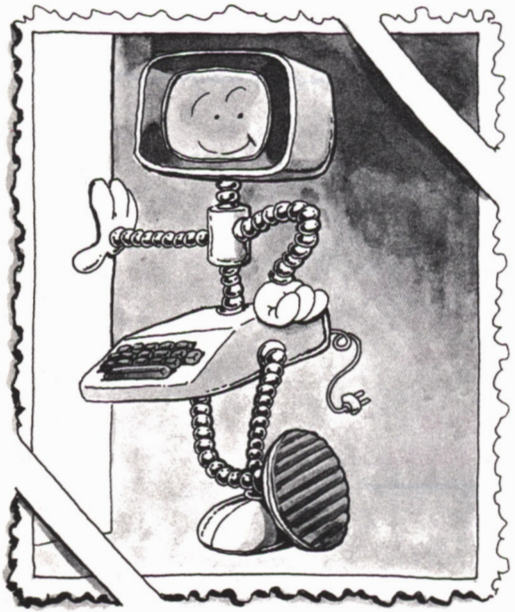


Lo siento... el travieso de Armafallos se ha salido con la suya otra vez. Nos ha fastidiado la foto del grupo...

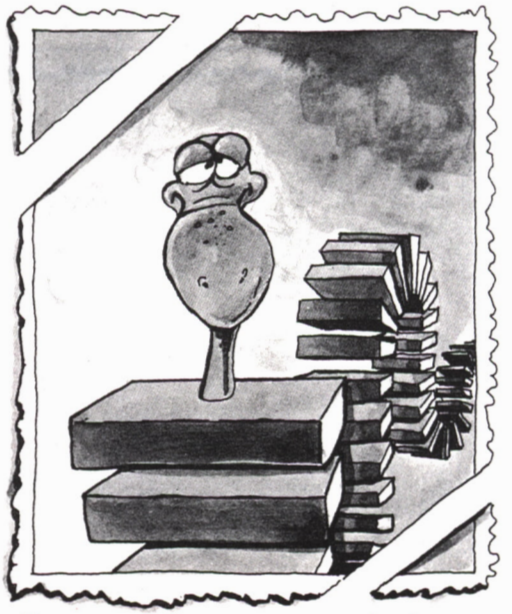
Te presento al Intérprete de BASIC. Cuando está despierto reside en la memoria de tu ordenador. Su misión es la de traducir tus instrucciones para el ordenador, ayudándote de todas las maneras que le sea posible.



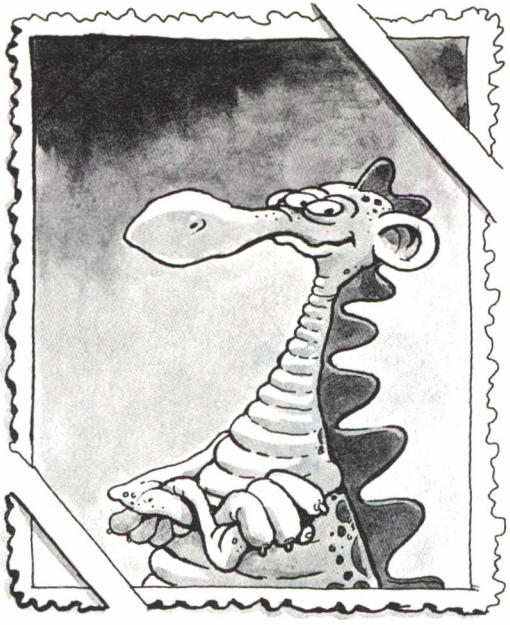
Jamás has de olvidar esta cara. Es la de Armafallos. Hará lo que sea para hacerte la vida imposible. Tendrás que esforzarte al máximo para que no se acerque a tus programas.



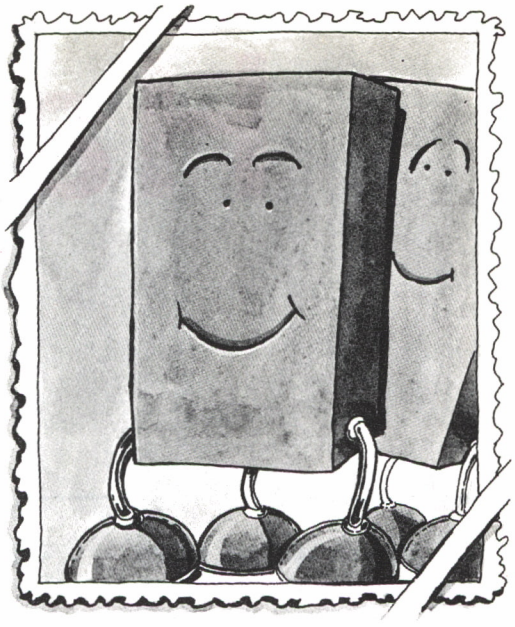
Aquí tenemos a tu buen amigo el Ordenador, siempre a tus órdenes.



No te precipites, que no es ningún monstruo, sino la Serpiente-Programa. Está formada por una serie de instrucciones de programa. Vas a aprender a construirla. Es muy dócil y fácil de manejar, una vez que la conozcas, pero ¡que no se le acerque Armafallos!

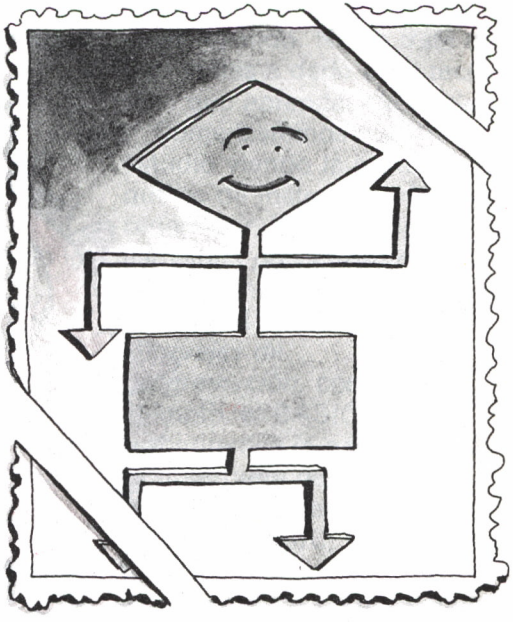


Te presento también a Dino. Es muy simpático y, aunque nunca haya estudiado en un colegio, podrá demostrarte lo fácil que es escribir programas en BASIC.



Aquí tenemos a las Instrucciones de BASIC, listas para incorporarse a tus órdenes en la Serpiente-Programa.

Esta criatura es una variable numérica, y lleva su nombre sobre la chaqueta, lo cual le da un valor. No está contento porque preferiría estar en su casilla reservada de memoria.



Aquí tenemos a tu mejor amigo, el Diagrama de Flujo. Te ayudará a crear programas que funcionen bien.

Hablando

BASIC

1

En la introducción empecé diciendo que dentro de una hora estarías escribiendo tu primer programa en BASIC, de modo que ¿por qué empezar con un capítulo dedicado a conceptos y definiciones básicos? ¿No estamos acaso perdiendo un tiempo precioso? Al contrario, si nuestro propósito es *aprender* para luego *retener* todo lo que vamos viendo, será preciso abandonar el aprendizaje de memoria a favor del entendimiento verdadero de lo que vamos a ver. La información que se cubre en este capítulo te ayudará a entender mejor qué es la programación, cómo se ejecuta un programa en BASIC, y el vocabulario de la informática (el mundo de los ordenadores).

Antes de que empecemos a escribir nuestro primer programa en BASIC, hay unas cuantas definiciones y conceptos que has de aprender. Una vez que entiendas estos términos, podré explicar siempre *todo lo que pasa* y *lo que hay que hacer,* de una manera muy sencilla pero a la vez precisa, para que estés al tanto de todo. Lee este capítulo con cuidado, asegurando que entiendas de verdad lo que estás leyendo. No se trata de que te líes a darle a las teclas, sino que aprendas de verdad.

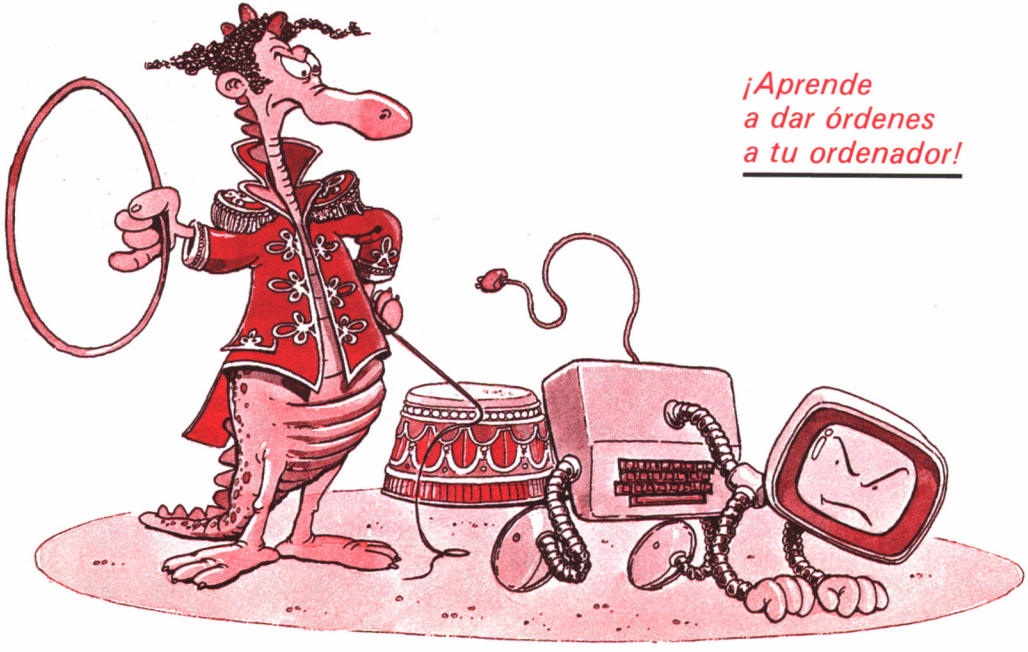
Comenzaremos nuestro estudio viendo la forma en la que se da instrucciones a un ordenador, lo que propiamente llamamos *programación*. A continuación, explicaremos la necesidad de los llamados *lenguajes de programación,* como, por ejemplo, el BASIC. Luego discutiremos la naturaleza de un intérprete de BASIC y exploraremos la historia del lenguaje BASIC, sus dialectos y sus usos. Por último, vamos a examinar los componentes de un *equipo de ordenador* y aprender un poco del argot técnico que se emplea para describir estos mismos componentes.

La programación

Tu ordenador es una máquina diseñada expresamente para manejar datos, ya sean en forma de texto o numéricos. Por ejemplo, puedes hacer que tu ordenador haga aparecer palabras y frases en una pantalla —lo cual se conoce como *proceso de textos*— o puedes hacer que convierta un peso expresado en gramos a su valor en kilogramos, lo cual se conoce como *proceso numérico*. Para hacer que tu ordenador lleve a cabo estos procesos, es necesario facilitar a la máquina unas instrucciones en un formato o «lenguaje» que sea capaz de comprender. Cualquier ordenador es capaz de «entender» (es decir, reconocer y ejecutar) únicamente un número reducido de instrucciones diferentes (digamos unas trescientas o cuatrocientas).

Aquellas instrucciones que un ordenador es capaz de entender directamente se llaman instrucciones en *lenguaje máquina*. Estas instrucciones se almacenan en un formato binario, es decir, en grupos de ceros y unos en la memoria de la máquina. Cada 0 ó 1 se llama *bit*, y un grupo de bits se llama *octeto* o *byte* (pronunciado «bait»).

Una secuencia de instrucciones que lleva a cabo alguna tarea se llama *programa* (una secuencia de instrucciones que no lleva nada a cabo es erróneo). Tu ordenador ejecuta un programa mediante la ejecución de cada instrucción en un orden fijo, Ahora bien, el escribir un programa (una secuencia de instrucciones) en lenguaje máquina, es decir, en formato binario, es un proceso lento y trabajoso.



¡Aprende a dar órdenes a tu ordenador!

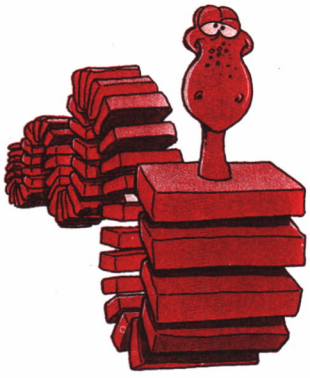
Sería fabuloso poder formular estas instrucciones oralmente o por escrito en nuestro propio idioma (castellano) y hacer que el ordenador las ejecutara. Desgraciadamente, esto no es posible, ya que el ordenador no entiende los idiomas que podamos hablar nosotros, ni en forma escrita ni oral. La razón a la que se debe este hecho es muy sencilla: un ordenador ejecuta sus órdenes al pie de la letra; es lógico y preciso y, por tanto, requiere que sus instrucciones sean claras, concisas y en la forma y secuencia exactas.

La dificultad que se encuentra con los idiomas humanos tiene su origen en la naturaleza propia de estos lenguajes, la posibilidad de construir frases ambiguas cuyo sentido depende del contexto o de los gestos y expresiones que las acompañan. Este tipo de comunicación no puede ser interpretado por un ordenador. Hasta el castellano escrito con la mayor exactitud sigue siendo demasiado impreciso para un ordenador. Por ejemplo, sería imposible decir a un robot computerizado que fuera «a la cocina a freír un huevo» y esperar un resultado positivo, a no ser que el robot haya sido previamente programado en este sentido. Hace falta entrenar (o programar) al ordenador antes de que sea capaz de desenvolverse en nuestro mundo. Aun así, pese a que conozca perfectamente tu cocina, por ejemplo, puede que fracase en la cocina de un amigo, ya que todo se encuentra colocado de distinta manera. En resumidas cuentas, la comunicación con un ordenador ha de ser siempre clara, concisa y concreta.

Por esta razón se inventaron unos «idiomas» simplificados para comunicar con los ordenadores. Recordarás que el lenguaje binario (conocido también como lenguaje máquina) es el que mejor entiende un ordenador, pero que a la vez es poco práctico para nuestro uso. Por ello, para facilitar las comunicaciones, se han inventado estos lenguajes, que se parecen en parte a los lenguajes humanos (especialmente al inglés) y se denominan *lenguajes de alto nivel.*

Para lograr una comunicación clara y eficaz con un ordenador, se suele emplear un número limitado de palabras derivadas del inglés, en un formato de órdenes predefinidas. Además, las frases o *sentencias* que especifican las instrucciones para el ordenador han de obedecer unas reglas gramaticales

«¡Me chifla el BASIC!
¡Anda, háblame en BASIC!»



«¿Te acuerdas de mí? Soy el Programa y estoy hecho de una serie de instrucciones.»

muy estrictas, cuyo conjunto forma la *sintaxis* del lenguaje. La combinación de un vocabulario limitado con una sintaxis definida se llama *lenguaje de programación,* siendo el BASIC un ejemplo de uno de estos lenguajes.

En resumen, un *lenguaje de programación* se compone de las reglas (sintaxis) y las palabras y símbolos (vocabulario) que permiten enviar tus órdenes a un ordenador en un formato que sea capaz de interpretar correctamente. Una secuencia de estas instrucciones se llama *programa.*

Veamos un ejemplo. Supongamos que queremos sumar las cifras

```
2 + 2
```

y hacer aparecer el resultado en la pantalla. En BASIC, escribiríamos:

```
1 R = 2 + 2
2 PRINT R
```

donde R es el resultado de la operación (*Nota:* en inglés, PRINT quiere decir «imprime» o «poner en pantalla»).

«¿Te acuerdas también de mí? Soy el Intérprete de BASIC, listo para traducir tus instrucciones al ordenador. Soy un programa y resido en la memoria de tu ordenador.»

Espera un momento... ¿no habíamos dicho antes que lo único que puede entender directamente el ordenador es el lenguaje máquina? ¿No acabamos de dar una instrucción al ordenador usando una palabra inglesa? Parece que hay una contradicción. Pues no la hay en realidad. Es cierto que el ordenador, sin más, no puede comprender el BASIC directamente, ni siquiera ningún otro *lenguaje de programación de alto nivel* (o sea, lenguajes que emplean frases parecidas al inglés). Para que un ordenador entienda un programa escrito en un lenguaje de alto nivel, como el BASIC, el programa ha de ser *traducido* primero por otro programa especial llamado, con razón, *intérprete.* En otras palabras, hablas con tu ordenador mediante la ayuda de un intérprete de modo que, para ejecutar un programa en BASIC, tu ordenador ha de disponer de un intérprete de BASIC. Veamos qué hace un intérprete.

El intérprete de BASIC

Un intérprete de BASIC lee cada instrucción que escribes en el teclado y la traduce automáticamente a una secuencia de instrucciones en lenguaje máquina que tu ordenador puede entender y ejecutar. Dado que todo este proceso ocurre dentro de tu ordenador, sucede de una manera totalmente imperceptible. Una vez que entre en acción el programa intérprete, a efectos prácticos tu ordenador ya puede hablar en

BASIC. Es posible que pueda hablar otros lenguajes de programación, sólo hace falta que posea los intérpretes correspondientes.

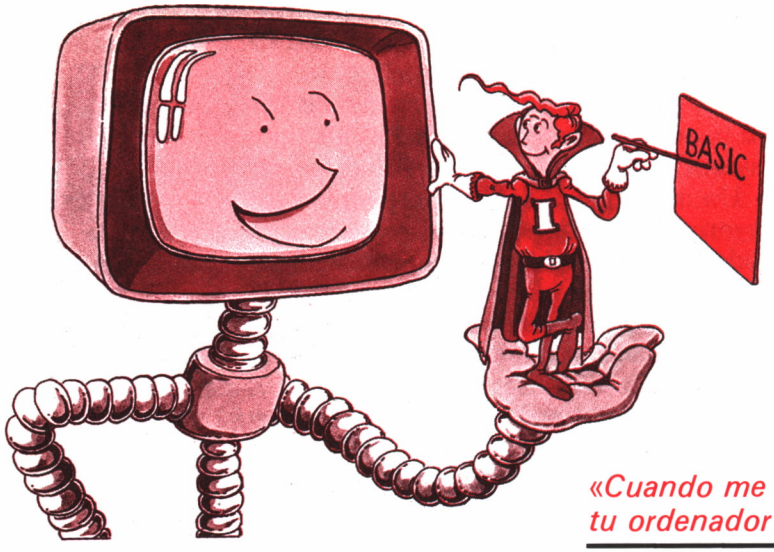
Hay varios tipos de intérpretes de BASIC que puedes usar con tu equipo. Hablaremos de los dos tipos más empleados: *residente* y *no residente*.

Un intérprete de BASIC *residente* viene incorporado en la mayoría de los equipos de microordenadores. Se le llama residente porque reside permanentemente en la memoria del ordenador. Puede que entre en funcionamiento automáticamente nada más encender el ordenador, o bien puede activarse mediante una instrucción, como «B» o «BASIC». Cuando aparece en la pantalla el símbolo (>) confirmándolo, sabrás que tu ordenador está listo para ejecutar tus instrucciones en BASIC.

Un intérprete residente en memoria normalmente tiene una limitación: sólo puede suministrarte una versión reducida de BASIC. Dado que un intérprete está «incorporado» en la memoria permanente del ordenador, ha de ocupar poco espacio, ya que la memoria total es limitada. Esta memoria tiene que contener, además del intérprete de BASIC, los programas a ejecutar, el espacio necesario para los cálculos, la organización del sistema y los datos sobre los que se va a trabajar. Estos requisitos de espacio limitan la extensión, y, en consecuencia, la complejidad del intérprete que reside en memoria. En los equipos que vienen provistos de poca memoria, el intérprete residente muchas veces es un «mini BASIC» que limita en gran parte lo que se puede hacer con él. Cualquier BASIC residente en memoria, sin embargo, será suficiente para aprender a programar en BASIC, al menos para lo que intentaremos hacer en este libro. Más tarde, cuando hayas aprendido a hacer programas más complicados, querrás más facilidades, por lo que será necesario un intérprete *no residente*.

Para poder facilitar más funciones, el intérprete ha de ser más complicado y, por tanto, más grande. En este caso se almacena en un dispositivo para almacenaje de memoria en masa, como es un cassette o un diskette (llamado este último, muchas veces, *floppy*), para evitar ocupar en todo momento la mayor parte de la memoria asequible. Con la mayoría de los equipos pequeños normalmente es preciso la compra de memoria adicional para acomodar un intérprete más completo.

Un intérprete más potente consistirá en una versión de BASIC denominada *BASIC completo, BASIC ampliado, BASIC de coma flotante* o *BASIC avanzado,* según quien lo suministre, o *BASIC de cassette* o *BASIC de disco,* según el medio magnético de almacenaje que se emplea.



«Cuando me alojo en la memoria, tu ordenador puede hablar en BASIC.»

Normalmente, todo programa escrito en un BASIC residente en memoria podrá ejecutarse sin modificaciones a la hora de emplear un BASIC completo. En estos casos, se dice que las dos versiones gozan de una *compatibilidad vertical.*

Para utilizar un BASIC de cassette o de disco, primero es preciso trasferir el intérprete del medio magnético a la memoria del ordenador; dicha operación se denomina *cargar* el intérprete. A continuación es necesario dar una instrucción específicamente reservada para activar el intérprete de BASIC completo, como puede ser la instrucción «FBASIC». A continuación el intérprete hace aparecer en pantalla un símbolo, que normalmente se diferencia del del BASIC residente para evitar confusiones. Sólo entonces es posible enviar a la máquina instrucciones en BASIC.

Pasemos ahora a ver qué es en sí el BASIC, cómo ha sido su desarrollo y los dialectos que de él se han derivado.

¿Qué es el BASIC?

El desarrollo de los lenguajes de alto nivel tuvo como finalidad principal hacer más fácil al usuario el dar instrucciones a un ordenador, es decir, programarlo. A lo largo de los últimos años se han diseñado centenares de lenguajes de programación.

En los primeros tiempos de la programación, los ordenadores se empleaban primordialmente para usos científicos, por lo que los primeros lenguajes de programación

fueron diseñados para facilitar el cálculo numérico. En este sentido podemos considerar que el lenguaje FORTRAN es el abuelo de estos lenguajes. Su nombre se deriva de las palabras iglesas FORmula TRANslator (traductor de fórmula), y fue desarrollado expresamente para resolver cálculos numéricos. Este lenguaje, sin embargo, poseía algunos defectos que hicieron necesaria la creación de nuevos lenguajes. El BASIC es uno de estos lenguajes; el COBOL, APL y Pascal son otros que han llegado a gozar de gran aceptación.

El desarrollo del BASIC fue todo un acontecimiento, ya que su diseño se hizo pensando en la sencillez y facilidad de aprendizaje. Es más, es un lenguaje *interactivo,* lo cual merece una explicación.

Las siglas BASIC vienen de las palabras Beginner's All-purpose Symbolic Instruction Code (algo así como Código de Instrucción Simbólica Polivalente para Novatos). Fue diseñado en 1964 por John Kemeny y Thomas Kurtz, de la Universidad de Dartmouth, quienes trabajaban con la ayuda de fondos otorgados por la Fundación Nacional de Ciencia, en Estados Unidos. El objetivo de sus creadores fue el de diseñar un lenguaje que un principiante pudiera utilizar con facilidad. Tuvieron éxito en su empeño, ya que hoy día el BASIC es uno de los lenguajes de programación más fáciles de aprender.

Debido a que el BASIC fue diseñado para que fuese interactivo —de hecho fue el primer lenguaje interactivo—, el usuario (¡ese eres tú!) podía relacionarse con el programa directamente a través de un terminal, en vez de manipular montones de tarjetas perforadas de IBM, como era el caso hasta entonces. El BASIC primero funcionó en el sistema de tiempo compartido GE225 de la Universidad de Dartmouth. Había terminales por toda la Universidad, de forma que muchos usuarios tenían acceso al sistema simultáneamente.

El éxito del BASIC fue verdaderamente estelar. La compañía General Electric decidió emplearlo inmediatamente para uso comercial. Kemeny y Kurtz publicaron el primer libro sobre el BASIC en el año 1967. Las empresas Hewlett Packard y Digital Equipment Corporation decidieron equipar la mayoría de sus ordenadores con este lenguaje.

El lenguaje BASIC posee dos ventajas sobre los lenguajes del tipo de FORTRAN:

1. *Para el usuario:* El BASIC es el lenguaje de más fácil aprendizaje, especialmente para los principiantes.
2. *Para el fabricante:* El BASIC es el lenguaje de más fácil incorporación a un equipo, ya que es un lenguaje sencillo que requiere una cantidad mínima de memoria.

Hubo un tercer factor que contribuyó de una manera significativa al enorme éxito del BASIC: la llegada de los microordenadores baratos. Al ser asequibles a gran número de personas a finales de los años setenta, el BASIC se convirtió en el lenguaje universal de estos pequeños ordenadores. Debido a que el intérprete de BASIC para una versión simplificada requiere tan sólo 4K (4.096 bytes) de memoria, hasta los ordenadores más pequeños podían disponer de un BASIC residente (recuerda que el BASIC residente es un intérprete que reside en la memoria permanente del ordenador). Los ordenadores de mayor potencia que han salido más recientemente disponen de memorias más grandes (64K o más, siendo 1K igual a 1.024 bytes) y, por tanto, son capaces de utilizar versiones de BASIC más potentes.

Hoy día casi todos los ordenadores disponen del lenguaje BASIC. A través de los últimos años los fabricantes han ido añadiendo funciones y ampliando el lenguaje, de manera que actualmente el BASIC es uno de los lenguajes menos estándars que hay. No hay dos dialectos del BASIC que sean iguales, hasta el punto de que el BASIC ahora es una familia de lenguajes en lugar de uno sólo. Pese a que se hayan propuesto muchas versiones estándar, hasta la fecha ninguna ha tenido éxito, ni parece que lo vaya a tener en un futuro previsible. ¿Quiere decir esto que tendrás que volver a aprender el BASIC cada vez que utilices un ordenador distinto? Afortunadamente, éste no es el caso. Una vez que conozcas las características esenciales que poseen todas las versiones en común, podrás aprender con facilidad las modificaciones que ofrece cualquier versión. Todos los dialectos comparten el mismo cuerpo básico de instrucciones, que aprenderás a lo largo de este libro.

¿Qué versión del BASIC?

Es probable que tu ordenador venga provisto de varias versiones de BASIC. Anteriormente hemos descrito los dos tipos principales del BASIC: el residente (generalmente un mini-BASIC) y el no residente (el BASIC completo o ampliado). Veamos brevemente algunas de sus diferencias.

Un mini-BASIC tiene menos facilidades o funciones que uno completo o ampliado. Una limitación frecuente con el mini-BASIC es que sólo es posible trabajar con números enteros, o sea, no trabaja con fracciones, por lo que se le suele llamar «BASIC entero». En cambio, una versión mejorada del BASIC que puede manejar fracciones de números se denomina «BASIC de punto flotante» (por el punto decimal), una característica muy deseable si se piensa hacer cálculos.

Un mini-BASIC normalmente almacena información sobre un cassette y no en diskette, e incluso es posible que sólo guarde programas y no datos, de manera que no puede manejar archivos. Generalmente sólo trata información que sea parte de un programa o que provenga del teclado, mientras un BASIC completo usando unidades de disco ofrece grandes ventajas y potencia a la hora de manipular tanto datos (archivos) como programas.

Algunos fabricantes suministran versiones de BASIC aun más avanzadas, que son específicas de cada uno. El usuario queda advertido de este hecho al ver que el fabricante ofrece un BASIC ampliado o «extendido», lo cual significa que hay unos recursos más potentes a la disposición del programador. Por otro lado, el uso de estos recursos especiales generalmente hace que un programa en su versión de BASIC sea incompatible con otros intérpretes de BASIC.

La versión que se emplea en este libro es un mini-BASIC universal para que puedas adquirir conocimientos válidos para todo tipo de BASIC. No obstante, iremos mencionando las variedades y modificaciones más comunes a lo largo del texto.

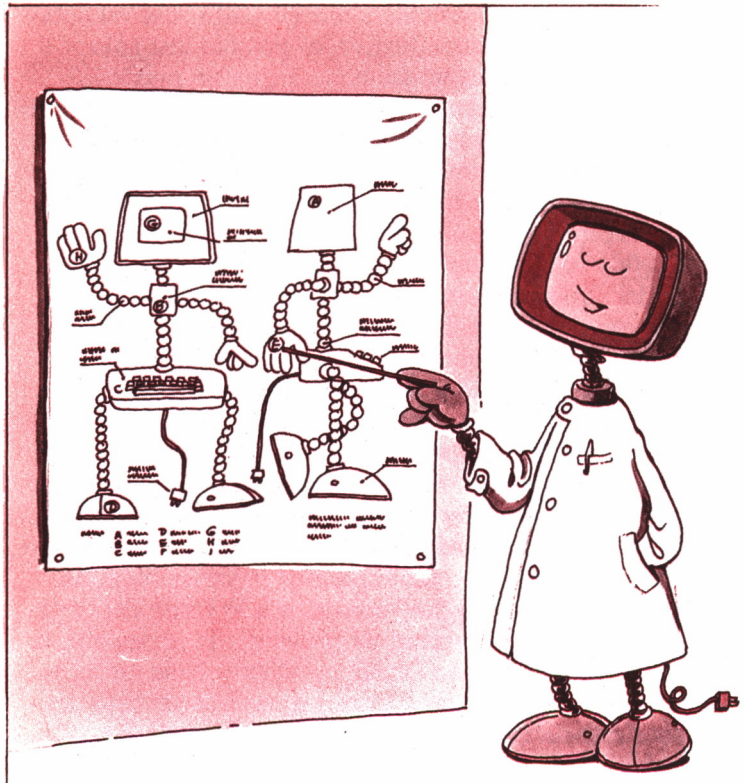
En resumidas cuentas, no has de preocuparte ahora con la versión de BASIC que vas a usar de momento. Más tarde, si deseas escribir programas más complejos, podrás servirte del manual de referencia correspondiente a la versión de BASIC que hayas elegido. Es más, en el último capítulo te informaremos sobre muchas de las características avanzadas del BASIC.

Ahora que hemos visto los lenguajes de programación en general y el BASIC en particular, pasemos a tu ordenador y la forma en la que procesa información.

Tu equipo

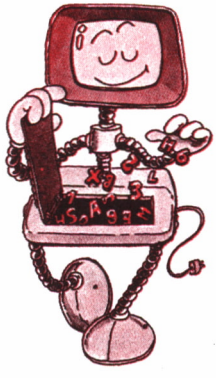
Tu ordenador maneja la información y se comunica contigo mediante un teclado y una pantalla, y opcionalmente a través de una impresora. El teclado se emplea para enviar información al ordenador. Cada vez que se pulse una tecla, el código electrónico que corresponde al carácter de esa tecla se envía al ordenador, donde es reconocida y ejecutada, o bien es ignorada. El teclado es tu *dispositivo de entrada de datos,* mediante el cual se suministra información al ordenador.

La *pantalla,* a veces llamada tubo de rayos catódicos (TRC), visualiza la información que el programa genera. Normalmente, cada carácter que pulsas en el teclado aparecerá en la pantalla, habiendo sido enviado primeramente



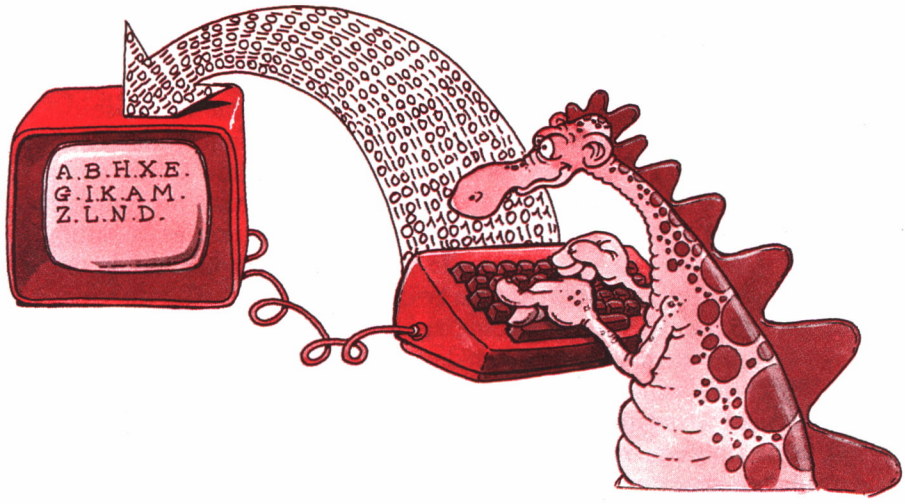
«Soy robusto, amigable y una gran ayuda una vez que me conozcas.»

al ordenador para luego «rebotar» a la pantalla. En general, no existe ninguna conexión directa entre el teclado y la pantalla, sino que todas las comunicaciones pasan a través del ordenador, como muestra el dibujo.



«Aquí está mi teclado.»

«Necesito que me digas algo para saber qué hacer.»



El teclado envía información al ordenador.

Según el diseño del sistema, el *ordenador* puede formar parte del teclado, la pantalla, las unidades de disco o estar separado de los demás componentes, pero, independientemente de su integración física en el sistema, constará siempre de una unidad central de proceso, una memoria y varios *interface* (conexiones para impresoras y otros dispositivos). Veamos cada uno de estos tres componentes pertenecientes al ordenador mismo.

La *unidad central de proceso* (CPU) recoge instrucciones de programa, una por una, y las ejecuta. Requiere pocos componentes, los cuales se denominan circuitos integrados o «chips». Todos los microordenadores utilizan un chip llamado *microprocesador* como elemento básico de su CPU. El dibujo siguiente muestra un chip típico.

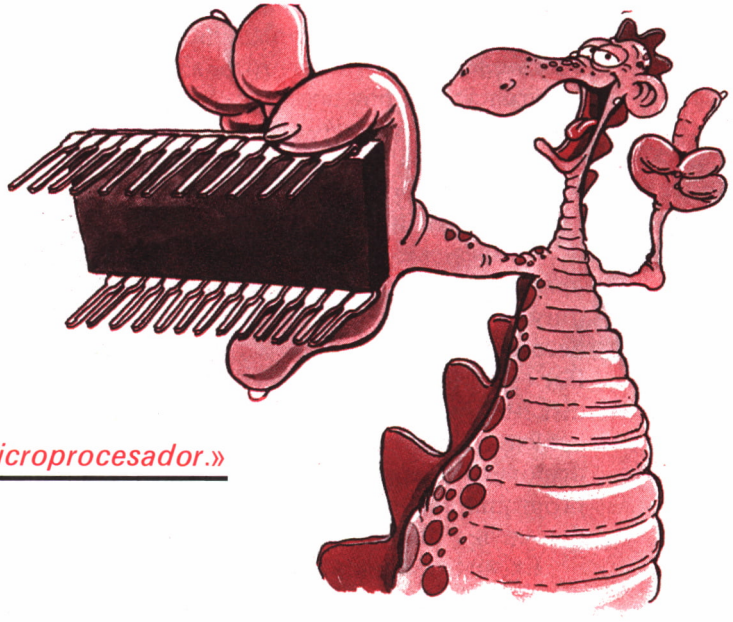
La *memoria* almacena tanto los programas como toda la información que los programas manejan, leen o generan durante su ejecución, Para ejecutar un programa, primero es necesario que se coloque dentro de la memoria del ordenador. Por ejemplo, si un programa proviene de un cassette o un diskette, ha de ser trasladado a la memoria del ordenador, lo cual se conoce como *cargar* el programa. La memoria ha de disponer de espacio suficiente como para almacenar tanto el programa más grande que se piensa utilizar, como los datos que vaya a manipular ese programa.

Existen dos tipos diferentes de memoria en tu ordenador: la memoria ROM y la memoria RAM. La memoria «normal»

que usarás para almacenar tus programas es la RAM, o memoria de acceso directo. La RAM se utiliza para guardar información y leerla posteriormente. Un chip de RAM se parece al del microprocesador del dibujo; la única diferencia es lo que hay dentro. Por desgracia, dado el estado actual de la tecnología, no es posible evitar que desaparezcan los datos si se corta el suministro de electricidad; la memoria es «volátil». Por esto mismo es sumamente importante que al final de una sesión de trabajo guardes tu programa en un medio no volátil, como un cassette o diskette, si has de conservarlo. Por otro lado, acuérdate de la necesidad de tener un intérprete de BASIC en memoria antes de poder ejecutar tu programa.

ROM significa memoria de sólo lectura. Las memorias de este tipo vienen provistas de programas cargados por el fabricante que no se pueden modificar, ya que esta memoria es no volátil y no puede borrarse. En general, el intérprete de BASIC reside en ROM, al igual que el *monitor,* un programa especial que se necesita para comunicarse una vez que se encienda el ordenador.

Si tu ROM no contiene nada, no sabrá qué hacer cuando escribes algo en el teclado. Como mínimo, la ROM ha de contener un monitor, cuyo programa examina la información enviada desde el teclado, actuando sobre funciones de régimen interior, tales como el arranque del intérprete de BASIC residente o la carga de un programa de un cassette.

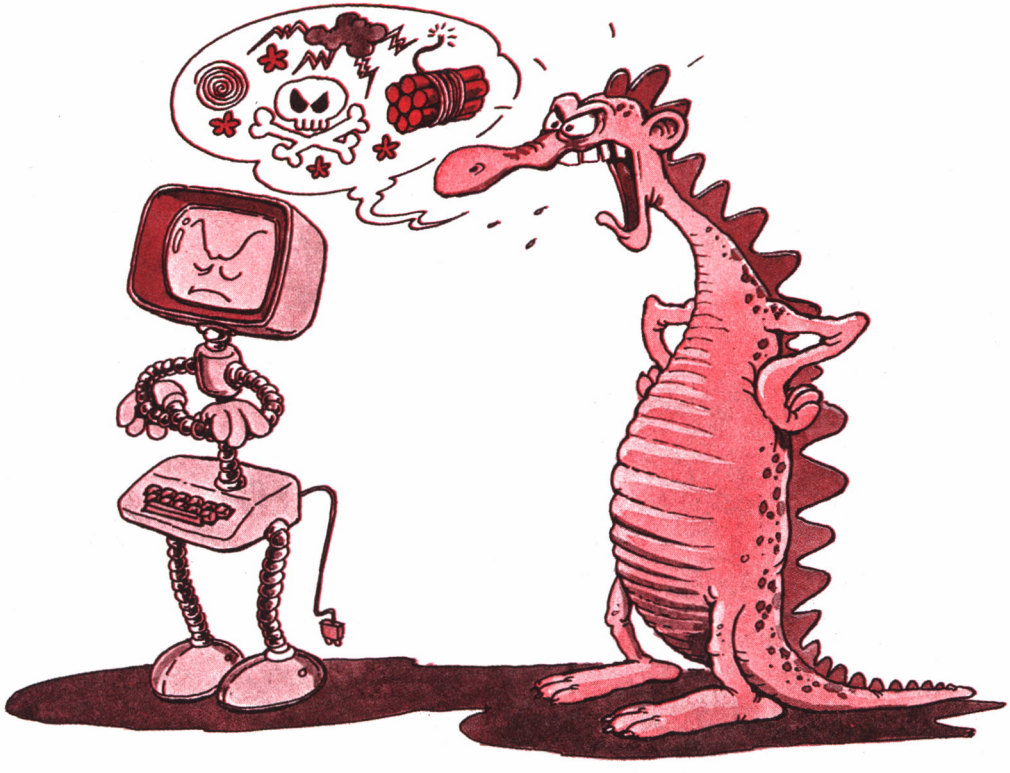


«Esto es un microprocesador.»

No puedes usar la ROM para almacenar otros programas; todos los que empleas en tu ordenador se cargan en memoria RAM. Se pueden adquirir cartuchos con programas para tu ordenador, en cuyo caso los programas residen en memoria ROM dentro del cartucho.

Existen al menos dos aparatos adicionales que se conectan al ordenador: un soporte de memoria y una impresora. Estos mecanismos se conectan mediante los *interfaces,* que son los componentes electrónicos necesarios para conectar con dispositivos especiales. El soporte de memoria puede consistir en un magnetófono cassette o en una o más unidades de diskette. (*Nota:* Ambos utilizan un medio magnético para archivar información y son capaces de almacenar bastante más información que la memoria electrónica interna de la computadora.) Estos aparatos requieren un interface específico que se conecta al circuito principal del ordenador y que les permite comunicarse con el microprocesador. La mayoría de

Tu ordenador se negará a reaccionar hasta que introduzcas un programa en su memoria.



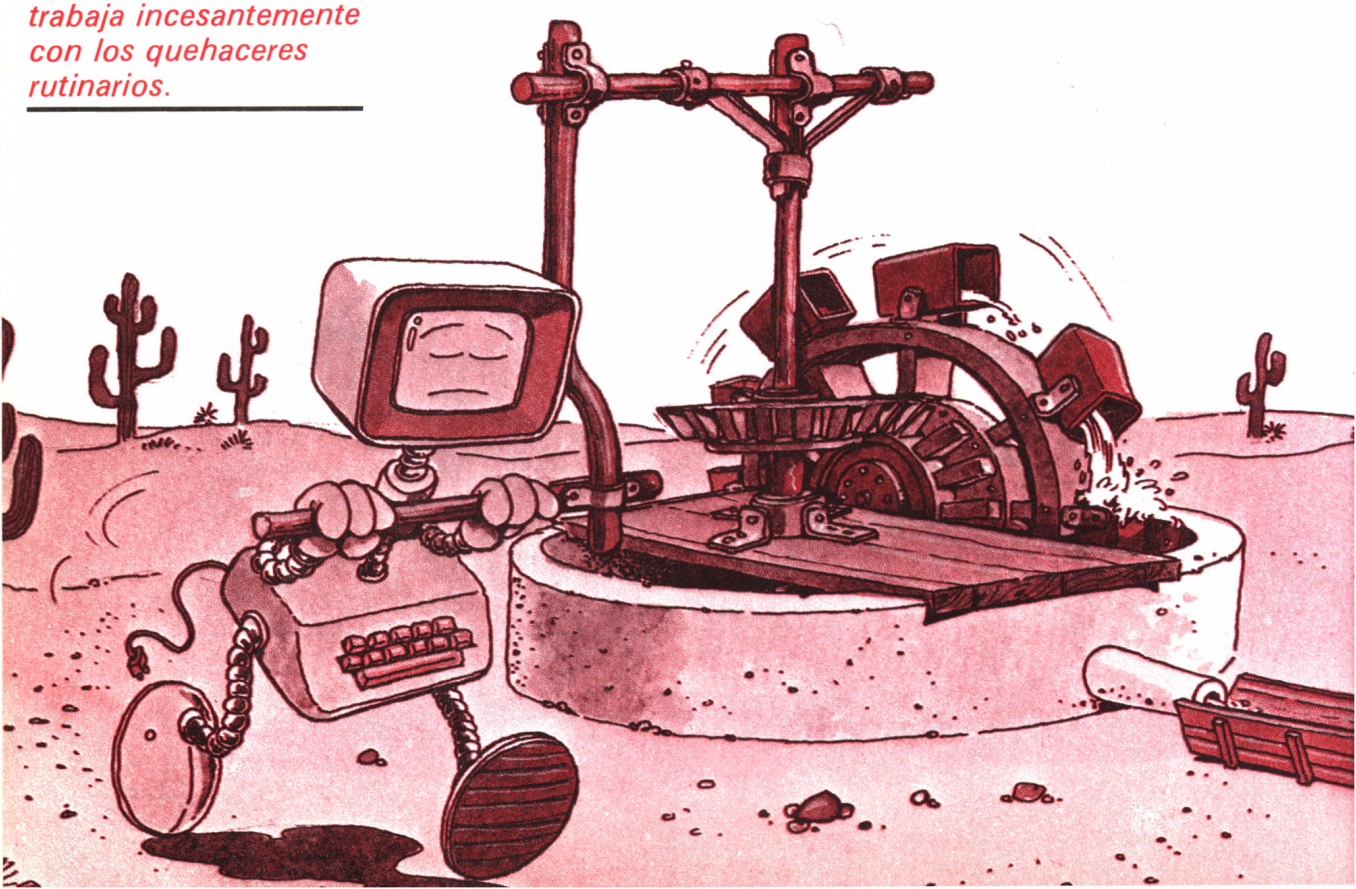
los ordenadores personales vienen provistos de un interface (acoplamiento) incorporado para la conexión a cassette, mientras que las unidades de disco y las impresoras normalmente precisan una tarjeta interface para conectarse al ordenador.

Una impresora sirve para obtener listados o copias permanentes de los programas o sus resultados. La impresora es, por tanto, un *dispositivo para la salida de datos (output)*, al igual que la pantalla, y existen instrucciones específicas en el BASIC para enviar información a cualquiera de los dos.

Por último, el *modem* es un aparato que se emplea con frecuencia. Te permite comunicarte con otro ordenador o terminal a través de las líneas normales de teléfono, siendo además muy útil a la hora de usar una red *(network)* comercial o de establecer acceso a *bancos de datos* (bibliotecas de información).

Bien, ya hemos visto todo el vocabulario que había que conocer, pero antes de pasar al capítulo 2 y comenzar a usar el ordenador, conviene tener en cuenta las siguientes advertencias.

El programa "monitor" trabaja incesantemente con los quehaceres rutinarios.

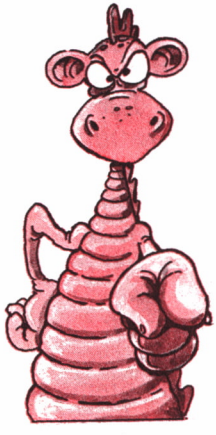


Los ordenadores y la sintaxis

Los ordenadores son rápidos, pacientes y muy exactos. Hacen lo que se les dice que han de hacer, y lo cumplen al pie de la letra. Si quieres comunicarte con éxito con tu ordenador, has de ser siempre exacto. Si te equivocas al escribir una instrucción en BASIC, no dañarás a tu ordenador, pero lo más probable es que tu programa no se ejecute correctamente. Se parará y te comunicará que ha habido un «syntax error».

Piensa que la sintaxis es el conjunto de reglas que especifican la manera correcta de escribir una instrucción en BASIC; reglas que son además sumamente rígidas. No te dejarán deletrear algo mal; donde haya que poner un punto no sirve ni una coma ni un punto y coma en su lugar. Cada símbolo o carácter tiene su interpretación exacta y única, su significado preciso. Cualquier desviación de las reglas significará el fracaso o, al menos, resultados totalmente inesperados.

Ten siempre en cuenta que, si no respetas las reglas, tu programa no se ejecutará. No intentes ser creativo, las reglas son sencillas, directas y fáciles de seguir al escribir líneas de programación. Es mejor esmerarse y ser creativo a la hora de definir el diseño global del programa y de planificar las tareas que se quieren llevar a cabo. En pocas palabras, es esencial que sigas cuidadosamente las instrucciones y recomendaciones que encontrarás en este libro.



«No te pases de listo... hay que ser exacto.»

Comuni
con tu or

2

Este capítulo te enseñará a hablar con tu ordenador. Aprenderás a dar instrucciones en BASIC y hacer que el ordenador ponga en pantalla palabras y frases. La información intercambiada entre el ordenador y tú incluirá los programas (instrucciones en BASIC que tú envías) y los datos (los números y caracteres que mandas o recibes).

Primero aprenderás a utilizar el teclado del ordenador, para que puedas empezar a enviar las instrucciones. En concreto, estudiarás la manera de mover el cursor por la pantalla y de corregir los errores de mecanografía. Luego enviarás tus primeras instrucciones en BASIC y harás aparecer mensajes en la pantalla. Comprenderás la diferencia entre la *ejecución inmediata* y la *ejecución diferida*. Por último, seguirás los pasos necesarios para escribir y ejecutar un programa sencillo.

Para cuando hayas terminado este capítulo, estarás familiarizado con los conocimientos básicos que se requieren para comunicar con tu ordenador.

cándote denador

Utilizando el teclado

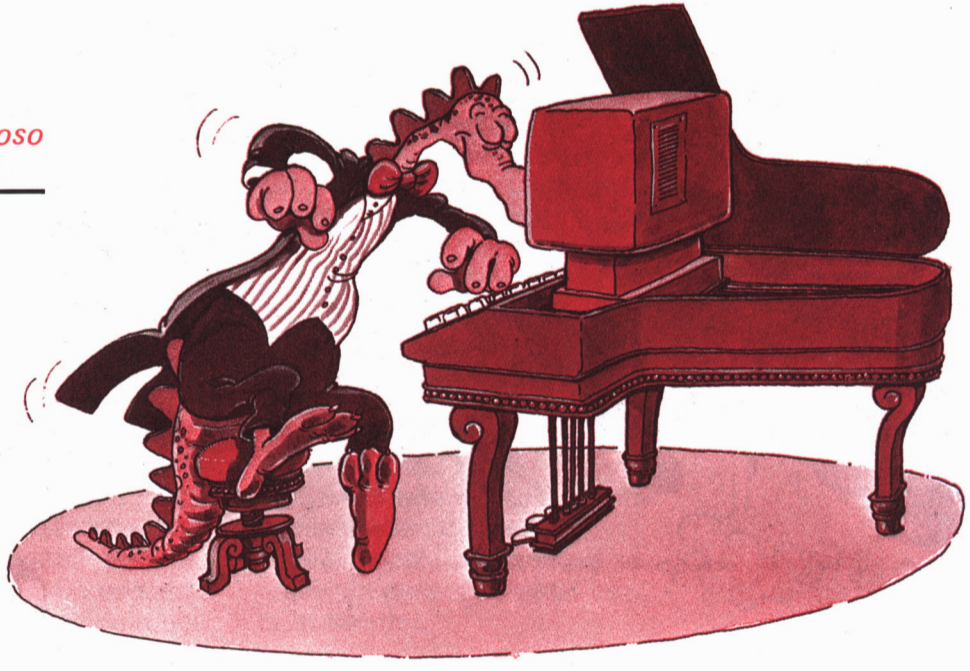
El dibujo nos muestra un teclado de ordenador. Se ve que es igual al de una máquina de escribir, excepto que posee algunas teclas más.

El teclado.

El teclado de un ordenador puede tener una variedad de configuraciones y combinaciones de teclas y, con la excepción de algunos detalles, tiene un formato estándar. Las teclas principales son:

1. Las letras del alfabeto (de A a Z).
2. Los números del 0 al 9.
3. Distintos símbolos, como, por ejemplo = , + , *, “ , $.
4. Una tecla de «retorno al margen izquierdo», marcada normalmente con la palabra RETURN, CR o ↵ .

Sé un virtuoso del teclado.



5. Una tecla SHIFT (mayúsculas) y una de control (CTRL).
6. La tecla RUBOUT (borrado) y la de ESC o BREAK (interrupción).

Algunos teclados añaden teclas para mayor facilidad de manejo, tales como un cuadro independiente de teclas numéricas y teclas con funciones especiales. Veamos ahora la misión y el uso de cada tipo de tecla.

Caracteres, números y símbolos

El uso que se hace de cada *letra, número* y *símbolo especial* es obvio: estas teclas tienen el mismo uso que en una máquina de escribir.

Return

En una máquina de escribir eléctrica, esta tecla retorna el carro al margen izquierdo y avanza el papel a la línea siguiente. En un ordenador, sin embargo, la tecla RETURN normalmente sirve para mover el *cursor* al principio de la línea siguiente en la pantalla. (El cursor es un pequeño cuadro o subrayado intermitente que indica la posición en la que aparecerá el próximo carácter en la pantalla.) En un ordenador, la tecla RETURN podría llamarse igualmente ENTER (entrar, o introducir), ya que su principal función es la de introducir un carácter o línea de texto en la memoria del ordenador. De hecho, algunos teclados ahora asignan a esta tecla la palabra ENTER, mientras otros simplemente la identifican con una flecha en ángulo recto (↵).

En todo caso, cada instrucción para el ordenador, incluyendo las instrucciones en BASIC, ha de terminar en RETURN, lo que indica «introduce la línea en memoria». En realidad, todo lo que escribas en una línea pasa inadvertido de cara al ordenador hasta que pulses la tecla RETURN, de manera que puedes corregir tus errores de mecanografía antes de introducir la línea en la memoria del ordenador.

El RETURN es importante sólo a la hora de mecanografiar, ya que, aunque sea necesario para enviar la instrucción, el RETURN no se almacena como parte de ella en la memoria. A lo largo de este capítulo introductorio, y con el fin de ayudarte en el aprendizaje, mostraremos todos los

caracteres que has de mecanografiar, incluyendo un símbolo (↵) al final de cada línea para representar la tecla RETURN.

Recuerda: tienes que pulsar RETURN después de escribir cada instrucción. Igualmente, si el ordenador te hace alguna pregunta, debes de pulsar también RETURN después de escribir la respuesta en el teclado.

Shift

La tecla SHIFT funciona de la misma manera que en una máquina de escribir: permite cambiar entre las mayúsculas y minúsculas correspondientes a los símbolos grabados en las teclas. En el caso de las teclas que no sean letras del alfabeto, la mayoría tienen dos símbolos, uno inferior y otro superior. Di no se pulsa la tecla SHIFT, se genera el símbolo inferior en la pantalla, mientras el pulsar SHIFT simultáneamente con una de estas teclas, genera el símbolo superior. Con las teclas alfabéticas, lo normal es que se genere mayúsculas o minúsculas, aunque en algunos casos sólo se puede generar mayúsculas, produciéndose en lugar de minúsculas otros símbolos o funciones. El BASIC estándar requiere que las instrucciones se escriban siempre con mayúsculas, por lo que usaremos solamente texto en mayúsculas en los programas de este libro. La ventaja de tener minúsculas en tu teclado está en el poder introducir o manejar textos en un formato normal.

Un gran número de los teclados vienen provistos también de una tecla marcada CAPS LOCK, que sirve para mantener la escritura en mayúsculas, como si la tecla SHIFT estuviese pulsada siempre. Una segunda pulsación de esta tecla anula este formato. La misión más importante que tiene la tecla SHIFT es la de disminuir el número total de teclas, ya que permite que cada tecla tenga el doble de símbolos de los que tendría normalmente.

Practica lo que acabas de aprender. Ve a tu ordenador y dale a las teclas que quieras. Pulsa cualquier tecla, es imposible dañar la máquina de esta manera. Experimenta con la tecla SHIFT a ver qué aparece en pantalla. Pulsa RETURN a ver qué pasa.

Control

La tecla CONTROL (o CTRL) se emplea para enviar instrucciones que se usan con gran frecuencia a tu ordenador, en una especie de taquigrafía. No tiene ningún equivalente en las máquinas de escribir. Se usa la CTRL de la misma manera

que la SHIFT: al pulsarla simultáneamente con otra tecla se genera otro carácter: un carácter de control. Por ejemplo, se genera la instrucción CTRL-A al pulsar las teclas CTRL y «A» simultáneamente, consiguiendo de esta manera formular una instrucción o un código de control con tan sólo pulsar dos teclas. (*Nota:* Los *códigos de control* sirven para facilitar el uso frecuente de ciertas instrucciones, como pueden ser el mover el cursor por la pantalla; es decir, hacia la derecha, la izquierda, arriba o abajo o al lado de algún punto específico, una palabra o una frase, así como para el borrado o inserción de un carácter o línea de texto.) El uso de los caracteres de control es específico a cada programa utilizado en tu ordenador, siendo su función exacta detallada por las instrucciones que acompaña el programa. En este libro no emplearemos códigos de control, excepto uno que puedes usar si quieres: el CTRL-C, cuya función, en algunos ordenadores, es la de interrumpir la ejecución de un programa.

Rubout

La tecla RUBOUT, o «borrado», tiene una función ôbvia: la de borrar un carácter. Hay teclados que no disponen de ella, dado que se puede conseguir lo mismo moviendo el cursor hacia atrás sobre la letra en cuestión.

ESC o BREAK

La tecla ESC o BREAK a veces forma parte del teclado con la función de un «código de control» estándar, común a todos los programas, permitiendo que interrumpas cualquier programa con sólo pulsar esta tecla. Cada programa, bien sea un intérprete de BASIC o uno tuyo, tiene una instrucción determinada para cesar su ejecución, como pueden ser END (final) o EXIT (salida). Por otro lado, si ocurre algo imprevisto y quieres interrumpir un programa de manera inmediata, en general puedes usar la tecla ESC. Con frecuencia es posible usar de la misma manera un código de control (como el CTRL-C), para lo que se emplearía la tecla CTRL para llevar a cabo la operación

El teclado numérico.

El teclado numérico

El teclado numérico (mostrado en el margen izquierdo) tiene mayor utilidad en las aplicaciones de gestión, en las que es necesario mecanografiar con rapidez. Este teclado independiente se emplea del mismo modo que una calculadora de mesa. Sus teclas simplemente duplican las funciones de las teclas normales del teclado principal (de 0 a 9, +, −, ×, ÷ y =).

Teclas de funciones especiales

Estas teclas facilitan el envío de instrucciones de uso frecuente al ordenador. El pulsar una de estas teclas tiene el mismo resultado que el escribir una instrucción de varios caracteres, como pueden ser el PRINT o EXECUTE.

Así pues, puedes pulsar una sola tecla para borrar una letra, insertar un bloque de texto o ejecutar un programa. Estas teclas son específicas a cada fabricante y con ellas se pueden ahorrar esfuerzos mecanográficos. Las teclas de más utilidad son: SCROLL (mover el texto hacia arriba y hacia abajo), CLEAR (anular el valor de todos los variables), PAGE UP (SCROLL hasta la página siguiente), PAGE DOWN (SCROLL hasta la página anterior), y las teclas que posicionan el cursor: ↑, ↓, ←, → y HOME (limpiar pantalla).

El cursor

Te acordarás seguramente de que el cursor es un cuadrito o subrayado que indica tu posición actual en la pantalla. Normalmente es intermitente para que lo puedas ver mejor. He aquí un cursor que te enseña dónde estás en la pantalla después de haber escrito la palabra HELLO:

```
> HELLO□
```

Mediante los movimientos que realizas con el cursor hacia arriba y abajo y hacia cada lado en la pantalla, puedes escribir encima y modificar cualquier trozo de texto que hayas escrito anteriormente, lo cual es muy cómodo para hacer cambios. Terminarás usando el cursor «a tope» para corregir errores de mecanografía.

La mayoría de los teclados vienen provistos de al menos cuatro teclas funcionales para posicionar el cursor en la pantalla: ↑ , ↓ , ← , → .

Aprende a mover el cursor por la pantalla.

Quizá sea buena idea que practiques en tu teclado ahora algunas de las cosas que acabas de aprender. Cuando te hayas familiarizado lo suficiente con el teclado, podrás empezar a ver cómo se envían instrucciones en BASIC al ordenador.

Hablar en BASIC

Para hablar en BASIC con tu ordenador tiene que haber un intérprete de BASIC en la memoria. Si tu ordenador lo tiene incorporado (residente), no tiene que hacer nada. Sin embargo, algunos sistemas requieren que escribas:

B ↵

o

BASIC ↵

para activar al intérprete. Si tu ordenador no incorpora el intérprete de BASIC, tendrás que cargarlo por medio de un diskette o un cassette.

Al activarse el intérprete, recibirás una confirmación al respecto, como por ejemplo:

```
XBASIC 2.1 READY
>
```

XBASIC es el nombre que se le da al intérprete; 2.1 es la versión. Continuamente están apareciendo en el mercado versiones sucesivas, por lo que el número te permite distinguir entre ellas. El símbolo > , llamado *prompt,* puede variar, aunque a lo largo de todo el libro emplearemos siempre este símbolo, el cual constituye un mensaje del intérprete; quiere decir: «Estoy listo. Dame instrucciones». Al verlo, sabes que tu intérprete de BASIC está listo para entrar en acción.

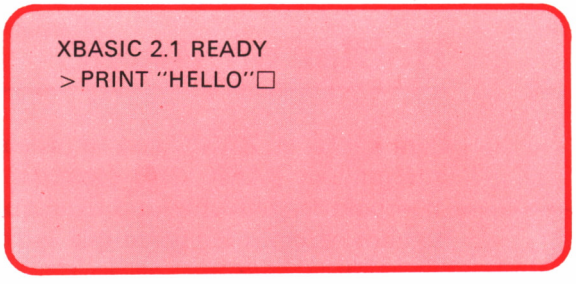
Prosigamos, asumiendo que lo siguiente sea verdad:

1. Un intérprete de BASIC se encuentra en la memoria del ordenador.
2. El símbolo > de BASIC aparece en tu pantalla. Si no lo hace, pulsa RETURN a ver qué sucede después. Si no da resultado, apaga el ordenador y enciéndolo de nuevo

Ahora enviaremos nuestra primera instrucción al ordenador. Escribe lo siguiente tal como se ve aquí:

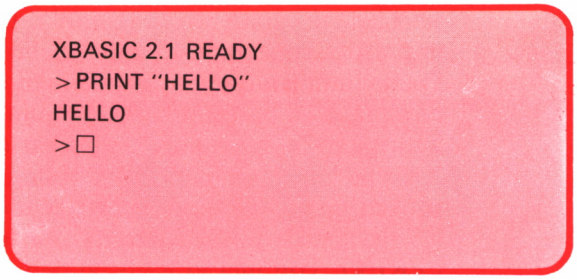
```
PRINT "HELLO"
```

La pantalla debería tener este aspecto:



El símbolo > al lado izquierdo te indica que el intérprete de BASIC espera alguna instrucción. Los caracteres a continucación son los que has escrito. No debe haber ocurrido nada todavía. ¿Recuerdas por qué?

Se debe a que has de pulsar la tecla RETURN para *introducir* tu instrucción al ordenador. Pulsa RETURN ahora. Tu pantalla ha de tener este aspecto:



El intérprete ha recibido tu instrucción y la ha ejecutado inmediatamente, poniendo la palabra HELLO en pantalla tal como se pidió. A continuación, el intérprete te envió un símbolo (>) nuevo para comunicarte que estaba a la espera de otra instrucción.

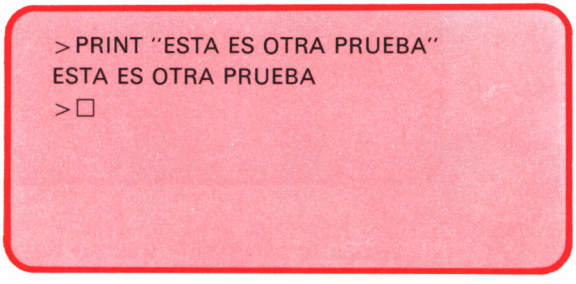
Veamos más detenidamente nuestra primera instrucción en BASIC:

```
PRINT "HELLO"
```

Esta instrucción se compone de dos partes: PRINT y «HELLO». PRINT es una *palabra reservada,* es decir, tiene un significado especial de cara al intérprete de BASIC. «HELLO» es el mensaje que se quiere imprimir en pantalla. Ha de colocarse entre comillas. Puedes colocar cualquier mensaje entre las comillas; probemos otro. Escribe:

```
PRINT "ESTA ES OTRA PRUEBA"
```

En tu pantalla debe aparecer lo siguiente:



Prueba otra vez con un mensaje a tu gusto, pero recuerda que si se te olvida una de las comillas o escribes la palabra PRINT mal, no funcionará la instrucción y recibirás a cambio un mensaje de error. Prueba a tu gusto; no hay peligro para la máquina.

Puede que te hayas preguntado por qué se utiliza la palabra inglesa equivalente a «imprimir» en castellano (o sea, PRINT), cuando en realidad no se trata de imprimir nada,

sino hacerlo aparecer en pantalla. Incluso si tienes una impresora conectada a tu ordenador, no imprimirá nada. Puede también que hayas adivinado la respuesta: los primeros terminales se parecían a las máquinas de escribir y, de hecho, imprimían la información en lugar de enviarla a la pantalla. Aunque haya cambiado la tecnología, el lenguaje BASIC no ha sufrido ningún cambio en este sentido. Hoy día se utiliza otra instrucción de BASIC para enviar información a la impresora: LPRINT, con una «L» inicial debido a que su principal función es la de *listar* los programas sobre papel.

Antes de seguir, cerciórate de que el *prompt* del BASIC aparezca en pantalla, es decir, que el intérprete esté preparado para aceptar otra instrucción. En caso contrario, no podrás ejecutar una instrucción de BASIC. Prueba pulsando RETURN o CTRL-C o, si no da resultado ninguno de los dos, apaga y enciende tu equipo.

Ahora vamos a escribir nuestro primer programa en BASIC, en vez de ejecutar una «sentencia» o instrucción única. Escribe:

```
10 PRINT "HOLA" ↵
20 PRINT "QUE TAL?" ↵
30 END ↵
```

Tu pantalla ha de tener este aspecto:

```
>10 PRINT "HOLA"
>20 PRINT "QUE TAL?"
>30 END
>□
```

Quizá te sorprenda que no haya ocurrido nada. El ordenador te responde tan sólo con el símbolo >. Estas tres líneas son más que tres instrucciones en BASIC seguidas: constituyen un pequeño *programa* en BASIC. Fíjate en el hecho de que cada línea lleva un número inicial, que se denomina número de línea o *etiqueta*. Indica al ordenador que queremos escribir un programa completo en lugar de ejecutar cada instrucción individualmente. Ahora escribe:

```
RUN ↵
```

(*Nota:* Esta instrucción puede variar según el intérprete.) Tu pantalla ahora ha de tener el aspecto siguiente:

```
HOLA
QUE TAL?
>□
```

¡Corre!

«A ver, ¿cómo va?»

o, alternativamente, y según el intérprete:

```
HOLA
QUE TAL?
END
>□
```

¿Cómo funciona esto? Primero creamos un programa de tres líneas y lo guardamos en memoria, línea por línea, al pulsar RETURN. Luego ejecutamos el programa mediante la instrucción RUN. Así se procede normalmente a la hora de escribir y ejecutar un programa, así que seguiremos este procedimiento de ahora en adelante. Cada línea se iniciará con una etiqueta numérica y el programa se ejecutará según esta numeración de forma automática.

Si en algún momento se te pasa iniciar una instrucción en BASIC con la identificación numérica necesaria, la instrucción se ejecutará al instante, sin que sea memorizada por la memoria del ordenador. Este tipo de operación se denomina *modo inmediato* o *modo de calculadora*. Si en algún momento escribes una línea de programación numerada a continuación del símbolo >, la línea se memorizará al pulsar la tecla RETURN, pero no se ejecutará hasta que escribas la instrucción RUN. Este modo de operación se llama *modo diferido* o normal. Veamos cómo funciona.

Nuestro programa de tres líneas ha sido almacenado en memoria. Puede ser ejecutado todas las veces que queramos, ampliado o modificado. Ahora escribe:

```
RUN↵
```

otra vez y verás en pantalla:

```
HOLA
QUE TAL?
>□
```

Hagamos que el ordenador nos muestre el contenido de su memoria escribiendo lo siguiente:

```
LIST ↵
```

a lo que la respuesta del ordenador en pantalla ha de ser:

```
10  PRINT "HOLA"
20  PRINT "QUE TAL?"
30  END
>□
```

Tu programa se lista en la pantalla tal y como se ha almacenado en la memoria del ordenador.

Hasta puedes guardar este tu primer programa en cassette o diskette, si dispones de alguno. La instrucción para lograr esto normalmente es:

```
SAVE ↵
```

Luego puedes recuperarlo cuando quieras al escribir:

```
LOAD ↵
```

Para demostrar la diferencia que existe entre el modo inmediato y el diferido, escribe:

```
PRINT "ADIOS" ↵
```

y verás en la pantalla:

```
>PRINT "ADIOS"
ADIOS
>□
```

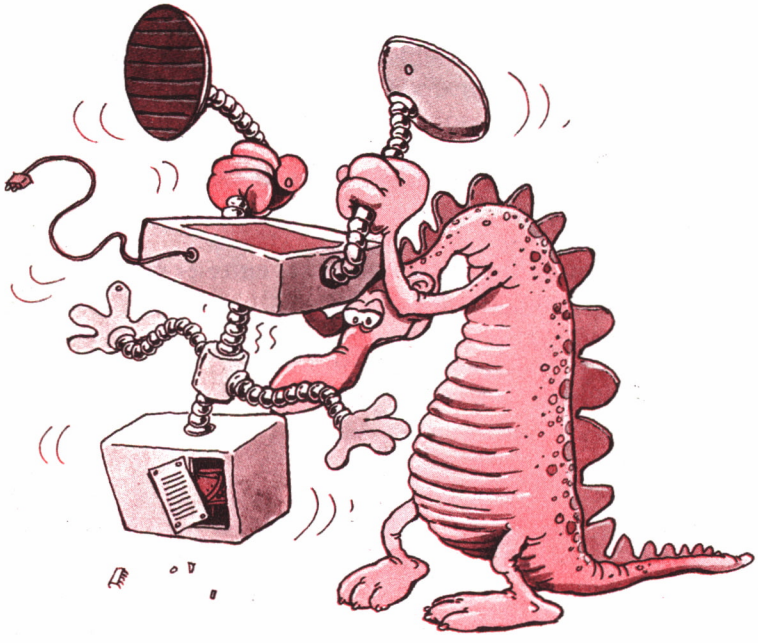
Ahora escribe:

```
LIST ↵
```

otra vez, y deberías ver en tu pantalla:

```
10  PRINT "HOLA"
20  PRINT "QUE TAL?"
30  END
>□
```

La instrucción nueva en modo inmediato no aparece, dado que no ha sido memorizada por el ordenador.



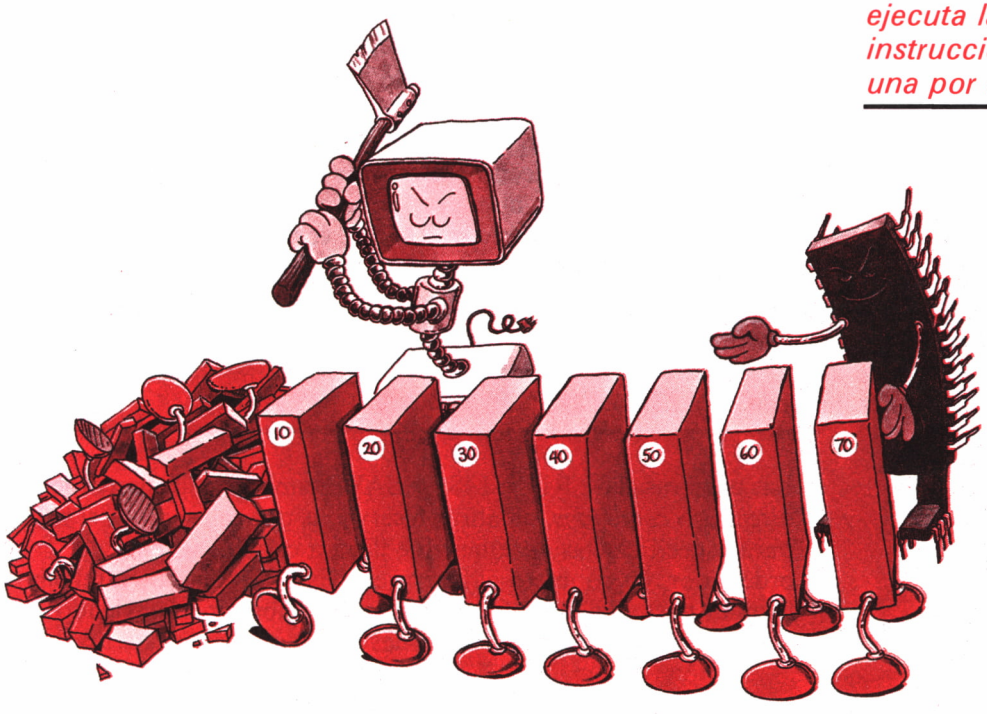
Si no empleaste el número de línea, ¡la instrucción se habrá esfumado!

Resumen del programa

En resumidas cuentas, hemos visto que una instrucción en modo inmediato se ejecuta nada más escribirla; no se guarda en la memoria del ordenador, y tendrás que escribirla de nuevo cada vez que quieres ejecutarla. Se lleva a cabo una *ejecución inmediata*. En la práctica, este modo de operación se usa relativamente poco, normalmente cuando se quiera verificar algunos valores después de que haya terminado un programa. Lánzate ahora a experimentar con instrucciones de BASIC a ver qué pasa.

Cuando una línea empieza con una numeración que la identifica, el modo de ejecución que le corresponde es el diferido, en el que cada línea del programa se guarda en la memoria del ordenador. Cuando la totalidad del programa se halla en memoria, con tan sólo dar la instrucción RUN se ejecutará el programa entero. Recuerda, sin embargo, que al apagar tu equipo se pierde el contenido de la memoria RAM, incluyendo cualquier programa que hayas escrito. Si has de guardar tu programa para su uso futuro, has de guardarlo mediante la instrucción SAVE en un cassette o diskette.

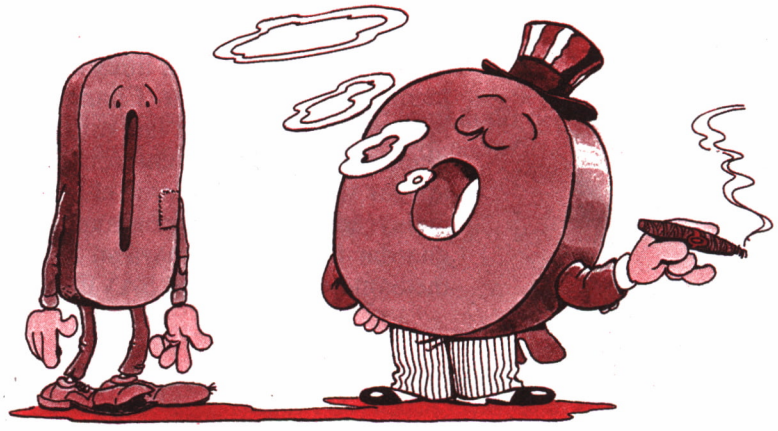
El ordenador ejecuta las instrucciones una por una.



Cada línea de un programa en BASIC ha de identificarse con un número, el cual especifica el orden exacto en el que se ejecutará el programa. Es decir, la línea 10 se ejecutará antes de la línea 20.

La instrucción END, en la línea 30 de nuestro ejemplo, es opcional en la mayoría de las versiones recientes de BASIC, pero forma parte necesaria de un programa con las versiones más antiguas. Esta instrucción indica al intérprete que ha llegado a un final «legítimo» durante la ejecución, en lugar de uno accidental.

Hay una gran diferencia entre el número 0 y la letra O.



Un último detalle que hemos de mencionar es la necesidad de que el número 0 se distinga de la letra O mayúscula. Una manera tradicional de señalar la diferencia ha sido la de colocar una barra diagonal sobre el cero: Ø. Hoy día, sin embargo, es posible tecnológicamente usar un símbolo más estrecho para el cero, de manera que el Ø se usa poco. En textos más antiguos sobre el BASIC, verás cómo la letra O o el cero tiene una barra diagonal para diferenciarlos entre sí.

Un programa más largo

Las palabras clave RUN, LIST y SAVE también se llaman órdenes directas o *comandos,* palabras reservadas que se usan por sí solas para especificar las acciones que lleva a cabo el sistema del ordenador, en concreto el intérprete. Estos comandos también forman parte del BASIC. A lo largo del libro iremos viendo progresivamente más instrucciones y comandos de este tipo.

Ahora vamos a desarrollar un programa más completo usando la instrucción PRINT y el comando NEW. Escribe lo siguiente:

```
NEW ↵
```

Luego:

```
LIST ↵
```

No ocurre nada. Ya no existe ningún programa en memoria; el comando NEW (que significa «nuevo») limpia o «renueva» la memoria del ordenador. Ahora escribe:

```
NEW
10  PRINT "ESTE"
20  PRINT "ES"
30  PRINT "OTRO"
40  PRINT "EJEMPLO"
50  END
```

Veamos lo que hace este programa. Escribe:

```
RUN ↵
```

Debes ver ahora en tu pantalla:

```
ESTE
ES
OTRO
EJEMPLO
>□
```

La instrucción END puede aparecer, o no, al final de la frase (debajo de la palabra EJEMPLO), según tu versión de BASIC. El programa imprime en la pantalla el texto que era de esperar. Verifiquemos que el programa haya sido debidamente almacenado en memoria:

```
LIST ↵
```

Visualizado en la pantalla tendrás lo siguiente:

```
>LIST
10  PRINT "ESTE"
20  PRINT "ES"
30  PRINT "OTRO"
40  PRINT "EJEMPLO"
50  END
>□
```

El comando NEW fue utilizado para vaciar la memoria del ordenador, haciendo sitio para el nuevo programa. Si no se usara el comando NEW, las líneas nuevas borrarían y reemplazarían a aquellas líneas guardadas anteriormente en memoria que tuviesen las mismas numeraciones de línea. Esto podría conducir a errores, al seguir existiendo líneas de programación «antiguas» en la memoria. Si no empleas el NEW, entonces cada vez que escribes una línea con la numeración 10, por ejemplo, la nueva versión ocupará el lugar que antes ocupara cualquier versión anterior. Pero cuidado, si habías escrito anteriormente una línea con la numeración 15, y luego hubieras escrito el programa que acabamos de hacer juntos (que no tiene ninguna línea con esa numeración) sin un comando NEW previo, la instrucción 15 se incorporaría automáticamente en el nuevo programa en el lugar correspondiente a su numero. Veamos una demostración de este fenómeno. Escribe:

```
15  PRINT "*****"
```

Luego:

```
RUN ↵
```

Y se visualizará en la pantalla:

```
ESTE
*****
ES
OTRO
EJEMPLO
>□
```

Como puedes comprobar, la nueva línea 15 se ha insertado automáticamente en el programa anterior entre la línea 10 y la 20. Listemos el programa; escribe:

```
LIST ↵
```

Y se verá en la pantalla:

```
10  PRINT "ESTE"
15  PRINT "*****"
20  PRINT "ES"
30  PRINT "OTRO"
40  PRINT "EJEMPLO"
50  END
```

Puedes comprobar que la línea 15 ha pasado a formar parte de tu programa. Recuerda que cada vez que escribes una línea de programación con un determinado número se coloca automáticamente en la memoria del ordenador y en la secuencia correcta.

Ahora vamos a demostrar que, por cada vez que utilizas un número de línea que *ya existe,* tu nueva línea ocupará automáticamente el lugar de la antigua. Empleemos este método para borrar la línea 15. Escribe:

```
15  PRINT "....." ↵
```

Luego:

```
RUN ↵
```

La pantalla debe presentar este aspecto:

```
ESTE
.....
ES
OTRO
EJEMPLO
>□
```

Como puede verse, tu nueva instrucción PRINT con la numeración 15 ha reemplazado a la que previamente residía en memoria. Verifícalo escribiendo:

```
LIST ↵
```

Verás en tu pantalla:

```
10  PRINT "ESTE"
15  PRINT "....."
20  PRINT "ES"
30  PRINT "OTRO"
40  PRINT "EJEMPLO"
50  END
>□
```

Para evitar incidentes desafortunados, al escribir cualquier programa nuevo, utiliza la instrucción NEW para borrar la memoria de tu ordenador y así evitar la interferencia de remanentes de otros programas.

Ahora vamos a borrar del todo la instrucción número 15. Esto se puede lograr de varias maneras; ahora simplemente escribiremos:

```
15 ↵
```

«Borro todo en un santiamén.»

Esta instrucción consiste solamente en la numeración de línea, por lo que se denomina *instrucción vacía*. Esta instrucción no hace nada, con la excepción de anular a la versión anterior. Ahora teclea la instrucción RUN. Se visualizará lo siguiente en tu pantalla:

```
ESTE
ES
OTRO
EJEMPLO
>□
```

Ten cuidado. Este procedimiento tiene sus inconvenientes. Si escribes:

```
20
```

sin querer, y luego le das al RETURN, borrarás la versión anterior de la línea 20, reemplazándola con una instrucción vacía que no contiene nada. Para disminuir la posibilidad de sorpresas ingratas, una buena idea es comprobar tus programas a través de un listado antes de ejecutarlos.

Resumen

Hemos aprendido a escribir programas rudimentarios en BASIC que visualizan información en la pantalla. Hemos escrito un programa en BASIC usando instrucciones numeradas. Hemos visto por qué se debe anticipar cada programa nuevo con el comando NEW, y terminarlo con la instrucción END. También hemos visto que un programa se almacena en la memoria del ordenador de forma automática según se va escribiendo en el teclado, y que se puede ejecutar al escribir el comando RUN. Hemos comprobado que se puede listar un programa en la pantalla mediante el comando LIST. Por último, hemos usado el comando SAVE para guardar un programa en una cinta o un diskette.

Hemos aprendido que la ejecución de un programa sigue el orden de la numeración de sus líneas. Si se duplica un número

de una línea con o sin intención, la instrucción nueva que resulta borrará automáticamente cualquier línea anterior con ese número. Es más, si en algún momento añades una línea nueva con un número nuevo, aunque esté fuera de secuencia, el intérprete la insertará automáticamente en su lugar debido dentro del programa.

En este capítulo hemos introducido muchos conceptos nuevos. Si de verdad quieres aprender a programar, es esencial que vayas practicando lo que has aprendido. A continuación encontrarás varios ejercicios para probar tus conocimientos. Sugerimos que los hagas detenidamente. Las respuestas a algunos ejercicios seleccionados se pueden encontrar al final del libro.

Ejercicios

2.1. Escribe un programa que imprima lo siguiente: «QUE PASE BUEN DIA».

2.2. Escribe un programa que imprima:

```
AAAAA
BBBB
CCC
DD
E
```

2.3. Escribe un programa que imprima:

```
********
*TITULO*
********
```

2.4. Define los términos siguientes:

a) Etiqueta.
b) Ejecución diferida.
c) Ejecución inmediata.
d) Instrucción vacía.
e) Cursor.
f) Tecla de control.

g) Cuadro independiente de teclas.
h) Palabra reservada.
i) Símbolo >.

2.5. ¿Cuál es la diferencia entre PRINT y LPRINT?

2.6. ¿Es posible ejecutar un programa entero escribiendo sus líneas una por una en modo inmediato?

2.7. ¿Por qué es conveniente emplear el comando NEW antes de escribir un programa nuevo?

2.8. ¿Es posible escribir líneas de programación numeradas sin seguir su secuencia numérica?

2.9. Da algunos ejemplos de comandos en BASIC.

2.10. ¿Es válida la siguiente instrucción para imprimir en pantalla la palabra EJEMPLO?:

```
PRINT EJEMPLO
```

2.11. ¿Para qué sirve la tecla RETURN?

2.12. Explica cómo se borra la línea 20 en un programa.

2.13. Si ya has escrito una línea con la numeración 30, y deseas reemplazarla con otra, ¿es necesario borrar la antigua antes de escribir la nueva?

2.14. Escribe un programa que imprima el dibujo a continuación en tu pantalla:

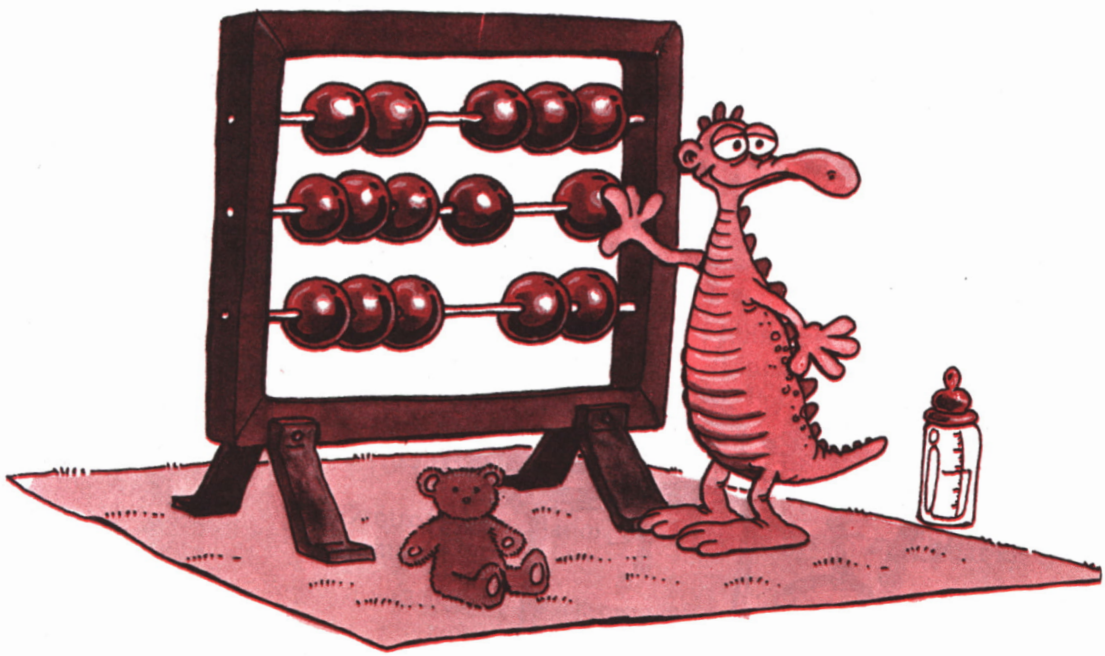
```
H    H  OOOOO  L      AAAAA
H    H  O   O  L      A   A
H    H  O   O  L      A   A
HHHHH  O   O  L      AAAAA
H    H  O   O  L      A   A
H    H  O   O  L      A   A
H    H  OOOOO  LLLLL  A   A
```

Haciendo

3

En este capítulo empezaremos a usar números; los visualizaremos en pantalla y haremos con ellos sumas, restas, multiplicaciones y divisiones. Aprenderemos a hacer cálculos usando los operadores aritméticos sencillos, y describiremos otros operadores importantes incorporados en el lenguaje BASIC.

cálculos en BASIC



Imprimir números

Hasta ahora sólo hemos impreso (visualizado en pantalla) textos. Pasemos ahora a la impresión de números; escribe:

```
PRINT 3
```

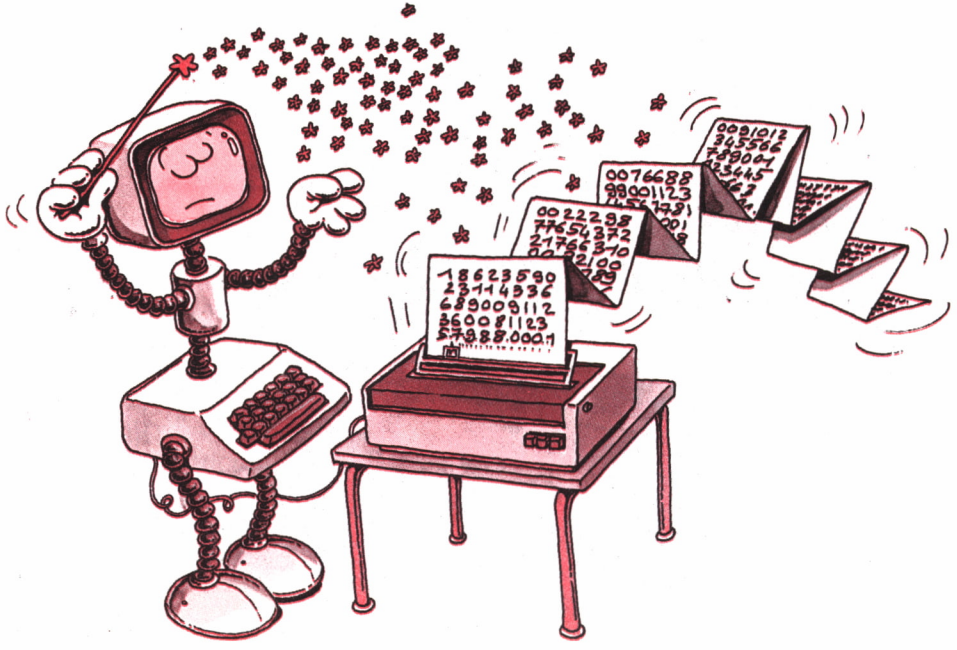
El resultado ha de ser:

```
3
```

Recordarás del capítulo anterior que esta instrucción está en el modo inmediato, en el cual una instrucción no precisa una numeración de línea y se ejecuta al instante. En este capítulo, escribiremos todos los ejemplos en este modo de operación, de manera que podrás ejecutarlos con tan sólo pulsar unas cuantas teclas.

Como eres buen observador, te habrás fijado ya que, en BASIC, los números no se tienen que colocar entre comillas, sino solamente los textos. El uso de las comillas permite que el intérprete distinga bien entre lo que son textos hechos por el usuario y lo que son palabras reservadas del lenguaje BASIC como, por ejemplo, la palabra PRINT. El texto que se encuentra entre comillas se llama *cadena* (en inglés, *string*), y puede estar compuesto de símbolos alfanuméricos (tanto letras como números).

«Te enseñaré la magia de imprimir los números.»



Ahora vamos a intentar imprimir algunos números grandes; digamos 100, 1000, 10000, etc. Alcanzado algún límite, tu intérprete se negará a imprimir y se visualizará el mensaje «NUMBER TOO LARGE» (número excesivamente alto). Cada intérprete de BASIC limita a un valor determinado el número entero más elevado que es capaz de manejar.

Si en algún momento necesitas trabajar con números enteros mayores al valor máximo admitido por tu intérprete, no tendrás más remedio generalmente que el de emplear otro intérprete que admite números más grandes.

Acuérdate que un intérprete de BASIC que admite el uso de números enteros se denomina un BASIC de *integer* («entero»). Este tipo de BASIC, sin embargo, no admite el uso de decimales. Veamos si tu intérprete admite números decimales. Escribe:

```
PRINT 1.5↵
```

Si ahora ves en pantalla el número 1.5, tu versión de BASIC sí puede manejar números decimales. (*Nota:* En inglés los números decimales y los millares se escriben «al revés» en cuanto a las comas y los puntos. Por ejemplo, lo que para nosotros se escribe 1.000,50, en los países anglosajones, y en algunos más, se escribe 1,000.50.) Si tu versión no puede trabajar con decimales, recibirás un mensaje indicando que ha habido un error, o quizá aparecerá el símbolo ?.

En el argot de la informática, los números decimales se denominan *números de coma flotante*. Un intérprete de BASIC que admite este tipo de números se llama *BASIC de coma flotante*.

La notación científica

Examinemos más a fondo los números decimales. Como en el caso de los números enteros, el intérprete sólo retendrá un número determinado de dígitos. Por ejemplo, el valor decimal de un tercio sería:

```
0.33333333333... (etc.)
```

Dentro del ordenador, este valor se puede almacenar en el formato:

```
0.3333333   (se retienen siete dígitos significativos después del cero)
```

Podemos decir que el valor correcto ha sido *truncado* (cortado) hasta siete dígitos. (*Nota:* Esto es sólo una aproximación, pero en general se mantiene en la mayoría de los casos.)

Si tu intérprete de BASIC admite los números decimales, entonces empleará también una *representación científica* para estos números. Cuando una cantidad es muy elevada, o bien muy reducida, se visualizará en la notación científica para ahorrar espacio.

Veamos un ejemplo:

```
3.2E6
```

significa

$$3.2. \times 10^6 = 32000000$$

10^6 representa 10 elevado a 6; es decir, 10 multiplicado por sí mismo 6 veces:

$$10 \times 10 \times 10 \times 10 \times 10 \times 10 = 1000000$$

Y de la misma manera:

```
1.12E-7
```

representa

$$1.12 \times 10^{-7} = 0.000000112$$

donde 10^{-7} representa 1/10 elevado a 7; es decir, 1/10 multiplicado por sí mismo 7 veces ($1/10 = 10^{-1}$).

Haciendo cálculos

Hagamos ahora unos cálculos aritméticos sencillos. Escribe:

```
PRINT 2 + 2 ↵
```

El resultado que ha de aparecer en tu pantalla es:

```
4
```

Acabamos de llevar a cabo nuestro primer cálculo aritmético. El símbolo más (+) se llama *operador*. Un operador es un símbolo que representa una operación que hay que llevar a cabo sobre uno o más operandos. El BASIC incorpora al menos cinco operadores aritméticos:

–	(menos)
+	(más)
*	(por)
/	(dividido entre)
^	o * * (elevado a) (a veces, en lugar de ^ se emplea ↑ o [])

Prueba otro ejemplo. Escribe:

```
PRINT 2 * 3 ↵
```

El resultado debe ser 6. El símbolo * es el que se usa para la multiplicación. El signo corriente de multiplicación, × , se podría confundir con la letra X, por lo que los lenguajes de programación utilizan el * en su lugar.

A continuación tenemos otros ejemplos de instrucciones aritméticas válidas:

```
PRINT 1 + 2 * 3 ↵
```

«Tendrás que practicar más.»

La respuesta es

```
7
```

```
PRINT 3 - 2 ↵
```

La respuesta es

```
1
```

```
PRINT 8/2 ↵
```

La respuesta es

```
4
```

```
PRINT 1 + 2 + 3 + 4 ↵
```

La respuesta es

```
10
```

Si tu versión de BASIC admite los números decimales, será admitida la instrucción siguiente:

```
PRINT (6/3 + 12/4) / 2 ↵
```

cuyo resultado es

```
2.5
```

Fíjate en el uso de los paréntesis para clarificar la agrupación de las operaciones. Probemos otro ejemplo. Escribe:

```
PRINT 2 + 3 + 4/2 ↵
```

que da el resultado de

```
7
```

La división (/) se llevó a cabo primero sobre el número 4. Esto se debe a que en BASIC, en el caso de que haya que elegir (es decir, si no se usan los paréntesis), la división (/) o la multiplicación (*) se efectuarán antes de las operaciones de sumar (+) o restar (−). Si tu intención hubiera sido la de dividir el grupo de números 2 + 3 + 4 entre 2, entonces habría sido necesario escribir:

```
PRINT (2 + 3 + 4) / 2
```

cambiando el resultado a

```
4.5
```

En este caso, pues, la división se habría efectuado sobre el conjunto (2 + 3 + 4). Es aconsejable utilizar los paréntesis a menudo, para evitar posibles confusiones. Por ejemplo, la expresión siguiente (o conjunto de valores y operadores):

$$\frac{1 + 2 + 3}{4 + 5} \times 3$$

se podría traducir a la expresión en BASIC de

```
((1 + 2 + 3) / (4 + 5)) * 3
```

o bien

```
(1 + 2 + 3) / (4 + 5) * 3
```

debido a que la ejecución se efectúa de izquierda a derecha cuando los operadores tienen la misma preferencia (importancia relativa el uno al otro); es decir, la división ocurre antes de la multiplicación.

Ahora bien, la segunda versión en BASIC:

```
(1 + 2 + 3) / (4 + 5) * 3
```

es equivalente a la expresión:

$$\frac{1 + 2 + 3}{(4 + 5) * 3}$$

Utiliza los paréntesis para identificar los conjuntos. Asegúrate de que haya siempre un paréntesis que cierra por cada uno que se haya abierto.

Usemos ahora nuestra nueva habilidad de cálculo para conseguir valores útiles.

Formatos de impresión

Si escribes:

```
PRINT "EL RESULTADO DE DOS MULTIPLICADO POR TRES ES", 2 * 3↵
```

Conseguirás el resultado siguiente:

```
EL RESULTADO DE DOS MULTIPLICADO POR TRES ES      6
```

En la instrucción PRINT anterior, hemos combinado texto con números, separados por una coma. Para ser más precisos, hemos utilizado una expresión, 2 * 3, en vez de un número. Ahora escribe:

```
PRINT "EL RESULTADO DE DOS MULTIPLICADO POR TRES ES, 2 * 3"↵
```

y conseguirás como respuesta:

```
EL RESULTADO DE DOS MULTIPLICADO POR TRES ES, 2 * 3
```

Esta es una sentencia válida en BASIC, pero no el resultado que querías. Recuerda que todo lo que se encuentre entre comillas se visualizará en pantalla de forma literal. Una coma o un punto y coma ha de colocarse fuera de las comillas para tener el efecto deseado.

Una instrucción PRINT se puede usar para imprimir más de un elemento en la misma línea. Cada elemento, sin embargo, ha de separarse de los demás mediante una coma o un punto y coma. El punto y coma dará como resultado una separación con un pequeño espacio de los elementos a imprimir, mientras la coma proporcionará un espacio más amplio. Como el tabulado de una máquina de escribir, la coma se emplea para crear tabulados o campos separados en la pantalla. Esta técnica de separación sirve para hacer visualizar en pantalla tablas de información.

Probemos esta característica del lenguaje BASIC. Escribe:

```
PRINT 1;2;3↵
```

Se ha de ver en pantalla:

```
 1 2 3
```

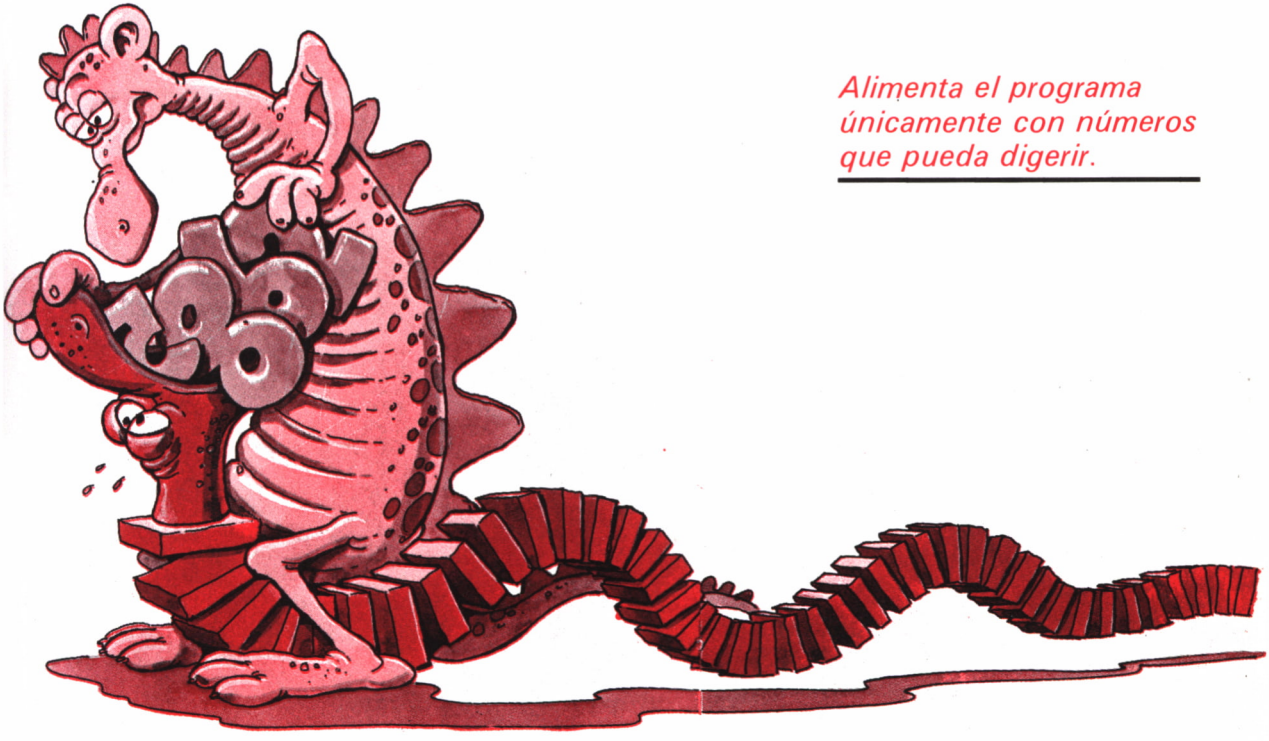
Ahora escribe:

```
PRINT 1,2,3 ↵
```

Tu pantalla ha de tener este aspecto:

```
 1    2.   3
```

Ahora vamos a calcular los impuestos sobre una venta por un importe de 1234 pesetas con una tasa del 6.5%.



Alimenta el programa únicamente con números que pueda digerir.

Supondremos que tu versión de BASIC admite números decimales. La sentencia, o instrucción completa, sería:

```
PRINT "LOS IMPUESTOS SUMAN"; 1234 * 6.5 / 100 ↵
```

Con el resultado de:

```
LOS IMPUESTOS SUMAN 80.21
```

También podríamos escribir:

```
PRINT "LOS IMPUESTOS SUMAN"; 1234 * 0.065 ↵
```

para obtener el mismo resultado, Hay muchas maneras equivalentes de escribir un programa.

Puedes imprimir varias cosas en una misma línea. Por ejemplo:

```
PRINT 1;2;3;4;5;6;7;8;9; "MUCHAS CIFRAS" ↵
```

Verás en pantalla:

```
123456789 MUCHAS CIFRAS
```

Ahora que hemos aprendido a efectuar algunos cálculos aritméticos sencillos y a hacer aparecer los resultados en pantalla, usemos estos nuevos conocimientos para resolver algunos problemas sencillos.

Ejemplos de aplicaciones

Calculemos el consumo de gasolina de un automóvil en millas por galón, para lo que la fórmula matemática sería:

CONSUMO = DISTANCIA (en millas) ÷ GASOLINA (en galones)

Digamos que la distancia recorrida fue 510 millas y la cantidad de gasolina utilizada fue 20.2 galones. Aquí tenemos la sentencia en BASIC:

```
PRINT "EL CONSUMO ES IGUAL A"; 510/20.2 ; "MPG" ↵
```

Para los que usan el sistema métrico, convirtamos esto a litros por kilómetro. Un galón es equivalente a 3.8 litros. Una milla equivale a 1.6 kilómetros. Así pues, el consumo en litros por kilómetro sería:

```
PRINT "CONSUMO ES IGUAL A"; (20.2 * 3.8) / (510 * 1.6) ; "L/KM" ↵
```

He aquí otro problema sencillo: dada una temperatura en Fahrenheit, su equivalente en grados Celsius se calcula mediante la fórmula:

GRADOS CELSIUS = (GRADOS FAHRENHEIT − 32) × 5/9

Para calcular el equivalente en grados Celsius de 79 °F, escribe:

```
PRINT "79 GRADOS F. = "; (79 - 32) * 5/ 9; "GRADOS C" ↵
```

Con el resultado de:

```
79 GRADOS F = 26.1111111 GRADOS C
```

Para entrar en detalles, fíjate que la fracción 5/9 no ha sido colocada entre paréntesis, ya que no importa si se ejecuta la * o la / primero.

Resumen

En este capítulo hemos aprendido a efectuar cálculos aritméticos y a hacer aparecer sus resultados y textos relacionados en la misma línea de pantalla. Hemos aplicado este nuevo conocimiento a la automatización del cálculo de algunas fórmulas sencillas escribiendo unas sentencias de una línea en BASIC.

Hasta ahora hemos especificado los valores numéricos dentro de la sentencia misma de BASIC. Ahora lo que queremos hacer es primero escribir un programa para luego especificar los valores desde el teclado repetidas veces, de manera que se pueda emplear una variedad de valores con el mismo programa, sin la necesidad de escribirlo de nuevo. Esto lo conseguiremos mediante el uso de las variables, que será el tema del siguiente capítulo.

Ejercicios

3.1. Escribe una línea en BASIC que calcule:

$$\frac{5 + 6}{1 + 2 \div 3}$$

3.2. Escribe una sentencia en BASIC que calcule:

$$1 + {}^1/_2 \; \frac{1}{1 + {}^1/_2}$$

3.3. Escribe una línea en BASIC que calcule el valor en grados Fahrenheit de 20 °Celsius.

3.4. Dada una velocidad de 100 km/h, calcula la velocidad equivalente en millas/hora (1 milla = 1,6 kilómetros).

3.5. Haz el cálculo del número de segundos en un día, una semana, un mes y un año.

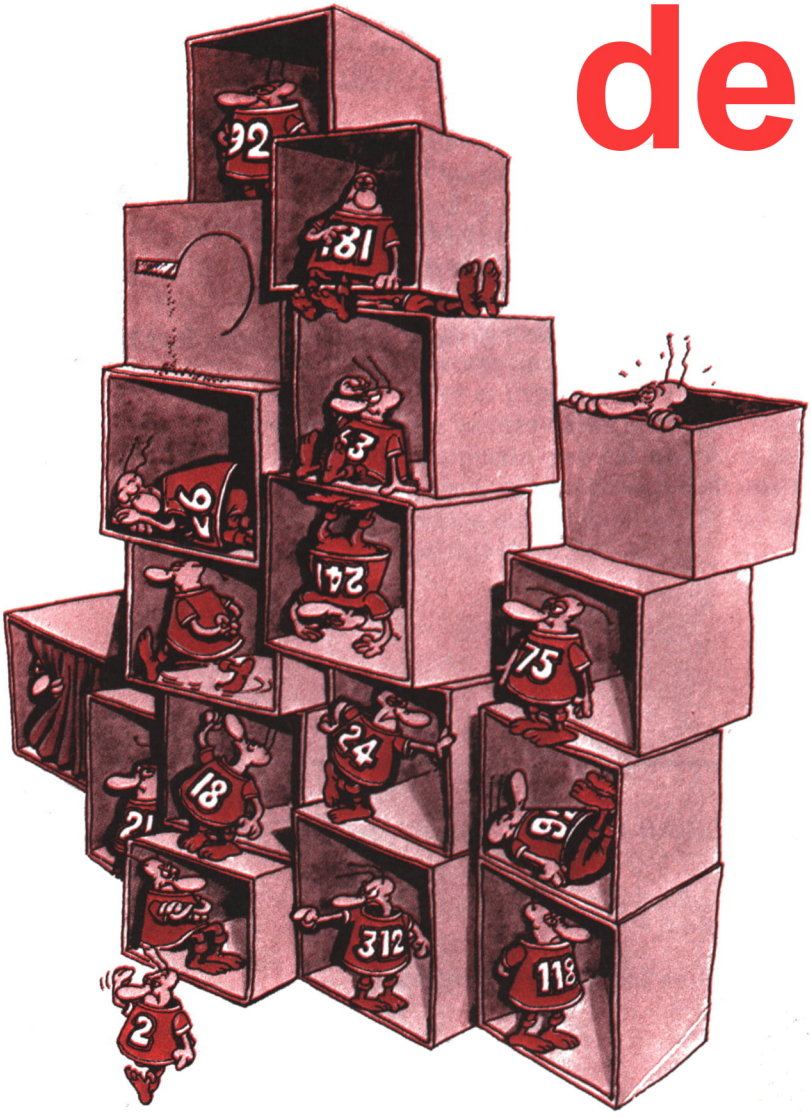
3.6. Suponiendo una velocidad media de 55 millas/hora, calcula el tiempo necesario para recorrer 350 millas.

3.7. Suponiendo un año de 365 días, calcula el número de días que has vivido hasta ahora.

3.8. Calcula el salario anual para una persona sobre la base de sus ganancias:

a) A la semana (52 semanas al año)
b) Cada dos semanas (multiplica por 26)
c) Mensual
d) Por hora (multiplica por 2080)

La memo de valores de las



rización y el uso variables

4

En este capítulo aprenderemos a escribir programas que se pueden usar repetidamente, sin cambio alguno, y que servirán para conseguir resultados distintos, según los datos que se suministran en el teclado. Hasta ahora, para conseguir el resultado de una operación aritmética, como por ejemplo 2 + 3, teníamos que incluir en la misma sentencia del programa los números sobre los que se iba a trabajar. Ahora aprenderemos a escribir programas que se pueden ejecutar con datos distintos cada vez. Los programas especificarán las operaciones a efectuar, mientras que los datos provendrán del teclado donde los suministrará el usuario a la hora de ejecutar el programa. De esta manera se hace que el programa sea utilizable más de una vez.

Adicionalmente introduciremos el concepto de una variable y aprenderemos a usar dos instrucciones nuevas: INPUT y LET.

Empecemos primero con la manera de suministrar información a un programa mientras se está ejecutando.

La instrucción «INPUT»

Escribe el programa siguiente. *(Nota:* Ya no colocaremos la flechita ↵ para indicar el RETURN al final de cada línea):

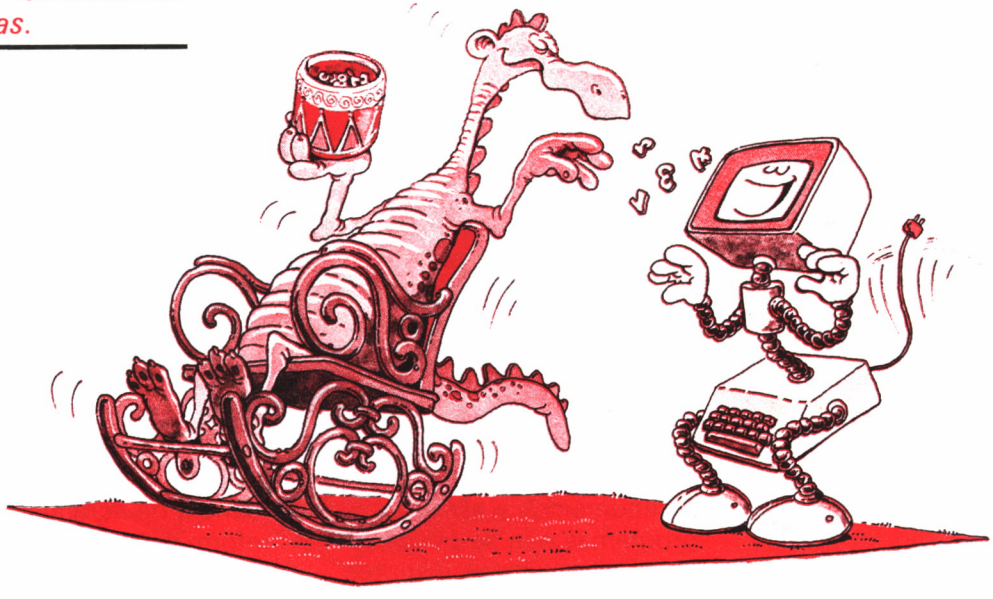
```
10  INPUT A
20  PRINT A; 2 * A; 3 * A
30  END
```

Ahora ejecuta este programa usando el comando RUN, como de costumbre. Deberías ver en tu pantalla algo parecido a esto:

```
?
```

Generalmente, aparece una interrogación, «?», en la pantalla junto al cursor intermitente para recordarte que has de mecanografiar la información requerida.

Con el INPUT verás qué fácilmente te acepta los números y las letras.



Ahora escribe un número, digamos el 3. Termina la entrada de datos pulsando la tecla RETURN como de costumbre. La pantalla ha de presentar el siguiente aspecto:

```
3 6 9
>
```

Tu programa ha sido ejecutado. Veamos qué es lo que sucedió. La primera línea fue:

```
10  INPUT A
```

Esta instrucción te pide que teclees alguna respuesta en el teclado. El programa puso en pantalla una ? y se paró, esperando que introdujeras datos. El valor de 3 que a continuación suministraste se leyó y fue almacenado en A. La «A» se llama *variable*. Es un nombre utilizado para guardar un valor. Dicho estrictamente, una variable es el nombre que se da a un sitio en memoria. Ejemplos de variables son: A, B, C, F, Z1, G2. La mayoría de las versiones de BASIC admiten nombres de variables de varias letras; por ejemplo: NUMERO1, SUM, TAX, RESULT.

La siguiente línea fue:

```
20  PRINT A; 2 * A; 3 * A
```

Esta instrucción tuvo como resultado la impresión de 3, 2 * 3, 3 * 3, 6:

```
3 6 9
```

Mediante la instrucción INPUT es también posible la entrada de varios valores simultáneamente, como en el siguiente ejemplo. Escribe:

```
10  INPUT A,B
20  PRINT A; A * 2; B; B * 2
30  END
```

Ahora ejecuta el programa con RUN. Deberías ver la típica «?» en la pantalla. Escribe dos números, digamos 2 y 3, separados por una coma y pulsa RETURN. Ahora has de ver en pantalla:

```
2 4 3 6
```

Examinemos todo el proceso. La primera línea del programa es:

```
10  INPUT A,B
```

Esta instrucción hizo que el ordenador te pidiera dos valores, los cuales a continuación se guardaron en memoria. Teniendo en cuenta que las variables son nombres de posiciones en memoria, podemos ver que originalmente estas posiciones estaban vacías, tomando posteriormente los valores 2 y 3, respectivamente. La segunda línea del programa fue:

```
20  PRINT A; A * 2; B; B * 2
```

Esta instrucción tuvo como resultado la impresión de 2, 2*2, 3, 3*2, 6:

```
2 4 3 6
```

Ejecutemos el programa de nuevo. Esta vez introduce los valores:

```
5,8
```

lo que debe conducir al siguiente resultado:

```
5 10 8 16
```

Podemos usar el programa repetidas veces, obteniendo resultados nuevos al escribir nuevos valores en el teclado. Hemos hecho que el programa sea reutilizable mediante el uso de nombres de variables (A y B), en lugar de valores predeterminados.

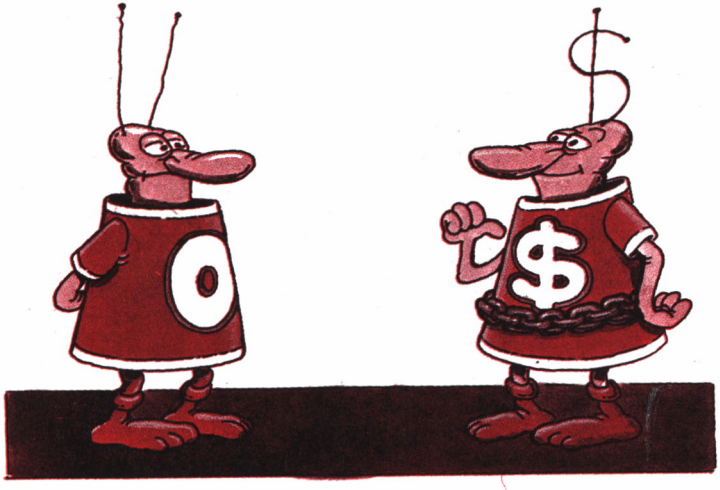
Con el fin de hacer que este programa sea de verdad reutilizable, mejoremos su estilo y facilidad de uso. Supongamos que queremos guardar el programa para poderlo ejecutar dentro de algunos días. Puede que se nos olvide qué hace el programa o el número de valores que hemos de suministrar. Podemos modificar el programa de manera que toda esta información se vea en la pantalla. Aquí tenemos una versión bastante mejorada:

```
10  PRINT "ESTE PROGRAMA MULTIPLICA DOS NUMEROS
      CUALESQUIERA POR 2"
20  PRINT "ESCRIBE DOS NUMEROS"
30  INPUT A,B
40  PRINT "PRIMER NUMERO : "; A, "SU DOBLE :"; 2 * A
50  PRINT "SEGUNDO NUMERO : "; B, "SU DOBLE :"; 2 * B
60  END
```

y el resultado que se verá en pantalla (*Nota:* A lo largo del texto destacaremos aquellos datos que suministra el usuario en negrita) será el siguiente:

```
ESTE PROGRAMA MULTIPLICA DOS NUMEROS CUALESQUIERA POR 2
ESCRIBE DOS NUMEROS
? 5,7
PRIMER NUMERO : 5          SU DOBLE : 10
SEGUNDO NUMERO : 7         SU DOBLE : 14
```

Hemos visto la manera en la que se suministran datos numéricos al programa mediante la instrucción INPUT. También hemos introducido el concepto de la variable. Veamos ahora cómo se puede emplear estos nuevos elementos con eficacia y cómo se desarrollan programas de mayor complejidad.



«Qué hay, Numérica; yo soy una cadena. Almaceno una cadena de caracteres. Me podrás reconocer por mis antenas.»

Las dos clases de variables

Existen *dos clases* de variables en el lenguaje BASIC: las *numéricas* y las de *cadena*. Las variables numéricas representan a valores numéricos; las variables de cadena representan textos. Las dos clases de variables tienen aspectos diferentes: variable de cadena se señala con el símbolo «$» después de su nombre.

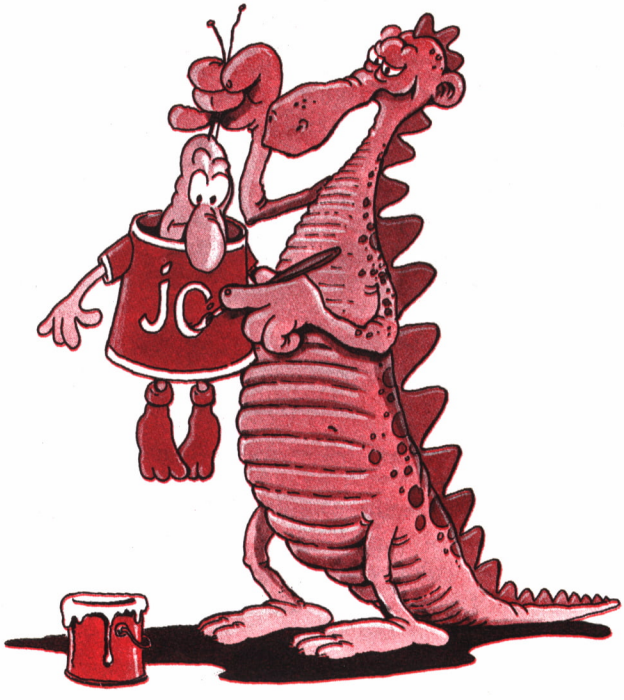
Asimismo, las dos clases se emplean para distintos fines; por ejemplo, puedes someter los números a operaciones matemáticas, mientras, como es de suponer, esto no es posible en el caso de los textos. Veamos primero las variables numéricas y luego las cadenas.

Las variables numéricas

Aprendamos las reglas que gobiernan la forma de dar nombre a una variable numérica, primer paso para saberlas usar eficazmente. Ya hemos usado dos variables numéricas, A y B, al principio de este capítulo, a las que asignamos valores numéricos desde el teclado mediante la instrucción INPUT. Veamos, pues, la manera de dar un nombre a una variable.

A la hora de dar nombres a las variables, todas las versiones de BASIC, incluida la versión original de BASIC de Dartmouth, permiten el uso de una letra seguida opcionalmente de un número de un solo dígito. A continuación tenemos algunos ejemplos de nombres de variables que son válidos:

A cada variable, un nombre.



A (una letra)

B (una letra)

Z (una letra)

A1 (una letra y un número)

A2 (una letra y un número)

B2 (una letra y un número)

Siguiendo nuestra definición, los nombres siguientes no son aceptables:

12 (no empieza con una letra)

A2B (demasiados caracteres)

BA (se permite solamente una letra)

1B (ha de empezar con una letra)

AB1 (demasiado largo; sólo se permite una letra)

La ventaja de los nombres cortos está en la reducción del tamaño y complejidad del intérprete de BASIC. En cambio, una desventaja que tienen estos nombres es que son difíciles de recordar; por ejemplo, el nombre completo RESULT es más descriptivo y, por tanto, más fácil de recordar que el «apodo» R. Para mayor facilidad de lectura, los nombres largos son permitidos en la mayoría de las nuevas versiones de BASIC. Normalmente, se puede usar cualquier número de letras consecutivas seguidas de unos dígitos opcionales, hasta cierta longitud máxima. Por ejemplo, los siguientes nombres, a pesar de ser más largos, serían ya aceptables:

PUNTOS	ALUMNO1
JUEGO	ALUMNO2
TANTEO	ALUMNO14
SUMA	CASO24

Mientras éstos no lo serían:

R2D2 (letras no consecutivas)

3VECES (comienza con un número)

A-UNO (símbolo no aceptable, «−»)

Es obvio que un programa escrito con nombres de variables largos y explícitos es más fácil de leer. En este

capítulo usaremos tanto nombres largos como cortos para que te habitúes a los dos formatos. Ten siempre en cuenta que el uso de nombres más largos es cuestión meramente de comodidad, no teniendo ningún efecto sobre el comportamiento del programa en sí.

Existe una limitación más en el uso de nombres de variables: no puedes emplear un nombre que a la vez es una *palabra reservada*; es decir, una palabra que guarda significado para tu intérprete de BASIC. Por ejemplo, no podrás usar las palabras LIST, END o RUN como nombres de variables. (*Nota:* En el apéndice B, al final del libro, figura una lista de las palabras reservadas más comunes, al igual que generalmente te la encontrarás en el manual de referencia para tu intérprete de BASIC.) Algunas versiones prohíben además el uso de una palabra reservada como *parte* de un nombre; por ejemplo, la palabra OR es una palabra reservada estándar, por lo que con algunos BASIC no puedes usar el nombre PORTAR para una variable.

Ahora que hemos estudiado la manera de asignar nombres «legales» («aceptables» o «permisibles» en el argot informático) a las variables numéricas, veamos la manera de hacer lo mismo en el caso de las variables de cadena.

«Soy una variable de cadena. Guardo los textos, y mi nombre termina en $».

Las variables de cadena

¿Exactamente qué es una «cadena»? Veamos algunos ejemplos:

"RESULTADO"

"ESTE ES UN EJEMPLO"

"ME LLAMO SEGISMUNDO"

"25 POR 4 = "

Fíjate en que la cadena se coloca entre comillas, ya que de no hacerlo así se podría confundir con el nombre de una variable. Una cadena puede incorporar una secuencia cualquiera de caracteres (letras, números y símbolos), con la excepción de las comillas mismas (cada versión de BASIC, sin embargo, dispone de algún truquito para superar esta limitación sobre las comillas). La longitud de una cadena siempre tiene algún límite, uno típico podría ser de unos 128 caracteres.

Hablemos ya de las variables de cadena. Cuando el valor almacenado en una variable es un texto (es decir, una «cadena» de caracteres), en lugar de un número, a esa variable

se la denomina *variable de cadena.* El nombre de una variable de cadena es exactamente como el de una variable numérica, excepto que ha de terminar en el símbolo «$». Así, algunos nombres cortos permisibles serán:

A$

R$

A1$

B5$

Si tu BASIC permite los nombres largos, los que aparecen a continuación también serían válidos:

NOMBRE$

PRIMER$

PUEBLO$

UNIDAD25$

Recuerda que un nombre del estilo de DANDO$ podría ser ilegal en el caso de algunas versiones de BASIC, al contener la palabra reservada AND. No te preocupes demasiado por esto, dado que lo normal es que te avise el intérprete nada más escribirlo en el teclado.

Hagamos una demostración del uso de estas variables con un pequeño programa que saluda al usuario por su nombre:

```
10 PRINT "SOY HAL, EL ORDENADOR."
20 PRINT "CUAL ES TU NOMBRE?";
30 INPUT PRIMER$
40 PRINT "Y CUAL ES TU APELLIDO?";
50 INPUT APDO$
60 PRINT "HOLA, "; PRIMER$; " "; APDO$; "!"
70 PRINT "AHORA SE COMO TE LLAMAS!"
80 PRINT "ME GUSTA "; PRIMER$; "COMO NOMBRE."
90 END
```

Si tu versión de BASIC acepta solamente los nombres cortos, utiliza P$ en lugar de PRIMER$, y A$ en lugar de APDO$.

Abajo mostramos el resultado de una ejecución de prueba. Ten en cuenta que todos los caracteres que tú has de escribir aparecen aquí en negrita.

```
>RUN
SOY HAL, EL ORDENADOR
CUAL ES TU NOMBRE? PEPE
Y COMO ES TU APELLIDO? SANCHEZ
HOLA, PEPE SANCHEZ
AHORA SE COMO TE LLAMAS!
ME GUSTA PEPE COMO NOMBRE
>
```

Ahora puedes comunicar con tu ordenador. Repasemos algunas de las características principales del programa, empezando con la línea 20.

```
20  PRINT "CUAL ES TU NOMBRE?";
```

Toma nota del hecho de que la línea termina con punto y coma, lo que indica al ordenador que ha de «visualizar el carácter siguiente a continuación del texto en lugar de la línea siguiente». El resultado de la instrucción en la línea 20 y tu respuesta al INPUT de la línea 30 es:

```
CUAL ES TU NOMBRE? PEPE
```

Si hubieras escrito:

```
20  PRINT "CUAL ES TU NOMBRE?"
```

sin punto y coma final, el resultado habría sido:

```
CUAL ES TU NOMBRE?
?PEPE
```

Por supuesto, puedes elegir la manera que más te guste. Recuerda, sin embargo, que si quieres que la respuesta que se escribe en el teclado aparezca en la misma línea que el mensaje que lo solicita, utiliza siempre el punto y coma al final de la instrucción PRINT. De no hacerlo así, la siguiente posición que adoptará el cursor en la pantalla será la del principio de la línea siguiente.

Basándonos en los conocimientos que hemos adquirido acerca de las variables numéricas y de cadena, continuemos nuestro diálogo con HAL, el ordenador. Vamos a añadir las líneas siguientes a nuestro programa:

```
90  PRINT "ESTOY PENSANDO UN NUMERO DEL 1 AL 100"
100 N=INT (RND (1) * 100)
110 PRINT "CUAL CREES QUE ES ";
120 INPUT I
130 IF I<N THEN PRINT "MAYOR... INTENTALO OTRA VEZ. ":
    GOTO 110
140 IF I>N THEN PRINT "MENOR... INTENTALO OTRA VEZ. ":
    GOTO 110
150 PRINT "BRAVO "; PRIMER$; ", LO HAS ACERTADO, ERA EL
    NUMERO "; N
160 END
```

Esto es un diálogo simple. De nuevo, los caracteres que aparecen en negrita son los que suministrarías tú.

```
ESTOY PENSANDO UN NUMERO DEL 1 AL 100
CUAL CREES QUE ES? 26
MAYOR... INTENTALO OTRA VEZ
```

Inspeccionemos el programa en detalle. La instrucción

```
110 PRINT "CUAL CREES QUE ES";
```

pone su mensaje en la pantalla. Otra vez se ha empleado el punto y coma para que los dos dígitos de la respuesta aparezcan en la misma línea. En la instrucción siguiente,

```
120 INPUT I
```

I es una variable numérica. Se le asignará el valor que has escrito en el teclado; en el ejemplo 26. De ahora en adelante, toda vez que se utilice el nombre I, el valor de 26 (o el que le hayas dado) lo sustituirá al leerlo el intérprete de BASIC. Esto ocurrirá en las líneas 130 y 140.

La línea

```
90 PRINT "ESTOY PENSANDO UN NUMERO DEL 1 AL 100";
```

es casi igual a la línea 110 (fíjate en el punto y coma final). La línea

```
100 N = INT (RND (1) * 100)
```

tal vez sea un poco difícil de entender. Su misión es que cada vez que juegues con el programa la variable N tenga un valor diferente, pero siempre comprendido entre 1 y 100. Este valor

que el ordenador «piensa» por sí mismo en esa línea es el número que tú tienes que intentar averiguar.

En las líneas 130 y 140 se comprueba si el número que has introducido (I) es mayor o menor que el valor de N; si es así, el programa te da un mensaje (mira detenidamente las instrucciones PRINT que están dentro de las líneas 130 y 140) y vuelve a la línea 110 (instrucción GOTO 11Ø) para que tengas una nueva oportunidad de acertar el número.

En caso de que aciertes, el programa llega a la línea 150 y te da un mensaje de felicitación en el que el intérprete BASIC sustituirá la variable PRIMER$ por tu nombre y la variable N por el valor que se determinó en la línea 100.

```
150 PRINT "BRAVO"; PRIMER$; "LO HAS ACERTADO, ERA EL
    NUMERO "; N
```

Al ejecutar esta instrucción el ordenador visualizará en la pantalla:

```
BRAVO PEPE, LO HAS ACERTADO, ERA EL NUMERO 43
```

Examinemos esta frase por partes:

BRAVO

(Es una cadena literal, lo que significa que forma parte de una línea de programación.)

PEPE

(Es el valor que el ordenador otorgó a la variable PRIMER$ cuando lo escribiste en el teclado. Ese valor se mantendrá hasta que des la orden NEW o le asignes un nuevo valor PRIMER$.)

LO HAS ACERTADO, ERA EL NUMERO

(Es otra cadena literal.)

43

(Es el resultado de la instrucción de la línea 100.)
En las líneas 130 y 140 usamos las instrucciones

IF («en caso que») THEN («entonces»)

Esta instrucción la estudiaremos en detalle en el capítulo 7, pero ya has visto que le vale al intérprete BASIC para hacer comparaciones. Otra cosa que quizá te hubiese gustado hacer

es conseguir que el programa se ejecute repetidamente (sin necesidad de escribir RUN cada vez). Aprenderemos a hacer esto en el capítulo 6, en el que estudiaremos la instrucción GOTO («ir a línea tal»), aunque ya la hemos usado al final de las líneas 130 y 140.

Ahora que estás familiarizado con las variables numéricas y de cadena, estudiemos la manera de usarlas en un programa más largo, primero asignando un valor a una variable, y luego usando una técnica que emplea un contador.

Asignar un valor a una variable (La instrucción LET)

Hasta ahora la única manera que hemos empleado para dar un valor a una variable ha sido a través de la instrucción INPUT. Por ejemplo, cuando se ejecuta la instrucción:

```
20  INPUT A
```

escribes un valor en el teclado, como por ejemplo 5.2 (seguido de RETURN) y el valor de A es 5.2.

«Oye, Z, guárdame este valor un segundo.»

Existe, sin embargo, otra manera de asignar un valor a la variable A: por medio de una instrucción de *asignación*. Por ejemplo:

```
10  A = 5.2
```

(*Nota:* Utiliza 5 en lugar de 5.2 si usas un BASIC de números enteros.) Esta instrucción sirve para asignar el valor de 5.2 a la variable A como parte integral del programa, sin que tengas que escribirlo en el teclado. Incluso podrías escribir:

```
10  B = 1
20  C = 2
30  A = B + C
```

Como puedes ver, el valor de A se establece en el valor de 2 + 1 = 3 cuando se ejecuta la línea 30.

En las primeras versiones de BASIC, la instrucción de asignación empezaba con la palabra reservada LET. En estas versiones, el ejemplo anterior se escribiría:

```
10  LET B = 1
20  LET C = 2
30  LET A = B + C
```

La finalidad de la instrucción LET era la de simplificar el diseño del intérprete al indicar explícitamente que la instrucción se trataba de una de asignación. Generalmente ya no se emplea la instrucción con la palabra LET. Su eliminación suprime uno de los pasos que ha de seguir el programador al escribir tales instrucciones.

Exploremos ahora la utilidad de la instrucción de asignación. Examinaremos dos ejemplos. En cada uno, calcularemos la suma y el promedio de dos números. A continuación tenemos el primer ejemplo, que no emplea la instrucción de asignación:

```
10  PRINT "DAME DOS NUMEROS"
20  PRINT "CALCULARÉ LA SUMA Y LA MEDIA DE LOS DOS"
30  PRINT "POR FAVOR, EL PRIMER NUMERO:";
40  INPUT A
50  PRINT "POR FAVOR, EL OTRO NUMERO:";
60  INPUT B
70  PRINT "LA SUMA DE"; A; "Y"; B; "ES: "; A + B
80  PRINT "LA MEDIA ES:"; (A + B) / 2
90  END
```

He aquí una ejecución típica:

```
>RUN
DAME DOS NUMEROS
CALCULARE LA SUMA Y LA MEDIA DE LOS DOS
POR FAVOR, EL PRIMER NUMERO: 24
POR FAVOR, EL OTRO NUMERO: 41
LA SUMA DE 24 Y 41 ES: 65
LA MEDIA ES: 32.5
>
```

Fíjate en el hecho de que la expresión A + B se emplea dos veces en las instrucciones de PRINT, lo cual es una inconveniencia de menor importancia, pero en un programa más largo podría resultar trabajoso y aumentaría la posibilidad de errores de mecanografía. Es más, si cambiáramos el programa para que usara otra fórmula, tendríamos que hacer bastantes modificaciones.

A continuación tenemos un programa equivalente que utiliza una *variable intermedia,* con el nombre de SUMA, para almacenar el resultado. Es más fácil de leer y reduce las posibilidades de equivocarse.

```
10  PRINT "DAME NUMEROS"
20  PRINT "CALCULARE LA SUMA Y LA MEDIA"
30  PRINT "POR FAVOR, EL PRIMER NUMERO:";
40  INPUT A
50  PRINT "POR FAVOR, EL SEGUNDO NUMERO:";
60  INPUT B
70  SUMA = A + B
80  MEDIA = SUMA/2
90  PRINT "LA SUMA DE LOS NUMEROS ES:"; SUMA
100 PRINT "LA MEDIA ES:"; MEDIA
110 END
```

Se emplean dos variables nuevas:

```
70  SUMA = A + B
80  MEDIA = SUMA/2
```

El uso de nombres de variables adicionales tiene dos ventajas: el programa es más fácil de leer y más fácil de modificar. Por ejemplo, supongamos que queremos modificar este programa para obtener el promedio de tres números. Para ello, escribiríamos:

```
62  PRINT "POR FAVOR, EL TERCER NUMERO:";
64  INPUT C
70  SUMA = A + B + C
80  MEDIA = SUMA/3
```

Dejaríamos el resto del programa sin cambios. Simplemente hemos introducido un tercer número y modificado las fórmulas en un solo sitio. El programa completo sería:

```
10  PRINT "DAME NUMEROS"
20  PRINT "CALCULARE LA SUMA Y LA MEDIA"
30  PRINT "POR FAVOR, EL PRIMER NUMERO:";
40  INPUT A
50  PRINT "POR FAVOR, EL SEGUNDO NUMERO:";
60  INPUT B
62  PRINT "POR FAVOR, EL TERCER NUMERO:";
64  INPUT C
70  SUMA=A+B+C
80  MEDIA = SUMA/3
90  PRINT "LA SUMA DE LOS NUMEROS ES:"; SUMA
100 PRINT "LA MEDIA ES:";
110 END
```

Una ejecución típica del programa podría ser:

```
DAME NUMEROS
CALCULARE LA SUMA Y LA MEDIA
POR FAVOR, EL PRIMER NUMERO:? 5
POR FAVOR, EL SEGUNDO NUMERO:? 3
POR FAVOR, EL TERCER NUMERO:? 10
LA SUMA DE LOS NUMEROS ES: 18
LA MEDIA ES: 6
```

Hemos aprendido dos métodos para hacer que se asocie un valor con una variable:

— Podemos emplear la instrucción INPUT, por medio de la cual el valor se asigna a la hora de ejecutar el programa.

— Podemos emplear la instrucción de asignación, mediante la cual un valor —o el método de calcular un valor (una fórmula)— se aloja dentro del programa mismo.

El primer método (el uso de la instrucción INPUT) es el preferido para aquellos casos en los que el valor que se suministra para asignar a la variable no es un valor calculado mediante una fórmula, o es un valor explícito que variará con cada ejecución del programa.

El segundo método (el uso de la instrucción de asignación) se debe emplear cuando el valor a asignar se calcula mediante una fórmula, o cuando el valor explícito (es decir, *no* calculado) sea igual para cada ejecución del programa.

Veamos ahora todas las normas que gobiernan la construcción de una instrucción de asignación.

La sintaxis de una instrucción

Las reglas (o sintaxis) que rigen a la hora de construir una instrucción de asignación son sencillas. El formato normal de una de estas instrucciones sigue la fórmula:

<variable> = <expresión>

Siempre debe haber una variable a la izquierda y una expresión a la derecha. Precisando más, podemos decir que una expresión es:

— un número o una variable, o

— un número o una variable seguido de un operador (como pueden ser +, −, *, /), y otra expresión.

He aquí algunas expresiones que sirven de ejemplo:

3	(número)
A	(variable)
2 + 2	(número, operador, número)
A + 2	(variable, operador, número)
A + B * 3	(variable, operador, expresión)

Las expresiones pueden colocarse entre paréntesis; por ejemplo:

3 + (A + 2) / 2

B + ((C * 2) + (D/2)) / 4

Si quieres, puedes pensar que una expresión es un valor, o algo que será calculado y resultará un valor (en otras palabras, una fórmula para calcular un valor).

El signo igual (=) empleado en la instrucción de asignación no se interpreta de la misma manera que cuando se trata de una operación matemática. En una expresión adopta el significado de «recibe el valor de». Por ejemplo, es posible escribir:

```
10  A = 1
20  A = A + 1
```

(La expresión A = A + 1 carece de sentido si de una operación matemática se tratara.) En BASIC, después de la ejecución de la línea 20 el valor de A será 1 (el valor anterior de A) + 1 = 2. Para evitar la confusión que pueda haber, muchos lenguajes de programación modernos usan el símbolo ← o := en lugar del signo igual (=). Recuerda que en una instrucción de asignación en BASIC, el signo = significa que la variable a la izquierda recibe el valor de la expresión a la derecha.

Aquí tenemos varios ejemplos de instrucciones de asignación correctamente expresadas:

```
A = -3 + 2   (-3 es un número entero negativo)
B = A - 1
C = (2*3) + (A/B)
MEDIA = SUMA/NUMERO
CUADRADO = A**2
X = B**2 - (4*A*C)
```

Examinemos esta última asignación para verificar si cumple con nuestra definición:

```
B**2
```

equivale a

<variable> <operador> <número>

seguido de

```
- (4 * A * C)
```

o sea,

<operador> <expresión entre paréntesis que cuenta como valor>

Entre los paréntesis:

```
4 * A * C
```

equivale a

<número> <operador> <variable> <operador> <variable>

Pues sí, es una expresión válida.

Las asignaciones siguientes, sin embargo, no son válidas:

B + C = SUMA	(se permite sólo una variable (y no una expresión) a la izquierda del signo =)
2 = A	(debe haber una variable y no un valor a la izquierda)
SUMA = B + C(D/3)	(falta el operador después de la C)
(MEDIA) = (B + C)/2	(sobran los paréntesis a la izquierda)
A =	(falta el valor a la derecha)

Por último, cabe señalar que, a la hora de ejecutarse una asignación, todas las variables a la derecha del signo = han de haber recibido un valor. Si escribes:

```
10  B = 2
20  SUMA = B + C
30  INPUT C
```

el programa fallará en la línea 20 al no tener un valor la C. Habrías acertado al escribir alternativamente:

```
10  B = 2
20  INPUT C
30  SUMA = B + C
```

Con esto finalizamos el aprendizaje de la sintaxis de las asignaciones. Apliquemos estos nuevos conocimientos a la introducción de una nueva técnica que utiliza las asignaciones: la técnica del contador. Usaremos esta técnica en muchos de los programas que vamos a escribir.

La variable como contador

Ten en cuenta que, como hemos dicho, una variable es simplemente un nombre que se ha dado a una dirección en memoria. Un valor puede guardarse en una dirección de memoria mediante el uso de una instrucción INPUT o una de asignación (=). En este ejemplo, queremos cambiar el valor de una variable constantemente, de forma que sirva como contador de las veces que ocurre alguna acción definida. La técnica que se utiliza para llevar esto a cabo se denomina de la *variable-contador*.

Veamos ahora la manera en la que las asignaciones sucesivas sirven para modificar el valor de la variable N. Escribe la instrucción siguiente en modo inmediato. (*Nota:* Este modo de operación generalmente se llama modo de *calculadora.*)

```
N = 1
```

El valor se almacena automáticamente en N. Verifiquémoslo. Escribe:

```
PRINT N
```

El valor 1 aparece en pantalla. Ahora escribe:

```
N = 2
```

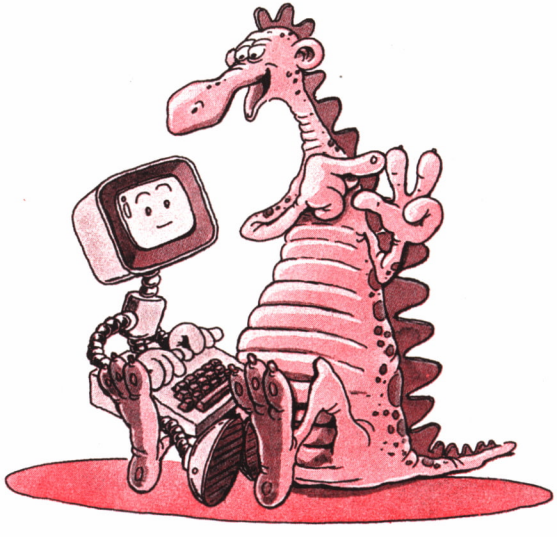
La N ahora contiene el valor de 2. Escribe:

```
PRINT N
```

La respuesta será:

```
2
```

«¡Juguemos a contar!»



El valor 2 ha sustituido al valor 1 en la variable N. Escribe:

```
N = 3
```

Ahora:

```
PRINT N
```

para verificar que el valor 3 se ha almacenado en la N. El mecanismo parece funcionar; pronto lo usaremos para contar. Dicho de otra manera, una variable puede servir para contar las veces que ocurre algo. Aquí tenemos un programa que cuenta las veces que suministras un número desde el teclado. Se para cuando le das al cero.

«Soy una variable-contador.»

```
10  SUMA = 0
20  SUMA = SUMA + 1
30  PRINT "INTRODUCE CUALQUIER NUMERO. ESCRIBE 0
    PARA TERMINAR";
40  INPUT NUMERO
50  PRINT "HAS INTRODUCIDO"; SUMA; "NUMEROS"
60  IF NUMERO < > 0 THEN 20
70  END
```

En el capítulo 6, examinaremos con detalle la función de la instrucción de la línea 60. No obstante, diremos que su significado es SI NUMERO no es igual (< >) a cero, ENTONCES ejecuta la instrucción 20. A continuación mostramos el resultado de una ejecución modelo:

```
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 5
HAS INTRODUCIDO 1 NUMERO
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 1
HAS INTRODUCIDO 2 NUMEROS
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 2
HAS INTRODUCIDO 3 NUMEROS
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 3
HAS INTRODUCIDO 4 NUMEROS
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 4
HAS INTRODUCIDO 5 NUMEROS
INTRODUCE CUALQUIER NUMERO.
ESCRIBE 0 PARA TERMINAR? 0
HAS INTRODUCIDO 6 NUMEROS
END
```

En este programa, la línea 10 *inicializa* el valor de SUMA con el valor de 0. Se incrementa en 1 cada vez que se introduce un número nuevo. Es, por tanto, una variable-contador. Nos encontraremos con muchos ejemplos de esta técnica según vayamos escribiendo más programas. Más adelante aprenderemos la forma de pulir este tipo de programa para evitar, por ejemplo, que se contabilice 0 como un número introducido cuando vamos a parar el programa.

Resumen

En este capítulo hemos examinado la forma de escribir programas que se pueden usar más de una vez. Estos programas dan resultados distintos según los valores que se introducen en el teclado, lo cual hemos logrado mediante las variables y las distintas maneras de asignarles valores.

Una variable ha de considerarse como un nombre que se da a una dirección de memoria, en la que es posible guardar o acumular valores o textos.

Hemos visto la manera de cambiar el contenido de una variable usando una instrucción INPUT o una de asignación. Ahora podemos escribir programas sencillos que automatizan una conversación y/o un cálculo simple.

Por otro lado, los programas que escribimos ahora sobrepasan sin excepción las diez líneas. Han llegado a ser largos; mantengamos un diseño claro al hacerlos. En el próximo capítulo, antes de proceder a la introducción de nuevas técnicas y herramientas, hablaremos de la manera de construir un programa que resulte claro y conciso.

Ejercicios

4.1. Haz un programa que lea cuatro números procedentes del teclado, visualizando en pantalla la suma, la media y el producto de ellos.

4.2. Asumiendo que tu versión de BASIC admite los nombres largos, determina si los nombres de las siguientes variables son admisibles o no:

a)	24B	*e)*	ALPHA2D	*i)*	PI
b)	B24	*f)*	EJEMPLO	*j)*	3$
c)	A + B	*g)*	INPUT	*k)*	TRES
d)	APLUSB	*h)*	INPUT1	*l)*	NOMBRE$

4.3. Escribe un programa que solicite el nombre del usuario y luego envía el mensaje, «CREO QUE YO CONOZCO A UN TAL (aquí el nombre)!»

4.4. Escribe un programa que solicite:

— el nombre de un objeto,
— el nombre de un mueble,
— el nombre de un amigo,

y luego dice: «TIENE TU AMIGO (nombre) UN (objeto) ENCIMA DE (mueble)?»

4.5. Escribe un programa que solicite el color de tus ojos y luego dice «ME GUSTAN LOS OJOS (color)»

4.6. ¿Son válidas las siguientes asignaciones?:

a)	A + 1 = A	*d)*	B + C = A
b)	A = A + A + A	*e)*	3 = 2 + 1
c)	A = B + C	*f)*	NUMERO = PRIMERO + ULTIMO *2

5 Cómo

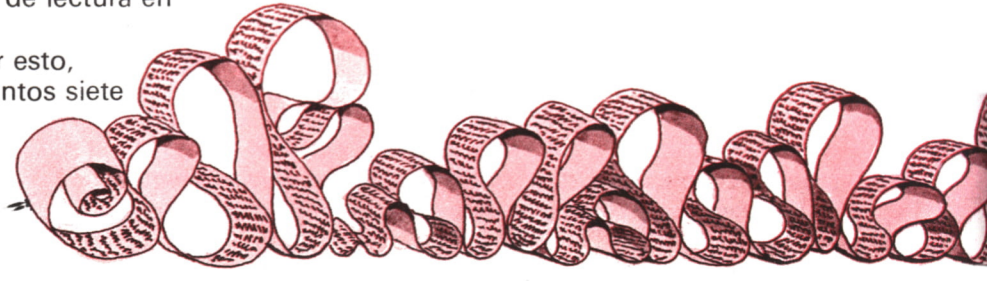
Hasta ahora hemos escrito programas cortos mediante el uso de tres tipos de instrucción: PRINT, INPUT y de asignación (=). También hemos visto la manera en la que se escriben expresiones aritméticas sencillas. En los capítulos siguientes aprenderemos nuevas técnicas y tipos de instrucción, escribiendo mediante su uso programas más largos. Antes de seguir adelante, sin embargo, hablemos de la manera de hacer programas coherentes y fáciles de leer.

Es importante que un programa sea fácil de leer. Si escribes un programa hoy y luego al cabo de unos días deseas usarlo o modificarlo, te llevará tiempo volver a descubrir cómo funciona. Como regla general, es preferible planificar cada programa antes de escribrirlo, haciendo que sea lo más fácil posible de leer según lo vas construyendo. Nuestro objetivo en este capítulo es el de ayudarte a mantener un alto nivel de claridad de lectura en tus programas.

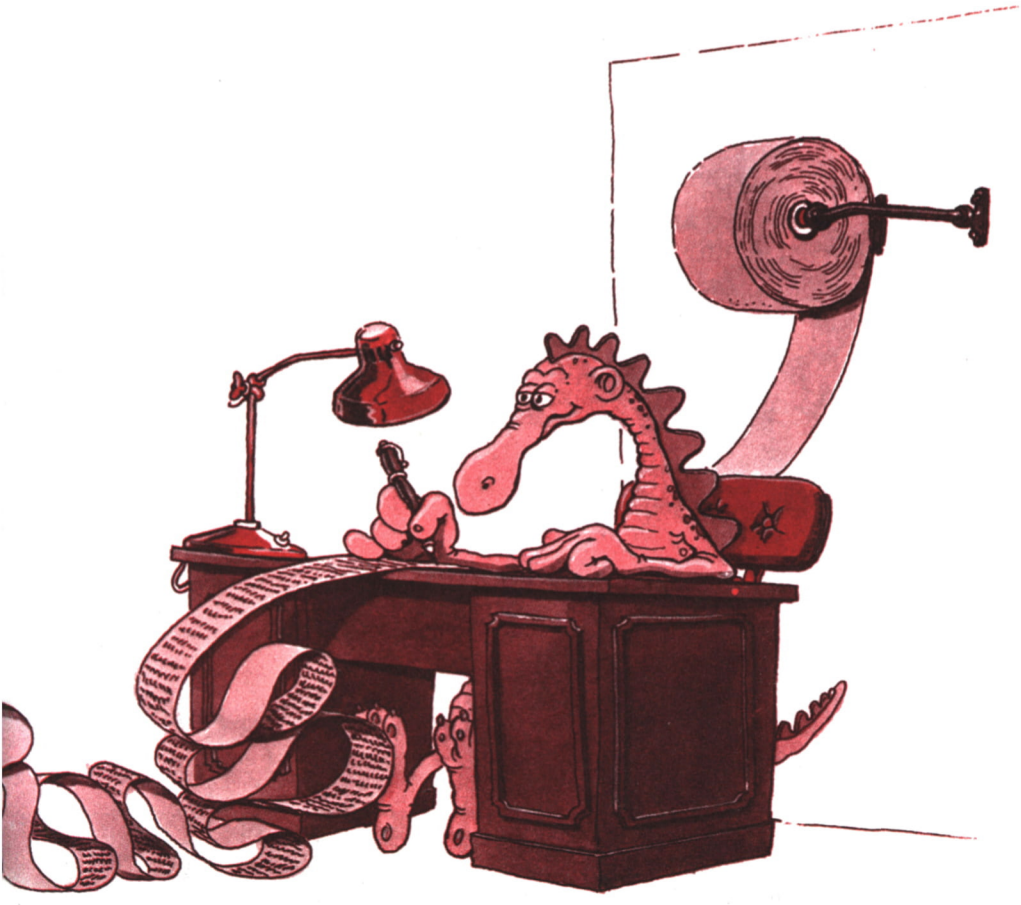
Para conseguir esto, examinaremos juntos siete técnicas:

1. El uso de la instrucción REM (para incorporar un comentario al programa; «REMark» en inglés = comentario).
2. El uso de más de una instrucción por línea.
3. El uso de espacios en blanco dentro de una instrucción.
4. El uso de la instrucción «nula» o vacía de PRINT y la instrucción CLE (para mejorar lo que se visualiza en pantalla).
5. El uso del «atajo con INPUT» (para reducir el número de líneas en el programa).
6. La selección de nombres significativos para las variables.
7. La numeración adecuada de líneas.

Pasemos ahora, pues, a examinar una por una cada técnica.



escribir programas claros



La instrucción REM

Aquí tenemos un ejemplo de la instrucción REM:

```
10  REM * * * PROGRAMA PARA SUMAR * * *
20  PRINT "DAME DOS NUMEROS:";
30  INPUT PRIMERO, SEGUNDO
40  PRINT "LA SUMA DE LOS DOS ES: "; PRIMERO + SEGUNDO
50  END
```

La instrucción REM se emplea para conseguir que sea más fácil seguir un programa mediante un comentario incorporado al mismo. Esta instrucción es ignorada por el intérprete a la hora de ejecutar el programa, teniendo un efecto nulo sobre el programa. Aquí tienes más ejemplos de comentarios que podrías usar:

```
25  REM AHORA LEE LOS DOS NUMEROS
45  REM TAMBIEN PODRIAMOS MULTIPLICARLOS COMO
    PARTE DE LA LINEA ANTERIOR
```

Los comentarios son invisibles para el Intérprete.

«¿Ves cómo las estrellitas me ayudan a seguir mi programa?»

Incorporándolos al programa tenemos:

```
10  REM * * * PROGRAMA PARA SUMAR * * *
20  PRINT "DAME DOS NUMEROS:";
25  REM AHORA LEE LOS DOS NUMEROS
30  INPUT PRIMERO, SEGUNDO
40  PRINT "LA SUMA DE LOS DOS ES: "; PRIMERO + SEGUNDO
45  REM TAMBIEN PODRIAMOS MULTIPLICARLOS COMO
    PARTE DE LA LINEA ANTERIOR
50  END
```

Para destacar mejor tus comentarios, puedes usar una variedad de símbolos:

```
10   REM * * * PROGRAMA PARA SUMAR * * *
60   REM-----SEGUNDA PARTE-----
100  REM=====PARTE FINAL=====
200  REM$$$$$CAMBIAR ESTA SECCION MAS
     ADELANTE$$$$$
```

Líneas de instrucciones múltiples

La mayoría de las versiones de BASIC te permiten escribir más de una instrucción en una línea de programación, separadas normalmente por dos puntos (:). Por ejemplo:

```
50  PRINT "ESCRIBE EL NUMERO"; : INPUT NUMERO
```

Lo cual es lo mismo que escribir:

```
50  PRINT "ESCRIBE EL NUMERO";
60  INPUT NUMERO
```

Ten en cuenta que sólo puede haber una numeración por cada línea, de manera que sólo hay un número a la izquierda cuando se colocan dos instrucciones en una línea. Otro ejemplo:

```
100  REM * * * CALCULAR TODO EN UNA LINEA * * *
110  SUMA = A + B : PRODUCTO = A * B : MEDIA = SUMA/2
```

Vemos que hay tres instrucciones en la línea 110.

La colocación de más de una instrucción en una línea es especialmente útil en al menos dos casos: la clarificación de una instrucción INPUT y la introducción de un comentario (REM) a la derecha de una instrucción. Un ejemplo de la clarificación de INPUT podría ser:

```
70  PRINT "INTRODUCE DOS NUMEROS"; : INPUT N1, N2
```

La línea 70 se ha escrito conforme con lo que luego se visualiza en pantalla, haciendo que sea más fácil seguir la actuación del programa.

A continuación tenemos un ejemplo de la colocación de un comentario al lado de la instrucción a la que se refiere:

```
60  RESULTADO = A + 2 * B + 5 : REM EL RESULTADO HA DE
    SER POSITIVO
```

Otro ejemplo sería:

```
50  TEMPERATURA = (FAHR - 32) * 5/9 : REM CONVERTIMOS
    A GRADOS CELSIUS
```

El uso de espacios en blanco

Con la excepción de los nombres, cadenas o datos introducidos como respuesta a la instrucción INPUT, los espacios en blanco son, generalmente, ignorados por el lenguaje BASIC. Por ejemplo, es posible escribir:

```
20  PRINT4+2*3
```

Sin embargo, es difícil leer esta instrucción. Para facilitar la lectura de tu programa, usa los espacios en blanco sin temor. Puedes usarlos:

— *Después de cada palabra reservada, como PRINT o INPUT.*

Como por ejemplo:

```
50  PRINT 4
60  INPUT NUMERO
```

— *Antes y después de cada operador.*

Como por ejemplo:

```
30  PRINT 4 + 2 * 3
40  RESULTADO = A1 / ((B - C) * D)
```

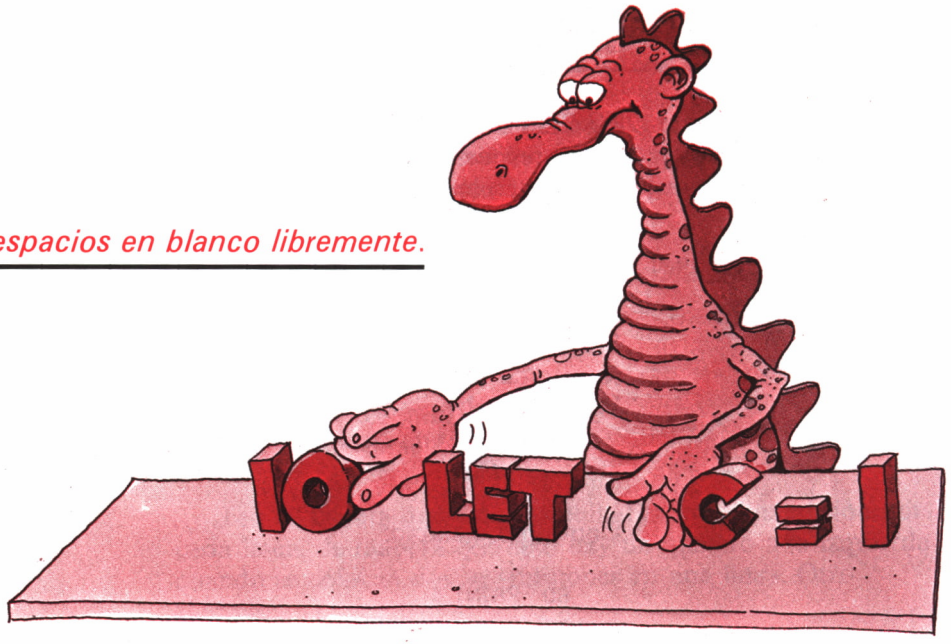
A lo mejor querrás también usar espacios en blanco antes de una palabra reservada, con el fin de alinear o sangrar las instrucciones de un programa. Por ejemplo:

```
10    PRINT "DIEZ"
20    PRINT "VEINTE"
90    PRINT "NOVENTA"
100   PRINT "CIEN"
200   PRINT "DOSCIENTOS"
```

Sin los dos espacios en blanco iniciales en las líneas 10, 20 y 90, el programa tendría el siguiente aspecto:

```
10  PRINT "DIEZ"
20  PRINT "VEINTE"
90  PRINT "NOVENTA"
100  PRINT "CIEN"
200  PRINT "DOSCIENTOS"
```

Utiliza los espacios en blanco libremente.



Por último, una combinación entre las instrucciones de REM y los espacios en blanco crea un programa cuyo texto está casi perfectamente compaginado:

```
1 REM ESTE PROGRAMA LLEVA MI INVENTARIO
2 REM COPYRIGHT YO MISMO 1984
3 REM ESTAS SON LAS VARIABLES:
4 REM C ES EL COLOR (1 A 10)
5 REM U ES EL NUMERO DE UNIDADES (HASTA 1000)
6 REM T ES EL TAMAÑO (1 A 50)
7 REM P ES EL PRECIO DE UNIDAD
8 REM V ES EL PRECIO DE VENTA AL PUBLICO
9 REM N ES LA CANTIDAD A PEDIR
```

Los espacios en blanco mejoran con mucho la facilidad de lectura. No obstante, ten cuidado de no usarlos en medio de una palabra reservada, el nombre de una variable, o durante la respuesta a un INPUT, a no ser que los espacios formen parte de una cadena.

Cuidando el aspecto de la pantalla

Hay dos técnicas nuevas que se pueden emplear para mejorar el aspecto de todo lo que hagas aparecer en pantalla: la instrucción CLE (de CLEar, o «borrar») y la instrucción vacía de PRINT.

La instrucción CLE borra la pantalla cuando se ejecuta. Incluso, a lo mejor, te interesa comenzar cada programa con:

```
10 NEW : CLE
```

Esta línea de dos instrucciones borra tanto la memoria (NEW) como la pantalla (CLE).

Otro truco que igual te interesa usar es el de escribir al final de tu programa:

```
110 CLE : LIST
```

Esto hará que se limpie la pantalla y el programa se liste, después de que haya sido ejecutado.

Algunas versiones de BASIC, aunque no todas, vienen provistas de la instrucción CLE o una similar (CLS, por ejemplo). Has de saber, sin embargo, que en la versión de BASIC de Microsoft esta instrucción anula el valor de todas las variables, en lugar de borrar la pantalla (*Nota:* Si tu versión no dispone de la instrucción CLE, en general es posible lograr los mismos resultados al escribir:

```
10 PRINT CHR$ (NUM)
```

donde NUM es el número de un carácter especial que limpia la pantalla, lo cual se especifica en el manual de tu ordenador.)

La instrucción vacía de PRINT se utiliza para visualizar una línea en blanco, lo cual se logra escribiendo simplemente:

```
50 PRINT
```

Si quieres saltar tres líneas en la pantalla, escribe:

```
50 PRINT : PRINT : PRINT
```

O, alternativamente:

```
50 PRINT
60 PRINT
70 PRINT
```

El «atajo» con INPUT

¡No te compliques la vida!

Cada vez que empleas una instrucción INPUT, has de indicar al usuario lo que debe introducir desde el teclado. Por ejemplo:

```
50  PRINT "ESCRIBE DOS NUMEROS ENTEROS";
60  INPUT EDAD1, EDAD2
```

La secuencia de PRINT - INPUT es tan frecuente que la mayoría de los BASIC modernos permiten un «atajo» con la instrucción INPUT, de manera que puedes escribir:

```
50  INPUT "ESCRIBE DOS NUMEROS ENTEROS"; EDAD1, EDAD2
```

lo cual es equivalente a las dos líneas que vimos anteriormente. Esta técnica reduce la mecanografía necesaria y muestra claramente lo que se va a ver en pantalla.
Si tu versión de BASIC no admite este pequeño «atajo», puedes todavía emplear el método de colocar dos instrucciones en una línea:

```
50  PRINT "ESCRIBE DOS NUMEROS ENTEROS"; : INPUT
    EDAD1, EDAD2
```

La selección de nombres de variables

Siempre has de procurar que el nombre que asignas a una variable te ayude a recordar lo que representa. De otra manera encontrarás dificultades en escribir programas largos, cometiendo muchos errores accidentalmente. Es más, si los nombres de variables que empleas no son obvios, podrías encontrarte en la situación de no poder descifrar tu propio programa algún tiempo después de haberlo escrito.

Si tu versión de BASIC sólo admite una letra con un número opcional para los nombres de variables, no hay mucho que puedas hacer para evitar algunos problemas. No obstante, algunos nombres con cierto nivel de representatividad podrían ser, por ejemplo:

```
10 REM USAREMOS:
20 REM  R  PARA LOS RESULTADOS
30 REM  P  PARA EL PRIMER NUMERO
40 REM  U  PARA EL ULTIMO NUMERO
```

Si tu versión admite nombres de varias letras, úsalos. Siguiendo el ejemplo anterior, lo mejoraríamos escribiendo:

```
10 REM LOS VARIABLES PARA EL SEGUNDO NUMERO SON:
20 REM  RESULTADO  PARA LOS RESULTADOS
30 REM  PRIMERO    PARA EL PRIMER NUMERO
40 REM  ULTIMO     PARA EL ULTIMO NUMERO
```

En esto hay dos limitaciones principales que hay que tener en cuenta:

— Tu versión de BASIC te limitará a un número máximo de caracteres.

— No podrás usar una palabra reservada, como PRINT, REM o INPUT, como el nombre de una variable.

Es buena práctica identificar al principio de tu programa todos los nombres de variables que vas a emplear, al igual que las fórmulas y expresiones matemáticas.

La numeración adecuada de las líneas

Hasta ahora en este capítulo hemos utilizado una numeración de líneas basada en múltiplos de 10:

```
10  (instrucción)
20  (instrucción)
30  (instrucción)
```

No obstante, es posible usar la secuencia que quieras, siempre y cuando utilices números enteros positivos y no excedas el límite de tu intérprete para los números de líneas. Por ejemplo, puedes escribir:

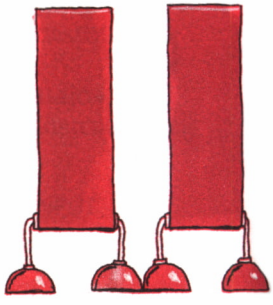
```
1  (instrucción)
2  (instrucción)
3  (instrucción)
```

o:

```
100  (instrucción)
200  (instrucción)
300  (instrucción)
```

También hemos tomado la precaución de dejar siempre un hueco entre numeraciones consecutivas para poder añadir correcciones o mejoras más adelante. Por ejemplo, aquí tenemos la primera versión de un programa:

```
10  REM * * PROGRAMA DE MULTIPLICACION * *
20  INPUT "DAME DOS NUMEROS"; N1, N2
30  PRINT "EL PRODUCTO ES: "; N1 * N2
40  END
```

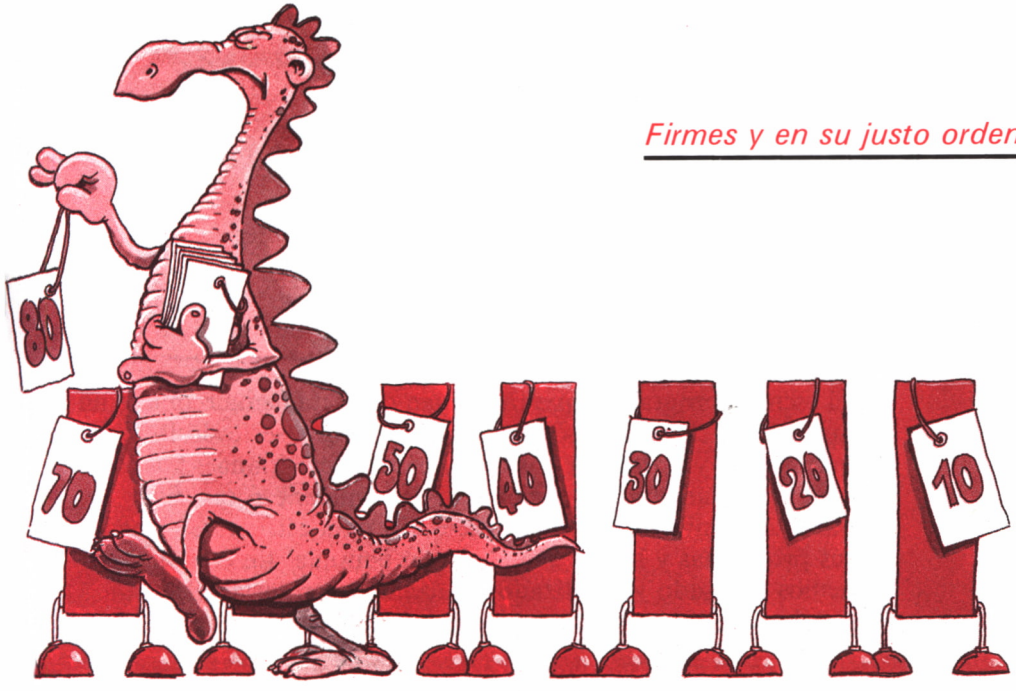


Ahora queremos facilitar la lectura del programa y mejorar su visualización en pantalla. Para lograrlo, añadimos las siguientes instrucciones:

```
5   NEW : CLE
15  PRINT "ESTE ES UN PROGRAMA DE MULTIPLICACION
    AUTOMATICA"
16  PRINT : PRINT : PRINT
35  PRINT : PRINT
```

Las nuevas instrucciones se insertarán de forma automática. Listemos el resultado:

```
>LIST
5   NEW : CLE
10  REM * * * PROGRAMA DE MULTIPLICACION * * *
15  PRINT "ESTE ES UN PROGRAMA DE MULTIPLICACION
    AUTOMATICA"
16  PRINT : PRINT : PRINT
20  INPUT "DAME DOS NUMEROS"; N1, N2
30  PRINT "EL PRODUCTO ES; "; N1 * N2
35  PRINT : PRINT
40  END
```



Firmes y en su justo orden.

Aquí tenemos una ejecución de muestra del programa:

```
ESTE ES UN PROGRAMA DE MULTIPLICACION AUTOMATICA

DAME DOS NUMEROS? 12, 15
EL PRODUCTO ES: 180
```

Si piensas hacer muchas correcciones o añadir bastantes instrucciones, a lo mejor quieres dejar mucho espacio entre cada numeración, como por ejemplo:

```
10   (instrucción)
50   (instrucción)
60   (instrucción)
100  (instrucción)
```

Muchas versiones de BASIC tienen un comando especial llamado RENUMBER que cambiará automáticamente la numeración de todas las líneas en tu programa fijando un incremento de 10 entre cada línea, lo cual es muy útil con programas largos en los que se te ha acabado el espacio entre líneas al insertar correcciones o modificaciones.

Resumen

El escribir programas que funcionan correctamente requiere un ingrediente clave: la disciplina. Es importante ser todo lo organizado y ordenado posible al escribir programas. Las chapuzas aumentan la posibilidad de cometer errores. En particular, invierte el tiempo necesario para estructurar bien tus programas y lo que visualizan en pantalla. En este capítulo hemos repasado y enfatizado las técnicas necesarias para escribir programas caracterizadas por su claridad.

Según vayas programando más en BASIC, no debes dejar de hacer caso a las sugerencias ofrecidas en este capítulo. Es esencial que adquieras buenos hábitos de programación; de no hacerlo así, tus programas resultarán imposibles de leer, o incluso llenos de fallos, según vayas escribiendo programas más y más complejos.

Ejercicios

5.1. Describe aquellas técnicas que sirven para mejorar la calidad de lo que se imprime en pantalla.

5.2. Verifica si son aceptables las siguientes fórmulas o expresiones:

a) A = A + 1
b) A = A + 1
c) PRINT ALPHA + 2
d) SUMA = 2 + (3 + (4/5)) / 2
e) IN PUT NUMERO
f) SUMA = 2 2 + 3 3

5.3. Explica por qué es necesario que la mayoría de los INPUT sigan a un mensaje previo.

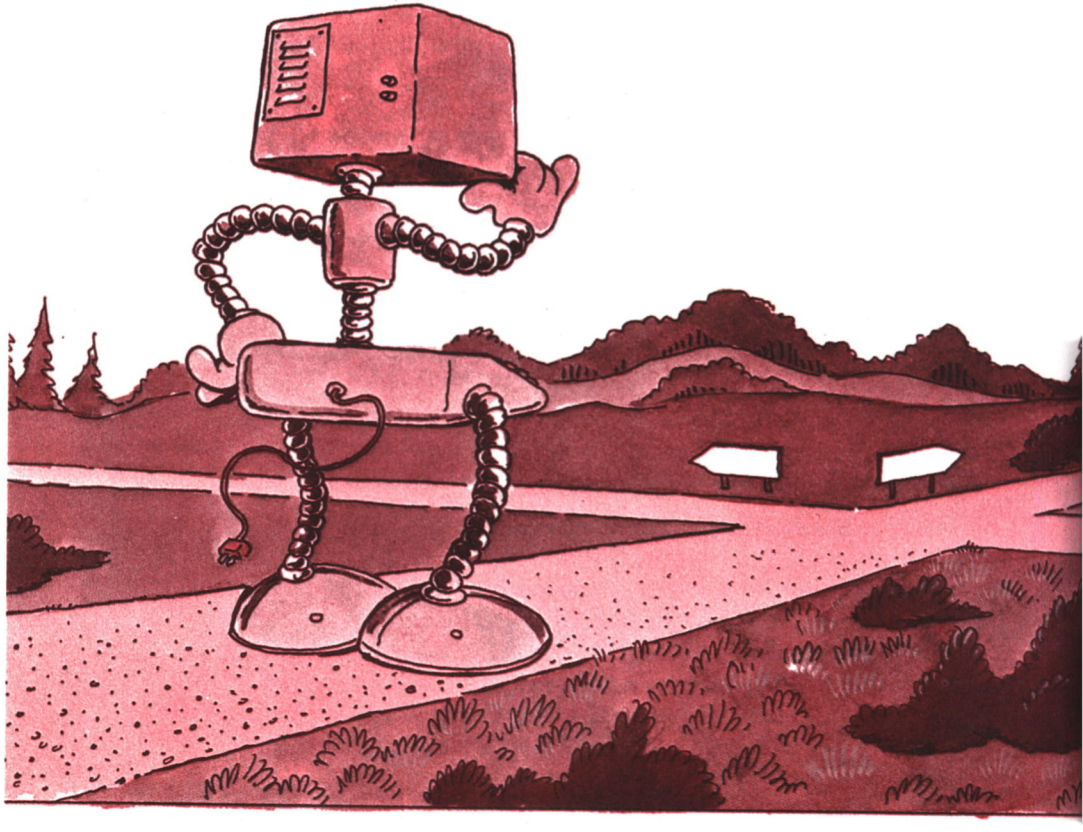
5.4. Escribe tres ejemplos de instrucciones INPUT con «atajos».

5.5. Para qué sirven los REM?

5.6. Cuál es el valor de A después de estas dos instrucciones?

```
30 A = 3
40 REM A = 4
```

El or
y las de



denador cisiones

6

Hasta ahora hemos aprendido a comunicarnos con el ordenador y a efectuar operaciones aritméticas sencillas. Sin embargo, nuestros programas han sido algo aburridos, puesto que fácilmente hubiéramos podido hacer las mismas tareas a mano. Esto se debe a que hemos utilizado únicamente los recursos elementales del ordenador, sin explorar y aprovecharnos de sus recursos más avanzados. Por ejemplo, los ordenadores son muy hábiles a la hora de realizar dos tareas en concreto: tomar decisiones complejas (basadas en la lógica y la comparación de valores) y ejecutar tareas repetitivas muchas veces en un tiempo muy corto. Esto es precisamente lo que aprenderemos a hacer con nuestros programas en este capítulo y el siguiente. Espercíficamente, aprenderemos a tomar decisiones; nuestros programas llegarán a ser «inteligentes», decidiendo lo que hay que hacer en un momento determinado.

En el lenguaje BASIC las decisiones se hacen comprobando un valor mediante la instrucción IF. Si las condiciones estipuladas se cumplen, cierta parte del programa se ejecuta y, si no, se ejecuta otra parte. Ahora vamos a aprender a usar la instrucción IF para hacer comprobaciones. También aprenderemos a utilizar la instrucción GOTO para obligar al programa a ejecutar una serie de instrucciones fuera de su secuencia numérica.

La instrucción IF

Se escribe la instrucción IF de la siguiente manera:

IF (condición) THEN (instrucción; es decir, tarea a hacer)

Aquí tenemos un ejemplo:

```
IF I = 1 THEN PRINT "UNO"
```

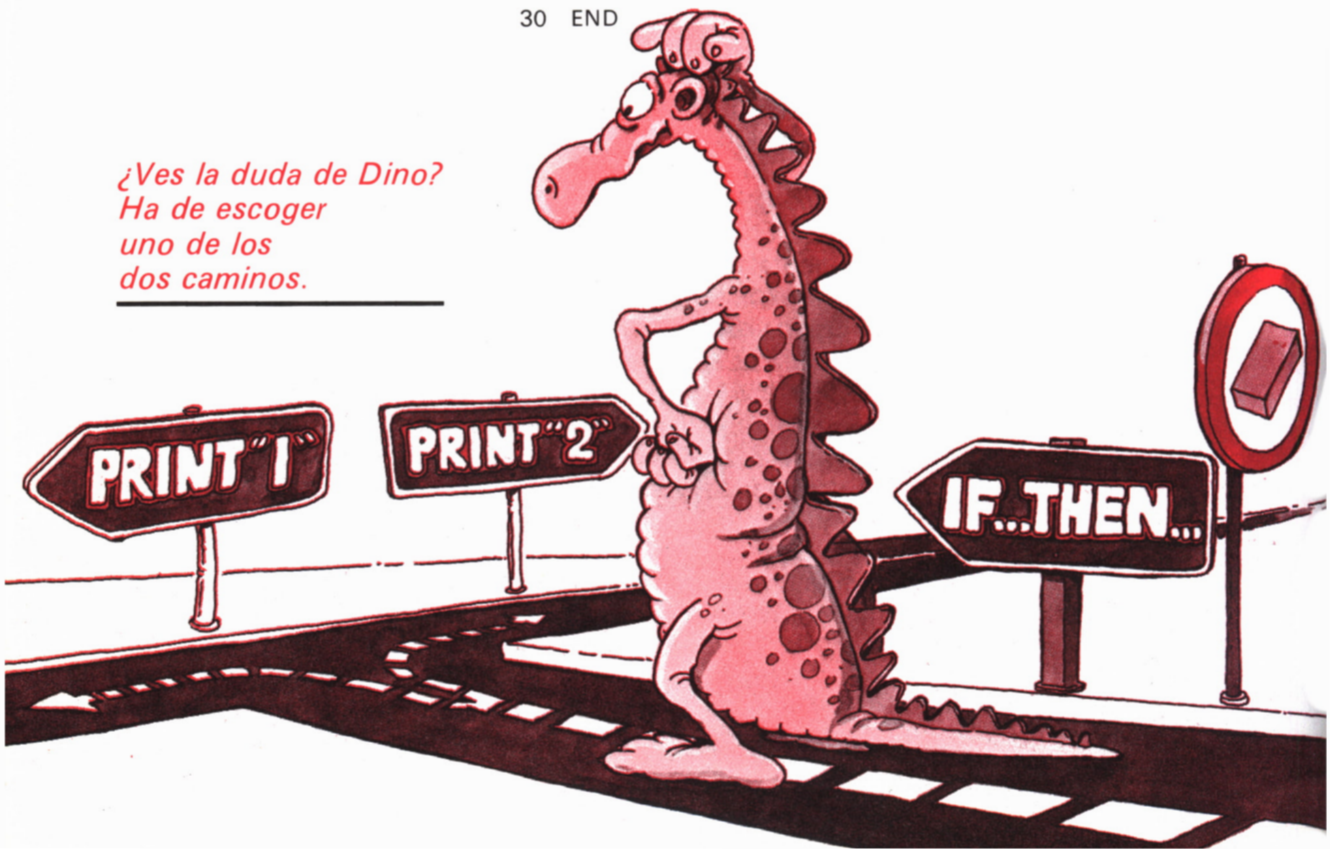
El efecto de esta instrucción debe quedar claro: SI (IF) el valor de la variable I es igual a 1 en el momento que se ejecuta esta instrucción, ENTONCES (THEN) la palabra UNO se imprime. Si I no es igual a 1, no ocurre nada y el programa pasa a la línea siguiente.

I = 1 se denomina *expresión* lógica. La expresión I = 1 es *verdad* cuando I es igual a 1; de otra manera se considera *falsa*.

La instrucción IF... THEN te permite comprobar el valor de una expresión y ejecutar una instrucción u otra —es decir, tomar una decisión— según los resultados de la comprobación. Otro ejemplo:

```
10  INPUT I
20  IF I = 1 THEN PRINT "UNO"
30  END
```

¿Ves la duda de Dino? Ha de escoger uno de los dos caminos.



Ejecuta el programa. Escribe «1» desde el teclado. Tu pantalla ha de tener el siguiente aspecto:

```
>RUN
? 1
UNO
>
```

Ejecuta el programa de nuevo, esta vez escribiendo el número «2» en el teclado. La pantalla ha de presentar este aspecto:

```
>RUN
? 2
>
```

En esta ocasión no se imprimió ningún mensaje en respuesta al número tecleado, el 2.

Ahora enseñemos al programa a reconocer los números del 1 al 4:

```
10  REM ESTE PROGRAMA RECONOCE LOS NUMEROS DEL 1
    AL 4
20  INPUT "ESCRIBE UN NUMERO ENTERO:"; NUMERO
30  IF NUMERO = 1 THEN PRINT "UNO"
40  IF NUMERO = 2 THEN PRINT "DOS"
50  IF NUMERO = 3 THEN PRINT "TRES"
60  IF NUMERO = 4 THEN PRINT "CUATRO"
70  END
```

Ejecutemos el programa. A continuación vemos dos ejemplos de las ejecuciones según su aspecto en la pantalla (con la negrita añadida):

```
>RUN
ESCRIBE UN NUMERO ENTERO:? 3
TRES
>RUN
ESCRIBE UN NUMERO ENTERO:? 5
>
```

Esto está bien, pero no llega a la perfección. En el mejor de los casos, nos gustaría que el programa respondiese con algún mensaje al escribir el número 5 en el teclado, algo como:

```
NO CONOZCO ESE NUMERO
```

o bien solicitar un número entero diferente.

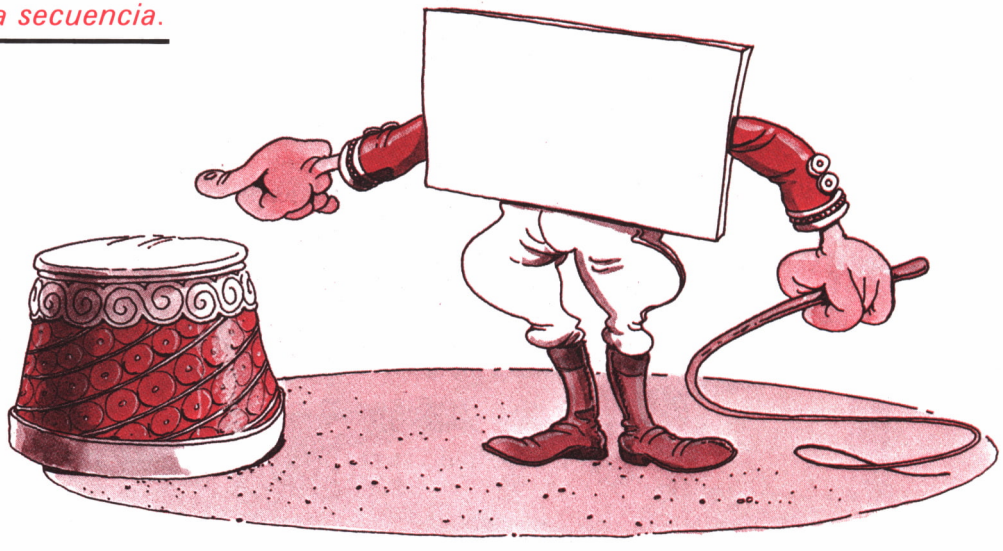
Hay una característica de la instrucción IF que puede hacer que esto ocurra. Por ejemplo, puedes escribir:

```
70  IF NUMERO = 5 THEN 20
```

donde 20 es el número de la línea a ejecutar si la comprobación determina que se cumple la condición. Esta instrucción viene a decir que si NUMERO es igual a 5, entonces hay que ejecutar como línea siguiente la 20. ¡Podemos saltarnos de la secuencia numérica establecida! Aquí tenemos un ejemplo:

```
10  INPUT I
20  IF I = 1 THEN 50
30  PRINT "NO ESCRIBISTE UN 1"
40  END
50  PRINT "ESCRIBISTE UN 1"
60  END
```

Podemos hacer que el programa salte fuera de la secuencia.



Ejecuta el programa, escribiendo un número 1 en el teclado. La pantalla ha de presentar este aspecto:

```
>RUN
? 1
ESCRIBISTE UN 1
>
```

Ejecuta el programa de nuevo, esta vez escribiendo un número 2 en el teclado. Tu pantalla tendrá este aspecto:

```
>RUN
? 2
NO ESCRIBISTE UN 1
>
```

¡Qué contento me ponen los fallos!

Nuestro programa ahora muestra «inteligencia», es decir, emite un mensaje apropiado, sea o no la respuesta al INPUT el número 1. A lo mejor te estás preguntando si no hubiéramos podido conseguir el mismo resultado utilizando la forma original de la instrucción IF, así que, vamos a verlo:

```
10  INPUT I
20  IF I = 1 THEN PRINT "ES UN UNO"
30  PRINT "NO ES UN UNO"
40  END
```

Ahora, ejecuta el programa y responde con un número 1. La pantalla tendrá visualizado:

```
>RUN
? 1
ES UN UNO
NO ES UN UNO
```

¡No funciona! Independientemente del resultado de la comprobación hecha por la instrucción IF, se ejecuta la instrucción que la sigue (línea 30 en este caso).

En el ejemplo primero recibimos el mensaje correcto al ejecutarse la instrucción IF:

```
ES UN UNO
```

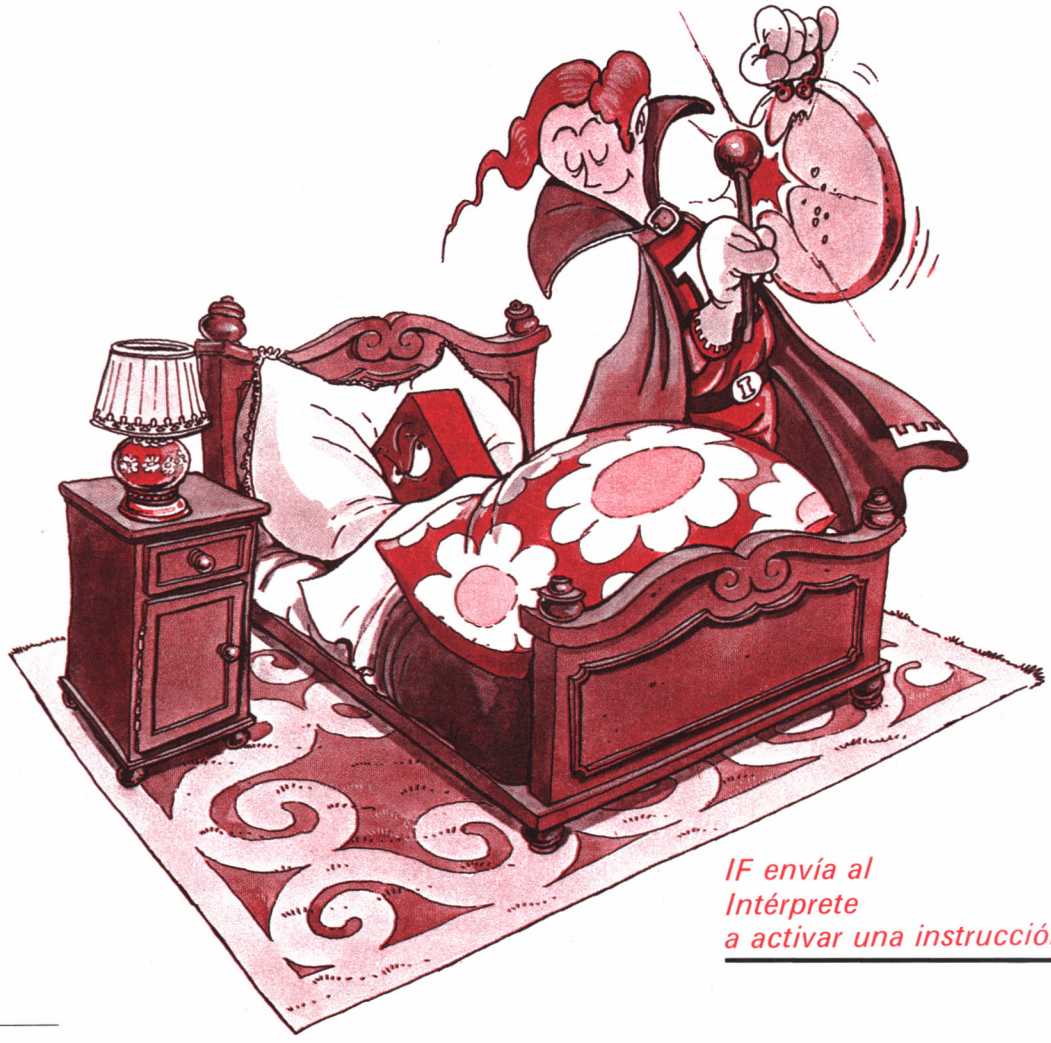
Entonces se imprime el siguiente mensaje de todas maneras:

```
NO ES UN UNO
```

La nueva forma de la instrucción IF elimina el problema:

```
IF I = 1 THEN 50
```

Usaremos esta instrucción frecuentemente en nuestros programas.



IF envía al Intérprete a activar una instrucción.

Vamos a reexaminar la instrucción IF para explotar su potencial al máximo. El formato estándar de la instrucción IF es el siguiente:

IF (expresión lógica) THEN (instrucción a ejecutar o número de línea)

Examinemos primero las expresiones lógicas y luego las instrucciones o líneas a ejecutar.

Las expresiones lógicas

En nuestro ejemplo, I = 1 es una *expresión* lógica; es decir, puede ser o *verdadera* o *falsa,* a los que se llama *valores lógicos.* Aquí tenemos algunos ejemplos:

I = 1	(I es igual a 1)
I > 4	(I es mayor a 4)
NUMERO < 100	(NUMERO es menor que 100)
AÑO < > 5	(AÑO no es igual a 5)
EDAD < 13	(EDAD es menor que 13)

Una expresión lógica combina los valores o las variables con los operadores lógicos. A efecto de que los conozcas todos, los operadores que podrás usar con las expresiones lógicas son:

=	igual a	
< >	no igual a	(en matemáticas se escribe ≠ o #)
<	menor que	
>	mayor que	
< =	menor que o igual a	(en matemáticas se escribe ⩽)
> =	mayor que o igual a	(en matemáticas se escribe ⩾)

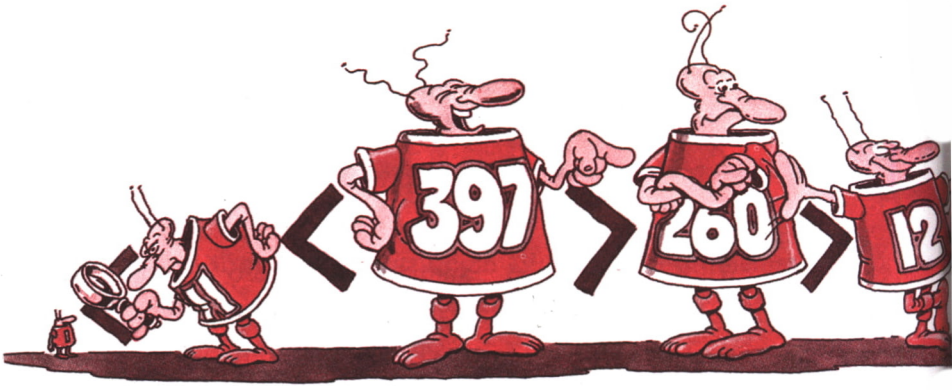
Otras expresiones lógicas aun más complejas podrían ser:

(NUMERO + 2) > 4

(EDAD − 5) > = 10

((2 * I − 5) / 2) < 10

2 > I



También es posible escribir:

$4 > 2$ (lo cual es siempre verdad)

$4 = 2$ (lo cual es siempre falso)

Algunas expresiones lógicas *no* válidas serían:

$2 < I < 0$	(sólo se puede emplear un operador relacional)
$(2\ EDAD - 2) < 5$	(expresión inválida, falta un $*$. Debe expresarse: $(2 * EDAD - 2) < 5$)

Hasta se pueden combinar las expresiones lógicas mediante los *operadores lógicos* AND, OR y NOT. Por ejemplo, se puede escribir:

```
IF (EDAD>9) AND (EDAD<20) THEN PRINT "ES UN ADOLESCENTE"
```

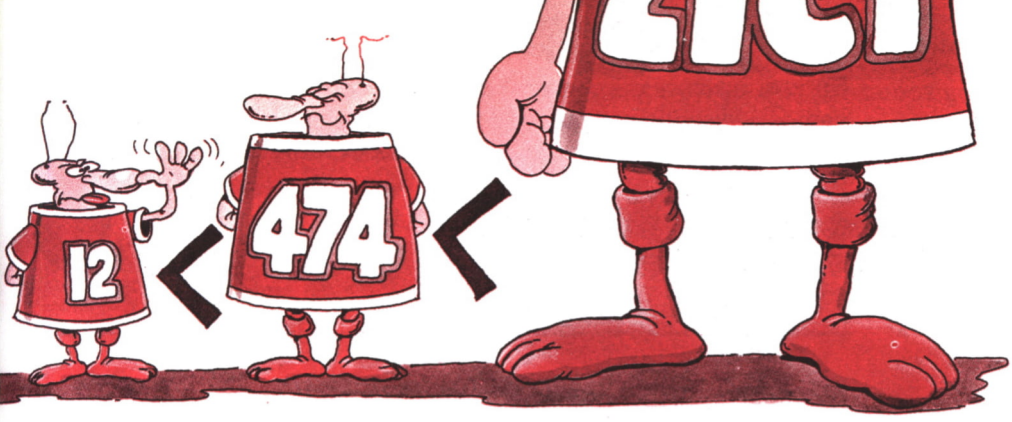
Esta instrucción imprimirá «ES UN ADOLESCENTE» cuando el valor de EDAD sea mayor que 9 y menor que 20. El AND se cumple, es decir, es verdadera, cuando *ambas* cláusulas sean verdaderas. Fíjate que *no* es posible escribir:

```
IF EDAD (>9 AND<20) THEN...
```

dado que cada instancia de dos paréntesis ha de encerrar una expresión válida.

He aquí otro ejemplo que utiliza dos expresiones lógicas:

```
20  INPUT RESPUESTA$
30  IF (RESPUESTA$="SI") o (RESPUESTA$="NO") THEN 60
40  PRINT "RESPUESTA INVALIDA"
    ...
60  PRINT "RESPUESTA VALIDA. PROSIGAMOS"
```



Los operadores de relación se pueden emplear con las variables.

Este segmento de un programa recoge una respuesta en la variable RESPUESTA$ (recuerda que el símbolo $ al final de un nombre indica que es una variable de cadena, es decir, una secuencia de caracteres). SI y NO son las únicas respuestas aceptables; el programa comprueba la validez de lo que se haya escrito en el teclado.

Si escribes SI, entonces (RESPUESTA$ = «SI») es verdad, y la instrucción IF es verdadera: la línea 60 se ejecuta a continuación y el programa imprime:

```
RESPUESTA VALIDA. PROSIGAMOS
```

De la misma manera, si escribes NO, ocurre exactamente lo mismo.

Si escribes cualquier otra cosa, recibes el mensaje diagnóstico:

```
RESPUESTA INVALIDA
```

Se satisface la condición OR, es decir, es verdad, cuando cualquiera de sus dos cláusulas sea verdadera, y falla cuando *ambas* sean inciertas.
Por ejemplo, si escribieras «YA», entonces (RESPUESTA$ = «SI») falla, y la segunda cláusula de la condición OR se comprueba (RESPUESTA$ = «NO»). También falla, y el programa emite el mensaje:

```
RESPUESTA INVALIDA
```

Por último, se puede usar el operador NOT para negar una condición. A continuación tenemos un ejemplo de una instrucción IF compleja:

```
IF ((MEDIA 3.5) AND (EXAMEN 3.0) AND NOT (ORAL 4.0)) THEN
    PRINT "SUSPENSO"
```

En esta instrucción comprobamos tres condiciones a la vez. No vamos a discutir instrucciones tan complejas en más detalle en este momento, pero puede que quieras experimentar con ellas por tu cuenta.

Instrucciones a ejecutar

Recordemos la definición de la instrucción IF:

IF (expresión lógica) THEN (instrucción a ejecutar
o número de línea)

Ya que estamos familiarizados con las expresiones lógicas, examinemos la segunda mitad de esta definición:

THEN (instrucción a ejecutar o número de línea)

La instrucción a ejecutar puede ser una asignación de valor, un INPUT o una instrucción PRINT. No son aceptables, en cambio, otra instrucción IF, o un comando, como por ejemplo REM, CLE, NEW o LIST.

Ahora que comprendemos la teoría de la instrucción IF, llevémosla a la práctica.

Un ejercicio aritmético

Empleando nuestros conocimientos recién adquiridos, vamos a desarrollar un programa que pone un «menú» en la pantalla. Según lo que elige el usuario, este programa educacional llevará a cabo adiciones, sustracciones, multiplicaciones y divisiones.

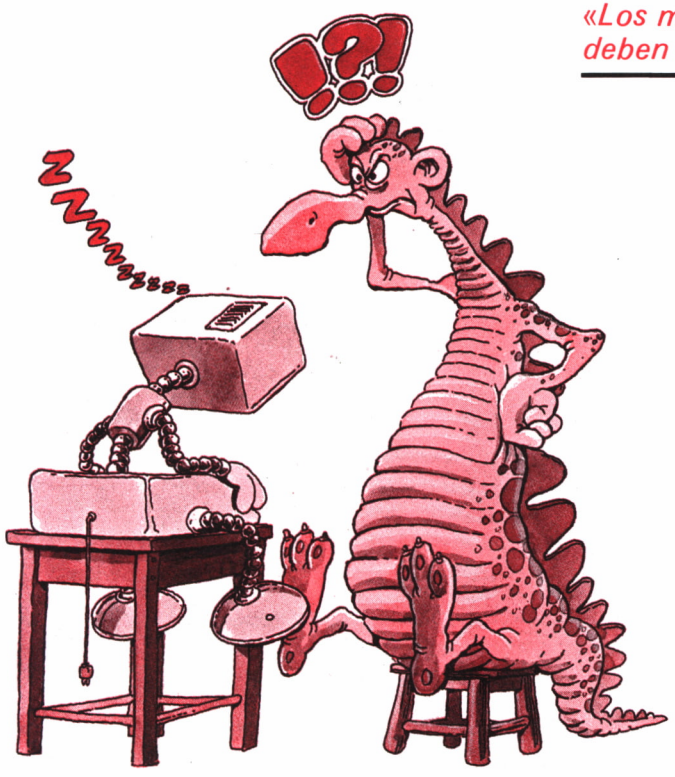
Aquí tenemos el diálogo que pensamos generar en la pantalla:

```
TE SALUDA EL PROFESOR-ORDENADOR
VOY A EVALUAR TUS CONOCIMIENTOS ARITMETICOS
QUE QUIERES PRACTICAR?
  -ADICION          (PULSA 1)
  -SUSTRACCION      (PULSA 2)
  -MULTIPLICACION   (PULSA 3)
  -DIVISION         (PULSA 4)
CUAL VAS A ELEGIR   (PULSA 1, 2, 3, ó 4)?: 3

-------------------------------------------

VALE. MULTIPLIQUEMOS.
CUANTO ES 2 POR 3: 6
CORRECTO. ENHORABUENA!
```

«Los mensajes en la pantalla deben de ser lo más claros posible.»



He aquí el programa que efectúa esto:

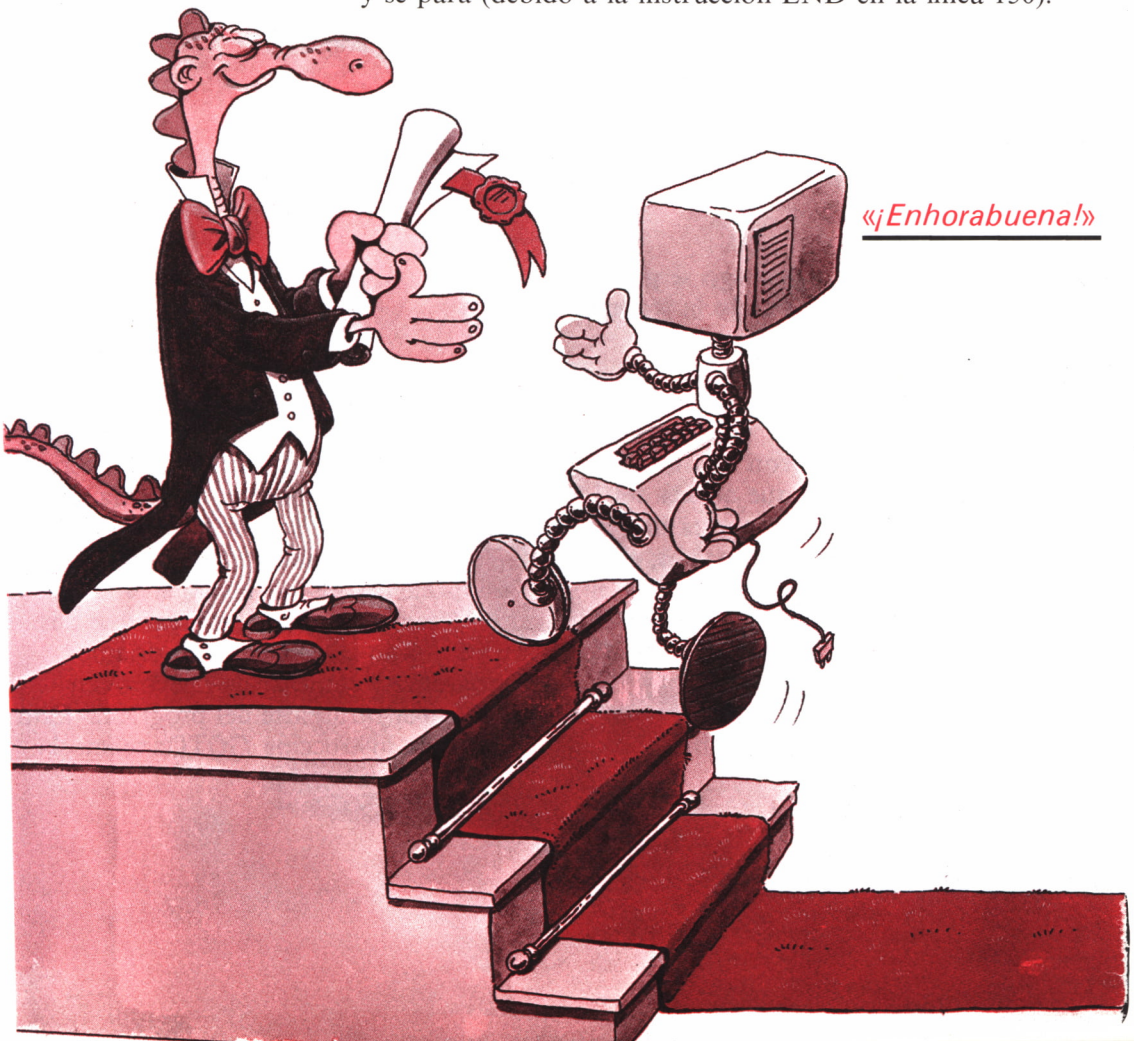
```
10  REM * * * ESTE PROGRAMA TE DA PRACTICA EN LA
    ARITMETICA * * *
20  PRINT "TE SALUDA EL PROFESOR-ORDENADOR"
30  PRINT "VOY A EVALUAR TUS CONOCIMIENTOS
    ARITMETICOS"
40  PRINT "QUE QUIERES PRACTICAR?"
50  PRINT " -ADICION          (PULSA 1)"
60  PRINT " -SUSTRACCION      (PULSA 2)"
70  PRINT " -MULTIPLICACION   (PULSA 3)"
80  PRINT " -DIVISION         (PULSA 4)"
90  INPUT "CUAL VAS A ELEGIR:"; ELECCION
100 IF (ELECCION = 1) THEN 200
110 IF (ELECCION = 2) THEN 300
120 IF (ELECCION = 3) THEN 400
130 IF (ELECCION = 4) THEN 500
140 PRINT "ELECCION INCORRECTA. TIENES QUE ELEGIR UN
    NUMERO ENTRE 1 y 4"
150 PRINT "ADIOS": END
190 REM-----ADICION-----
200 PRINT "VALE. SUMEMOS"
210 INPUT "CUANTO ES 4 + 7 :"; NUMERO
220 IF (NUMERO <> 11) THEN 600
230 PRINT "CORRECTO. ENHORABUENA!" : END
290 REM-----SUSTRACCION-----
300 PRINT "VALE. RESTEMOS"
310 INPUT "CUANTO ES 9 - 5 :"; NUMERO
320 IF (NUMERO<>4) THEN 600
330 PRINT "CORRECTO. ENHORABUENA!" : END
390 REM-----MULTIPLICACION-----
400 PRINT "VALE. MULTIPLIQUEMOS"
410 INPUT "CUANTO ES 2 por 3 :"; NUMERO
420 IF (NUMERO<>6) THEN 600
430 PRINT "CORRECTO. ENHORABUENA!" : END
490 REM-----DIVISION-----
500 PRINT "VALE. DIVIDAMOS"
510 INPUT "CUANTO ES 9 ENTRE 3 :"; NUMERO
520 IF (NUMERO<>3) THEN 600
530 PRINT "CORRECTO. ENHORABUENA!" : END
590 REM-----TERMINACION CASO DE ERRORES-----
600 PRINT "LO SIENTO. TE HAS EQUIVOCADO. ADIOS." : END
```

Por su tamaño el programa tiene un aspecto algo imponente, pero en realidad es bastante sencillo. Examinémoslo.

Las instrucciones 20 a 90 forman el «menú» en la pantalla. El programa comprueba la selección del usuario en las instrucciones de 100 a 130 (los paréntesis después de cada IF no son necesarios; se incluyen para facilitar la lectura del programa). Si el usuario escribe «1», entonces se cumple la condición de (ELECCION = 1), y la instrucción 200 es la siguiente en ejecutarse. Si el usuario escribiese algo que no fuera 1, 2, 3 ó 4, entonces la instrucción 140 se ejecuta y el programa emite el mensaje:

```
ELECCION INCORRECTA. TIENES QUE ELEGIR UN NUMERO
      ENTRE 1 y 4
ADIOS
>
```

y se para (debido a la instrucción END en la línea 150).



«¡Enhorabuena!»

En nuestro ejemplo anterior, escribimos «3». La línea 100 no se cumple, por lo que se ejecuta la línea 110 a continuación. A su vez, la línea 110 no se cumple, originando que se ejecute la línea 120 a continuación. La condición en esta línea se mantiene, ya que (ELECCION = 3) es cierto, haciendo que la próxima línea ejecutada sea la 400. El segmento de programa correspondiente aparece a continuación:

```
400  PRINT "VALE. MULTIPLIQUEMOS"
410  INPUT "CUANTO ES 2 por 3 :"; NUMERO
420  IF (NUMERO<>6) THEN 600
430  PRINT "CORRECTO. ENHORABUENA!" : END
```

En nuestro ejemplo, escribimos «6» como respuesta a la línea 410. Al ejecutarse la línea 420, (NUMERO < > 6) no se mantiene ya que NUMERO = 6 (recuerda que < > significa «no igual a»). De esta manera la siguiente línea que se ejecuta es la 430, y el programa responde con:

```
VALE. ENHORABUENA!
>
```

y luego se para, ya que la línea 430 contiene *dos* instrucciones, siendo la segunda:

```
: END
```

Al examinar el programa con más detalle, puede que des con una nueva fuente de frustración: si se escribe un número que no sea del 1 al 4 como respuesta al menú, o si se da una respuesta incorrecta para cualquiera de los problemas aritméticos, el programa se para bruscamente. Sería ideal que el programa continuase. Por ejemplo, sería bueno si el programa solicitara otra selección después de indicar que un número que no esté entre 1 y 4 no es aceptable. Podríamos volver al principio del programa y empezar de nuevo o, en general, trasladarnos a cualquier otra parte del programa. Esto es posible mediante el uso de la instrucción GOTO; veámosla.

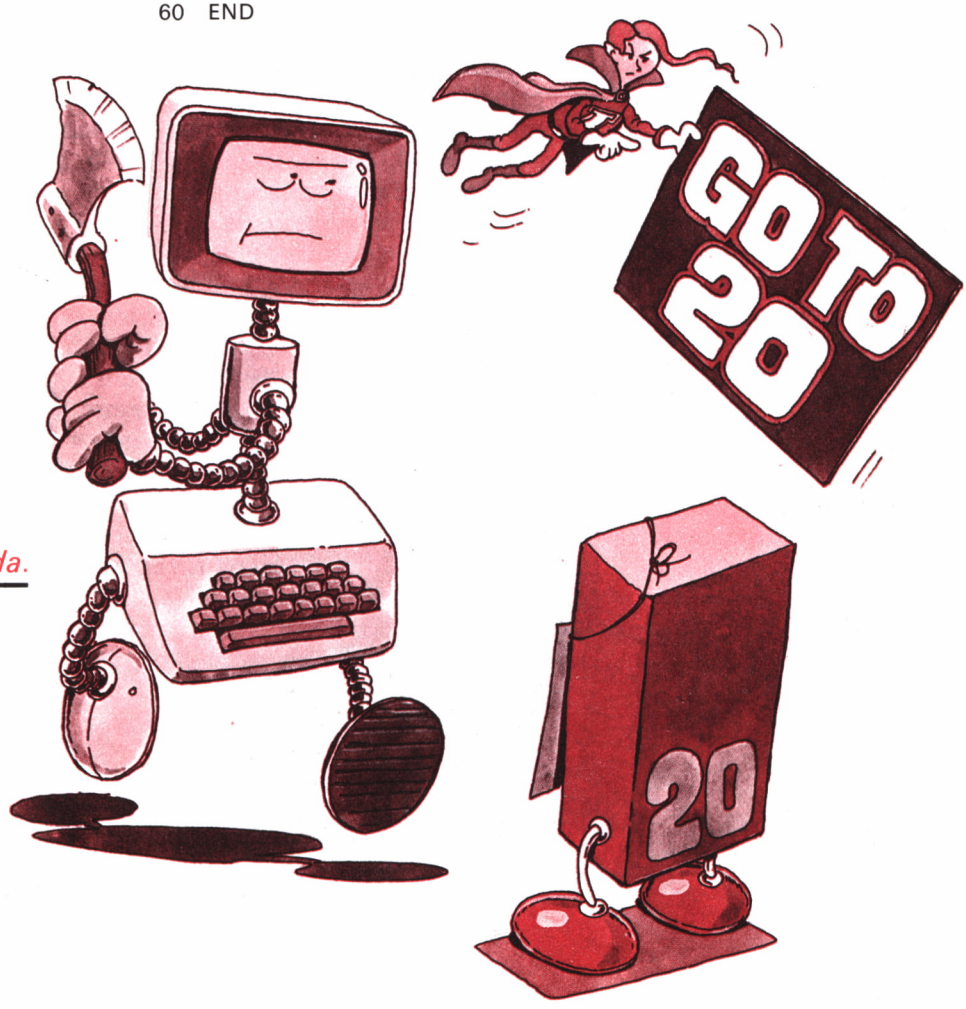
La instrucción GOTO

Se escribe la instrucción GOTO de la siguiente manera:

GOTO (número de línea)

lo que obliga a que el programa ejecute la línea especificada. Veamos un ejemplo:

```
10  PRINT "ESTE PROGRAMA RECONOCE EL NUMERO 1.
    ESCRIBE 0 PARA PARAR."
20  INPUT "ESCRIBE UN NUMERO:"; NUMERO
30  IF NUMERO = 1 THEN PRINT "UNO"
40  IF NUMERO = 0 THEN 60
50  GOTO 20
60  END
```

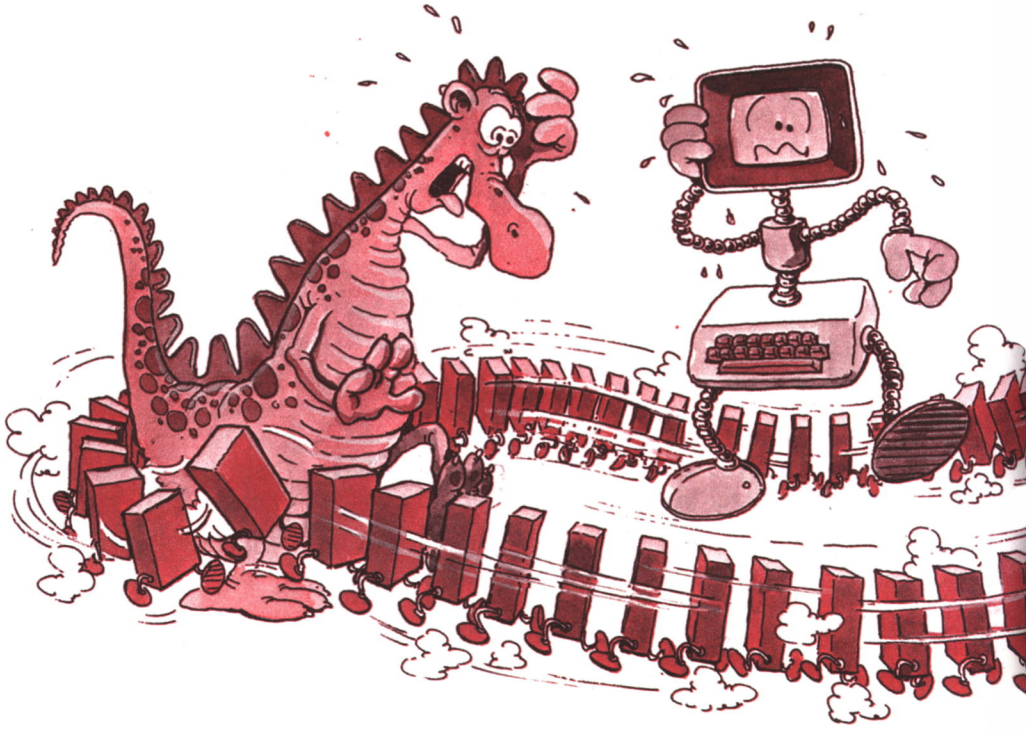


GOTO precipita la ejecución de la instrucción señalada.

Aquí tenemos una ejecución de ejemplo:

```
>RUN
ESTE PROGRAMA RECONOCE EL NUMERO 1. ESCRIBE 0 PARA
    PARAR.
ESCRIBE UN NUMERO:? 1
UNO
ESCRIBE UN NUMERO:? 5
ESCRIBE UN NUMERO:? 25
ESCRIBE UN NUMERO:? 1
UNO
ESCRIBE UN NUMERO:? 0
>
```

Cada vez que escribes el número 1 el programa lo reconoce e imprime UNO. Al escribir cualquier otra cosa, el número se ignora y el programa pide otro número. El programa vuelve al principio continuamente: esto se denomina *bucle*. Se dice que el programa vuelve sobre sí mismo (completa un ciclo). Si se escribe un 0, se detecta en la línea 40, la ejecución salta a la línea 60 y el programa se para.



Ahora quitemos la línea 40, quedándonos el programa:

```
10  PRINT "ESTE PROGRAMA RECONOCE EL NUMERO 1.
    ESCRIBE 0 PARA PARAR."
20  INPUT "ESCRIBE UN NUMERO:"; NUMERO
30  IF NUMERO = 1 THEN PRINT "UNO"
50  GOTO 20
60  END
```

Y una ejecución típica:

```
ESTE PROGRAMA RECONOCE EL NUMERO UNO. ESCRIBE 0
    PARA PARAR.
ESCRIBE UN NUMERO:? 2
ESCRIBE UN NUMERO:? 1
UNO
ESCRIBE UN NUMERO:? 5
ESCRIBE UN NUMERO:? 0
ESCRIBE UN NUMERO:?
```

«¡Para este bucle, que me mareo!»

Como el aprendiz del mago, hemos creado un problema terrible: ¡el programa jamás se parará! Esto constituye un error de programación muy común, llamado *bucle sin fin*. El programa podría seguir ejecutándose continuamente, pero no te preocupes, no hará ningún daño. Para parar el programa, has de pulsar las teclas designadas por el fabricante del intérprete para interrumpir un programa en BASIC (prueba con CTRL-C). Si no hallas ninguna solución, y no te acuerdas de lo que hay que hacer, apaga y enciende el ordenador. Recuerda, de todas maneras, que, si apagas el ordenador, perderás todo lo que hayas escrito hasta ese momento y que no hayas guardado en cassette o diskette. Nos esforzaremos para evitar esta situación desagradable programando una salida en cada programa de ahora en adelante.

Habiendo tenido nuestro primer contacto con la instrucción GOTO, volvamos un momento a nuestra definición de la instrucción IF y simplifiquémosla.

Vuelta a la instrucción IF

Recuerda que uno de los formatos de esta instrucción es el de:

IF (expresión lógica) THEN (número de línea)

Siendo el otro formato:

IF (expresión lógica) THEN (instrucción a ejecutar)

Aquí tenemos un ejemplo del primer formato:

```
IF NUMERO = 0 THEN 60
```

Esto es equivalente a:

```
IF NUMERO = 0 THEN GOTO 60
```

Dado que GOTO 60 es una instrucción de ejecución normal, podrás reconocer que el formato

```
THEN 60
```

es realmente una manera conveniente y más corta de escribir

```
THEN GOTO 60
```

Así que, en realidad, el formato general de la instrucción IF es, de hecho, aún más sencillo que en nuestra definición anterior:

IF (expresión lógica) THEN (instrucción a ejecutar)

Existe una simplificación más: el uso opcional de THEN antes de GOTO. Con la mayoría de versiones de BASIC, los formatos siguientes son todos equivalentes:

```
IF NUMERO = 0 THEN 100
IF NUMERO = 0 THEN GOTO 100
IF NUMERO = 0 GOTO 100
```

Ahora vamos a demostrar el uso de las instrucciones IF y GOTO mediante el ejemplo de varios programas.

Contar el número uno

En el capítulo 5 introdujimos la técnica del contador. Usémoslo ahora para contar el número de veces que se escribe el número 1 en el programa de la sección anterior. A continuación tenemos la versión modificada:

```
1   REM CONTADOR DEL NUMERO 1
10  PRINT "CONTARE LAS VECES QUE ESCRIBES EL NUMERO
    1"
20  PRINT "ESCRIBE 0 PARA PARAR"
30  SUMA = 0
40  INPUT "ESCRIBE UN NUMERO: "; NUMERO
50  IF NUMERO = THEN 100
60  IF NUMERO <> 1 THEN GOTO 40
70  SUMA = SUMA + 1
80  PRINT "UNO. TOTAL ACTUAL: "; SUMA
90  GOTO 40
100 END
```

La ejecución correspondiente sería:

```
CONTARE LAS VECES QUE ESCRIBES EL NUMERO 1
ESCRIBE 0 PARA PARAR
ESCRIBE UN NUMERO:? 10
ESCRIBE UN NUMERO:? 1
UNO. TOTAL ACTUAL: 1
ESCRIBE UN NUMERO:? 9
ESCRIBE UN NUMERO:? 5
ESCRIBE UN NUMERO:? 1
UNO. TOTAL ACTUAL: 2
ESCRIBE UN NUMERO:? 2
ESCRIBE UN NUMERO:? 1
UNO. TOTAL ACTUAL: 3
ESCRIBE UN NUMERO:? 410
ESCRIBE UN NUMERO:?
```

Examinemos el programa. Las líneas 10 y 20 ponen mensajes en la pantalla:

```
10  PRINT "CONTARE LAS VECES QUE ESCRIBES EL NUMERO
    1"
20  PRINT "ESCRIBE 0 PARA PARAR"
```

La instrucción 30 inicializa el contador SUMA en el valor de cero:

```
30 SUMA = 0
```

A continuación se recoge un número del teclado:

```
40 INPUT "ESCRIBE UN NUMERO: "; NUMERO
```

Si el número es cero, hemos terminado:

```
50 IF NUMERO = 0 THEN 100
```

donde 100 es la instrucción de END. Supongamos que el número introducido fuese un 10, a ver qué pasaría:

```
60 IF NUMERO<>1 THEN GOTO 40
```

Si el número no es igual a 1, saltamos hacia atrás a la línea 40 para volver a pedir un número. Si el número es 1, seguimos adelante:

```
70 SUMA = SUMA + 1
```

La variable-contador SUMA se incrementa en uno. Recuerda el significado de una instrucción de asignación. La línea 70 se interpreta de la siguiente manera:

SUMA recibe el valor nuevo de (valor antiguo de SUMA) + 1

En esta ocasión la variable SUMA recibe el valor de 0 + 1 = 1. La próxima instrucción es:

```
80 PRINT "UNO. TOTAL ACTUAL: "; SUMA
```

Entonces el programa hace un bucle y vuelve a la línea 40, solicitando otro número:

```
90 GOTO 40
```

Vuelta al ejercicio aritmético

Acuérdate del programa de ejercicios aritméticos que desarrollamos al principio de este capítulo. Lamentamos el hecho de que fuera demasiado sencillo y no podía hacer un «reciclaje». Ahora sí conocemos los medios necesarios para hacer que se repita.

Dado que el programa es bastante largo, veamos únicamente la sección que viene a cuento. Primero, tenemos la sección que solicita que el usuario seleccione un número entre 1 y 4:

```
90   INPUT "CUAL VAS A ELEGIR:"; ELECCION
100  IF (ELECCION = 1) THEN 200
110  IF (ELECCION = 2) THEN 300
120  IF (ELECCION = 3) THEN 400
130  IF (ELECCION = 4) THEN 500
140  PRINT "ELECCION INACEPTABLE. TIENES QUE ELEGIR UN
     NUMERO ENTRE 1 y 4"
150  PRINT "ADIOS" : END
```

Vamos allá con nuestra mejora:

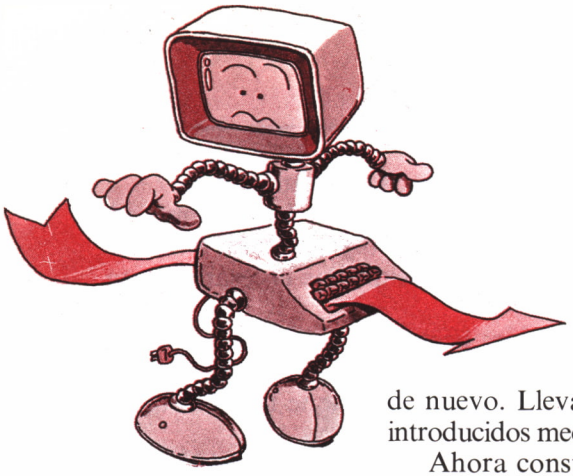
```
150  GOTO 90
```

Ya está. Verifícalo por tu cuenta.

Ahora bien, también nos gustaría que el programa nos presentara más de una pregunta aritmética, por ejemplo, 10 preguntas. Esto lo podemos conseguir mediante la adición de instrucciones de GOTO y un contador.

Comprobar los datos

Los ejemplos que acabamos de ver nos demuestran una regla muy importante a la hora de escribir un programa: cuando se pida datos que se van a suministrar desde el teclado, no se puede suponer que se vayan a suministrar correctamente. El usuario podría pulsar una tecla errónea, con o sin propósito. Para evitar el comportamiento extraño o erróneo de un programa, siempre hay que comprobar *(validar)* el suministro de datos (input). Si la información que se escribe en el teclado no es válida, has de generar un mensaje cortés y solicitar los datos



«Ten cuidado con los datos que me introduces.»

de nuevo. Llevaremos a cabo esta comprobación de los datos introducidos mediante INPUT en la mayoría de nuestros programas.

Ahora construyamos dos programas completos que sirven para tomar decisiones autónomas.

Conversión de kilometraje

En el capítulo 3 aprendimos a hacer una conversión sencilla de millas a kilómteos. A continuación tenemos una manera para automatizar esta conversión:

```
10   REM * * * CONVERSION DE KILOMETRAJE * * *
20   REM
30   PRINT "CONVIERTO MILLAS A KILOMETROS"
40   PRINT "ESCRIBE 0 PARA PARAR"
50   INPUT "CUANTAS MILLAS "; MILLAS
60   IF MILLAS = 0 THEN GOTO 100
70   KM = MILLAS * 1.6
80   PRINT MILLAS; " MILLAS =  "; KM; " KILOMETROS"
90   GOTO 50
100  END
```

La ejecución correspondiente sería:

```
CONVIERTO MILLAS A KILOMETROS
ESCRIBE 0 PARA PARAR
CUANTAS MILLAS? 7
    7 MILLAS = 11.2 KILOMETROS
CUANTAS MILLAS? 10
    10 MILLAS = 16 KILOMETROS
CUANTAS MILLAS? 0
```

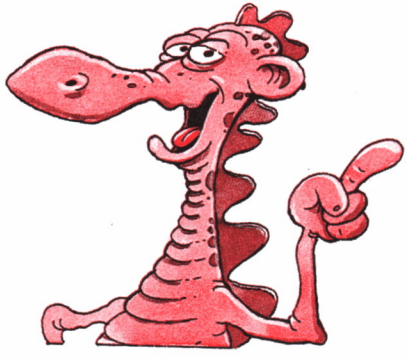
Adivina el número

Vamos a mejorar y completar el programa que hicimos antes y que permitía jugar con el ordenador intentando averiguar un número generado al azar. El programa nos invita a averiguar el número que ha pensado en el menor número de intentos. El listado completo es el siguiente:

```
10  REM * * * ADIVINA EL NUMERO * * *
20  CLS
30  REM * * PRESENTACION * *
40  INPUT "COMO TE LLAMAS"; NOMBRE$
50  PRINT "HOLA "; NOMBRE$;", VOY A PENSAR UN NUMERO
    DEL 1 AL 100"
60  PRINT "TIENES QUE INTENTAR AVERIGUARLO LO ANTES
    POSIBLE"
70  REM
80  REM * * * GENERACION ALEATORIA DEL NUMERO * * *
90  N = INT (RND (1) * 100) : NI = 0
100 REM * * * LA INSTRUCCION ANTERIOR HACE QUE "N"
    TOME UN VALOR CUALQUIERA ENTRE 1 y 100 * * *
110 REM
120 REM * * * ENTRADA DE LOS INTENTOS * * *
130 PRINT: INPUT "QUE NUMERO CREES QUE ES "; I
140 NI=NI+1
150 REM * * * RUTINA DE COMPARACION * * *
160 IF I<0 OR I>100 THEN GOTO 130
170 IF I = N GOTO 260
180 IF I<N THEN R$ = "MAYOR"
190 IF I>N THEN R$ = "MENOR"
200 REM
210 REM * * * MENSAJE DE ERROR Y NUMERO DE INTENTOS
    * * *
220 REM
230 PRINT R$; "...INTENTALO OTRA VEZ. - INTENTO "; NI; "-"
240 GOTO 130
250 REM
260 REM * * * MENSAJE DE FELICITACION * * *
270 PRINT "BRAVO "; NOMBRE$:", LO HAS ACERTADO EN ";
    N; " INTENTOS"
280 REM
290 REM * * * MODULO FINAL * * *
300 PRINT "VAMOS A JUGAR OTRA VEZ..."
310 PRINT: INPUT "PARA SEGUIR PULSA "S" "; S$
320 IF S$="S" THEN GOTO 10
330 END
```

Nota: Si tu intérprete de BASIC no admite nombres de variables largos puedes cambiar NOMBRE$ por NB$ u otro

similar. (Ten cuidado de no usar nombres que contengan palabras reservadas, ya que tal vez no los admita el intérprete.) Aunque te parezca largo, este programa es muy sencillo, y hay cosas interesantes en él. Observa, por ejemplo, la forma en que se comprueba en la línea 160 que el número que has introducido está dentro del rango permitido, asegurándonos que es mayor que 0 y menor de 100. Fíjate también en que solamente hay un mensaje de error (línea 230) donde sustituimos la variable R$ por «MAYOR» o «MENOR», según sea el resultado de la comparación efectuada en las líneas 180 y 190. En la línea 140 se incrementa en una unidad la variable NI (Número de Intentos) cada vez que cometes un fallo. La instrucción PRINT de la línea 230 escribirá cada vez en la pantalla el número del intento, utilizando esa variable NI.



Resumen

Mediante el uso de las instrucciones IF y GOTO hemos visto la manera de construir programas que pueden comprobar valores y tomar decisiones en consecuencia. También hemos estudiado la manera de crear bucles dentro de los programas, de manera que una parte de un programa se pueda repetir indefinidamente. De forma complementaria, hemos aprendido a validar los datos que se suministran al ordenador a través del teclado. Ahora hemos aprendido todos los conocimientos básicos necesarios para escribir programas normales en BASIC, habiendo examinado a la vez varios ejemplos para ilustrar lo que íbamos viendo. Ahora vamos a hacer que nuestros programas sean más cómodos de escribir.

Debido a la frecuencia y la importancia de los bucles y la automación en los programas, el lenguaje BASIC nos ofrece más facilidades a través de instrucciones adicionales, las cuales discutiremos en el próximo capítulo.

Ejercicios

6.1. ¿Qué utilidad tiene la instrucción IF?

6.2. ¿Qué resultado producirá:

```
10 INPUT RESPUESTA$
20 IF (RESPUESTA$ = "SI") THEN PRINT "GRACIAS"
30 IF (RESPUESTA$ = "NO") THEN PRINT "QUE PENA!"
40 PRINT "SI O NO" : GOTO 10
```

Si el resultado es ilógico, sugiere alguna alternativa.

6.3. Verificar si son válidas las siguientes expresiones lógicas:

a) A = 4
b) B = 2 OR C = 3
c) A > 5
d) 5 > A
e) 1 > 2
f) SUMA > NUMERO
g) LETRA$ = "A"

6.4. Verificar si es válida la siguiente línea:

```
10 IF A = 5 THEN IF B = 2 THEN 18
```

6.5. ¿Qué es un bucle en la programación?

Cómo las

7

Con la ayuda de las instrucciones IF y GOTO, podemos repetir un segmento de un programa todas las veces que deseemos. El segmento de programa correspondiente se denomina bucle; la mayoría de los programas emplean estos bucles. En este capítulo estudiaremos mejores técnicas para crear los bucles, desarrollando a la vez programas sofisticados que automatizan ciertas tareas.

Comenzaremos este capítulo con un repaso de la técnica IF/GOTO para generar un bucle. Luego pasaremos a la introducción de una nueva instrucción, la de FOR...NEXT, diseñada especialmente para facilitar la creación de bucles. Es una instrucción que emplearemos a fondo en nuestros programas.

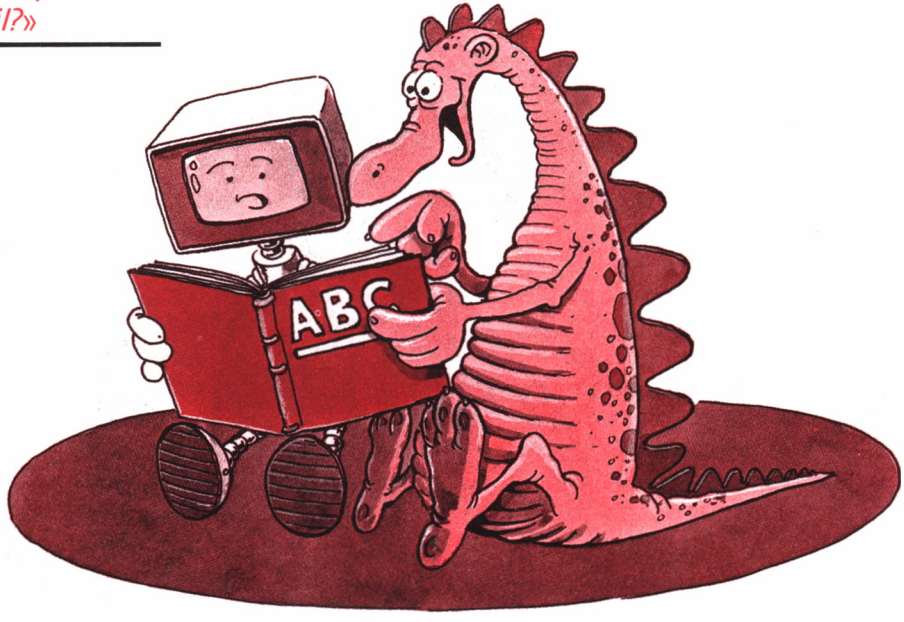
automatizar repeticiones

La técnica IF/GOTO

Vamos a empezar con el análisis de un programa que automatiza un bucle mediante la técnica IF/GOTO. A la vez que examinamos el programa aprovecharemos para señalar ciertas características que son comunes a todos los bucles. Por ejemplo, analizaremos el uso de la variable-contador, el incremento, la inicialización y la comprobación previa a la salida. A continuación tenemos el programa: contabiliza la suma de los primeros diez números enteros.

```
1   REM * * * SUMA DE LOS PRIMEROS DIEZ NUMEROS
    ENTEROS * * *
10  SUMA = 0
20  I = 1
30  SUMA = SUMA + I
40  I = I + 1
50  IF I = 11 THEN 70
60  GOTO 30
70  PRINT "LA SUMA DE LOS PRIMEROS 10 NUMEROS
    ENTEROS ES: "; SUMA
80  END
```

«¿Qué te parece si pasamos a algo más difícil?»



En este programa se usan dos variables: SUMA e I. La variable SUMA acumula la suma de los primeros diez números enteros según los vamos añadiendo; es el equivalente de un subtotal en una calculadora. La I es el número que se está añadiendo a la SUMA.

Recuerda que una variable ha de tener un valor la primera vez que se emplea. Por tanto, antes de que empleemos las variables SUMA e I en una fórmula, hemos de establecer su valor en lo que llamamos un valor *inicial* (0 y 1, respectivamente). Esto se logra con las instrucciones en líneas 10 y 20, que sirven de instrucciones de *inicialización*.

«¿Te acuerdas de mí?»

La siguiente línea es:

```
30  SUMA = SUMA + I
```

Esta instrucción añade el valor actual del número I al subtotal actual de SUMA. Cuando se ejecuta por primera vez esta instrucción, el valor de SUMA es 0 y el de I es 1, con el resultado de asignar el valor de 0 + 1 = 1 a la variable SUMA. Es decir, como consecuencia de la ejecución de esta instrucción podemos decir que la variable SUMA contiene el valor de 1.

La siguiente línea es:

```
40  I = I + 1
```

El valor actual de I es 1, por lo que el efecto de esta instrucción es el de dar a I el nuevo valor de 2. Este incremento constituye la técnica de la variable-contador: con el fin de generar el siguiente número entero el valor de I se aumenta en 1. Al mismo tiempo, el valor de I indica la cantidad de números que se han añadido hasta el momento. O sea, la I se emplea como número entero actual y como contador.

«¡Has caído de nuevo!»

Así que lo único que nos queda por hacer es volver a la línea 30 y continuar sumando enteros:

```
50  GOTO 30
```

¡Falso! Este programa —en teoría— nunca se parará (en realidad se interrumpirá una vez que el valor de SUMA supere el máximo que permite tu intérprete). Obviamente no es la situación que deseamos: queremos que el programa se pare después de ejecutar el bucle diez veces. Tenemos que introducir una comprobación:

```
50  IF I = 11 THEN 70
```

Cuando la variable I alcanza el valor de 11, la línea 70 se ejecuta y el programa se para. Este proceso se llama *salida* del bucle. Asegurémonos que el número 11 (y no 10) es, en efecto, el que se debe usar en la línea 50. Si escribimos:

```
50  IF I = 10 THEN 70
```

No funciona. Una vez que la I llegue al valor de 10, la variable SUMA contiene únicamente la suma de 1 a 9. El bucle ha de ejecutarse una vez más.

Recuerda que cada bucle contiene un contador, cuyo valor has de comprobar cuidadosamente, ya que determina la salida del bucle. En nuestro ejemplo, siempre y cuando el valor de I no sea igual a 11, el bucle se repetirá:

```
60  GOTO 30
```

Cuando la I alcanza el valor de 11, los primeros diez números ha sido sumados. Esto se debe a que en nuestro programa la adición (SUMA = SUMA + I) ocurre antes del incremento (I = I + 1). Las últimas dos líneas del programa constituyen la *salida* del bucle:

```
70  PRINT "LA SUMA DE LOS PRIMEROS 10 NUMEROS
    ENTEROS ES: "; SUMA
80  END
```

Una ejecución del programa produce el resultado:

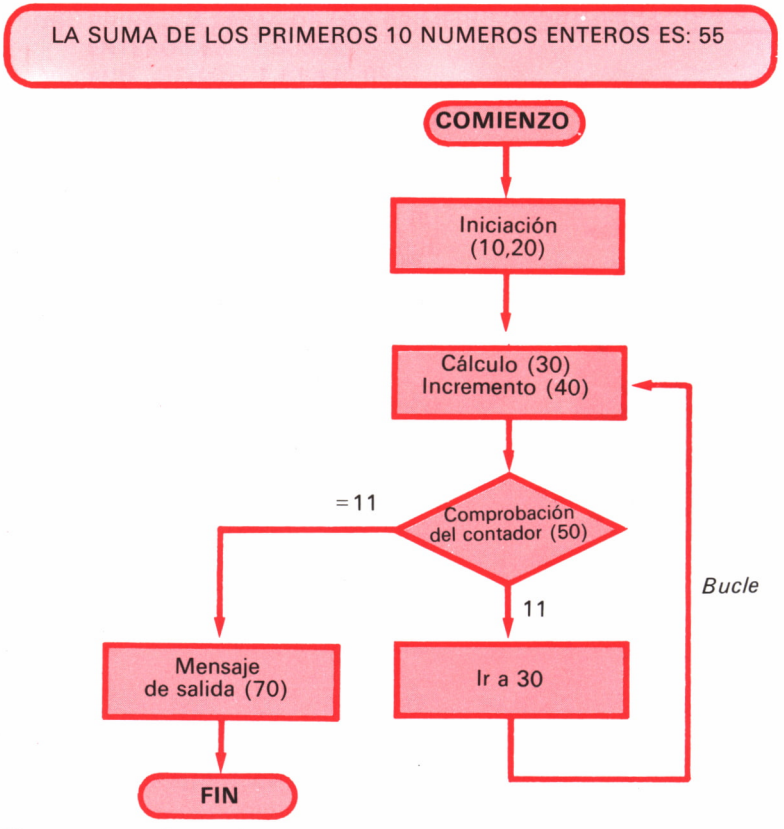


Figura 7.1. Flujo de control en el programa de sumas de enteros.

La figura 7.1 muestra el flujo de control en el progrrama. Los números entre paréntesis son los números de línea. Este diagrama se llama *diagrama de flujo*. Cubriremos en detalle el tema de los diagramas de flujo en el capítulo 8. De momento es suficiente seguir la organización general del programa: inicialización, cálculo e incremento, comprobación y salida. Todos los segmentos de programa con bucles llevan a cabo estos procesos.

Las alternativas

Juguemos un poco con nuestro programa de sumar números enteros para fijar nuestros conocimientos de la programación. De esta manera podremos demostrar las muchas alternativas que se pueden barajar en la construcción de un programa. Por ejemplo, en la línea 50 podríamos haber escrito:

```
50  IF I > 10 THEN 70
```

y el resultado sería el mismo (cuando la I alcanza el valor de 11, es mayor que 10). También hubiéramos podido escribir:

```
40  IF I = 10 THEN 70
50  I = I + 1
```

en vez de:

```
40  I = I + 1
50  IF I = 11 THEN 70
```

Con este cambio se comprueba el valor de I antes de que se incremente. Toma nota de que en esta ocasión se comprueba el valor de 10 en la variable I (en lugar de 11).

También podíamos escribir:

```
50  IF I < 11 THEN 30
60  REM
```

Puedes determinar por tu cuenta que todas estas versiones sí funcionan. Son, todas ellas, alternativas válidas y equivalentes. Hasta un programa tan corto como éste de sumas se puede escribir de más de una manera, todas ellas equivalentes. No existe una manera única de escribir un programa. Al igual que en el lenguaje hablado, en el de ordenador puede expresarse el mismo concepto de muchas maneras distintas.

Este programa corto ha servido para demostrar el uso de un bucle conjuntamente con la variable-contador. Igualmente hemos podido examinar las fases típicas de este tipo de programa: inicialización, cálculo, incremento, comprobación y salida. En vista del uso frecuente que tienen los bucles en los programas, el BASIC ha sido provisto de una instrucción especial para facilitar su utilización: la instrucción FOR... NEXT.

La instrucción FOR... NEXT

Gran parte de la programación necesaria para realizar un bucle se encuentra automatizado mediante la instrucción FOR... NEXT. Explicaremos su uso y modo de operación a través de ejemplos concretos.

A continuación veremos una manera de reconstruir nuestro programa de adición, empleando esta nueva instrucción:

```
1   REM * * * ADICION DE ENTEROS - VERSION 2 * * *
10  SUMA = 0
20  FOR I = 1 TO 10
30  SUMA = SUMA + I
40  NEXT I
50  PRINT "LA SUMA DE LOS 10 PRIMEROS ENTEROS
    ES: "; SUMA
60  END
```

Fíjate en que este programa tiene dos instrucciones menos que la primera versión. Es más corta y más fácil de leer. Veámoslo en detalle. La primera instrucción que se puede ejecutar inicializa el valor de SUMA en 0:

```
10  SUMA = 0
```

La próxima instrucción es la de FOR:

```
20  FOR I = 1 TO 10
```

Esta instrucción tiene varias funciones:

— Señala el principio del bucle automático (es aquí donde empieza el bucle).

— Especifica que la I (la variable-contador) empieza con el valor inicial de 1 cuando primero se ejecuta esta instrucción. Esto elimina la necesidad de una instrucción de inicialización separada para la variable I.

— El valor de I se incrementa en 1 (hasta un valor máximo de 10) cada vez que la instrucción se reactiva mediante la instrucción complementaria NEXT. Se realiza una comprobación automática y cuando la I tiene un valor superior a 10, el bucle se deja de ejecutar, ejecutándose en su lugar la instrucción que sigue a NEXT (lo que constituye la salida del bucle).

El cuerpo principal del bucle contiene simplemente la acumulación de la suma:

```
30  SUMA = SUMA + 1
```

La instrucción NEXT:

```
40  NEXT I
```

señala el final del bucle y causa la reactivación de la FOR. Ocupa el lugar de dos instrucciones en la versión anterior:

```
40  I = I + 1
60  GOTO 30
```

Cada vez que se ejecuta la instrucción NEXT I, el programa salta al principio del bucle, o sea, la instrucción FOR. Al activarse la FOR:

— Se incrementa el valor de I en 1.

— El valor nuevo de I se compara automáticamente con el número 10.

Siempre que valor de I no sea superior a 10, la ejecución continúa. La repetición del bucle se interrumpe cuando el valor de I sea igual a 10 y se da con la instrucción NEXT. En este momento, la salida se efectúa y se ejecuta la línea 50 (después de NEXT). Esta secuencia se detalla en la figura 7.2 (diagrama de flujo), que demuestra que la instrucción FOR automatiza tres tareas:

— La inicialización de la variable-contador (se establece el valor inicial de I en 1).

— El incremento de la variable-contador (la I se incrementa en 1 cada vez).

— La comprobación de la variable-contador, comparándolo con un valor máximo (la I se compara con el número 10).

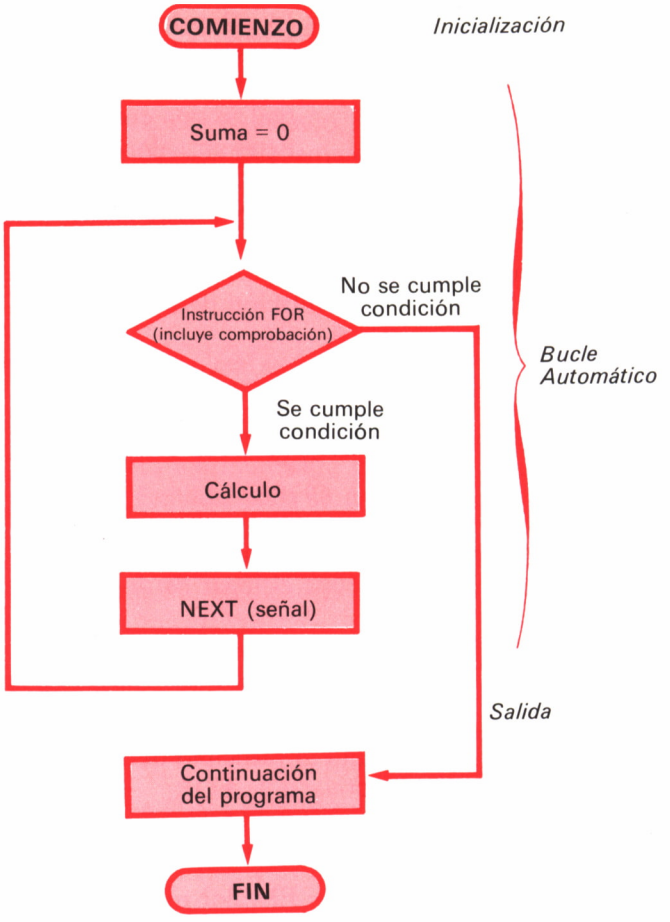


Figura 7.2. Repetición automática con FOR... NEXT.

La instrucción NEXT simplemente señala el final del bucle y ocasiona una instrucción de «GOTO la instrucción FOR». Después de usar la instrucción FOR unas cuantas veces, podrás apreciar la forma en la que simplifica el diseño de los bucles y clarifica el programa.

La instrucción FOR... NEXT resulta cómoda. No tienes que usarla, pero probablemente encontrarás que es muy valiosa. En general, cuanto más claro sea el programa, más se reducen las posibilidades de cometer errores. Pasamos ahora a algunos ejemplos prácticos para demostrar el uso de la instrucción FOR... NEXT y el uso de los bucles automáticos.

Suma de N números enteros

Vamos a calcular la suma de un número N de enteros. Esta vez el usuario especifica el valor de N en el teclado. Aquí tienes el programa:

```
10  REM * * SUMA DE NUMEROS ENTEROS * *
20  SUMA = 0
30  INPUT "SUMARE UN NUMERO N DE ENTEROS. ESCRIBE
    N:"; N
40  FOR I = 1 TO N
50  SUMA = SUMA + I
60  NEXT I
70  PRINT "LA SUMA DE"; N; "NUMEROS ENTEROS ES"; SUMA
80  END
```

Una ejecución de prueba da el resultado siguiente:

```
SUMARE UN NUMERO N DE NUMEROS ENTEROS. ESCRIBE N:? 5
LA SUMA DE 5 NUMEROS ENTEROS ES 15
```

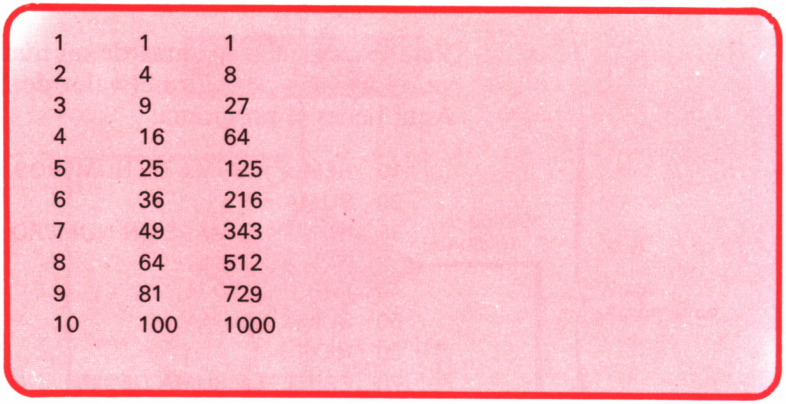
Comprenderás el programa con facilidad. Esta vez hacemos un bucle de 1 a N, donde N se suministra desde el teclado (línea 30). Puedes mejorar el programa validando la entrada de datos: N ha de ser mayor a 1. El intérprete de BASIC verificará automáticamente que N sea un número entero cuando ejecuta la instrucción FOR. Intenta engañarlo.

Tablas de valores

Mediante la potente instrucción FOR... NEXT, demostraremos con qué facilidad se pueden automatizar los cálculos e imprimir tablas de valores. Aquí tienes una tabla de los cuadrados (un número multiplicado por sí mismo) y los números elevados a la tercera potencia (multiplicado por sí mismo dos veces):

```
10  REM TABLA DE NUMEROS ELEVADOS A LA SEGUNDA Y
    TERCERA POTENCIAS
20  REM PARA LOS PRIMEROS DIEZ ENTEROS
30  FOR I = 1 TO 10
40  PRINT I, I ^2,I ^3
50  NEXT I
60  END
```

Con el siguiente resultado:



```
1     1     1
2     4     8
3     9     27
4     16    64
5     25    125
6     36    216
7     49    343
8     64    512
9     81    729
10    100   1000
```

Examinemos más de cerca la instrucción número 40:

```
40  PRINT I,I^2, I^3
```

Los símbolos I ^ 2 significan la variable I elevada a la segunda potencia; es decir, I * I. Por ejemplo, si I = 2, entonces I ^ 2 es igual a 2 * 2 = 4. De la misma manera, I ^ 3 significa la I elevada a la tercera potencia; o sea, I * I * I. Si la I = 4, entonces I ^ 3 = 4 * 4 * 4 = 64.

Fíjate bien en nuestro empleo en este caso de la coma en la instrucción PRINT, que consigue que los resultados se alineen en columnas según se vayan listando. La coma obliga a que se realice un espaciado automático (o *tabulación*) en la línea, siendo el espaciado exacto función del intérprete de BASIC del que dispongas. Por ejemplo, podría ser de unos 14 caracteres.

Para que te sirva de práctica, podrías volver a escribir el programa para que te diera los cuadrados y los cubos de los primeros N enteros, donde el valor de N se lee del teclado. Aprendimos a hacer esto en la sección anterior.

Líneas de estrellitas

Este programa sencillo imprime un número N de estrellitas, donde el valor de N se especifica desde el teclado:

```
10  REM * LINEAS DE ESTRELLITAS *
20  PRINT "VISUALIZARE EN PANTALLA UNAS LINEAS DE
    ESTRELLITAS"
30  INPUT "DIME CUANTAS LINEAS :"; N
40  REM N ES EL NUMERO DE LINEAS A IMPRIMIR
50  FOR I = 1 TO N
60  PRINT "**********************************************************"
70  NEXT I
80  END
```

Una ejecución modelo del programa sería:

```
VISUALIZARE EN PANTALLA UNAS LINEAS DE ESTRELLITAS
DIME CUANTAS LINEAS :? 6
**********************************************************
**********************************************************
**********************************************************
**********************************************************
**********************************************************
**********************************************************
```

Repetimos, cada vez que el usuario suministra un dato desde el teclado, es aconsejable validarlo para evitar resultados inesperados en el programa. Esperamos que la persona que usa este programa introduzca un número positivo. Supongamos además que quieres que no sean más de 20 líneas de estrellitas. Comunicarías esto al usuario mediante una instrucción PRINT apropiada, empleando asimismo una instrucción de validación parecida a:

```
IF (N<1) OR (N>20) GOTO 20
```

Los bucles avanzados

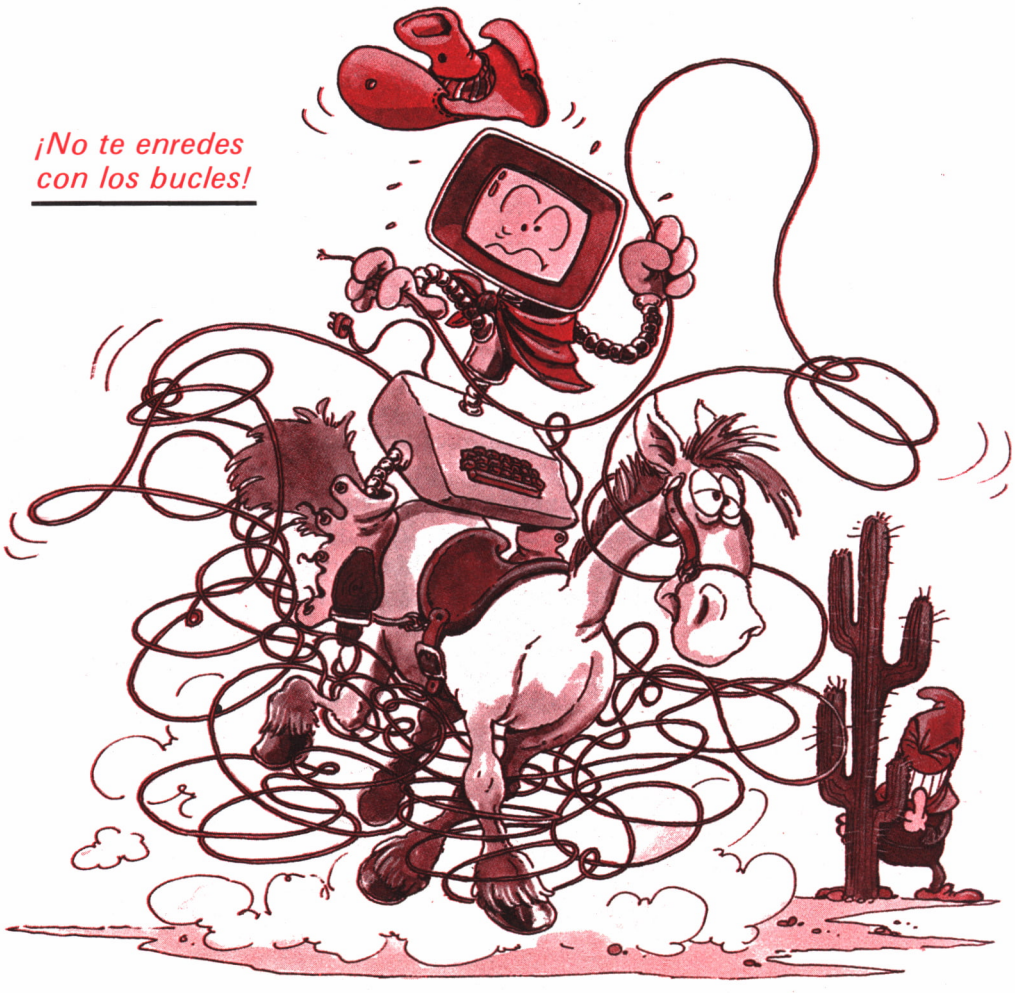
La instrucción FOR... NEXT ofrece dos características avanzadas que aún nos falta por explorar:

— Puedes fijar los incrementos del contador en el valor de cualquier número entero, como por ejemplo 2, 3, 4, o incluso −1 (en lugar de incrementar por el valor de 1 únicamente). Esta característica se denomina del *paso variable*.

— Puedes crear un bucle dentro de otro, lo cual se llama *bucle anidado*, o *bucle enclavado*.

Exploremos, pues, estas dos técnicas.

¡No te enredes con los bucles!



Paso variable

Aquí tenemos un ejemplo de paso variable:

```
FOR I = 1 TO 5 STEP 2
```

Por cada vez que se vuelve al principio del bucle, el valor de I se verá incrementado en 2. También sería posible escribir:

```
FOR I = 10 TO - 5 STEP - 1
```

usando un *paso* (step) *negativo* como incremento. Debido a que el límite superior para la variable de contador (−5 en este caso) es menor que el valor inicial (10), constituye un «paso negativo» para el intérprete. El valor de I se disminuirá en 1 cada vez. El primer valor asignado a I será el de 10, a continuación el de 9, el siguiente de 8, etc. El último será el de −5. Dicho de otra manera, el valor de I será igual a cada valor siguiente, uno por cada ciclo del bucle: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, −1, −2, −3, −4, −5. El paso negativo es una facilidad cómoda que puede ser interesante emplear en un momento determinado.

Tu variable puede variar sus pasos.

Los bucles anidados

La técnica del bucle anidado es una facilidad tan importante como potente que se emplea para automatizar procesos complejos. Un bucle anidado resulta de la colocación de un grupo de instrucciones FOR... NEXT dentro de un bucle; o sea, un bucle completo dentro de otro bucle.

En general, es posible usar todas las instrucciones que uno quiera entre las instrucciones FOR y NEXT. Cabe, pues, que se quiera colocar hasta otro bucle entre estas instrucciones. Cuando éste sea el caso, el segundo bucle se llama bucle anidado. Se aclara este concepto en la figura 7.3.

Notarás que el uso de los bucles anidados hace que el programa sea más difícil de leer. Como remedio, sugerimos que emplees la *sangría,* otra técnica útil para la clarificación de la estructura de un programa. La figura 7.4 muestra una versión sangrada de la figura 7.3.

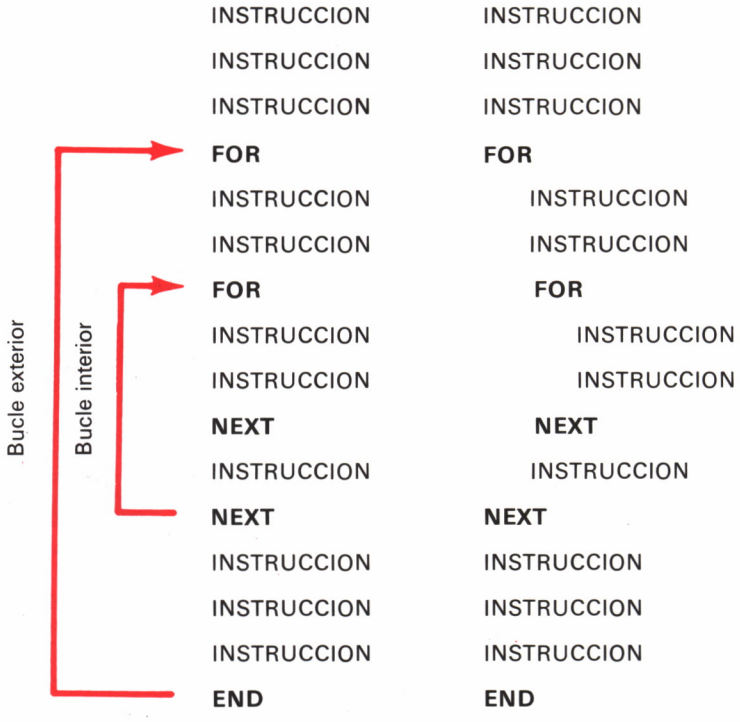
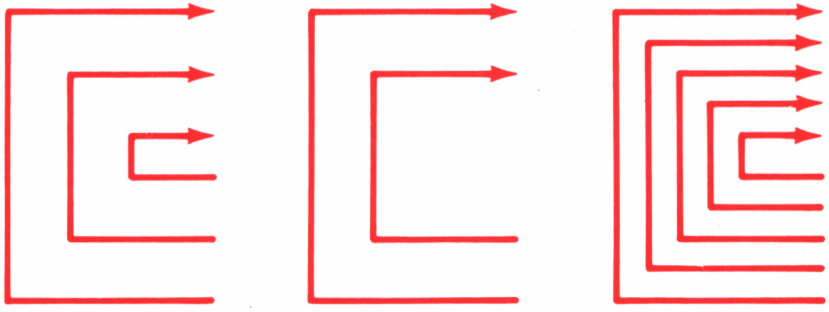


Figura 7.3. Bucle anidado.

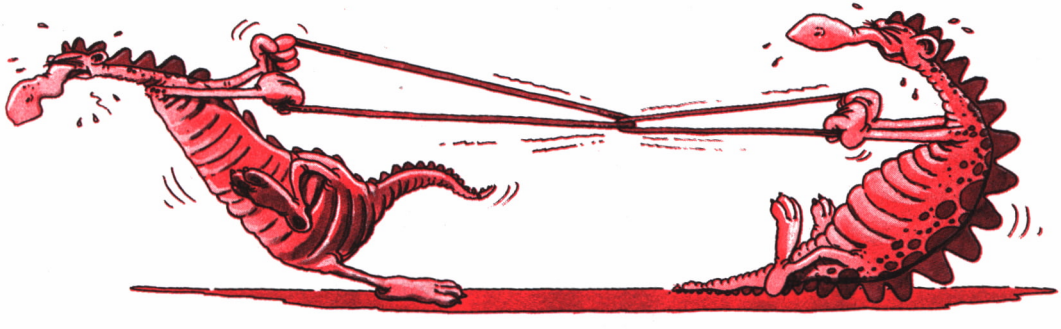
Figura 7.4. Programa sangrado.

Se pueden anidar los bucles hasta el nivel máximo admitido por el intérprete o la misma capacidad de memoria de tu equipo. En cambio, no está permitido cruzar los bucles. Los siguientes bucles son admitidos:

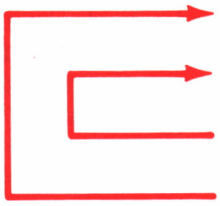


Estos bucles, en cambio, no lo son:

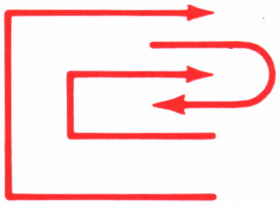
¡No mezcles los bucles!



Además, no puedes saltar (o sea, usar un GOTO) desde un lugar interior del bucle exterior a un lugar dentro del bucle interior:



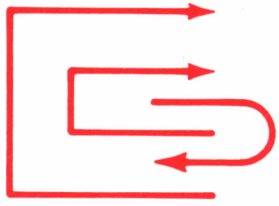
Anidado correcto



Salto ilegal
(con IF o GOTO)

Sin embargo, no presenta problemas saltar desde dentro del bucle interior:

Salto correcto



Salto correcto

A continuación veremos un ejemplo de bucle anidado. El programa muestra una aceleración del tiempo, visto en horas y minutos, en un reloj simulado:

```
10  REM * * * RELOJ SIMULADO * * *
20  FOR HORA = 0 TO 23
30    FOR MINUTO = 0 TO 59
40      PRINT "SON LAS "; HORA; "HORAS Y ";
          MINUTO; "MINUTOS"
50    NEXT MINUTO
60  NEXT HORA
70  PRINT "FINAL DEL DIA"
80  END
```

La ejecución de este programa da el resultado siguiente (visto parcialmente):

```
SON LAS 0 HORAS Y 0 MINUTOS
SON LAS 0 HORAS Y 1 MINUTOS
SON LAS 0 HORAS Y 2 MINUTOS
SON LAS 0 HORAS Y 3 MINUTOS
SON LAS 0 HORAS Y 4 MINUTOS
  . . .
SON LAS 0 HORAS Y 59 MINUTOS
SON LAS 1 HORAS Y  0 MINUTOS
```

Otras características

Hemos de añadir un comentario final. Los valores y las expresiones decimales generalmente son admisibles en una instrucción FOR... NEXT. Por ejemplo, es posible escribir:

```
FOR MEDIDA = 0.1 TO 13.5 STEP 0.2
FOR NUMERO = N TO (N * 2) STEP 1
```

Sin embargo, no recomendamos que se emplee esta técnica, por lo que no discutiremos estas características avanzadas en este capítulo.

Resumen

Los bucles se emplean extensamente para automatizar la repetición de un segmento de programa, lo cual se consigue mediante la instrucción FOR... NEXT en el lenguaje BASIC. En la mayoría de los casos, la instrucción FOR... NEXT sirve para reemplazar otras instrucciones de BASIC. En este capítulo, hemos examinado los usos típicos de la instrucción FOR... NEXT, incluyendo los bucles anidados, y hemos desarrollado una serie de programas avanzados. Ya que has aprendido todas las técnicas básicas de la programación, estás casi listo para empezar a escribir tus propios programas. En el próximo capítulo explicaremos la manera de empezar.

Ejercicios

7.1. Haz visualizar los primeros 15 números enteros en una línea (4 instrucciones).

7.2. Escribe un programa que lee la hora, suministrada en horas y minutos desde el teclado, y la visualiza en pantalla de la siguiente manera:

Input:	3 (horas),	31 (minutos)
Pantalla:	H H H	(3 letras)
	M M M	(3 letras)
	M	(1 letra)

(Una pista: Puedes usar PRINT LETRA$ varias veces para imprimir varios caracteres.)

7.3. ¿Cuál es la variable-contador en un bucle?

7.4. ¿Se puede saltar al medio de un bucle?

7.5. Haz visualizar una tabla que convierte las onzas a gramos (1 onza = 28 gramos).

7.6. Calcula la suma de los primeros N números enteros impares, donde el valor de N se suministra desde el teclado, y haz que se vea en pantalla en el caso de cada número.

7.7. Lee del teclado las notas que sacaron cinco alumnos que se examinaron cuatro veces cada uno con puntuaciones de 0 a 10. Haz aparecer en pantalla las notas, y luego el total y el promedio para cada alumno.

7.8. Haz que se vea en pantalla una tabla de impuestos para unos precios que van desde $1 hasta $100 en incrementos de $1, suministrando la tasa de impuestos desde el teclado.

La creación

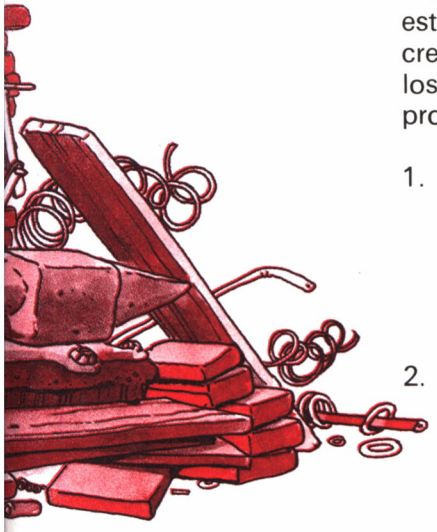
de un programa

8

La programación constituye el diseño de un programa que automatiza una tarea. Esto lo hemos hecho sin pasos intermedios, escribiendo directamente una secuencia de instrucciones en BASIC. Esta técnica es suficiente para los programas simples, pero no lo es para los que sean complejos.

En este capítulo, vamos a estudiar la manera correcta de crear un programa, siguiendo los cinco pasos de este proceso:

1. Especificar la secuencia de los distintos pasos necesarios para resolver el problema, lo que se denomina el diseño del *algoritmo*.
2. Dibujar un diagrama que muestra la secuencia de procesos y los pasos lógicos, cuya denominación es la de *diagrama de flujo*.
3. Escribir el programa en BASIC. Este paso se llama *codificación*.
4. Verificar y probar el programa, lo cual se llama *depuración* del programa (o, alternativamente, *debugging)*.
5. Clarificar y documentar el programa, a lo que llamamos *documentación*.



Hasta ahora sólo hemos practicado el segundo y el quinto paso del proceso de creación, pero esta secuencia es válida únicamente en el caso de los programas cortos. Antes de comenzar, pues, con los programas largos, repasemos detenidamente la secuencia completa de las fases de desarrollo de un programa.

Diseño del algoritmo

Los programas se diseñan para tratar de resolver algún problema o automatizar alguna tarea. Hasta ahora hemos diseñado programas capaces de resolver problemas sencillos. La secuencia de los pasos necesarios para resolver estos problemas ha sido generalmente obvia, de manera que la fase de diseño ha quedado en un segundo plano. Lo normal, sin embargo, es el tener que diseñar primero la solución del problema que se tiene a mano. Para que nos sirva a la hora de escribir los programas, la solución ha de especificarse en un formato que consiste en una serie de pasos definidos. Este formato, o secuencia de pasos, se llama *algoritmo*. Para ser precisos, un algoritmo se define como una especificación, paso a paso, de la solución de un problema. Técnicamente hablando, un algoritmo también ha de tener una terminación concreta, no puede continuar indefinidamente.

Tomemos el ejemplo de un problema sencillo: digamos que queremos convertir un peso expresado en onzas a su equivalente en gramos. Te acordarás de que una onza equivale a 28.35 gramos. La solución es evidente: multiplicamos el peso en onzas por 28.35. Podríamos decir que esto es un algoritmo simple de un único paso.

Tomemos ahora otro problema algo más complicado: vamos a leer un número suministrado en el teclado y comprobar si cae dentro de una clasificación de números determinada. Vamos a decir que el número es aceptable si cae entre 0 y 100. La secuencia de los pasos necesarios para resolver este problema es la siguiente:

1. Leer el número.
2. Comprobar si el número es superior a 0. En caso afirmativo, seguir adelante y, si no, rechazar el número.
3. Comprobar si el número es inferior a 100. En caso afirmativo, seguir adelante y, si no, rechazar el número.

Constituye, pues un algoritmo de tres pasos.

En la práctica, la mayoría de los problemas que se nos presentan son más complicados, y sus soluciones requieren unos algoritmos más largos y complejos. Podemos tomar algunos ejemplos de la vida cotidiana. Encontrarás algunos más en cualquier libro de cocina o manual de operaciones de un coche o electrodoméstico.

«Voy a demostrar el algoritmo del huevo cocido.»

Veamos el algoritmo correspondiente a la cocción de un huevo duro. Los pasos serían:

1. Coger una cacerola.
2. Llenarla de agua.
3. Encender el gas o la placa eléctrica.
4. Colocar la cacerola sobre el fuego o la placa.
5. Esperar a que el agua hierva.
6. Poner un huevo en el agua hirviendo.
7. Anotar la hora.
8. Al cabo de tres minutos, sacar el huevo del agua.
9. Apagar el gas o la placa.

Parece muy sencillito este algoritmo, pero si fuera a realizar esta operación un robot computerizado, tendría que ser bastante más preciso. Por ejemplo, tendríamos que especificar con exactitud la cacerola a emplear, la cantidad de agua a poner en ella, etc.

Muchos de los algoritmos que se ven hoy día en los libros de texto dan por supuesto que el ususario posee ciertos conocimientos culturales o técnicos, y, por tanto, son en general poco adecuados. Dicho de otra manera, presuponen que el usuario puede suministrar la información que falta «entre líneas». Desafortunadamente, es por esta misma razón que muchos de estos manuales resultan difíciles de comprender.

Intentaremos que no nos suceda lo mismo en este libro. Esperemos que nuestros algoritmos se especifiquen lo suficiente como para transformarse en programas operativos.

Tomemos un último ejemplo: el algoritmo de arrancar un coche. Si suponemos que el coche funciona correctamente, el algoritmo resulta ser bastante sencillo:

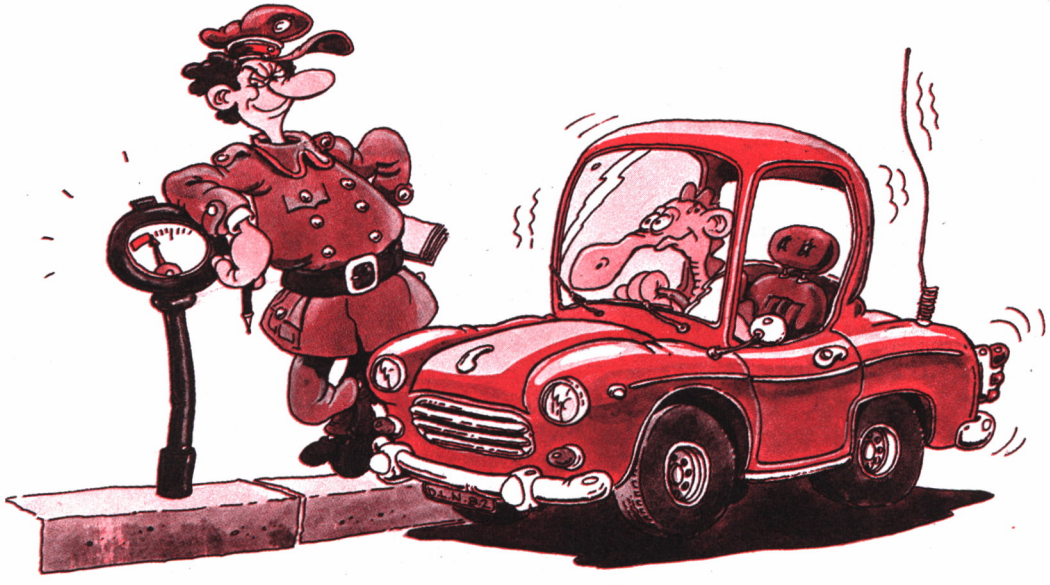
1. Insertar la llave en su sitio.
2. Girarla hacia la derecha hasta el tope.
3. Soltarla mientras se hace una ligera presión sobre el acelerador.

Sin embargo, sabemos que es muy posible que el coche no arranque a la primera. Esto se debe a que inciden varios factores sobre la situación: la temperatura ambiental o la condición mecánica del motor. El preparar un algoritmo para arrancar un coche bajo todas las condiciones posibles llenaría varias páginas, si tomamos en cuenta todo lo que podría suceder.

En la vida cotidiana podemos simplificar a menudo los pasos de un algoritmo. En cambio, cuando se trata de un programa para un ordenador, esto simplemente no es posible. El algoritmo ha de ser tan correcto como completo.

Al diseñar un algoritmo para una solución computerizada, hay que ser meticuloso, anticipando todas las situaciones razonables que puedan surgir. De lo contrario, el programa podría fracasar. Programar con éxito requiere una actitud especial: debes desconfiar constantemente de todo lo que hagas, siempre suponiendo que puede tener un fallo o que falta algo.

Arrancar un coche constituye un algoritmo interesante.

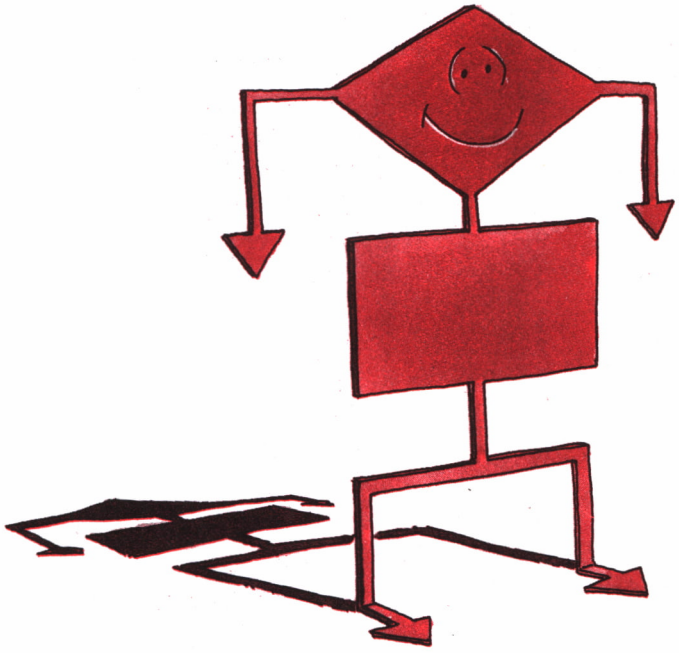


Nunca has de suponer que un dato suministrado desde el teclado vaya a ser razonable o lógico. Has de examinarlo y comprobarlo, así como prepararte para los posibles errores. Demostraremos estos conceptos más adelante en este capítulo, cuando hagamos un estudio de un caso real.

En resumen, para automatizar la solución a un problema mediante un programa en ordenador, hay que empezar con la preparación del algoritmo. La versión final del algoritmo ha de ser perfecta, pese a que al principio raramente lo es. De hecho, lo más probable es que diseñes una solución aproximada (o sea, un algoritmo primitivo) al principio, refinándola según vas progresando, hasta que alcance lo que para ti es la perfección. No dejes de asegurarte que el algoritmo esté completo antes de empezar a escribir las líneas de instrucción en serio.

Recuerda, ¡un algoritmo debe funcionar!

*«Soy el diagrama de flujo.
Estoy aquí para ayudarte.
Por favor, utilízame.»*



Los diagramas de flujo

Bueno, pues ya tenemos el algoritmo. Lo primero que nos viene a la mente es la idea de traducirlo directamente a un programa de BASIC y ejecutarlo rápidamente. ¡Para el carro! Existe aún un paso intermedio más que podrá ahorrarte muchas horas de programación: la creación de los diagramas de flujo. Si te saltas este paso, es casi seguro que no vas a escribir un programa que funcione debidamente, y lo más probable es que malgastes un montón de tiempo intentando corregir el programa, sin ninguna garantía de éxito. En cambio, una vez que logres tener un diagrama de flujo bueno, el escribir el programa está a un paso fácil.

Un diagrama de flujo es simplemente un diagrama que detalla la secuencia de operaciones del programa. La figura 8.1 muestra el diagrama de flujo de los pasos correspondientes a la cocción de un huevo duro.

Como podrás comprobar, este diagrama es simplemente una representación gráfica del algoritmo que hemos visto anteriormente. En este caso, cada casilla del diagrama representa un paso del algoritmo. Su misión es la de mostrar la secuencia de las operaciones a lo largo del tiempo. Más adelante, una vez que estés familiarizado con los diagramas de flujo, seguramente suprimirás el paso del diseño del algoritmo para pasar directamente a ellos, ya que el diagrama es realmente una representación gráfica del algoritmo.

En el caso de un algoritmo tan sencillo como el del huevo duro, el diagrama de flujo no es muy útil y puede ser suprimido. El verdadero valor de un diagrama de flujo se hace patente cuando se proceda al diseño de algoritmos más complejos que están compuestos de muchas elecciones y decisiones.

Vamos a diseñar ahora un programa que solicita tu fecha de nacimiento, la fecha actual, y luego calcula tu edad. El algoritmo es obvio. El diagrama de flujo se puede apreciar en la figura 8.2. Las casillas en forma de rombo del diagrama indican una comprobación, es decir, una elección en la secuencia de ejecución. A efectos de facilitar la conversión del diagrama en un programa, por ahora cerciórate de que cada elección sólo presenta dos alternativas: «sí» o «no», marcando las flechas según el caso. Ahora examina el diagrama de flujo de la figura 8.2 y comprueba que las flechas que proceden de los rombos estén señaladas o «sí» o «no», según el resultado de la comprobación (ver figura 8.3).

En general, existen tres maneras para dibujar las flechas, como vemos en la figura 8.4. Puedes elegir la manera que más te guste, siendo tu propósito el de hacer que el diagrama sea fácil de leer. La posición de las flechas para «sí» y «no» es perfectamente intercambiable; es decir, la del «sí» se puede colocar en el lado derecho si lo prefieres.

Volvamos al diagrama de flujo de la figura 8.2 para el cálculo de edad, y examinemos el algoritmo al que corresponde.

Primero, se solicita la fecha de hoy. Esta operación corresponde a la primera casilla rectangular del diagrama (marcada con un 1). Segundo, se solicita tu fecha de nacimiento, lo que corresponde a la casilla marcada con un 2.

Figura 8.1.
Huevo pasado por agua.

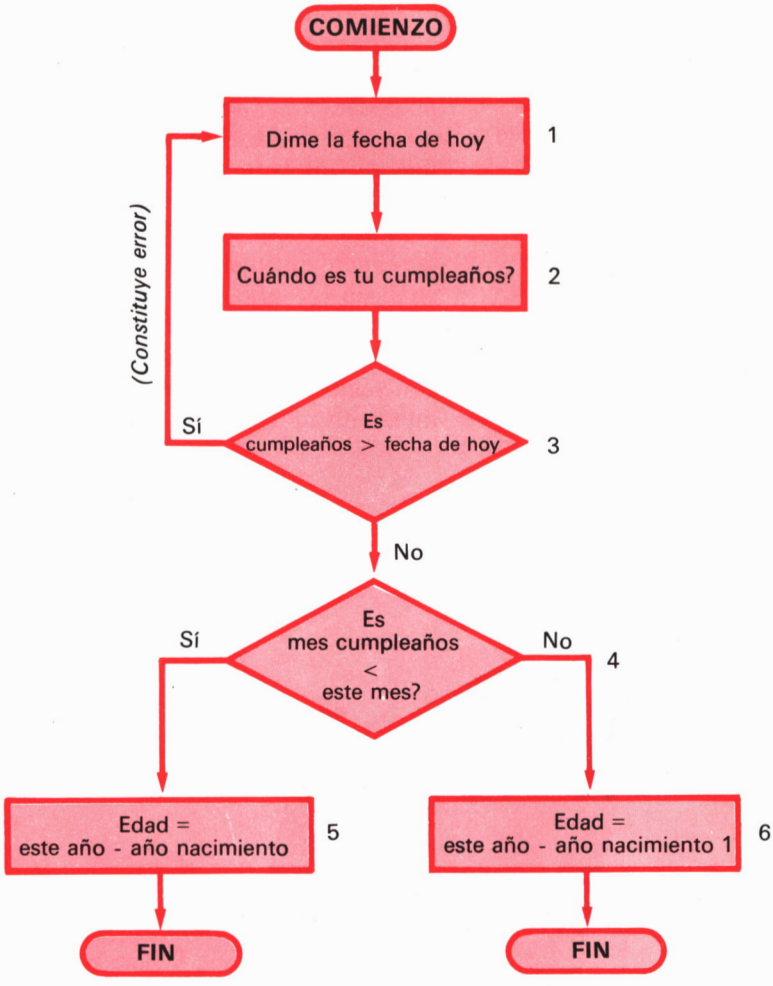
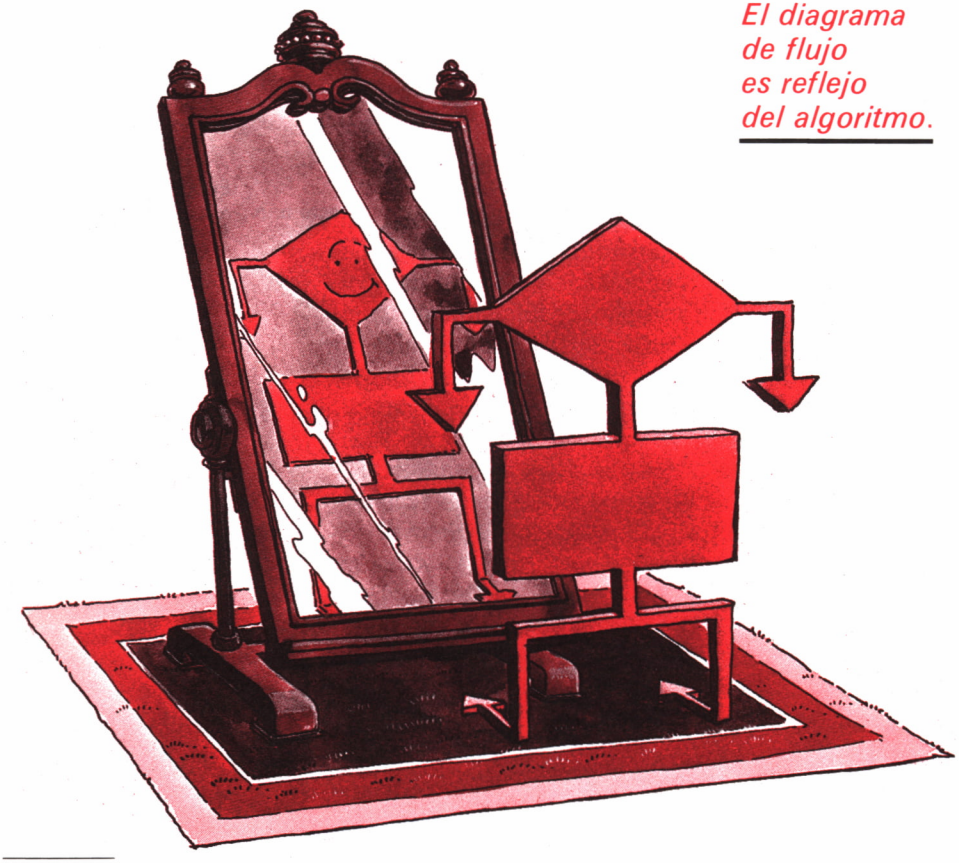


Figura 8.2. Diagrama de flujo para cálculo de edad.

Figura 8.3. Rombo.

A continuación hemos de comprobar si la fecha de nacimiento que se suministra es razonable o no. Tenemos que ver si la fecha es anterior a la de hoy. Si el valor de la fecha de nacimiento es superior al de la fecha actual, se reconoce el error y el proceso se repite. Si no, se supondrá que la fecha de nacimiento es la correcta. Esta operación corresponde a la casilla en forma de rombo (número 3) en el diagrama.

Para refinar el proceso aún más, podríamos rechazar igualmente aquellas fechas de nacimiento que resulten en una edad de 150 años o más, ya que es muy poco probable que sean válidas. Sin embargo, el aceptar tales fechas de nacimiento no supone que vaya a haber resultados seriamente contrarios a nuestro propósito, por lo que probablemente es mejor no molestarnos en prevenir contra ellas.



El diagrama de flujo es reflejo del algoritmo.

En la casilla número 4 del diagrama, se determina si el mes de tu fecha de nacimiento es inferior al mes de la fecha de hoy. Si es así, ya has celebrado tu cumpleaños este año y tu edad se puede calcular como la diferencia entre el año actual y el año de tu nacimiento (casilla número 5). Por ejemplo, si has nacido en febrero de 1946 y estamos en el mes de mayo de 1984, tu edad es igual a 1984 − 1946 = 38.

En caso contrario (casilla número 6), tu edad se calcula usando el año en curso menos el año en que naciste menos 1. Por ejemplo, si naciste en junio de 1942 y estamos en mayo de 1984, tu edad es igual a 1984 − 1942 − 1 = 41. A efectos de simplificar el ejemplo, no comprobaremos el día del mes, añadiendo esta mejora al programa más adelante.

Los pasos del algoritmo habrán quedado claros. Pasemos a examinar los símbolos del diagrama de flujo.

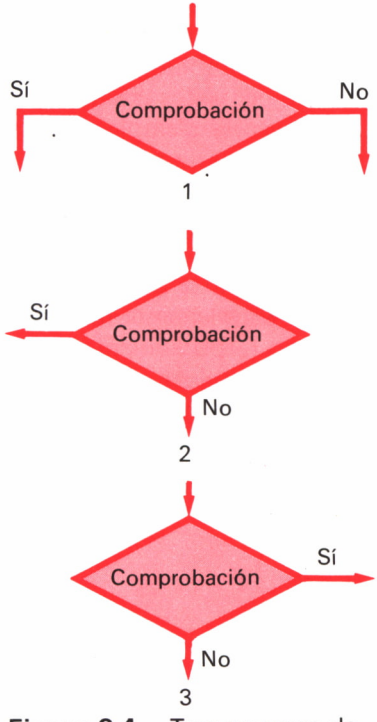


Figura 8.4. Tres maneras de dibujar las flechas.

Figura 8.5. Otra configuración para la casilla de decisión.

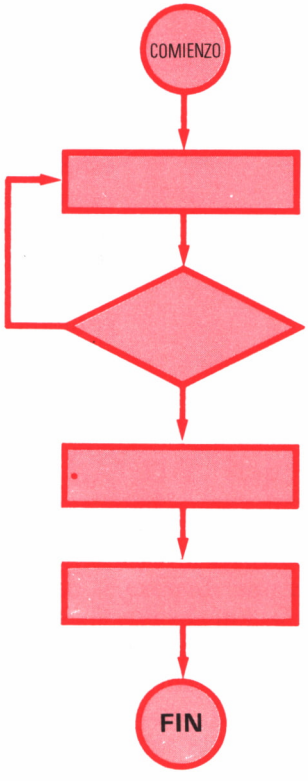


Figura 8.6. Símbolos de comienzo y fin.

Los símbolos del diagrama de flujo

En un diagrama de flujo se emplean las casillas rectangulares para representar los cálculos y las acciones directas que no requieren que se elija entre alternativas, como pueden ser las instrucciones de INPUT o PRINT. Las casillas en forma de rombo denotan las comprobaciones o las elecciones; siempre han de tener por lo menos dos flechas saliendo de ellas. Por último, cada algoritmo ha de tener un principio y un fin, representados por las casillas ovaladas marcadas COMIENZO y FIN.

Los símbolos que se emplean en los diagramas de flujo no están estandardizados. Se han propuesto muchas normas, pero hasta ahora ninguna ha gozado de una aceptación universal. La casilla rectangular se usa siempre, mientras el rombo puede ser sustituido con una casilla con las esquinas redondeadas, como vemos en la figura 8.5. Además, los símbolos de COMIENZO y FIN pueden aparecer dentro de pequeños círculos, como se ve en la figura 8.6. Por último, a veces se emplean unos símbolos especiales para denotar el uso de periféricos específicos; por ejemplo, una operación de PRINT puede representarse en cualquiera de las dos maneras que vemos en la figura 8.7.

En la práctica no hay que preocuparse demasiado por los símbolos. El diagrama de flujo es simplemente una manera cómoda de visualizar un algoritmo (especialmente cuando tiene muchas elecciones). Incluso puedes emplear otra serie de símbolos si te parece mejor, aunque una ventaja de usar símbolos ya aceptados es la de poder compartir tus programas con los demás, aparte de que te será más fácil leer o seguir otros diagramas de flujo.

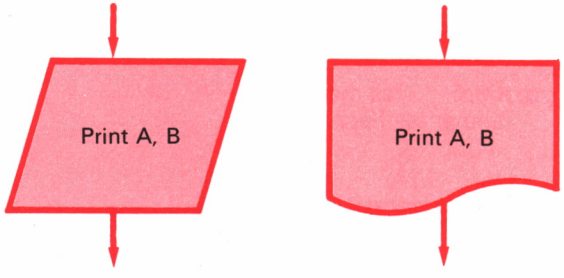


Figura 8.7. Símbolos para el PRINT.

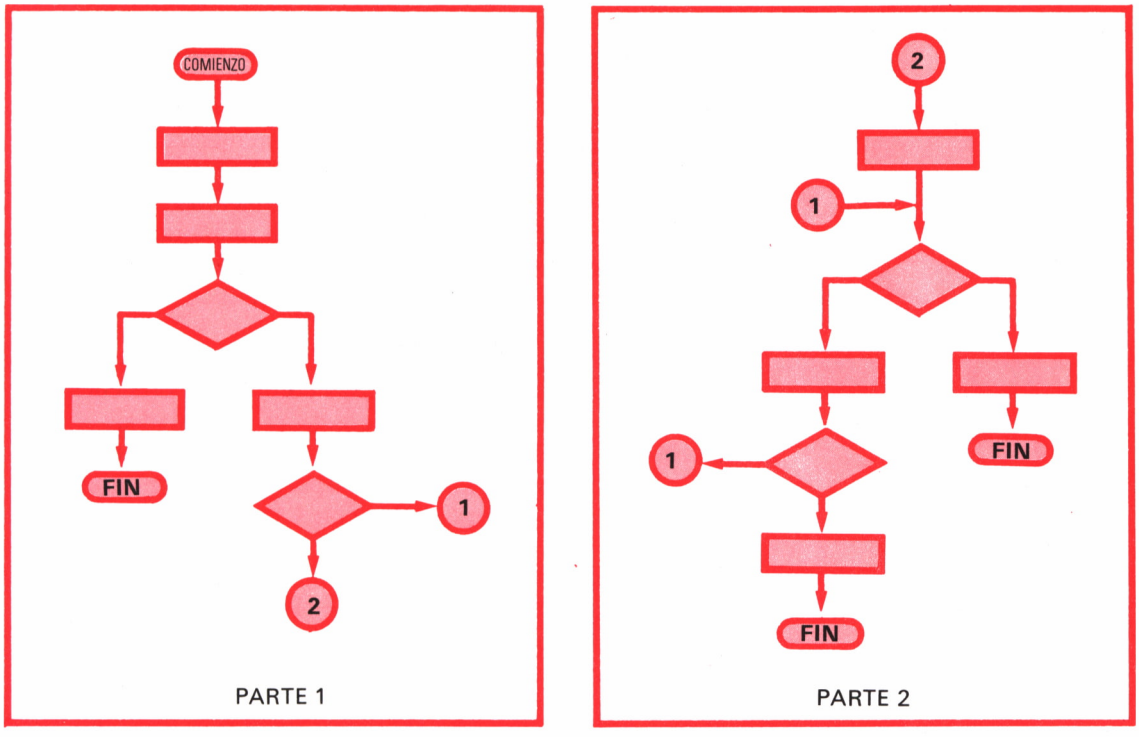


Figura 8.8. Subdivisión del diagrama de flujo.

Subdivisión de la tabla de flujo

Existe otra práctica útil para este paso en la creación de un programa. Si un diagrama de flujo se extiende sobre más espacio que el que hay disponible en una página, se puede separar por segmentos. A cada flecha que se corta se le asigna un número o un nombre, que será el mismo que se pondrá en la siguiente página donde sigue el diagrama. La figura 8.8 nos muestra un ejemplo de esta manera de cortar e identificar las flechas de un diagrama.

Cómo refinar el diagrama de flujo

Las instrucciones que se colocan dentro de las casillas del diagrama de flujo se pueden escribir de la manera que sea más cómoda. No son instrucciones de BASIC. Cuando escribes un diagrama por primera vez, seguramente emplearás instrucciones poco precisas, como «dime tu fecha de nacimiento»; más tarde querrás mejorar estas instrucciones y diseñar un diagrama que sea más fácil de traducir al lenguaje de programación.

Si encuentras que las instrucciones que contienen las casillas son lo suficientemente precisas como para proceder a la escritura del programa, obviamente no hay necesidad de modificarlas. Si, por otro lado, crees que las instrucciones son demasiado complejas o vagas para empezar con el programa directamente, has de reemplazarlas con una secuencia más detallada.

Como ejemplo, recuerda el diagrama de flujo en la figura 8.2 que comprueba el mes de la fecha de nacimiento, pero no el día, a la hora de determinar si el cumpleaños ha ocurrido o no durante el mes actual. Mejoremos el programa para que verifique no sólo el mes, sino el día también. El segmento correspondiente del diagrama de flujo inicial (aproximado) se ve en la figura 8.9. La versión mejorada del diagrama se puede apreciar en la figura 8.10.

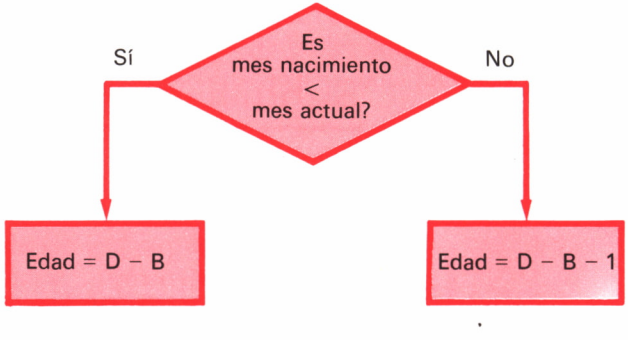


Figura 8.9. Un algoritmo aproximado.

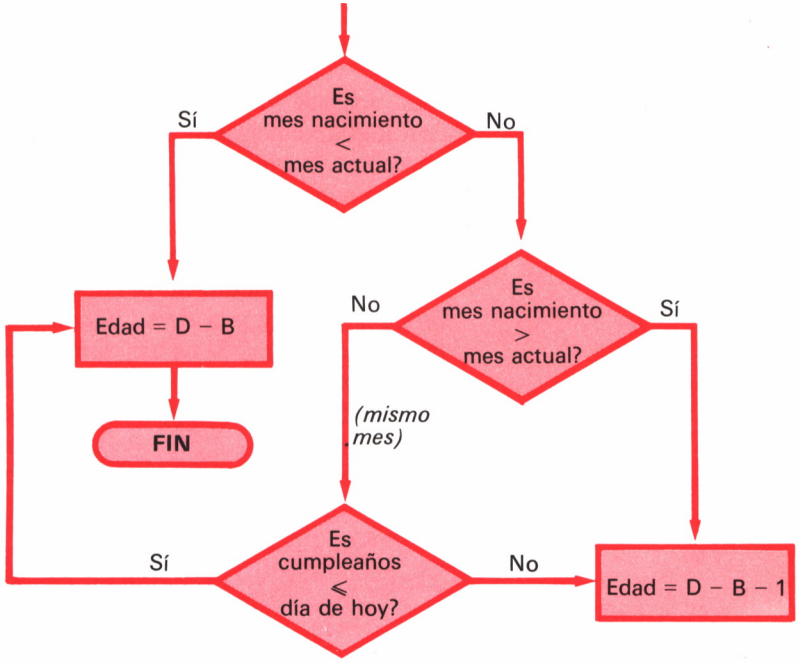


Figura 8.10. Un diagrama de flujo más refinado.

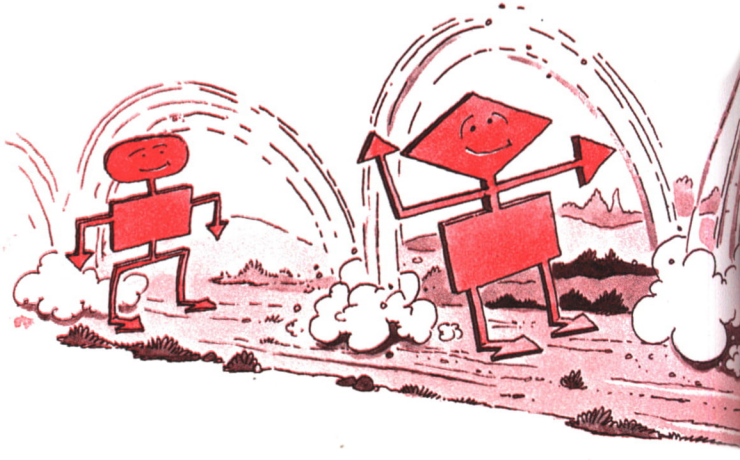
En la práctica normalmente no es necesario escribir el algoritmo, sino que se procede directamente a la fase del diagrama de flujo. Este procedimiento es aceptable. Sin embargo, a veces incluso los programadores experimentados (y los no muy experimentados) se saltan la fase del diagrama de flujo también, comenzando directamente con la escritura del programa sobre papel. Esta manera de proceder se presta a cometer errores y a perder mucho tiempo; recomiendo enérgicamente que siempre escribas un diagrama de flujo antes de programar. Según vayas adquiriendo experiencia podrás usar diagramas menos detallados, realizando solamente el diagrama de la estructura general del programa.

El ensayo manual

Una vez que hayas escrito el diagrama de flujo, podrás ensayar con algunos ejemplos. Asegúrate que los resultados sean los correctos, o que aparenten serlo. Este proceso se denomina *ensayo manual,* para diferenciarlo de los ensayos hechos en el ordenador.

Para tomar un ejemplo, vuelve a examinar el diagrama de flujo para el cálculo de la edad en la figura 8.2 y pruébalo con la fecha de hoy y la de tu nacimiento, según se estipula. ¿Te da la edad correcta? Si es así, parece que todo marcha bien. Si no, existe un error. Pruébalo ahora con otros valores, usando fechas de nacimiento tanto anteriores como posteriores a la fecha de hoy. ¿Funciona? Si funciona, el algoritmo probablemente está bien estructurado. Si no, hay un error. El ensayo manual es un procedimiento rápido para asegurarte que no exista ningún error fundamental en la estructuración de tu programa.

Si te saltas
todas las normas,
caerás en un error.



Habiendo conseguido un diagrama de flujo que parece funcionar, hemos dado todos los pasos preliminares a la escritura del programa mismo. Prosigamos, pues, a la creación del programa.

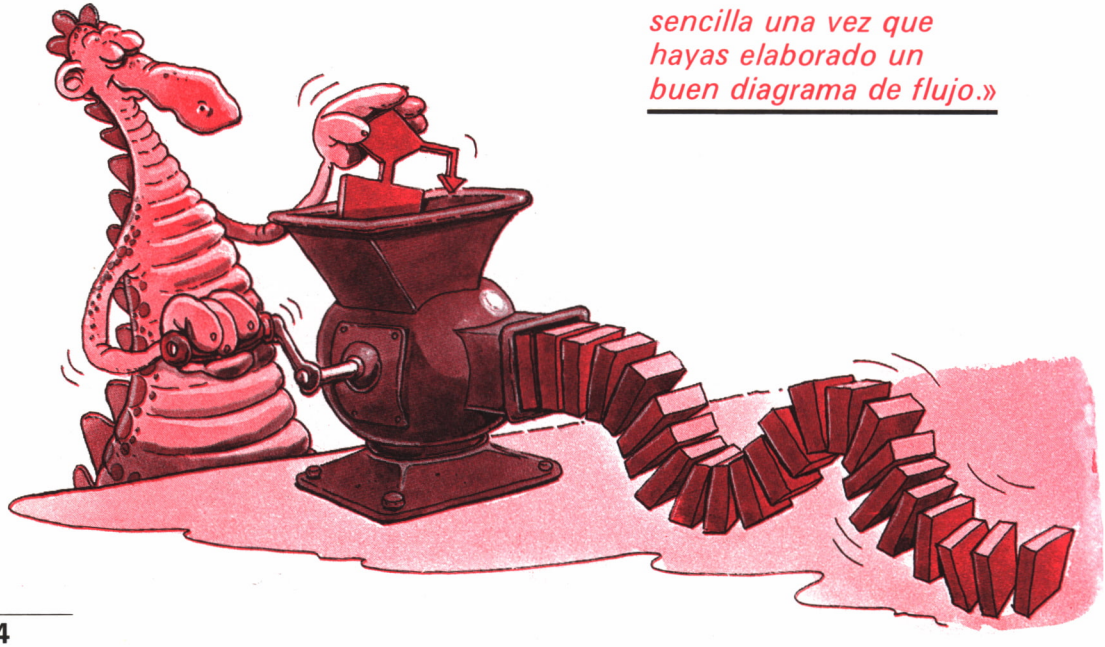
La codificación

La escritura de las instrucciones se llama *codificación*. Empleamos el término *programación* normalmente para referirnos al proceso entero de la creación del programa: el diseño del algoritmo, el diseño de diagrama de flujo, la codificación, la depuración y la comprobación de su funcionamiento. La codificación consiste en la traducción del contenido del diagrama de flujo a unas instrucciones de programa expresadas en un lenguaje de programación, en este caso, expresadas en BASIC.

Esto es precisamente lo que hemos estado aprendiendo hasta este capítulo: cómo traducir varias frases, fórmulas, comprobaciones y condiciones a las instrucciones de BASIC.

La clave de la codificación fluida es el diseño de un diagrama de flujo con detalle suficiente como para poder transformar sin dificultad cada casilla del diagrama en varias instrucciones de BASIC sencillas. En general, en las fases de principiante del programador, cada casilla del diagrama se traduce a un par de instrucciones de BASIC, digamos una o dos; es decir, existe una correspondencia bastante directa entre

«La codificación resulta sencilla una vez que hayas elaborado un buen diagrama de flujo.»

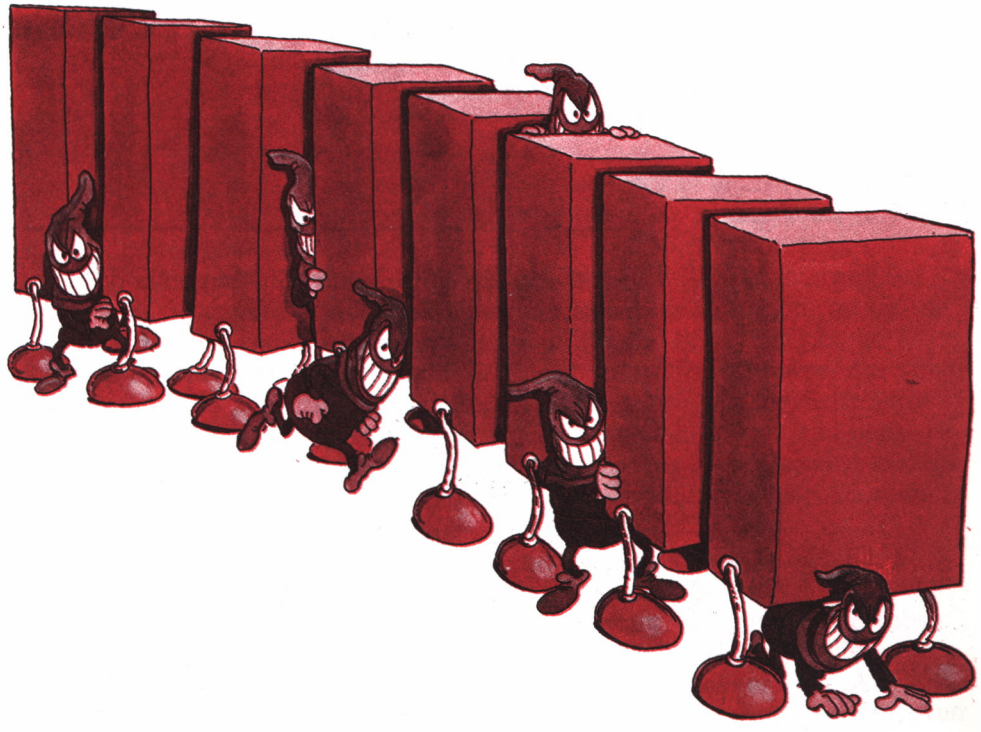


las casillas y las instrucciones. Más tarde, una vez que el programador haya desarrollado sus conocimientos, es posible diseñar diagramas de flujo menos precisos, en los que cada casilla equivale a muchas instrucciones.

Aunque pueda parecer mentira, la fase de codificación muchas veces es la que ocupa menos tiempo en el desarrollo de un programa. La comprobación del comportamiento del programa muchas veces requiere bastante más tiempo que la codificación en sí. Es evidente, pues, la importancia que tiene el diseño de un buen diagrama de flujo: permite reducir al máximo los errores y el tiempo necesario para eliminarlos.

Recuerda la importancia que tiene el hacer que tu programa sea exacto, conciso y fácil de leer a la hora de codificarlo, ya que esto incide sobre el tiempo necesario para su ejecución, comprobación y modificación.

Ojo con Armafallos. Que sea limpio y exacto el programa.



La exactitud

Escribe tu programa con gran precisión, ya que cualquier fallo en la colocación de un símbolo de puntuación probablemente hará que fracase el programa.

La claridad

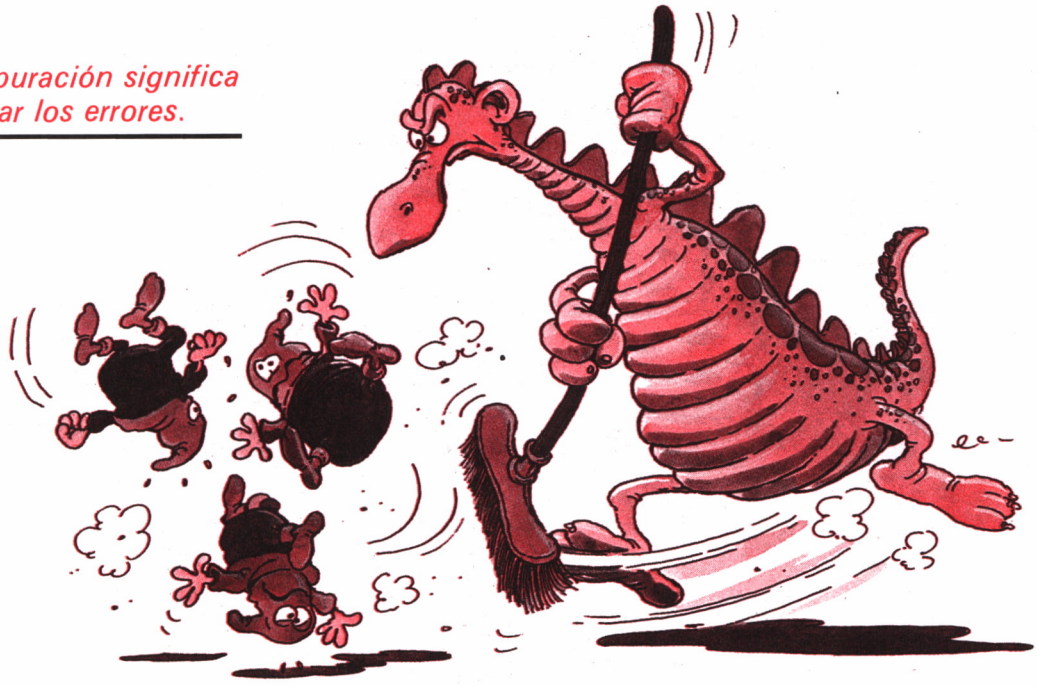
Utiliza nombres de variables que sean fáciles de recordar. Deja intervalos o espacios amplios en la secuencia de numeración de líneas por si luego resulta necesario intercalar nuevas instrucciones. Usa también un comentario completo (mediante la instrucción REM) a lo largo de todo el programa para aclarar los procesos que lleva a cabo.

Ahora que has diseñado el algoritmo, dibujado el diagrama de flujo y escrito el programa correspondiente, esperarás que funcione. Pues no. Lo sentimos mucho, pero en la mayoría de los casos *los programas no funcionan, como era de esperar, a la primera.* Lo normal es que hagan falta varios intentos o el poseer mucha experiencia antes de que un programa algo largo funcione correctamente. Esto nos lleva al siguiente tema: la depuración del programa.

La depuración

Has traducido tu programa del diagrama de flujo a las instrucciones de BASIC. De momento, sigue sobre papel, así que lo que falta por hacer ahora es escribir el programa en el teclado del ordenador para introducirlo a la memoria, ejecutarlo y averiguar si funciona o no. Este proceso es el de la *comprobación* y *depuración* del programa. Los errores de un programa constituyen sus *fallos,* y la eliminación de estos fallos se hace durante la *depuración.* Cada vez que se detecta un fallo, en primer lugar hay que corregirlo y volver a ejecutar el programa. Pese a todos tus esfuerzos en escribir el programa correctamente, raramente un programa largo estará exento de errores a la primera. Esto se debe a la facilidad con la que se puede omitir un carácter, o incluso una línea entera, durante la introducción del programa en memoria, hasta el extremo de que incluso los mejores programadores se ven obligados a

La depuración significa eliminar los errores.



invertir un tiempo considerable en la depuración de sus programas. No te desanimes si tienes que corregir un programa varias veces antes de que funcione bien; es normal.

Afortunadamente, tu intérprete de BASIC te ayudará a diagnosticar algunos de los problemas. Si el programa contiene algún fallo que puede ser detectado por el intérprete, la ejecución del programa se interrumpirá y el intérprete te enviará un mensaje describiendo el problema como, por ejemplo, «SYNTAX ERROR IN LINE 84). El intérprete es más útil a la hora de detectar los *errores de sintaxis* (el uso de símbolos u operaciones no admisibles). Por desgracia, no te puede ayudar a identificar los errores más insidiosos, que son los del *diseño* o de la *lógica* del programa. La localización de estos errores es de tu incumbencia, por lo que vale la pena invertir el tiempo necesario para diseñar los diagramas de flujo con diligencia. Por esta misma razón es importante, además, validar y comprobar el rango de cada número que se suministra a un programa, o que es generado por él. En el caso de que el programa contenga un error de lógica, este método ayudará a aislar el segmento de programa defectuoso.

Normalmente, y en el caso de un programa sencillo, el intérprete detectará algunos errores de mecanografía. Al corregirlos, el programa funcionará de la manera deseada. Aun

El Intérprete ayuda a diagnosticar los errores de sintaxis.

así, es buena idea asegugarte de que funcione correctamente mediante varios ensayos con distintos valores o condiciones, por si acaso tuviese algún fallo lógico. En la mayoría de los casos, sin embargo, podrás averiguar que el programa se comporta de la manera esperada.

Consejos prácticos

Un consejo práctico que querrás seguir: deberías distribuir varias instrucciones de PRINT extras a lo largo del programa para verificar continuamente el valor de ciertas variables claves. Esto te ayudará a detectar algún valor extraño y a localizar las instrucciones que los han engendrado. Una instrucción típica de este tipo sería:

```
1235  PRINT "COMPROBACION DEL VALOR DE LA
      MEDIA :"; MEDIA
```

Luego, al ver que el programa funciona bien, se pueden eliminar estas instrucciones de sobra. Esta técnica se llama *seguir* una variable.

Otro procedimiento práctico es el de usar el *modo inmediato de operación* para verificar el valor de varias variables en el programa cuando éste se interrumpe o es interrumpido por el intérprete. Por ejemplo, podrías escribir:

```
>PRINT MEDIA
```

y luego:

```
>PRINT SUMA
```

para determinar el valor actual de estas dos variables.

Las claves del éxito en la depuración son la experiencia y las medidas de prevención. Cuanto mayor sea la inversión en el diseño y la preparación del diagrama de flujo, menores serán las necesidades de depuración.

Ahora has conseguido que funcione bien tu programa. Al encontrar que todo marcha bien, no querrás tocarlo más. Sin embargo, puede que mucho más tarde se tropiece con algún fallo, o se quiera volver a usar el programa o compartirlo con algún amigo. Para que sea perfectamente reutilizable, falta dar el último paso: hay que documentar el programa antes de olvidar su manera de funcionar.

La documentación

Acabas de diseñar y codificar un programa. Conoces muy bien su forma de operar y lo que hace cada instrucción. Te sorprenderá, pues, con qué rapidez se te olvida lo que hace el programa y las dificultades que puede haber en leer o entender el programa más adelante. Si piensas volver a trabajar con él o corregir sus posibles errores en una fecha futura, es vital que clarifiques la operación del programa pronto y de varias maneras. Esto conlleva la clarificación del programa mismo, además de todo lo que puede necesitar una explicación, bien sobre papel o bien como comentario incorporado al programa en la forma de instrucciones de REM.

A continuación tenemos un resumen de las distintas técnicas que hemos discutido para clarificar el progrma:

Sigue un esquema claro: Separa las secciones con líneas en blanco o instrucciones vacías. Utiliza la alineación y la sangría para dar claridad. Puede que también quieras usar números de línea que tengan todos la misma longitud. De esta manera todas las instrucciones del programa se alinearán verticalmente. Además, cada vez que empleas una instrucción de FOR... NEXT, resulta conveniente sangrar el bloque de instrucciones que se encuentra entre la FOR y la NEXT.

Por último, se emplean los espacios en blanco para clarificar las instrucciones complicadas, especialmente las que contienen expresiones matemáticas, y los paréntesis para aclarar el resultado de algún cálculo.

Explica tus métodos: Usa las instrucciones REM para explicar las fórmulas, las comprobaciones, los nombres o los procedimientos. También es aconsejable incluir una breve descripción de los métodos o maneras de proceder que estás empleando pero que no son evidentes o que no se explican del

Hay que documentar al programa.

todo con las instrucciones de PRINT existentes en el programa.

Corrige los diagramas de flujo: Has de hacer un diagrama o juego de diagramas que corresponda exactamente a la versión final del programa. Muchas veces se hacen cambios al programa durante el proceso de depuración: asegúrate que éstos se reflejen en el diagrama de flujo original para que no encuentres dificultades en hacer cambios o correcciones más adelante.

Cambia los números de línea: Durante la fase de depuración muchas veces es necesario insertar nuevas líneas en el programa. Cuando estés satisfecho con el programa, es aconsejable cambiar la numeración de las líneas, de tal manera que el intervalo entre los números de línea sea siempre igual. Esto facilitará cambios sucesivos en el programa. Existen en el mercado distintos programas que hacen esto automáticamente, asignando un número nuevo a todas las líneas del programa. Si no te es posible adquirirlo, puede ser conveniente, de todas maneras, el que inviertas algo de tiempo en una nueva numeración que tendrá como resultado una secuencia limpia. No obstante, el cambio de numeración es una comodidad opcional; no incide sobre el programa en sí.

Resumen

En este capítulo hemos visto los cinco pasos que llevan a la versión final de un programa: el diseño, el dibujo del diagrama de flujo, la codificación, la depuración y la documentación. Repasemos cada paso brevemente.

Cualquier programa requiere el diseño de un algoritmo, lo cual se puede llevar a cabo mentalmente, si no sobre papel. El algoritmo puede consistir en una serie de fórmulas o apuntes que describen los pasos esenciales del programa.

El siguiente paso es el del diseño del diagrama de flujo que describe la secuencia entera de las operaciones. Hay que considerar que éste es un paso necesario para cualquier programa que tiene ya un número importante de líneas. Recuerda: cuanto mejor sea el diseño del diagrama de flujo, mayores son las posibilidades de que el programa resulte de la forma deseada.

El siguiente paso es el de la codificación, o sea, la traducción del diagrama de flujo a instrucciones en BASIC. La práctica hará que esta fase se lleve a cabo con mayor rapidez. De hecho, al cabo del tiempo llega a ser la fase que menos tiempo requiere. Luego viene la fase de la comprobación y la depuración del programa. Esta fase siempre es necesaria y muchas veces es la más larga. Cada programa ha de ser examinado en detalle.

Por último, la calidad de la documentación determinará con qué facilidad se podrá usar o modificar el programa en una fecha posterior a la de su creación.

Ejercicios

8.1. Describe las cinco fases que constituyen el desarrollo de un programa.

8.2. ¿Cuál es la diferencia entre la codificación y la programación?

8.3. ¿Qué finalidad tiene la depuración?

8.4. ¿Cómo se realiza el trazado de una variable?

8.5. ¿Por qué es aconsejable cambiar la numeración de las líneas después de hacer muchos cambios?

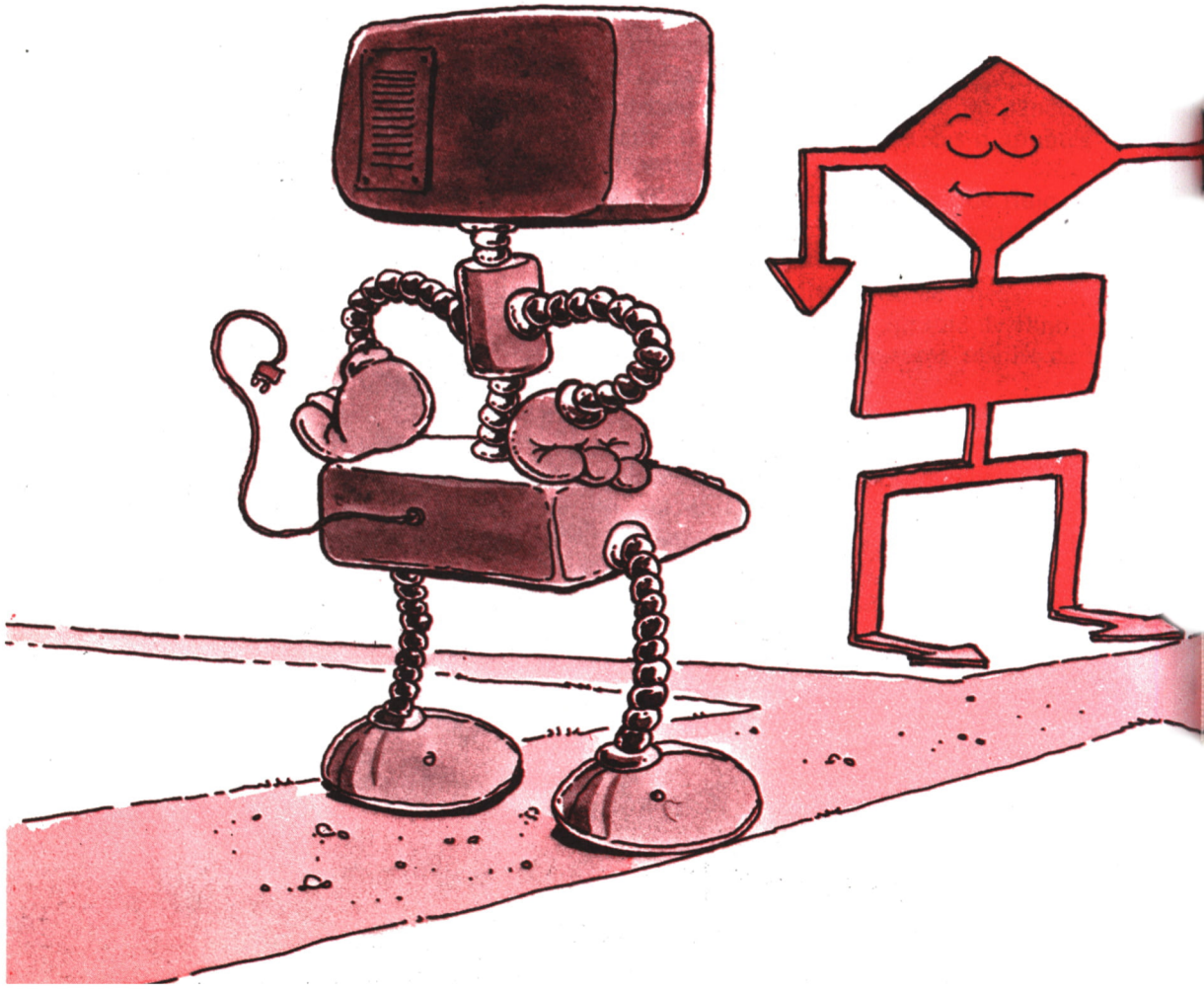
8.6. ¿Qué es un diagrama de flujo?

8.7. Escribe un diagrama de flujo que sirva para describir cómo se pone en marcha un coche o algún aparato mecánico o eléctrico.

8.8. ¿Cuáles son las ventajas de un programa que está escrito con claridad?

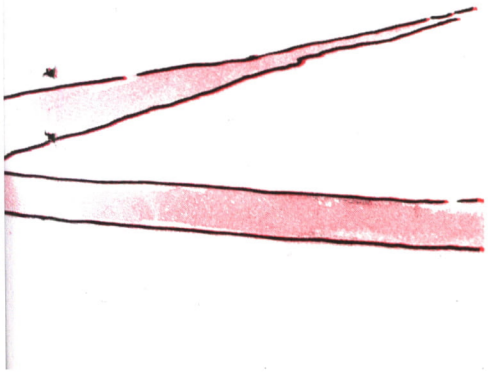
8.9. Describe las técnicas que se pueden emplear para aclarar el funcionamiento de un programa.

Estudio de
La
al sist



un caso real: conversión ema métrico

9



Ahora vamos a desarrollar un programa completo, comentando cada paso en su momento. El problema a resolver es el siguiente:

Necesitamos escribir un programa que convierta automáticamente un peso que se exprese en onzas a su equivalente expresado en gramos. El programa ha de hacer una de estas dos cosas: convertir un número suministrado desde el teclado, o visualizar una tabla de conversión de peso para todos los números entre dos valores específicos.

El diseño del algoritmo

La secuencia general de los pasos a seguir para resolver el problema es bastante concreta: solicitaremos al usuario que indique lo que quiere hacer (una conversión simple o una tabla de valores) y luego realizamos la operación que se ha especificado. Muy por encima podemos decir que éste es el algoritmo, pero vamos a concretarlo un poco más.

Una onza equivale a 28,35 gramos, por lo que la conversión de onzas a gramos se lleva a cabo mediante la fórmula:

$$P_{gramos} = P_{onzas} \times 28.35$$

o, más brevemente:

$$P_g = P_{oz} \times 28.35$$

El algoritmo básico es, entonces:

- — Especificar si conversión sencilla o tabla de valores.
- — Si es una conversión, pedir peso en onzas.
- — Convertir a gramos (mediante la fórmula de arriba) y visualizar el resultado.
- — END.
- — Si es una tabla, pedir valor máximo en onzas.
- — Convertir a gramos y visualizar los resultados hasta el valor máximo.
- — END.

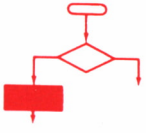
En la práctica no es imprescindible llegar a escribir el algoritmo, siempre y cuando se prepara el diagrama de flujo.

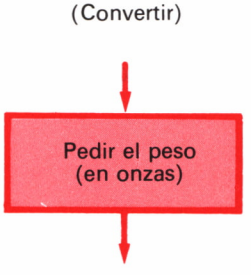
Preparación del diagrama de flujo

Para preparar el diagrama de flujo, sabemos que lo primero que hay que establecer es si el usuario desea convertir un valor solo o una tabla de valores. A continuación tenemos la sección del diagrama correspondiente:

Como se ve, se trata de una casilla indicativa de una decisión, con dos posibles respuestas (es decir, ramas): CONVERTIR y TABLA.

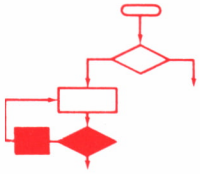
Examinemos primero la decisión de CONVERTIR: el usuario desea convertir un peso expresado en onzas a su equivalente en gramos. Tenemos que solicitar el valor de este peso. Esto implica que en el diagrama pongamos:





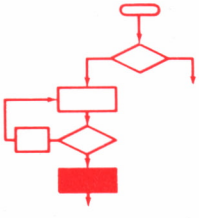
Ahora sería posible convertir este valor a gramos. No obstante, vamos a hacer que el diagrama sea más sofisticado.

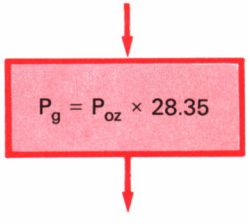
Como precaución comprobaremos el valor que nos facilita el usuario, determinando su posible validez. Con esta modificación el diagrama adquiere el aspecto siguiente:



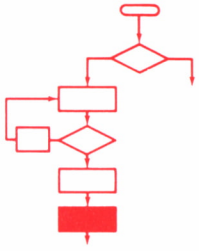
Habrás notado que hemos añadido una casilla para comprobar la validez del dato suministrado. Si el dato es aceptable, seguimos adelante; de lo contrario, se envía un mensaje de diagnóstico como, por ejemplo, «VALOR INADMISIBLE, PRUEBE DE NUEVO», y el programa solicita otro valor.

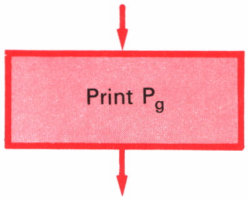
Ahora hemos conseguido un valor admisible para el peso que podemos convertir a gramos, o sea:



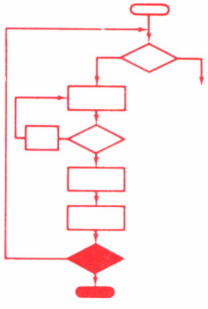


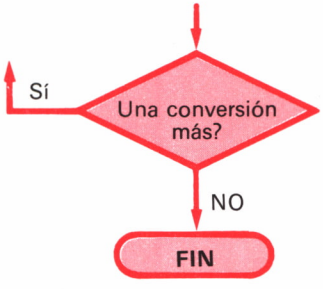
Ya podemos poner los resultados en pantalla. La casilla para esto sería:



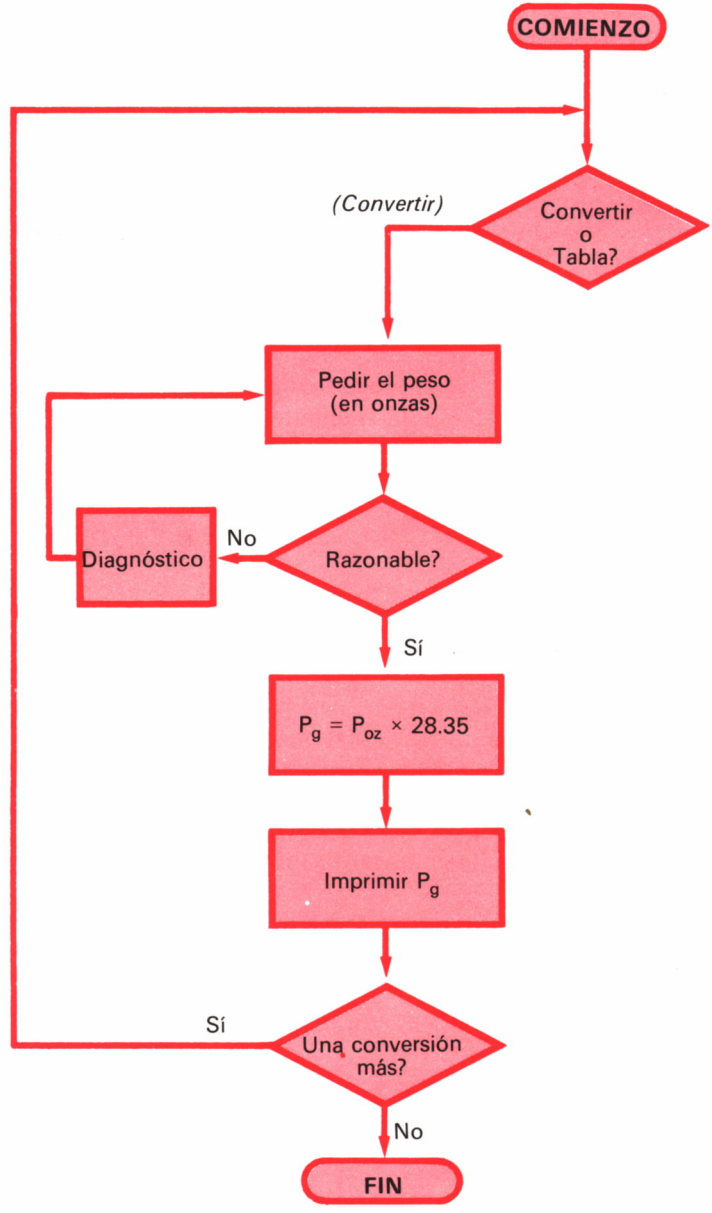


La conversión sencilla ya está realizada, por lo que sería posible terminar esta sección del diagrama sin más. Sin embargo, vamos a facilitarle más la labor al usuario y preguntarle si desea realizar otra conversión. Esto nos lleva a la siguiente casilla:



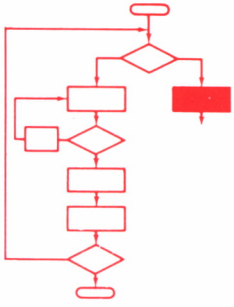


La palabra COMIENZO bajo la flecha izquierda indica que ésta conectará con el principio del programa. Hasta ahora el diagrama ha ido adquiriendo el siguiente aspecto:

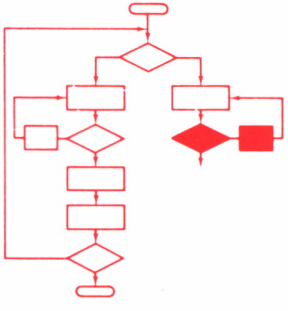


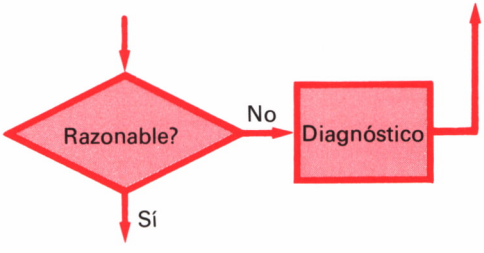
Volvamos ahora a la primera bifurcación para determinar lo que sucederá si el usuario opta por confeccionar una tabla de conversión. Para realizarla deberá seguir la flecha marcada con la palabra TABLA.

Tenemos que saber el valor máximo que se va a convertir. Esto se logra mediante la casilla siguiente:

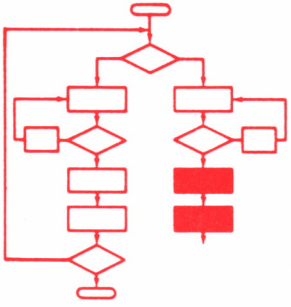


De nuevo vamos a cerciorarnos de que sea aceptable este número: como sucedía anteriormente, el programa no avanzará hasta que sea especificado un valor aceptable para la variable MAX.

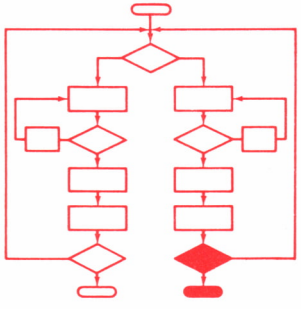




Una vez que se suministra un valor adecuado, podemos proseguir con la visualización en pantalla de una tabla que convierte onzas a gramos hasta el valor máximo deseado:



Por último, vamos a añadir la misma facilidad que empleamos en el caso de la conversión sencilla. Preguntamos al usuario si quiere realizar alguna conversión más:



La figura 9.1 nos muestra el diagrama entero, que ha resultado ser bastante típico. El contenido de cada casilla se ha especificado en términos amplios. Algunas se codificarán con tan sólo un par de instrucciones, mientras otras van a requerir bastante más código. No obstante, la secuencia es lo más importante; el hecho de que algunas de las casillas estén más especificadas no tiene gran importancia. Ten en cuenta que en el diagrama lo que ha de ser exacto es la secuencia de los procesos; los detalles dentro de las casillas se pueden poner de

la manera que más fácil te parezca. No existe ninguna razón especial para pasarse mucho tiempo mejorando el contenido de las casillas, siempre que pienses que vas a poder traducir de ellas al lenguaje BASIC sin dificultad.

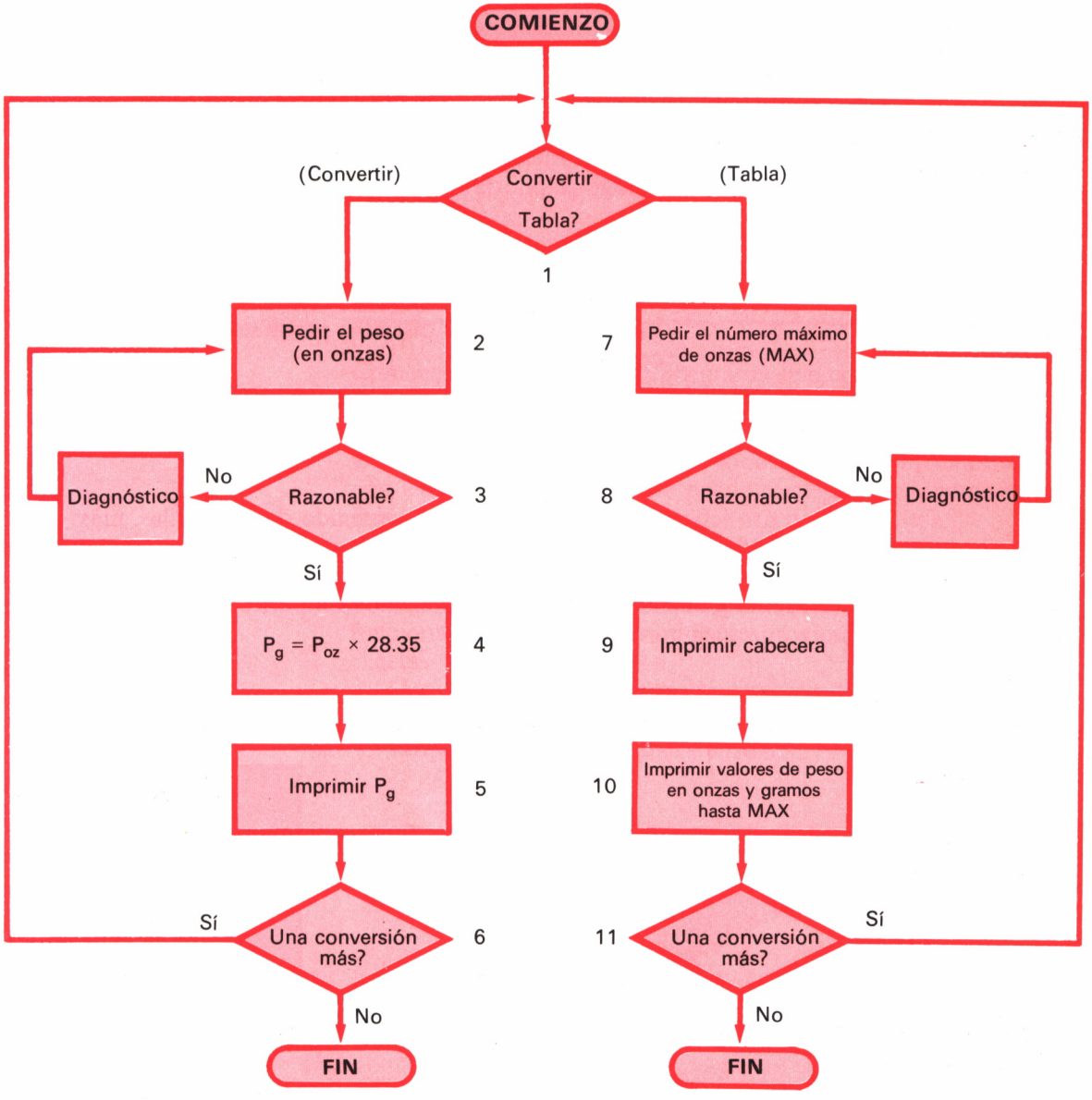


Figura 9.1. Diagrama de flujo de conversión de peso.

La estructura del diagrama ha de ser fácil de leer e interpretar. Las probabilidades de escribir un programa correctamente se ven incrementadas notablemente cuando se trabaja desde un diagrama que está bien hecho.

Para ser completos, podríamos considerar algunas mejoras o alternativas como las siguientes:

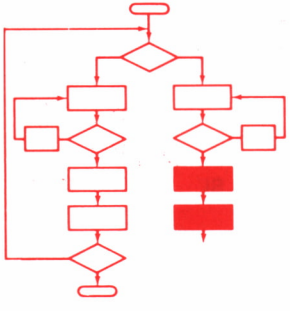
— Se podría explicar en detalle la manera en que se va a comprobar la validez del peso (en este caso, nos dedicaremos solamente a ver si P_{oz} es un número positivo).

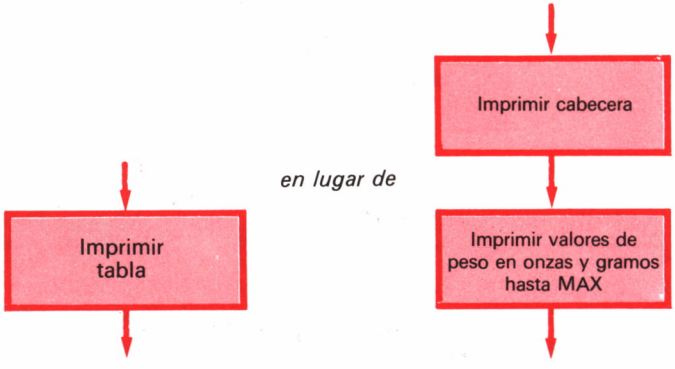
— Tanto el mensaje de diagnóstico como gran parte del diálogo se podrían hacer mucho más explícitos.

Como regla general, es mejor *hacer sólo lo imprescindible,* o sea, hacer únicamente lo que sea preciso para que:

1. La secuencia de los distintos pasos sea completa y correcta.
2. Cada casilla sea comprensible y se preste a la conversión de su contenido a las instrucciones de un programa.

Cuanto más simple sea el contenido de las casillas, más fácil de comprender resultará el diagrama. Asimismo, cuanto más detallado sea este contenido, más fácil será la codificación. Siguiendo estos consejos, podemos simplificar el diagrama al escribir:





Las dos versiones son válidas. La mejor para ti es la que te parezca más fácil de usar. En este caso, yo preferí reflejar directamente los pasos de programación, por lo que el contenido de la casilla es más explícito.

Siempre existe la opción de modificar el contenido de una casilla, o cualquier otra parte de un diagrama de flujo, sobre una hoja de papel aparte. Esto facilita la codificación o el análisis de alguna alternativa.

Bien, pues ahora disponemos de un diagrama de flujo completo, así que vamos a comprobarlo a mano con algunos números. El modo de hacerlo es evidente, por lo que te lo dejamos para que practiques.

Codificación

Comencemos ahora con el programa que corresponde a nuestro diagrama de flujo. Para que el programa resulte más fácil de leer, vamos a suponer que tu versión de BASIC admite los nombres de variables largos (nombres que emplean más de una letra) y que el intérprete pueda manejar operaciones de coma flotante (es decir, admite el uso de decimales).

En caso contrario, habrás de cambiar los nombres de las variables. Si tu versión de BASIC es además un BASIC «Integer» (de números enteros únicamente), el programa podrá funcionar, pero la conversión será aproximada en lugar de exacta.

Convirtamos el contenido de cada casilla del diagrama a sus instrucciones de BASIC correspondientes.

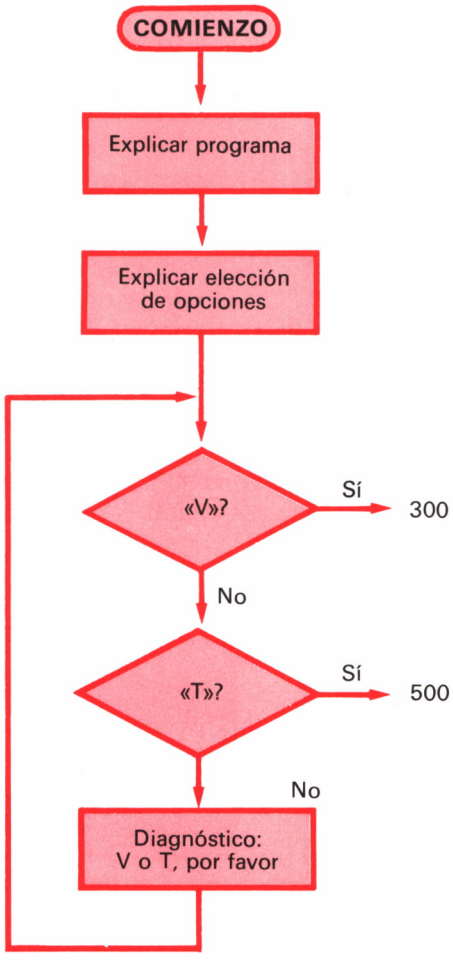
Empecemos con la primera casilla:

Las instrucciones en lenguaje de programación correspondientes son:

```
100 REM * * * CONVERSION DE ONZAS A GRAMOS * * *
110 REM ESTE PROGRAMA PUEDE REALIZAR UNA
    CONVERSION DIRECTA
120 REM O VISUALIZAR UNA TABLA DE VALORES
130 REM PRIMERO, DETERMINAR TIPO DE CONVERSION:
    DIRECTA O DE TABLA
140 PRINT "VOY A CONVERTIR ONZAS A GRAMOS"
150 PRINT "SI DESEA CONVERTIR UN VALOR DIRECTAMENTE,
    ESCRIBA V."
160 PRINT "SI DESEA VER UNA TABLA DE VALORES, ESCRIBA
    T."
170 PRINT "SU ELECCION (V O T)";
180 INPUT ELECCION$
190 IF ELECCION$="V" THEN GOTO 300
200 REM LINEA 300 ES LA CONVERSION DE UN VALOR
210 IF ELECCION$ = "T" THEN GOTO 500
220 REM LINEA 500 ES LA CONVERSION DE UNA TABLA
230 REM SI EL CARACTER NO FUE NI V NI T, EL INPUT NO ES
    VALIDO
240 PRINT "V O T POR FAVOR: ";
250 GOTO 170
```

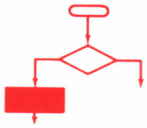
Según vayas adquiriendo más experiencia, podrás ir directamente de la casilla 1 del diagrama de flujo a las instrucciones correspondientes. No obstante, al principio tendrás que escribir previamente la versión detallada del diagrama.

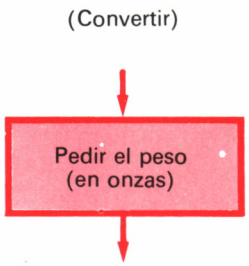
Aquí tenemos la versión detallada de la casilla 1:



Al examinar esta versión detallada, notarás el mayor grado de correspondencia entre ésta y las instrucciones en BASIC. Además, verás que hemos introducido una *comprobación de validez* para ELECCION$. No vamos a suponer que el usuario vaya a cooperar con nosotros siempre y escribir «V» o «T». Averígualo por tu cuenta. *Recuerda:* cada vez que solicitas información mediante un INPUT, has de validar el dato o los datos en cuestión.

El resto del programa es más sencillo. Programemos la segunda casilla:





Las instrucciones correspondientes son:

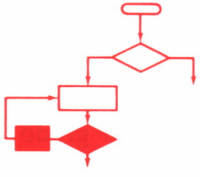
```
300  REM * * * CONVERSION DE UN VALOR * * *
310  PRINT "ESCRIBE EL PESO EN ONZAS...";
320  INPUT POZ
```

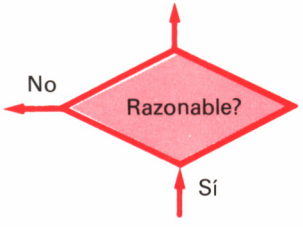
Alternativamente, podríamos escribir:

```
310  INPUT "ESCRIBE EL PESO EN ONZAS..."; POZ
```

En este caso yo decidí separar las instrucciones de PRINT y de INPUT para demostrar con más claridad la correspondencia entre las líneas del programa y el diagrama de flujo. Las dos versiones, sin embargo, pueden usarse indistintamente.

Sigamos adelante. La casilla 3 es:





Su equivalente en el programa es:

```
330  IF POZ<0 THEN GOTO 310
340  REM EL PESO HA DE SER POSITIVO
350  REM PODEMOS AÑADIR AQUI UN MENSAJE EXPLICITO
```

El equivalente de la casilla 4 es:

```
360  PG = POZ * 28.35
```

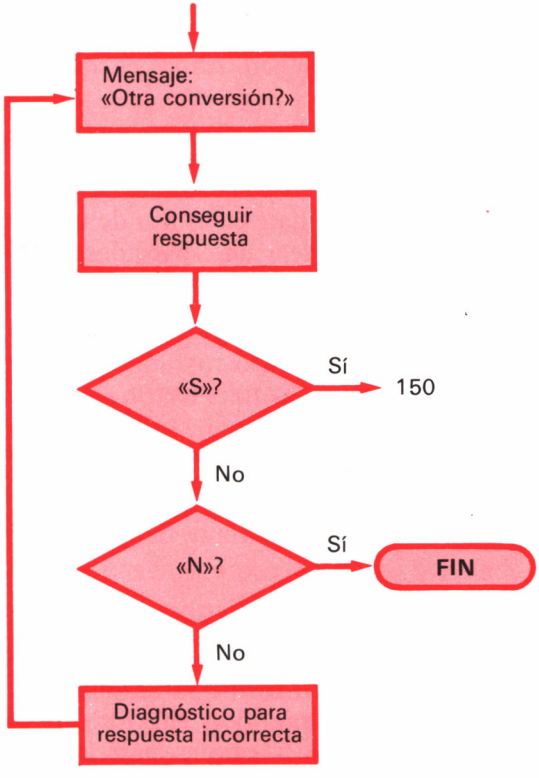
Y para la casilla 5:

```
370  PRINT POZ; "ONZAS EQUIVALEN A"; PG; "GRAMOS"
```

Y para la casilla 6:

```
410  PRINT "QUIERE EFECTUAR OTRA CONVERSION? (S)I O
     (N)O:"
420  INPUT OTRO$
430  IF OTRO$ = "S" THEN GOTO 150
440  IF OTRO$ = "N" THEN END
450  REM INPUT NO HA SIDO NI S NI N. INFORMAR AL
     USUARIO
460  PRINT "S O N POR FAVOR"
470  GOTO 380
```

Si lo anterior te parece difícil de entender, aquí tienes su equivalente en diagrama de flujo:



Examinemos ahora la mitad derecha del diagrama de la figura 9.1. A continuación tenemos el equivalente de la casilla 7:

```
520 PRINT "VOY A VISUALIZAR UNA TABLA DE CONVERSION"
530 PRINT "DE ONZAS A GRAMOS"
540 PRINT "DIGAME EL NUMERO MAXIMO DE ONZAS:";
550 INPUT MAX
```

Y para la casilla 8:

```
560 REM EL VALOR DE MAX HA DE SER IGUAL O MAYOR A 1.
    SI NO, SE RECHAZA
570 IF MAX<1 THEN GOTO 540
```

Para mayor claridad, saltemos una línea en la pantalla antes de imprimir la tabla:

```
580 PRINT
```

Recuerda: es una instrucción vacía. Imprime una línea en blanco. Ahora el equivalente de la casilla 9:

```
590 PRINT "ONZAS", "GRAMOS"
```

Fíjate en el uso de una coma en lugar del punto y coma para crear un espaciado agradable de las columnas.

Y ahora la casilla 10:

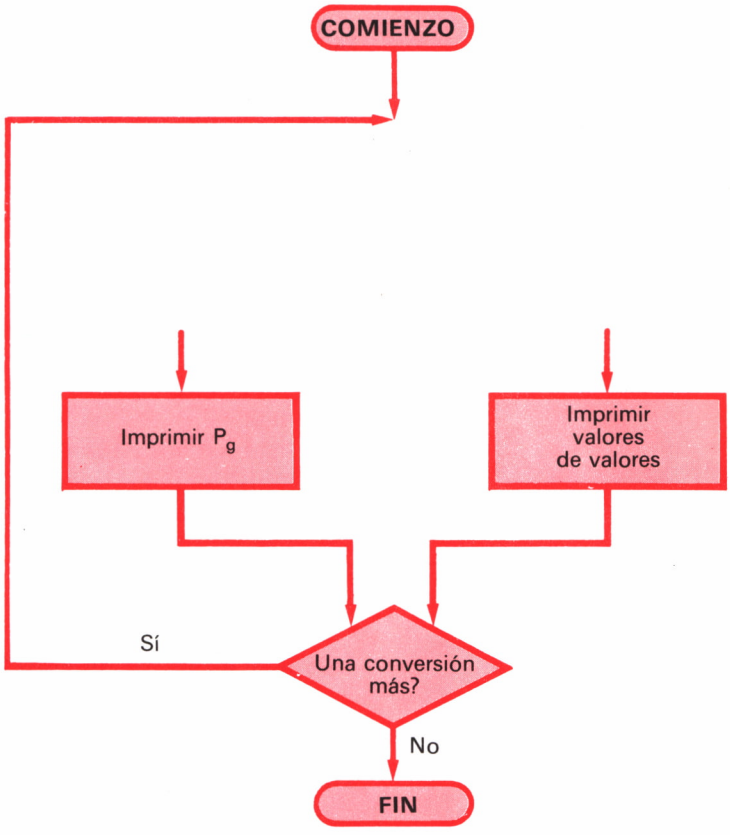
```
600 I = 1
610 PRINT I, I * 28.35
620 I = I + 1
630 IF I<= MAX GOTO 610
```

Por último, tenemos la casilla 11:

```
650 GOTO 380
```

La casilla 11 es igual a la casilla 6, así que nos permite simplificar el programa con un salto (GOTO) a las instrucciones para la casilla 6.

La modificación correspondiente del diagrama sería:



Este es el programa completo:

```
100  REM * * * CONVERSION DE ONZAS A GRAMOS * * *
110  REM ESTE PROGRAMA PUEDE REALIZAR UNA
     CONVERSION DIRECTA
120  REM O VISUALIZAR UNA TABLA DE VALORES
130  REM PRIMERO, DETERMINAR TIPO DE CONVERSION:
     DIRECTA O DE TABLA
140  PRINT "VOY A CONVERTIR ONZAS A GRAMOS"
150  PRINT "SI DESEA CONVERTIR UN VALOR DIRECTAMENTE,
     ESCRIBA V."
160  PRINT "SI DESEA VER UNA TABLA DE VALORES, ESCRIBA
     T."
170  PRINT "SU ELECCION (V O T)";
180  INPUT ELECCION$
190  IF ELECCION$="V" THEN GOTO 300
200  REM LINEA 300 ES LA CONVERSION DE UN VALOR
210  IF ELECCION$ = "T" THEN GOTO 500
```

```
220 REM LINEA 500 ES LA CONVERSION DE UNA TABLA
230 REM SI EL CARACTER NO FUE NI V NI T, EL INPUT NO ES
    VALIDO
240 PRINT "V O T POR FAVOR:";
250 GOTO 170
260 REM
270 REM
300 REM * * * CONVERSION DE UN VALOR * * *
310 PRINT "ESCRIBA EL PESO EN ONZAS...";
320 INPUT POZ
330 IF POZ<0 THEN GOTO 310
340 REM EL PESO HA DE SER POSITIVO
350 REM PODEMOS AÑADIR UN MENSAJE EXPLICITO AQUI
360 PG = POZ * 28.35
370 PRINT POZ; "ONZAS EQUIVALEN A"; PG; "GRAMOS"
380 REM
390 REM * * * RUTINA DE SALIDA * * *
400 PRINT
410 PRINT "QUIERE EFECTUAR OTRA CONVERSION? (S)I O
    (N)O:"
420 INPUT OTRO$
430 IF OTRO$ = "S" THEN GOTO 150
440 IF OTRO$ = "N" THEN END
450 REM INPUT NO HA SIDO NI S NI N. INFORMAR AL
    USUARIO
460 PRINT "S O N POR FAVOR"
470 GOTO 380
480 REM
490 REM
500 REM * * * CONVERSION DE UNA TABLA * * *
510 REM SOLICITAR LIMITE SUPERIOR
520 PRINT "VOY A VISUALIZAR UNA TABLA DE CONVERSION"
530 PRINT "DE ONZAS A GRAMOS"
540 PRINT "DIGAME EL NUMERO MAXIMO DE ONZAS:";
550 INPUT MAX
560 REM EL VALOR DE MAX HA DE SER IGUAL O MAYOR A 1.
    SI NO, SE RECHAZA
570 IF MAX<1 THEN GOTO 540
580 PRINT
590 PRINT "ONZAS", "GRAMOS"
600 I = 1
610 PRINT I, I * 28.35
620 I = I + 1
630 IF I<= MAX GOTO 610
640 REM EL VALOR DE I AHORA ES > MAX. FINAL DE LA
    TABLA
650 GOTO 380
660 END
```

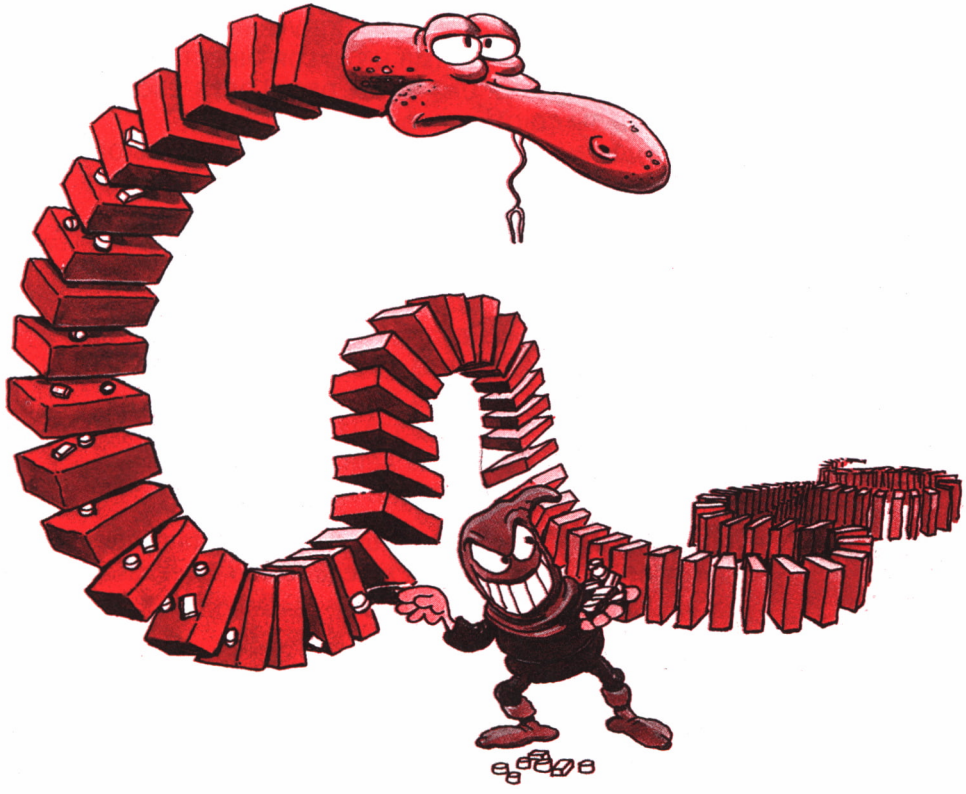
Se han incorporado algunas mejoras al programa. Por ejemplo, se han incluido varias instrucciones REM para mayor facilidad de lectura (véanse las líneas 260, 270, 300, 380, 480, 490 y 500).

Se podrían incorporar algunas modificaciones más. Por ejemplo, las instrucciones entre las líneas 600 y 630 podrían ser reemplazadas por un bucle FOR... NEXT. Esto contribuiría a la facilidad con que se lee el programa, pero no merece el esfuerzo que conlleva.

Recuerda: cualquier cambio que se hace a un programa que ya funciona puede causar nuevos fallos. Cambia un programa únicamente si se consigue una mejora notable.

Tenemos un programa terminado, así que vamos a probarlo.

Recuerda: ¡cualquier cambio que hagas puede provocar más fallos!



Comprobación

Ejecutemos el programa. A continuación tenemos el resultado de una prueba:

```
RUN
VOY A CONVERTIR ONZAS A GRAMOS
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? 3
  3 ONZAS EQUIVALEN A 85.05 GRAMOS
```

Y aquí tenemos otra prueba:

```
QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O:? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? T
VOY A VISUALIZAR UNA TABLA DE CONVERSION
DE ONZAS A GRAMOS
DIGAME EL NUMERO MAXIMO DE ONZAS:? 4

ONZAS       GRAMOS
  1           28.35
  2           56.7
  3           85.05
  4           113.4
```

Parece que el programa funciona correctamente tanto en el caso de una conversión sencilla como en el caso de una tabla de valores.

Intentemos engañar al programa:

```
QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O:? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? D
V O T POR FAVOR: SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? 7
  7 ONZAS EQUIVALEN A 198.45 GRAMOS
```

Comprobemos la opción de repetir la operación:

```
QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O;? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? 85
  85 ONZAS EQUIVALEN A 2409.75 GRAMOS

QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O:? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? 2.5
  2.5 ONZAS EQUIVALEN A 70.875 GRAMOS

QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O:? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? 3.1
  3.1. ONZAS EQUIVALEN A 87.885 GRAMOS
```

¿Intentamos hacer otra trampa? Venga:

```
QUIERE EFECTUAR OTRA CONVERSION? (S)I O (N)O:? S
SI DESEA CONVERTIR UN VALOR DIRECTAMENTE, ESCRIBA V.
SI DESEA VER UNA TABLA DE VALORES, ESCRIBA T.
SU ELECCION (V O T)? V
ESCRIBA EL PESO EN ONZAS...? -5
ESCRIBA EL PESO EN ONZAS...? □
```

Pues parece que funciona bien. De todas maneras, deberías probarlo varias veces antes de darte por satisfecho.

En el caso de este programa hemos tenido mucho cuidado, ¡y mucha suerte! El programa nos ha funcionado como era debido a la primera.

Resumen

En este capítulo hemos visto todos los pasos necesarios para escribir un programa que resuelva un problema dado. Ahora sería muy productivo si cerrases el libro, hicieras un diagrama de flujo, y te pusieras a convertirlo en un programa funcional.

La clave del éxito en la programación es la práctica constante.

Ejercicios

9.1. Añade al programa de este capítulo la opción de convertir gramos a onzas.

9.2. Amplía el programa para incluir la conversión de distancia, siendo:

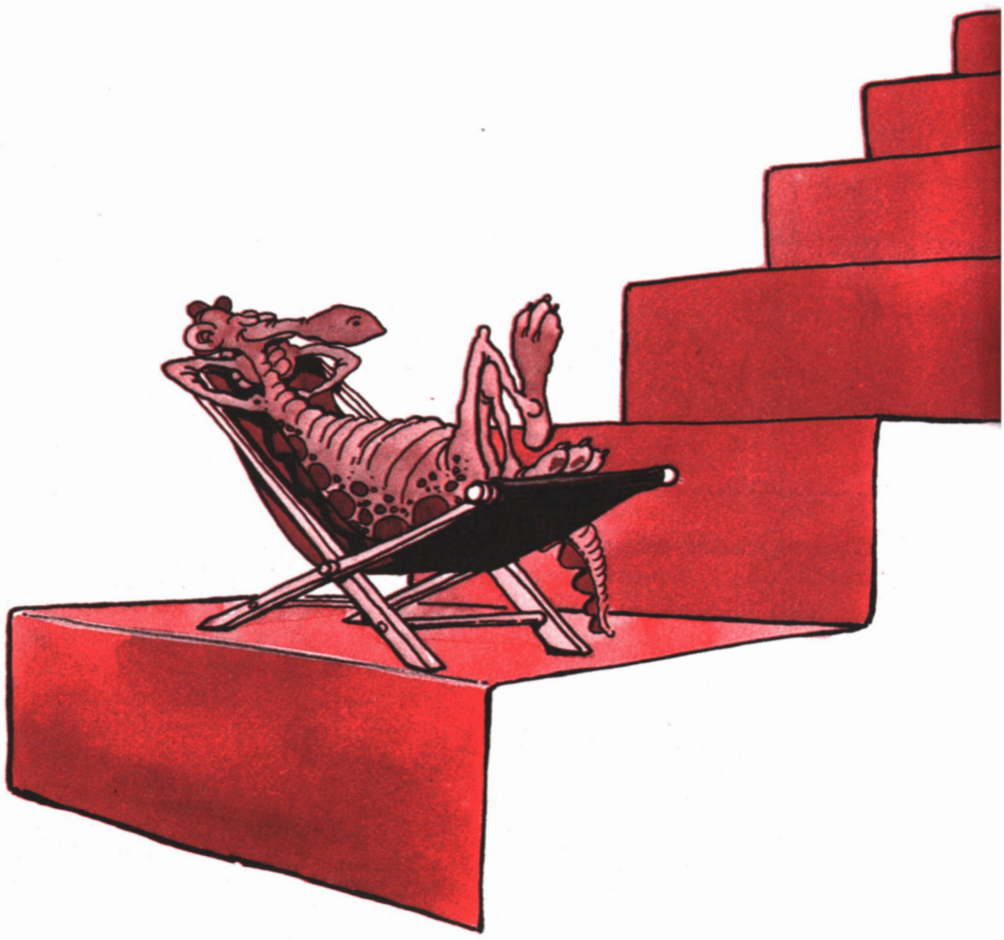
1 metro = 39.37079 pulgadas
1 km = 0.62138 millas
1 pulgada = 25.3995 mm
1 pie = 30.479 cm
1 yarda = 0.91438 m
1 milla = 1609.3149 m

9.3. Amplía el programa para incluir la conversión de temperatura:

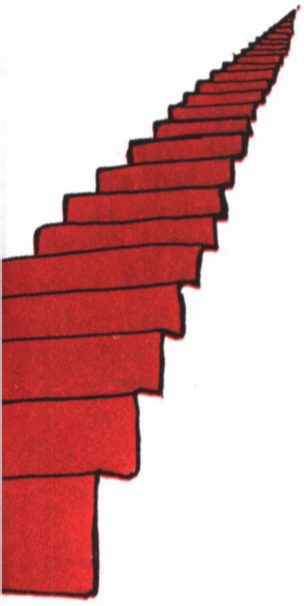
$C = (F - 32) \times 5/9$
$F = (9/5) \times C - 32$

9.4. Sugiere otras modificaciones para mejorar o facilitar la lectura de nuestro programa.

El sig



uiente paso



10

Has aprendido a escribir programas en BASIC por tu cuenta. En este capítulo vamos a examinar el siguiente paso que puedes dar para mejorar tus capacidades y conocimientos de programación. Repasaremos lo que se puede hacer con el lenguaje BASIC, y luego describiremos otros conocimientos y procedimientos que te ayudarán a crear programas más complejos con mayor facilidad.

Lo que puedes lograr con el BASIC

Se puede escribir un programa en BASIC para automatizar la mayoría de las tareas, a no ser que requieran unos cálculos matemáticos de gran precisión, la realización de decisiones complejas, o una respuesta muy rápida (como en el caso del control de procesos en tiempo-real). Encontrarás que el BASIC se presta bien a las aplicaciones simples de negocio, tales como el proceso de datos, las listas de direcciones para envíos y los cálculos financieros más comunes. Con ciertas ampliaciones del lenguaje, el BASIC también sirve para los gráficos y los juegos.

Otras áreas de aplicación típicas para el BASIC incluyen la enseñanza asistida por ordenador (EAO), el mantenimiento de archivos personales y de negocio, los cálculos matemáticos y técnicos de un grado limitado de precisión y muchas aplicaciones más. Las aplicaciones generalmente se ven limitadas más que nada por las habilidades que posee el programador.

Con los conocimientos que has adquirido hasta ahora deberás ser capaz de escribir una amplia variedad de programas en BASIC. No obstante, y según vayas progresando, querrás ampliar tus conocimientos y emplear algunas herramientas de la programación que sean más potentes y más cómodas. Esto nos lleva al tema de las próximas secciones de este capítulo.

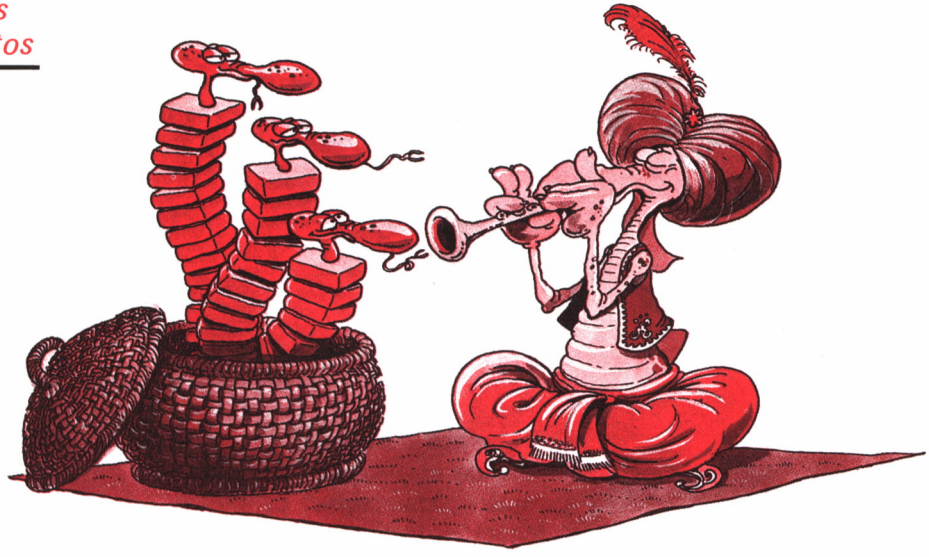
Mejora tus conocimientos

Existen tres pasos fundamentales que puedes dar para incrementar tus habilidades como programador en BASIC:

1. Obtener más experiencia.
2. Obtener mejores conocimientos de todos los recursos de tu versión concreta de BASIC.
3. Aprender más técnicas de programación.

Veamos estos pasos, uno por uno.

Aumenta tus conocimientos



Más experiencia

La clave de la eficacia en la programación es la práctica: escribe todos los programas que puedas, y haz que funcionen perfectamente. Desarrolla buenos hábitos de programación, siguiendo todos los consejos que se han presentado en este libro. Si tus programas funcionan de forma consistente después de tan sólo unos cuantos intentos, lo más probable es que estés en el buen camino para llegar a ser un programador eficaz y disciplinado. Si no funcionan bien, vigila tus hábitos de programador, o quizá vuelve a leer algunas partes de este libro y luego practica más. *Recuerda:* no hay forma de llegar a ser buen programador a no ser que escribas muchos programas. Este libro te ayudará a comenzar, pero en ningún momento podrá sustituir lo que sólo la experiencia puede dar.

Practica «a tope.»

Características específicas del BASIC

Cada intérprete de BASIC ofrece ciertas capacidades específicas, incluyendo las instrucciones, los comandos, abreviaturas, ampliaciones del «BASIC estándar» (tales como los gráficos y el sonido) y un entorno operativo (que incluye los comandos de almacenaje en disco o cassette, manejo de archivos, un programa editor y más cosas). Puedes mejorar tus

habilidades de programación mediante el estudio de estas características y posibilidades adicionales. En la próxima sección describiremos las instrucciones adicionales de BASIC que se encuentran en la mayoría de los intérpretes. Las características, como los gráficos, el sonido, y archivos son particulares de tu ordenador y su intérprete. Ganarás mucho a través de su aprendizaje.

Técnicas adicionales

Una vez que desarrolles programas más extensos (por ejemplo, de más de una página), vas a querer aprender las técnicas normales para resolver problemas corrientes, tales como la ordenación de elementos, así como su clasificación, el formato de los datos, sistemas de archivo y la creación de estructuras de información. Se discuten estos temas a fondo en los textos y manuales de programación.

Más sobre el BASIC

Cada intérprete de BASIC ofrece una serie de características bastante «estandardizadas», además de muchas ampliaciones que son peculiares del intérprete en cuestión. Las características que tienen en común la mayoría de los intérpretes de BASIC incluyen todo lo que hemos estudiado hasta ahora, además de seis clases de instrucciones adicionales. Vamos a examinar brevemente cada una de estas clases, que posteriormente podrás estudiar con más detalle en otros libros más avanzados. Estas seis clases son:

1. *Las funciones:* Las funciones pueden ser expresiones que están incorporadas en el lenguaje o bien son definidas por el usuario, operan sobre una variable determinada y realizan un cálculo o una acción específica. El lenguaje BASIC siempre incorpora algunas funciones (tales como ABS, COS, EXP, IN, FIX, RND, SGN, SIN, SQR o TAN). Estas funciones realizan de forma automática varias tareas comunes. Otras funciones pueden ser definidas por el usuario. Examinemos primero algunas de las funciones que típicamente se hallan incorporadas en el intérprete.

— La función INT calcula el valor entero de un número decimal, eliminando la parte fraccional del número. Por ejemplo, la función INT (1.234) da como resultado el valor de 1.

— Asimismo, la función ABS calcula el valor absoluto de un número; por ejemplo, ABS (−5.2) da el valor de 5.

Estudia todo lo posible.

— La función SQR calcula la raíz cuadrada (*square root,* en inglés) de un número, de manera que SQR(4) tiene como resultado el valor de 2.

El usuario mismo puede definir otras funciones. Una de estas funciones consiste, esencialmente, en una fórmula creada por el programador al que se ha puesto un nombre, que opera sobre una variable, y que se utiliza en el programa en distintas ocasiones. De esta manera, cada vez que se nombra a la función, se calcula la fórmula en cuestión usando el valor actual de la variable. Constituye una notación abreviada muy cómoda de usar.

A continuación tenemos un ejemplo de una función definida por el usuario que calcula el 2 por 100 del valor de X:

```
10  DEF FNA(X) = X * 2/100
```

Y ahora un ejemplo del uso de esta función:

```
20  PRINT FNA(50)
```

Lo que nos da como resultado el valor de 1.

Veamos las instrucciones con más detalle. Las letras FN significan función, por lo que FNA es el nombre de la función A. La X es la variable de «relleno». No tiene ningún valor cuando se escribe la DEFinición; el valor real o la variable se coloca en su lugar a la hora de emplear la función, como en nuestro ejemplo de FNA(50).

2. *Las subrutinas:* Una subrutina es una serie de instrucciones dentro de un programa en BASIC que tiene su propia «dirección» (número de línea) a la que el programa se puede dirigir todas las veces que sea requerido. Cada vez que el programa se dirige a esta «dirección» durante el curso de la ejecución del programa, se ejecuta la serie completa de instrucciones que se encuentran en la subrutina, terminando con la instrucción RETURN («volver») que devuelve la ejecución a su punto de partida. Las subrutinas son especialmente útiles para la ejecución de aquellos segmentos del programa que han de repetirse frecuentemente, y evitan la necesidad de repetir muchas líneas de instrucción.

Aquí tenemos un ejemplo:

```
500  REM ESTA SUBRUTINA SACA MEDIAS
510  PRINT "VOY A CALCULAR LA MEDIA DE A Y B"
520  PRINT "A = "; A; "B = "; B
530  PROMEDIO = (A + B)/2
540  PRINT "LA MEDIA ES "; MEDIA
550  RETURN
```

Siempre que a las variables A y B se les haya asignado algún valor previamente, el programa principal podrá dirigirse a la subrutina mediante una instrucción del tipo:

```
50  GOSUB 500
```

(*Nota:* GOSUB viene del inglés GO to SUBroutine, o «ir a la subrutina».) El programa puede volver a dirigirse a la subrutina:

```
170  GOSUB 500
```

dando un resultado diferente, si es que el valor de A y B ha cambiado también. La utilización de las subrutinas sirve para simplificar un programa, hacerlo más corto y ahorrar tiempo al programador. Hasta es posible crear una biblioteca de subrutinas de más uso para su uso en más de un programa. Sin embargo, al hacer esto hay que tener cuidado con los nombres de las variables y los números de línea.

3. *Operadores de cadenas:* Estos operadores permiten que se manipulen secciones de texto con comodidad al operar sobre «cadenas» de caracteres. Estas operaciones pueden incluir la modificación de las cadenas, su medida, la conexión entre ellas, su separación, la inserción de texto a la izquierda, la derecha o en cualquier posición dentro de la cadena, la sustitución de caracteres o la comparación de cadenas de caracteres. Como se puede apreciar, estos operadores son muy útiles para las aplicaciones en el procesamiento de palabras y textos.

4. *La estructura de datos:* El lenguaje BASIC admite variables con uno o dos subíndices. Estas variables corresponden a los vectores y matrices matemáticas. El uso de estas variables con subíndice te permite estructurar las variables en listados o agrupaciones similares, de manera que se puede referir el programa a cualquier elemento que forma parte de la lista. Por ejemplo, los distintos elementos de una lista denominada CLIENTE tendrían nombres como CLIENTE(1), CLIENTE(2), etc. Si la lista tuviera diez elementos, se podrían listar todos ellos al programar:

```
50  FOR N = 1 TO 10
60  PRINT CLIENT(N)
70  NEXT N
```

Se pueden comparar dos elementos consecutivos al escribir:

```
100  IF CLIENTE(K) < CLIENTE (K + 1) THEN 500
```

Lo que representa una gran ventaja en la programación.

5. *Los archivos:* Toda versión de BASIC que se base en los diskettes incorpora las características necesarias para almacenar y/o recoger datos de unos archivos sobre diskette. Estas posibilidades varían según tu equipo y el intérprete, y vienen explicadas en la documentación suministrada por el fabricante.

6. *Otros recursos:* Otros recursos que son típicos de muchos intérpretes buenos pueden incluir las siguientes instrucciones: ON... GOTO, READ... DATA y PRINT USING. Deberías explorar estas instrucciones en el manual o libro de referencia de tu intérprete, ya que facilitan mucho la programación.

También es frecuente que el intérprete admita algunas abreviaturas; por ejemplo, a veces es posible escribir:

```
?25 * 32
```

en lugar de la versión equivalente con la instrucción PRINT, lo cual es una ventaja a la hora de emplear tu equipo como una calculadora.

Otra mejora que a veces se encuentra es un modo de operación de doble precisión, el cual permite mejorar la precisión de los cálculos matemáticos. Asimismo, puede existir un tabulador (TAB) para facilitar la impresión de tablas sobre la pantalla.

Por último, recuerda que existen comandos especiales que varían con el intérprete para generar gráficos, realizar sonidos especiales o para facilitar el editaje; por ejemplo, existen comandos que te permiten modificar el programa y sus archivos correspondientes, a la vez que ayudan en la ejecución y la depuración de un programa. Estos últimos comandos pueden incluir la posibilidad de controlar la pantalla, fijar puntos en los que se parará automáticamente el programa, seguir el valor de las variables que se han elegido durante la ejecución, y la deceleración y aceleración de la velocidad de visualización en pantalla.

Conclusión

El objetivo de este libro ha sido el de instruirte rápida y eficazmente en la manera de escribir tus primeros programas en BASIC. Si te has quedado «enganchado» por el BASIC, ahora querrás saber más. El próximo paso que has de dar, entonces, es el de completar todos los ejercicios del libro y luego desarrollar algunos programas tuyos propios.

Según vayas progresando, querrás aprender técnicas más avanzadas. Para tal fin, se incluye una lista de libros de programación al final de este libro.

Me alegraría pensar que estás de acuerdo conmigo cuando digo que el aprendizaje del BASIC puede ser tanto fácil como agradable, y espero que ahora desees seguir adelante en tus estudios. De la misma manera espero que no tengas apuros a la hora de dirigirte a mí con las sugerencias que tengas para futuras mejoras para este libro.

APENDICE A

Respuestas a los ejercicios

2

2.2.

```
10 PRINT "AAAAA"
20 PRINT "BBBB"
30 PRINT "CCC"
40 PRINT "DD"
50 PRINT "E"
```

2.4. *a)* En el BASIC, una etiqueta es un número de línea, que ha de colocarse delante de cada línea que forma parte de un programa.

b) La modalidad de ejecución diferida es la que se emplea normalmente a la hora de introducir un programa en la memoria del ordenador. Las instrucciones de BASIC se escriben con una numeración de las líneas y son memorizadas por el ordenador para ejecutarse posteriormente.

c) La modalidad de ejecución inmediata es aquella en la que se escribe una instrucción sin un número de línea, de forma que se ejecuta al instante. También se denomina modalidad de calculadora.

d) Una instrucción vacía es aquella que no provoca ninguna acción por parte del ordenador. Por lo general, consiste solamente en un número de línea o etiqueta, sin que exista en ella nada más. A veces incluye la instrucción REM.

e) El cursor es un símbolo especial que aparece en la pantalla (como, por ejemplo, un cuadro o subrayado) que indica la posición actual en la pantalla. Generalmente tiene el formato de una luz intermitente para facilitar su localización.

f) La tecla de control (o CTRL), cuando se pulsa simultaneamente con una tecla alfabética, emite un comando especial. Las teclas de control hacen que sea más fácil enviar al ordenador los comandos que se usan con frecuencia, al ser necesario pulsar tan sólo dos teclas.

g) Un cuadro independiente de teclas es un teclado especial, que generalmente consiste en un juego reducido de teclas para fines especiales que se posiciona a la derecha del teclado principal. Normalmente se emplea para cálculos numéricos y movimientos del curso.

h) Una palabra reservada es aquella que tiene un significado específico en el lenguaje BASIC, por lo que no se puede emplear como una variable en un programa.

i) El símbolo > es un carácter o mensaje generado por un programa que indica que el programa espera a que el usuario introduzca información. La mayoría de las versiones de BASIC utilizan este símbolo para significar «escriba la siguiente instrucción». El símbolo «?» significa «introduzca información.»

2.6. Sí, pero constituye una manera bastante torpe de hacerlo, ya que si se desea ejecutar el programa de nuevo, será necesario escribir todas las instrucciones de nuevo. Además, las bifurcaciones condicionales (que veremos más adelante) no funcionan correctamente en la modalidad inmediata.

2.8. Sí, el BASIC colocará cada instrucción en el lugar apropiado dentro del programa, de manera que todas las etiquetas estén en orden numérico.

2.10. No, es preciso escribir:

```
PRINT "EJEMPLO"
```

2.12. Para borrar la línea 20 de un programa se escribe una instrucción vacía con la etiqueta 20.

3

3.2.

```
PRINT 1 + (1/2) * (1 / (1 + (1/2)))
o
PRINT 1 + .5 / (1 + .5)
```

3.4.

```
PRINT 100 * 1.6
```

3.6.

```
PRINT 350 / 55
```

4

4.1.

```
10  INPUT A, B, C, D
20  SUMA = A + B + C + D
30  MED = SUMA / 4
40  PROD = A * B * C * D
50  PRINT SUMA, MED, PROD
60  END
```

4.2.

a = no	e = sí	i = sí
b = sí	f = sí	j = no
c = no	g = no	k = sí
d = sí	h = no	l = sí

4.4.

```
10  PRINT "DAME EL NOMBRE DE UN OBJETO ";
20  INPUT O$
30  PRINT "DAME EL NOMBRE DE UN MUEBLE ";
40  INPUT M$
50  PRINT "DAME EL NOMBRE DE UN AMIGO ";
60  INPUT A$
```

```
70  PRINT
80  PRINT "TIENE TU AMIGO "; A$; " UN "; O$; " ENCIMA
    DE "; M$; "?"
90  END
```

4.6. Las asignaciones b, c, y f son válidas.

5

5.2.

a = sí d = sí
b = sí e = no
c = sí f = no

5.4.

```
INPUT "TU NOMBRE: "; NOMB$
INPUT "CUANTAS SON 2 + 3?"; C
INPUT "LOS ULTIMOS DOS NUMEROS SON..."; A, B
```

5.6. A = 3

6

6.1. La instrucción IF permite que el programa tome decisiones, cambiando así su comportamiento según las variaciones en los datos introducidos o calculados.

6.3.

a = sí e = no
b = sí f = sí
c = sí g = sí
d = sí

6.4. Sí.

6.5. Un bucle repite la ejecución de una parte de un programa. Para evitar un bucle sin salida (uno que

no deja de repetirse), siempre ha de incluirse una comprobación detro del bucle, la cual al cumplirse da lugar a que el programa salga del bucle.

7

7.2.

```
10  INPUT "HORAS, MINUTOS :"; HORAS, MINUTOS
20  REM COMPROBAR LOS DATOS
30  IF HORAS> = 0 AND HORAS < 72 AND MINUTOS > =
    0 AND MINUTOS < 60 THEN 70
40  PRINT "INTENTELO DE NUEVO"
50  GOTO 10
60  REM IMPRIMIR UNA SERIE DE LA LETRA H
70  IF HORAS = 0 THEN 110
80  FOR A = 1 TO HORAS
90  PRINT "H";
100 NEXT A
110 PRINT
120 REM IMPRIMIR UNA SERIE DE LA LETRA M
130 IF MINUTOS = 0 THEN 170
140 FOR A = TO MINUTOS
150 PRINT "M";
160 NEXT A
170 PRINT
180 END
```

7.4. Se puede saltar a dentro de un bucle que no sea gobernado por una instrucción de FOR... NEXT.

7.6.

```
10  PRINT "PROGRAMA PARA SUMAR TODOS LOS
    NUMEROS ENTEROS IMPARES"
20  PRINT "HASTA LIMITE DEFINIDO POR EL USUARIO"
30  INPUT "ENTERO IMPAR MAXIMO PARA
    SUMAR "; NUM
40  REM COMPROBAR VALIDEZ DE LOS DATOS
50  REM INT (NUM/2) < > NUM/2 COMPRUEBA SI ES
    IMPAR (VER CAP. 10)
60  IF NUM > 0 AND NUM < 10000 AND INT (NUM/2) <
    > NUM/2 THEN 90
70  PRINT "NUMERO INACEPTABLE... INTENTALO DE
    NUEVO"
80  GOTO 30
90  REM HACER LA TABLA
```

```
100 PRINT "NUMERO", "SUMA"
110 PRINT
120 FOR N = 1 TO NUM STEP 2
130 SUMA = SUMA + N
140 PRINT N, SUMA
150 NEXT N
160 END
```

7.8.

```
10 INPUT "PORCENTAJE DE LA TASA DE
   IMPUESTOS "; IMP
20 IF IMP < 1 OR TAX > 100 THEN 10
30 PRINT "PRECIO", "IMP", "PRECIO + IMP"
40 FOR PRECIO = 1 TO 100
50 PRINT PRECIO, PRECIO * IMP/100, PRECIO + PRECIO *
   IMP/100
60 NEXT PRECIO
70 END
```

8

8.2. La codificación es solamente un paso en la programación. La programación incluye los pasos del diseño del algoritmo, la creación del diagrama de flujo, la codificación, la depuración y la documentación.

8.4. Se puede trazar una variable mediante la inserción de instrucciones PRINT para mostrar el valor de la variable en los puntos críticos del programa. A veces, el mismo intérprete facilita un comando específico para este fin: TRACE.

8.6. El diagrama de flujo es un diagrama simbólico que muestra de forma gráfica la secuencia de lo que ocurre durante la ejecución del programa.

8.8. Los programas escritos con claridad presentan la doble ventaja de ser fáciles de entender y, por tanto, de modificar. Esto permite que tanto el programador que lo ha escrito como otros programadores puedan cambiarlo sin dificultad.

APENDICE B

Algunas palabras reservadas en el lenguaje BASIC

Esta lista te ayudará a evitar el empleo de nombres ilegales para las variables, aunque es muy probable que tu intérprete no reconozca algunas de ellas como palabras reservadas.

ABS
AND
ARCCOS
ARCSIN
ARCTAN
AT
AUTO
BREAK
CALL
CHR$
CLEAR
CLOCK
CLOSE
CLS
COLOR
CON
CONT
COPY
COS
DATA
DATE
DEFDBL
DEFFN
DEFOMT
DEFSNG
DEFSTR
DEFUSR
DELETE
DIM
DSP
EDIT
ELSE
END
ENTER
EOF
ERR
ERROR
EXP
FIELD
FIX
FN
FOR
FORMAT
FREE
FUNCTION
GET
GOSUB
GOTO
GRAPHICS
HUN
IF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LET
LSET
LEN
LINE
LIST
LN
LOAD
LOC
LOG
LPRINT
MEM
MERGE
MID$
MKD$
MKI$
MKS$
NEW
NEXT
NOT
ON
OPEN
OR
OUT
PAUSE
PEEK
PLOT
POINT
POKE
POP
POS
POSN
POWER
PRINT
PUT
RANDOM
READ
REM
RENAME
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SET
SGN
SIN
SLOW
SQR
STEP
STOP
STRING$
STR$
TAB
TAN
TEXT
THEN
TIME$
TO
TRACE
TROFF
TRON
UNPLOT
USING
USR
VAL
VARPTR
VERIFY
VLIN
VTAB

APENDICE C

Glosario de términos *

alfanumérico *(alphanumeric):* Conjunto compuesto por todos los caracteres alfabéticos y numéricos; o sea, de A a Z y de 0 a 9.

algoritmo *(algorithm):* Secuencia de los distintos pasos en la solución de un problema dado.

archivo *(file):* Agrupación de datos al que se le ha dado un hombre. Normalmente se almacena un programa en un disco en el formato de un archivo.

asignación *(assignment):* Operación por la que se da cierto valor a una variable. En lenguaje BASIC es indicada mediante el símbolo «=».

BASIC *(Beginners All-purpose Symbolic Instruction Code):* Código de Instrucción Simbólico Polivalente para Principiantes. Un lenguaje de programación de alto nivel diseñado expresamente para que sea fácil de aprender.

bifurcación *(jump):* Un salto dentro del programa; es decir, la ejecución de una secuencia fuera de secuencia.

binario *(binary):* Sistema de números que emplea únicamente el 0 y el 1.

bit *(bit):* Contracción de las palabras inglesas «binary digit». Un bit puede tener el valor de 0 ó 1.

bucle *(loop):* Secuencia de instrucciones que se ejecutan repetidas veces, hasta que se cumpla una condición, siendo ésta normalmente la adquisición de un valor por parte de una variable.

* El término equivalente inglés figura entre paréntesis, cuando sea diferente.

bucle anidado *(nested loop):* Bucle que se encuentra dentro de otro.

bucle sin salida *(endless loop):* Es aquel bucle que no posee una salida programada y que se ejecutará hasta que se interrumpa el programa. Constituye un error cometido frecuentemente por los programadores, cuando el bucle no comprueba ninguna condición para efectuar la salida o cuando esta comprobación siempre tiene un mismo resultado. La salida de un bucle de este tipo requiere el uso de un carácter especial de interrupción, normalmente el CTRL C.

cadena *(string):* Secuencia de caracteres (en diferencia con los *números)*. En el BASIC, varían los nombres de las variables según se refieran a las cadenas o a los números. Por ejemplo, VERBO$ < es una variable de cadena, mientras NUMERO sería una variable numérica.

carga *(loading):* Transferencia de datos o programas a la memoria del ordenador.

circuito integrado *(integrated circuit):* Circuito electrónico realizado sobre un pequeño trozo de silicona y compuesto de numerosos transistores y funciones lógicas.

circuito integrado o micrológico *(chip):* Circuito integrado que reside sobre un pequeño trozo de silicona, montado a su vez sobre plástico o cerámica.

codificación *(coding):* Es proceso de convertir un algoritmo o un diagrama de flujo en una secuencia de instrucciones en lenguaje de programación. Constituye una de las fases de la programación.

comando *(command):* Palabra reservada cuya función tiene que ver primordialmente con el funcionamiento interno del equipo, como puede ser el borrado de pantalla, el arranque de un programa, o el acceso a los ficheros. La especificación de un comando activa a un programa especializado dentro del ordenador con el fin de llevar a cabo una operación.

CPU *(Central Processing Unit):* Módulo electrónico que se encarga de buscar, decodificar y ejecutar en la secuencia correcta las instrucciones almacenadas en la memoria. En la mayoría de los microordenadores, la CPU y la memoria se alojan sobre el mismo circuito impreso o «tarjeta». La CPU utiliza normalmente un microprocesador y algunos otros componentes.

CRT *(Cathode Ray Tube):* Tubo de rayos catódicos, parecido a la pantalla de un televisor.

cursor *(cursor):* Símbolo que se emplea para indicar la posición en la que aparecerá el próximo carácter en la pantalla. Consiste muchas veces en un pequeño cuadro o subrayado intermitente. Existen unas teclas especiales para efectuar los movimientos del cursor sobre la pantalla.

datos *(data):* Los textos y números sobre los que puede trabajar un programa.

depuración *(debugging):* Es el proceso de eliminar los errores de un programa. Puede ser largo y pesado, por lo que no hay que ahorrar esfuerzos a la hora del diseño del programa, con el fin de que tenga el menor número de errores posible.

diagrama de flujo *(flowchart):* Es la representación gráfica de un algoritmo.

disco *(disk):* Medio magnético en el que se pueden almacenar datos y programas. En un disco los programas se organizan de una manera que permite buscarlos por nombre. Los discos tienen gran capacidad de almacenamiento de información, constituyendo el dispositivo de memoria en masa más utilizado con los equipos de microordenadores.

disquette *(diskette):* Disco flexible; es decir, un disco blando de 8″ o 5-1/4″, cuyo diseño facilita el almacenamiento a bajo coste de programas y datos.

editor *(editor):* Programa cuyo diseño facilita la entrada y la modificación de textos, inclusive la eliminación de errores y la modificación de programas durante su introducción.

equipo físico *(hardware):* Todos los componentes físicos incluidos en un equipo, como son el ordenador, los discos y el monitor. Se diferencia del *software* (las instrucciones y los programas).

etiqueta *(label):* Numeración de una línea en BASIC.

expresión *(expression):* Combinación de operandos o variables separados con operadores. Una expresión representa una fórmula y realiza un cálculo específico. Cuando el intérprete evalúa una expresión a la hora de ejecutar el programa, de esta evaluación resulta un valor.

expresión lógica *(logical expression):* Combinación de operandos o variables separados con operadores lógicos (=, >, etc.).

fallo *(bug):* Error de programación. Es preferible evitarlos para no tener que corregirlo después de buscarlos.

gráficos *(graphics):* Los dibujos o imágenes que se visualizan en la pantalla, generados mediante una configuración de pequeños puntos. El empleo del color mejora sustancialmente la presentación de los gráficos.

IC *(Integrated Circuit):* Circuito integrado.

inicialización *(initialization):* Aquella fase dentro de un programa durante la cual se asignan los valores iniciales a las variables. Cada bucle requiere su propia fase de inicialización.

instrucción *(instruction):* Es la orden que al entregarse al intérprete afecta a los datos con los que se está trabajando. (Los comandos, en cambio, no afectan a los datos, sino que realizan funciones internas o facilitan el dieño o uso de un programa.)

instrucción vacía *(empty statement):* Una instrucción que no inicia ninguna acción. Por ejemplo, la instrucción

```
10 REM
```

se puede emplear para crear un hueco en el programa y así hacer que sea más fácil de leer.

interfaz *(interface):* Los circuitos electrónicos que permiten conectar un dispositivo al ordenador. Los discos, las impresoras y los magnetófonos requieren interfaces específicos para su conexión.

intérprete *(interpreter):* Programa que se hace cargo de traducir las instrucciones de un lenguaje de programación (como el BASIC) al lenguaje binario del ordenador y luego ejecutarlas. Una vez que el intérprete haya sido instalado en el ordenador, éste da la impresión de entender las instrucciones en BASIC.

I/O *(Input/Output):* Las comunicaciones que entran y salen del ordenador (la entrada/salida de información).

K: Abreviatura de «kilo», utilizada para indicar la capacidad de memoria del ordenador en bytes (octetos) (IK´1024 bytes).

lenguaje de alto nivel *(high.level language):* Lenguaje de programación cuyo diseño está pensado para facilitar la manera en la que se dan instrucciones al ordenador. El BASIC es un ejemplo de estos lenguajes.

lenguaje de máquina *(machine language):* Lenguaje binario que entiende el ordenador sin interpretación; el conjunto de instrucciones que manipulan la información binaria dentro de la CPU y la memoria.

Logical o **Logicial** *(software):* Conjunto de los programas, incluyendo los del sistema operativo (software de base) y los aplicativos (software aplicativo).

lógico-a *(logical):* Es toda variable o expresión que puede tener un valor de verdadera o falsa.

memoria *(memory):* Medio que almacena información. La memoria interna suele alojarse sobre la misma placa que la CPU y guarda los datos y los programas en forma de bytes. Las unidades más comunes de almacenamiento de memoria en masa son los cassettes y los discos.

microordenador *(microcomputer):* Ordenador cuya unidad central de proceso consiste en un microprocesador.

microprocesador *(microprocessor):* Circuito integrado que realiza gran parte de las funciones de una CPU en un mismo micrológico (chip). Los microprocesadores de hoy en día pueden incorporar hasta decenas de miles de transistores en un sólo micrológico y pueden incorporar inclusive la memoria.

monitor *(monitor):* Un programa de poca extensión, necesario para hacer funcionar al sistema de un ordenador. Recoge los caracteres del teclado, los visualiza en la pantalla y realiza los intercambios básicos de datos

entre el teclado, la pantalla y los periféricos normales. Con este nombre suele designarse también a TRC.

MPU *(Microprocessor Unit):* Un micrológico, o «chip».

no-residente *(non-resident):* Un programa que no reside en la memoria permanente del ordenador (la ROM). Un programa no-residente se almacena en cassette o disquette.

número de coma fija *(fixed point number):* Es aquel número entero en el que la coma decimal se mantiene en posición fija a la derecha del último dígito.

número de coma flotante *(floating point number):* Es un número decimal. Un número fijo de dígitos se emplean internamente para representar cualquier número y, según se aplican las operaciones a ese número, varía la posición de la coma, «flotando» hacia la derecha o la izquierda.

octeto *(byte):* Grupo de ocho bits.

operador *(operator):* Símbolo que representa cualquier operación válida sobre los valores, es decir, + (sumar), * (multiplicar), etc.

operador de relación *(relational operator):* Operador lógico que establece una relación de superior/inferior entre unos valores (>, <, etc.).

ordenador *(computer):* Es el equipo cuyos elementos incluyen una unidad central de proceso, una memoria, interfaces básicos que permiten que se comunique con el mundo a su alrededor, y una fuente de alimentación eléctrica. El equipo puede incluir de forma opcional un teclado, una pantalla y unidades de disquette. Un ordenador posee la capacidad de almacenar y ejecutar programas. Se comunica con el mundo de su alrededor mediante dispositivos de entrada y salida de datos (input/output). El dispositivo más corriente para la entrada de datos es el teclado, mientras el de la salida de datos es la pantalla y, opcionalmente, una impresora. La memoria adicional se suministra normalmente a través de unidades de disco o magnetófonos de cassette.

palabra reservada *(reserved word):* Es toda palabra que posee un valor previamente definido de cara al intérprete. No puede emplearse como nombre de variable.

periférico *(peripheral):* Dispositivo que se conecta a un ordenador. Puede ser, por ejemplo, una impresora, una unidad de disco o un terminal.

precisión doble *(double precission):* Número que tiene el doble del número de dígitos de los que tiene su representación normal. Cada intérprete representa los números con un número fijo de dígitos. La precisión doble se emplea normalmente para cálculos científicos o financieros que manipulan números muy elevados o que realizan cálculos muy complicados.

programa *(program):* Secuencia de instrucciones que pueden ser ejecutadas por un ordenador. Cada programa se escribe en un lenguaje determinado de programación y ha de cargarse en la memoria del ordenador previo a su ejecución.

RAM *(Random Access Memory):* Memoria de acceso directo; aquella porción de la memoria del ordenador en la que es posible tanto almacenar datos temporalmente como leerlos (siendo el resto de la memoria la de ROM).

residente *(resident):* Programa que se aloja permanentemente dentro de la memoria del ordenador (es decir, ROM).

ROM *(Read Only Memory):* Memoria de sólo lectura. Esta memoria no puede ser modificada por el programador. Normalmente contiene parte del monitor o a todo él, y en ocasiones una versión sencilla de intérprete.

RUN *(RUN):* Comando que inicia la ejecución de un programa.

sentencia *(sentence):* Instrucción o comando en BASIC que forma parte de un programa.

sistema operativo *(operating system):* Programa de régimen interno que dispone de las extensas facilidades necesarias para todo tipo de intercambios de datos o proceso de los mismos para emplear los recursos de un ordenador. El sistema operativo maneja los archivos en disco, realiza conversiones de formato y se encarga de iniciar o parar la ejecución de programas.

sintaxis *(syntax):* Es el conjunto de las reglas que gobiernan la validez de las instrucciones de un lenguaje de ordenador. El lenguaje BASIC posee una sintaxis muy sencilla, y el intérprete siempre busca e identifica los errores de sintaxis.

terminal *(terminal):* La constituye la combinación de una pantalla y un teclado, o una impresora y un teclado, usados para comunicarse con el ordenador.

unidad de disco *(disk drive):* Mecanismo empleado para leer datos desde un disco o escribirlos.

variable *(variable):* Ubicación de memoria que tiene un nombre y puede adquirir distintos valores en momentos diferentes. El lenguaje BASIC hace distinción entre las variables de cadena y las numéricas. El nombre de una variable de cadena siempre ha de terminar en «$».

Indice alfabético

OTROS TITULOS PUBLICADOS EN ANAYA MULTIMEDIA

TU PRIMER LIBRO DEL ZX SPECTRUM - J. Dewhirst & R. Tennison.

PROGRAMACION EN BASIC: UN METODO PRACTICO H. Dachslager/M. Hayashi/R. Zucker.

BITS Y BYTES: INICIACION A LA INFORMATICA - Rachelle S. Heller & C. Dianne Martin.

EL ORDENADOR EN EL AULA - Egidio Pentiraro.

ASTRONOMIA: EL UNIVERSO EN TU ORDENADOR (ZX SPECTRUM) - Maurice Gavin.

LIBRO GIGANTE DE LOS JUEGOS PARA ORDENADOR Tim Hartnell.

EL ORDENADOR Y TUS HIJOS - Ray Hammond.

EL ORDENADOR PERSONAL: COMO ELEGIRLO Y UTILIZARLO - Arnoldo Cavalcoli.

Editados por:

EDICIONES ANAYA MULTIMEDIA, S.A.
Villafranca, 22
28028 MADRID

Revistas especializadas

EL ORDENADOR PERSONAL
Ferraz, 11, 3.º
28008 MADRID

MICROS
Ediciones Arcadia
Víctor de la Serna, 4
28016 MADRID

TU MICRO
Avenida Alfonso XIII, 141
28016 MADRID

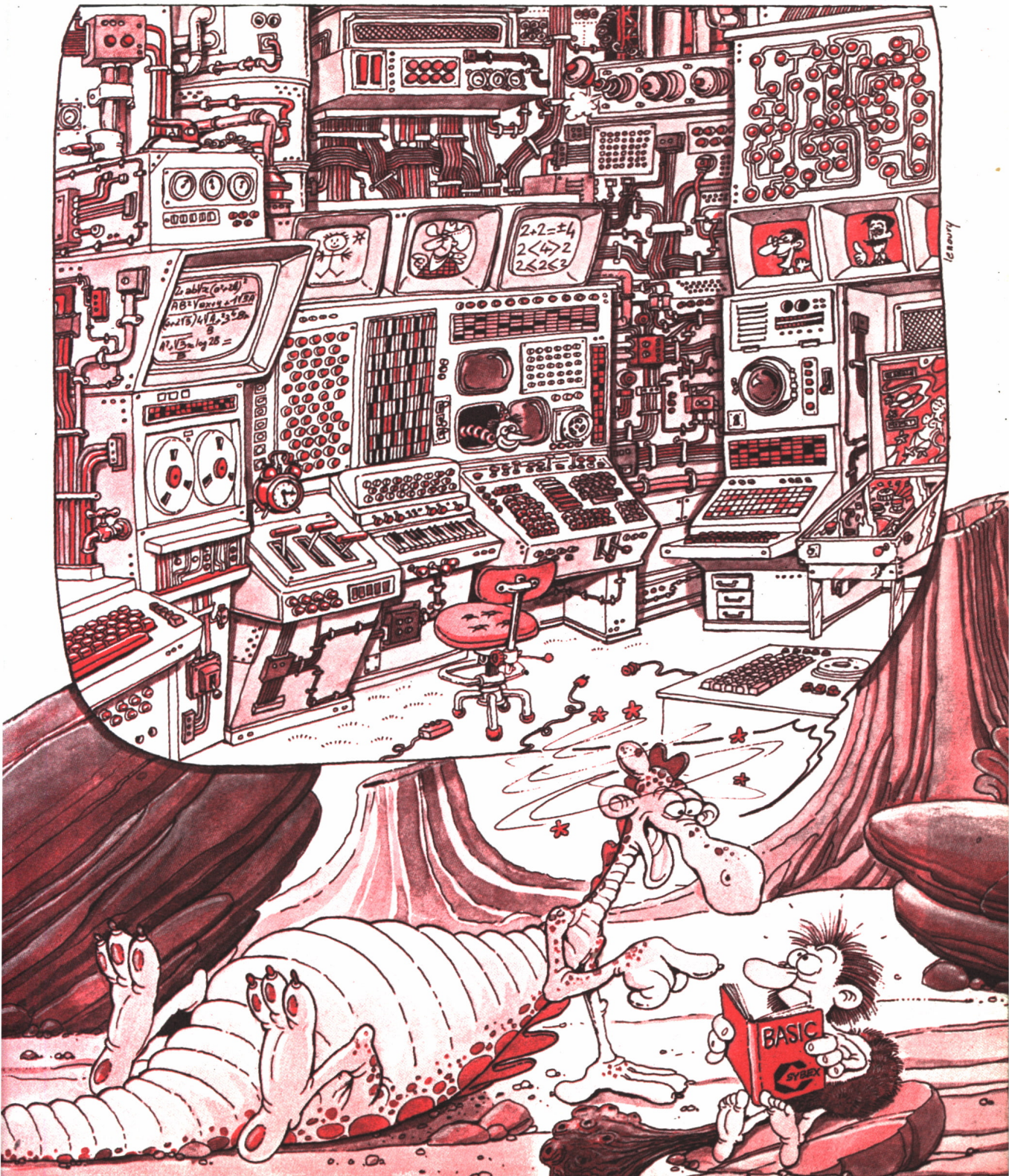
ORDENADOR POPULAR
Ediciones y Suscripciones, S.A.
Jerez, 3
28016 MADRID

INFORMATICA TEST
Haymarket, S.A.
Travesera de Gracia, 17-21
BARCELONA-21

CHIP
Ediciones Arcadia
Víctor de la Serna, 4
28016 MADRID

ZX
Apartado 784
MADRID

¡Ya me he leído todos los libros de Anaya Multimedia!



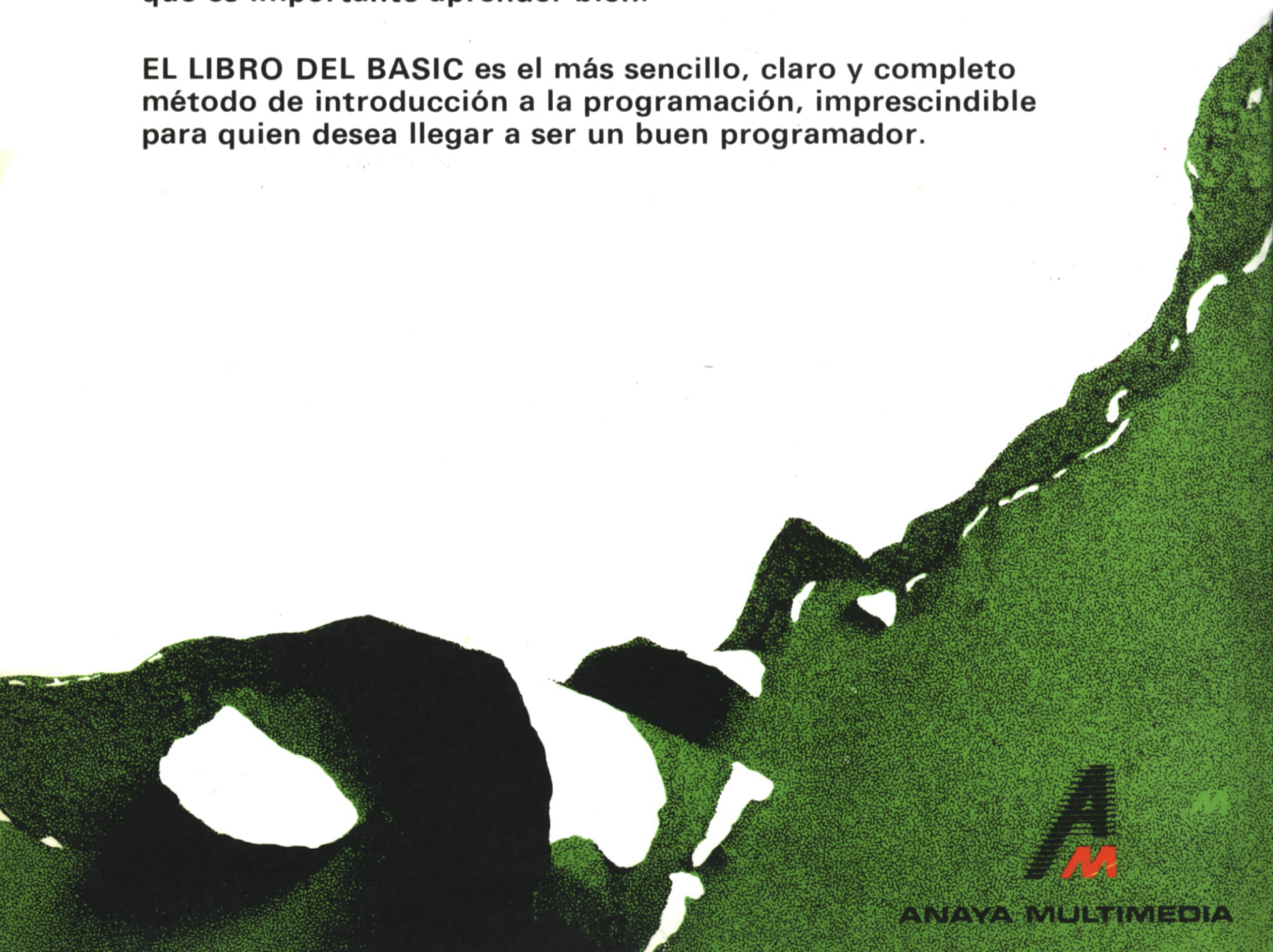
EL LIBRO DEL BASIC enseña los principios fundamentales de la programación, necesarios para poder entender y usar un microordenador.

Está escrito en un estilo claro y ameno, dirigido a «jóvenes» de 8 a 88 años, siendo el libro ideal para quien no tiene ninguna experiencia previa en el manejo o programación de microordenadores.

Puede usarse con cualquier microordenador: El BASIC es el lenguaje de programación más popular y difundido.

Es divertido, está repleto de simpáticos dibujos y diagramas, que aclaran perfectamente todos los conceptos que es importante aprender bien.

EL LIBRO DEL BASIC es el más sencillo, claro y completo método de introducción a la programación, imprescindible para quien desea llegar a ser un buen programador.



ANAYA MULTIMEDIA