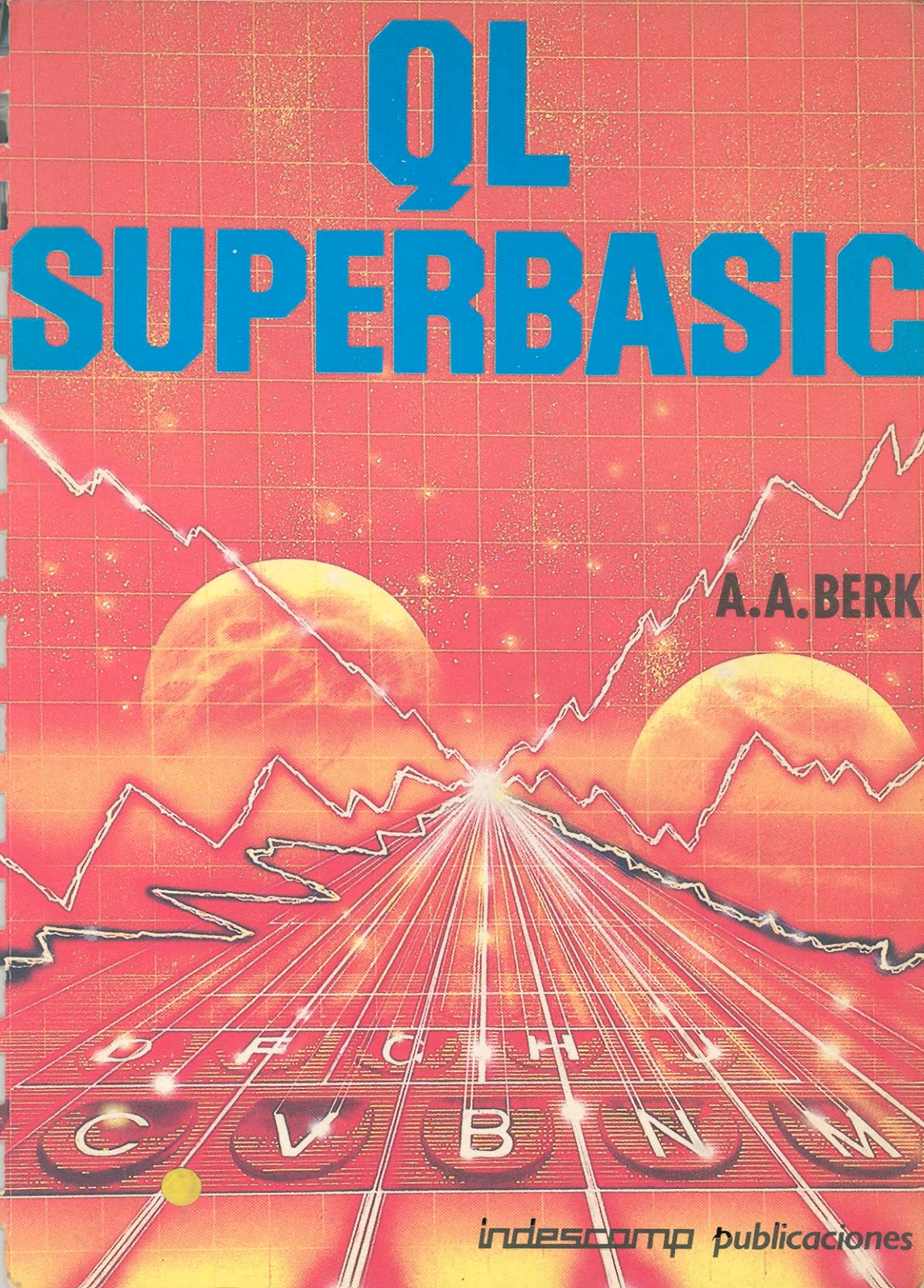
QL SUPERBASIC

A.A.BERK



indescomp publicaciones

QL SuperBasic

A. A. Berk

indescomp

Granada Technical Books
Granada Publishing Ltd

Editado por **INDESCOMP, S.A.**
Avda. Mediterráneo, 9 - 28007 MADRID (ESPAÑA)

Traduce, compone e imprime: CONORG, S.A.

I.S.B.N.: 84-86176-30-1

Depósito Legal: M-9324-1985

Contenido

Prefacio

El nombre QL es un acrónimo de 'Quantum Leap', que representa un 'salto **cuántico**'; ya que su concepción marca un avance que lo separa de lo normal en el campo de los microordenadores. Tiene una memoria y una capacidad de cómputo similar a muchas máquinas industriales de tamaño medio, pero su coste lo hace asequible a todo el mundo.

Las "microductoras" internas significan también que el depósito de datos y programas para un uso posterior es directo, y siempre está accesible. Puedes archivar documentos largos procedentes del tratamiento de textos, de diagramas gráficos habituales en los negocios, o de información de la clase "base de datos" para su ulterior recuperación y análisis.

Este libro concierne, sin embargo, al lenguaje de programación contenido **dentro** de la máquina, y disponible al ponerla en marcha. Tienes una máquina capaz de aceptar comandos e instrucciones usando un **dialecto** de uno de los más populares lenguajes de programación conocidos en el mundo. Ese dialecto se llama **SuperBASIC.** Es una forma mejorada del BASIC, y quedarás agradablemente sorprendido por la estructura y facilidades que te ofrece, en relación con los lenguajes a los que estás acostumbrado.

Este libro se ha escrito de manera que sea aconsejable tanto para el principiante voluntarioso, como para la persona experimentada que necesita ejemplos ensayados y comprobados que le ayuden a comprender el SuperBASIC. El material también está adecuado para la consulta posterior, y será provechoso con este propósito incluso después de que hayas aprendido el lenguaje.

La primera versión del QL que apareció en el mercado, tenía unas cuantas "pifias programales" dentro de su sistema operativo, y muchas equivocaciones en los manuales que venían con la máquina. Debe tenerse bien presente que **todos y cada uno de los muchos ejemplos de este libro han sido ensayados y comprobados en la máquina,** por lo que los encontrarás valiosísimos como ayuda para dilucidar y resarcirte de los errores en los manuales, y al mismo tiempo conseguir un buen punto de contacto para el uso del SuperBASIC.

Dr. A. A. Berk

Capítulo Uno

Presentación del QL

Este capítulo supone que has seguido las instrucciones del manual del QL para conectar tu ordenador, y tienes ahora un monitor y un teclado delante de ti con el que trabajar. Observaremos algunas de las operaciones simples que debes ser capaz de efectuar con esos equipos para comprender el resto del libro.

El ordenador

El hardware del ordenador QL ocupa una placa grande con circuitos electrónicos encajada exactamente debajo del teclado, dentro de la carcasa de plástico negra. Dicha carcasa también aloja las dos unidades micro**ductoras** a la derecha. Hay varias ranuras y zócalos en las partes posterior y laterales de la carcasa, a través de los cuales se tiene acceso a la placa interna con los circuitos electrónicos. Eso permite que se conecten a la unidad principal una gran variedad de accesorios y equipos, desde impresoras hasta ampliadores de memoria. Ninguno de ellos nos preocupa ahora, ya que no examinaremos el hardware en ningún aspecto.

La característica principal de la máquina, en lo que nos concierne, es su habilidad para comunicarse con nosotros a través de la pantalla y del teclado. Vía estas dos unidades 'periféricas' aprenderemos a ordenarla que efectúe cualquier tarea de la que sea capaz, escribiéndola una **sucesión de instrucciones,** que constituye el **programa.**

Si deseas añadir una impresora a la máquina, deberás comprar una que esté constatada como capaz de conectarse al QL, y asegurarte que dispones de los cables correctos para hacer la labor de conexión.

SuperBASIC

Para comunicarse con algo, sea mecano o humano, debes primero descubrir las palabras **básicas** que es capaz de comprender. Eso se denomina **su lenguaje,** y tu primer trabajo es aprender el lenguaje del QL, que recibe el nombre de SuperBASIC.

Los elementos primarios del SuperBASIC se denominan **sentencias.** Cada sentencia constituye un "mandato" al ordenador para que lleve a cabo una determinada **acción.** Por ejemplo, hay sentencias para hacer que escriba algo en la pantalla, que sume dos números, que emita un pitido usando el zumbador interno, y así sucesivamente. Un programa es simplemente un **conjunto ordenado de sentencias,** escritas usando las palabras especiales -parecidas a inglés- que son **clave** en ese lenguaje.

Utilización del QL

Se supone que has desembalado la máquina, conectado todo, y estás ahora sentado en frente de la pantalla dispuesto para proseguir.

Antes de pasar adelante, observa la parte derecha del teclado, y encuentra el botón de **restauración (reset)** que sobresale en el costado de la carcasa. Oprímelo unas cuantas veces para ver el efecto sobre la máquina. Este botón se usa para sacarte de situaciones en las que has especificado algo mal y la máquina 'se queda colgada' y deja de responder a tus acciones, o cuando deseas comenzar completamente de nuevo con **las condiciones iniciales** en la máquina. Cuando lo pulsas, cualquier programa que pueda haber en la máquina se pierde. Si lo habías depositado en la microcinta, siempre podrás recuperarlo, ya que no se ha ido de ahí en absoluto. ¡No te habitúes a pulsar el botón de restauración, ya que es una maniobra bastante drástica!

Cada vez que pulsas el botón, la pantalla cambiará mostrando una trama aleatoria de motas coloreadas, durante un segundo aproximadamente. Con eso se ilustra el repertorio completo de puntos luminosos que pueden mostrarse separadamente y en unos cuantos colores, mediante un programa de computadora. La imagen se borra luego y pasa a mostrar dos recuadros de color, uno de los cuales es un par de alternativas para que elijas, y el otro un mensaje de reserva de derechos. Las alternativas conciernen a dos de las **teclas funcionales** situadas en la parte izquierda del teclado. F1 y F2 son las dos teclas superiores de las cinco que existen, y que están rotuladas F1 a F5. Estas cinco teclas se usan para propósitos especiales en los programas, pero cuando se pone en marcha al principio, dos de ellas se emplean para que informes al QL si estás usando un receptor de TV normal o un **monitor** especial de color. Un monitor de color dará una calidad de imagen considerablemente mejor a la de un receptor de TV, pero son relativamente caros. Una TV ciertamente es adecuada para el aprendizaje, la experimentación y la confección de programas.

Si tienes un receptor de TV, pulsando F2 te asegurarás que las imágenes producidas por el QL no desaparecen por fuera del borde de la pantalla. Sin embargo, eso se logra sacrificando parte de la cantidad total del área de la pantalla, que en los demás casos estaría disponible.

Tengas un receptor de TV o un monitor, restaura las condiciones iniciales y luego pulsa F1 y observa el pequeño piloto cuadrado con luz roja oscura delante de la ranura correspondiente a la microductóra situada a la izquierda. Si no lo has observado, vuelve a restaurar y ensaya de nuevo. Deberás ver que ese piloto se ilumina y deberás oir un ligero ruido indicativo de que está en marcha y girando la microductora.

El teclado

Siguiendo las instrucciones del párrafo anterior, habrás conseguido que la imagen cambie mostrando tres **ventanas:** blanca, roja y negra. A través de esas tres ventanas es como el QL te comunica los diversos tipos de información que le has solicitado. Por ejemplo, ensaya pulsando cualquier letra y manteniendo pulsada la tecla durante dos o tres segundos. Después de un breve intervalo, la pantalla mostrará una línea de letras que se repiten una y otra vez. Aparecen en la ventana negra de la parte inferior de la pantalla. El QL te está comunicando el hecho de haber reconocido que has pulsado una tecla determinada, exponiendo la letra rotulada en la caperuza de la tecla. Dicha ventana del fondo es la denominada **ventana de comando;** y en ella verás cómo escribe los comandos a medida que tú se los tecleas.

Las otras dos ventanas las emplea para exponer la información salida de tus programas, y el propio programa que la produce. Lo veremos ulteriormente. Por el momento, vamos a usar la ventana de comandos.

Debes tener un renglón de letras en la pantalla, terminando con un "chapón" en rojo que parpadea, y es lo que llamamos **cursor,** porque siempre va **corriendo** por la ventana de comando para mostrar dónde aparecerá el siguiente símbolo que teclees. Ensaya pulsando otra tecla mientras vigilas el cursor. Si mantienes la tecla pulsada el suficiente tiempo, el carácter que aparece en la pantalla comienza automáticamente a repetirse en forma bastante rápida. Tendrás que acostumbrarte a pulsar las teclas durante el tiempo justo, o todo lo que impongas por teclado provocará errores de escritura. No te preocupes sobre ningún error que cometas al **tipografiar**, ya que aprenderás cómo corregirlos más adelante; ahora continúa tecleando.

Tendrás que usar la larga barra ESPACIADORA situada en la parte inferior del teclado para que aparezcan **espacios en blanco** que separan unas palabras de otras. Si continúas tecleando letras, el cursor llegará finalmente y desaparecerá por el borde derecho de la pantalla. De hecho, se ha pasado al borde izquierdo de la pantalla, pero no podrás verlo hasta que hayas tecleado suficientes letras para hacer que vuelva a exhibirse en la posición justa. Ensaya con eso.

Si estás usando una TV, la imagen en pantalla no será muy clara, y ya volveremos a hablar de la imagen normal de TV. Sin embargo, debes tener bien en cuenta que siempre hay tres ventanas en la imagen, ya que será importante más adelante.

Ahora pulsa el botón de restauración de las condiciones iniciales, y elige F2 tengas un receptor de TV o un monitor. El cursor parpadeante volverá a ser visible, pero esta vez es de color **magenta** (carmesí oscuro). Ensaya otra vez pulsando una letra durante un segundo o dos, y verás que aparecen en la ventana de comandos símbolos mayores y más fáciles de leer. Ensaya tecleando por ejemplo, tu nombre y dirección, para conseguir el "tacto" del teclado.

A medida que continúes tecleando, la línea finalmente será tan larga que no encaje en la pantalla, y el cursor retrocederá hasta el extremo izquierdo de la pantalla. Todo lo que teclees a partir de ese punto, hará que la imagen **se desplace** hacia arriba en la pantalla. Continúa tecleando y verás que una larga línea de texto se distribuye sobre varias líneas consecutivas de pantalla, a medida que tecleas.

La ventana negra de comandos sólo puede mostrar **cuatro líneas de texto,** y si tecleas más, de manera que provoques que se deslice la imagen hacia arriba, verás que desaparece el texto de la parte superior para dejar espacio para el que estás tecleando; y desaparece de manera que ya es irrecuperable directamente en pantalla. (Este corrimiento de la imagen se denomina en inglés "scroll").

Las letras que has producido al pulsar habrán sido expuestas hasta ahora en **minúsculas.** Para producir mayúsculas, pulsa una de las **teclas de turno** -rotuladas SHIFT- y la tecla correspondiente manteniendo pulsada al mismo tiempo la tecla SHIFT. Si dejas de mantenerla pulsada, las letras subsiguientes volverán a aparecer en minúsculas. Practica escribiendo tu nombre y dirección de nuevo, pero esta vez usando las iniciales en mayúsculas, si antes no lo hiciste así. También ahora debes concentrarte en coger el tino al teclado y a la pantalla. Encontrarás que para mayor comodidad, hay una tecla de turno -rotulada SHIFT- en cada uno de los extremos del teclado.

Observarás que algunas de las teclas llevan marcados dos símbolos en la caperuza superior. Los situados arriba se obtienen siempre usando la tecla SHIFT de igual manera que para las mayúsculas. Pásate ahora unos cuantos minutos recorriendo cada carácter del teclado y ensayando con y sin pulsar SHIFT para ver el efecto. Eso también te mostrará el completo **repertorio de caracteres** que puedes usar en el QL.

Para los programas en general, es habitual **imponer** los comandos e instrucciones usando letras mayúsculas. Como sería una tontería tener que mantener pulsada una de las teclas SHIFT durante todo el tiempo, y especialmente dado que con eso también se pasa el "turno" de las teclas numéricas en la fila superior del teclado, y entra el "turno" de los símbolos situados por encima de los números y que se usan bastante menos frecuentemente. Es normal por tanto, que en un teclado de ordenador se disponga de una tecla de **enclavamiento de mayúsculas** -rotulada CAPS LOCK- que cambia las teclas de letras a la correspondiente mayúscula, pero deja las teclas de números y de otros símbolos igual, i.e. produciendo el rotulado en la parte de abajo. Pulsa solamente una vez la tecla CAPS LOCK y ensaya con todas las teclas. Verás que los números y otros símbolos que no son letras están como si continuaras en el modo minúsculas, pero todas las letras son ahora mayúsculas. Esta modalidad de operación del teclado permanece en vigor bien hasta que se restauran las condiciones iniciales, o hasta que se pulsa otra vez la tecla CAPS LOCK. Por tanto, cada vez que se pulsa la tecla de **enclavamiento de mayúsculas,** actúa como un "basculador" que cambia el modo de operación.

Te encontrarás además que algunas de las teclas no producen un símbolo visivo en pantalla. Por ejemplo, todas las teclas de función y la tecla rotulada ESC -que corresponde a **escape,** no hacen que se exponga nada en pantalla. Se usan para otros fines. Encontrarás que sucede lo mismo con la tecla rotulada CTRL -por **control-** y la tecla ALT -de **alternativo.** Esta tecla ALT es utilizada por los programas de Psion que vienen con tu máquina, y en breve veremos cómo se usa la tecla CTRL. También habrás jugado a estas alturas con las teclas que llevan marcada una **flecha** en la caperuza, y por supuesto con la tecla marcada ENTER -que concluye toda operación de **introducción** de datos y comandos (solemos decir "adentro"). Todas ellas afectan a la imagen expuesta, pero no necesariamente exponen algún símbolo. Sin embargo, puede que al pulsar ENTER, te hayas encontrado con que aparece el mensaje **"bad line"** que te indica que es una **mala línea,** y de hecho es la manera con que el ordenador te dice que no comprende lo que le has tecleado, por lo que no se ve obligado a cumplimentar lo que le hayas mandado. Siempre está comprobando si el comando es válido, y todo lo que ha visto hasta ahora será una retahíla de letras sin ningún significado.

Edición

A no ser que seas afortunado, probablemente te encontrarás con que ya has cometido, como mínimo, un error al teclear. Es importante que seas capaz de **editar** (y recuerda su primitivo significado de "sacar a la luz pública") lo que has tecleado equivocadamente para alterarlo o sustituirlo por lo correcto; en definitiva, para **revisarlo.**

Ensaya tecleando una letra, y luego pulsa la tecla de control. Mientras la mantienes pulsada, presiona la tecla **flecha-izquierda** -i.e. aquélla en que la flecha apunta hacia la izquierda. La letra que acabas de teclear desaparecerá, y el cursor retrocederá una posición hacia la izquierda. Si mantienes pulsadas simultáneamente CTRL y la flecha izquierda, el cursor continúa retrocediendo y borrando todo a medida que pasa por encima. Finalmente, se detendrá al alcanzar el principio de la línea que estabas tecleando, y esperará que le mandes una nueva acción. Este método de **tachar el anterior** es una de las maneras de "editar" tus trabajos.

Teclea unas cuantas letras, y pulsa otra vez la tecla flecha-izquierda, pero esta vez sin pulsar la tecla de control. Con eso, haces que el cursor retroceda a lo largo de la línea y se sitúe sobre cualquiera de los símbolos que previamente has tecleado. Todavía puedes apreciar el símbolo, ya que el cursor es **transparente.** También aquí, si mantienes la tecla de flecha pulsada lo suficiente, repetirá automáticamente su acción, y el cursor se mueve bastante rápido a lo largo de la línea. Para desplazarlo en el sentido contrario a lo largo de la línea, obviamente pulsa la tecla **flecha-derecha.** Por estos medios, puedes situar el cursor **sobre** cualquier símbolo que haya en la línea. Observa que el cursor no sobrepasa ni el comienzo ni el final de una línea de símbolos.

Ahora sitúa el cursor sobre cualquier símbolo, y pulsa la tecla flecha-derecha mientras mantienes pulsada la tecla de control. Eso provoca que el símbolo señalado exactamente por el cursor sea el que desaparezca, y que los situados a la derecha del cursor se corran para ocupar el espacio que quedaba libre. De esta manera, puedes **borrar el actual** símbolo señalado por el cursor, en contrapartida a la situación anterior en que borrabas el colocado inmediatamente a su izquierda. Ensaya ambas formas de borrar.

Prueba ahora a teclear cualquier letra sin tener pulsada la tecla de control. Verás que las letras se **insertan** dentro de la línea y en la posición señalada por el cursor en ese momento, y que el texto situado a la derecha del cursor se desplaza para dejar el espacio necesario para incluir la letra. Con estos medios, puedes cambiar cualquier letra de una línea. Simplemente sitúa el cursor sobre la letra que quieres cambiar, suprímela usando CTRL con la flecha-derecha, y teclea la nueva letra que quieres insertar. Ensáyalo todo.

Ahora eres capaz de efectuar un intento de teclear una frase larga exactamente y usando las **facilidades de edición** que te hemos presentado antes. Si la concluyes -pulsando la tecla ENTER- dicha frase será llevada "adentro" del ordenador propiamente dicho, para comprobar su validez y... emitirte el mensaje de error "mala línea" (a no ser que aciertes por casualidad).

Un comando

Para ver cómo debe usarse la ventana de comandos, vamos a practicar usando el comando "pite", para lo que hay que mandarle BEEP. Con él, simplemente hacemos que genere un tono continuo puro y lo emita a través del altavoz interno incorporado en la carcasa.

Para dar un **comando** al ordenador, debes ante todo tener una línea en blanco dentro de la ventana de comandos, con el cursor situado en la parte izquierda. La manera más rápida de conseguirlo es pulsando ENTER y no preocupándose del posible mensaje de error.

El ordenador está esperando ahora que le demos un "mandato" para ponerse a trabajar. Teclea el siguiente, exactamente tal y como se muestra:

BEEP 3000,100

Es decir, teclea las cuatro letras, seguidas del "espacio" o "blanco", luego las cuatro cifras, después una coma y luego las otras tres cifras. El cursor debe estar justamente detrás del último cero escrito. Si has cometido un error al teclear, **edita la línea** hasta que aparezca exactamente como la que te hemos mostrado. Ese es un comando válido, pero el ordenador "sólo lo escucha y lo repica en pantalla" y no procede a obedecerlo hasta que lo concluyas pulsando ENTER.

Eso es una regla general, ya que estás recibiendo la oportunidad de corregir la línea tecleada hasta que estés satisfecho, y antes de que el ordenador compruebe lo mandado y pase a **ejecutar ese comando.** Pulsa pues ENTER y verás -o mejor, oirás- el resultado del comando que le has dado. El cursor también retrocederá de un salto al comienzo de la siguiente línea, y los caracteres del comando dado se deslizarán hacia arriba una línea -a no ser que mantengas tu dedo sobre la tecla ENTER un tiempo demasiado largo, en cuyo caso el comando probablemente desaparezca por la parte superior. No te preocupes ahora de eso. Una vez que el comando ha sido efectuado de esa manera, no puedes volver a utilizarlo sin volverlo a teclear enteramente.

Si cometes una equivocación, y no te percatas antes de mandarla "adentro", será detectada por el ordenador y te lo dirá. Ensaya tecleando lo anterior, pero olvídate del número 100 del final. El ordenador te dirá que así no puede pitar correctamente, porque el primer número -el primer **parámetro** de ese comando- es el que le dice cuánto ha de durar el pitido, mientras que el segundo número -el segundo parámetro del comando- le dice el tono con el que tiene que pitar. Si uno de esos **parámetros** falta, no sabe cómo cumplimentar el comando, y te envía un mensaje de error.

Antes de continuar experimentando con el comando BEEP, deberás saber cómo detenerlo cuando lo necesites. Si tecleas lo siguiente, verás a lo que me refiero:

BEEP 0,100

El pitido todavía prosigue, y así lo hará bien hasta que restaures las condiciones iniciales o bien hasta que le teclees BEEP por sí solo, seguido inmediatamente de la pulsación de ENTER. Ensaya a detener el pitido por uno u otro de esos métodos, y luego continúa experimentando cambiando los parámetros del comando.

Las "bandas" de valores admitidos para dichos parámetros es de 0 a 32767 para el primero, y de 0 a 255 para el segundo de ellos.

Resumen

- Cuando se pulsa el botón de **restauración** (reset), se pierde todo lo que hay en la memoria en ese momento y el ordenador comienza a trabajar de nuevo a partir de las **condiciones iniciales.**
- El teclado está al principio en el **turno** de letras minúsculas, y se pueden inscribir las letras en mayúsculas pulsando la tecla SHIFT simultáneamente con la letra en cuestión. De igual manera se consiguen los símbolos marcados en la parte superior de la caperuza de las teclas.
- CAPS LOCK provocará que el teclado pase permanentemente al turno de mayúsculas -queda enclavado a mayúsculas- que sólo afecta a las teclas de letras. Las restantes teclas con números y signos no se ven afectadas.
- CTRL con **flecha-izquierda** suprime el símbolo situado inmediatamente antes de la posición corriente del cursor, en tanto que CTRL con la flecha-derecha suprime el carácter señalado exactamente por el cursor en ese momento.
- Automáticamente se **insertan** los símbolos en la línea de comando, en la posición señalada por el cursor, y se desplaza el resto del texto situado a la derecha una posición también a la derecha.
- La pulsación de ENTER hace que el ordenador lleve **adentro** la línea tecleada y comience a examinar su validez como comando para cumplimentarla, y... producir el mensaje de error correspondiente, cuando no la puede interpretar.

Capítulo Dos

Introducción a la Programación

Si eres un principiante en programación, o si eres un experto pero estás algo oxidado porque no has usado un ordenador desde hace tiempo, encontrarás este capítulo muy útil como inicio o reinicio del tema. El capítulo contiene alguna información importante sobre SuperBASIC, e incluso aunque seas experto, también debieras leer por lo menos el resumen del final del capítulo antes de pasar al siguiente.

Este libro concierne a uno de los aspectos particulares del QL. Intenta describir el **software**, o colección de programas propia de la máquina. El QL viene con una programación muy avanzada, con programas ya preparados, que han sido desarrollados en computadoras grandes y depositados en los cartuchos de cinta insertables en las microductoras, para que los uses como desees. Sin embargo, estos programas tienen como meta uno o dos aspectos de la utilización de un ordenador, y sólo podrás conseguir la potencia completa de la máquina si aprendes a **confeccionar programas por ti mismo.**

Los programas son simplemente listas de instrucciones escritas usando palabras que el ordenador puede comprender. Siempre y cuando conozcas dichas palabras, y la acción concreta que realizan, puedes hacer que el ordenador efectúe cualquier tarea que puedas imaginar, y muchas más que no imaginas todavía. Hay unas cuantas reglas sencillas que aprender, y con ello toda la potencia del ordenador queda completamente a tu disposición.

Para hacer que un ordenador haga precisamente lo que deseas, debes primero aprender a creer ciegamente y aplicar completamente la Primera Ley de la Informática -"los ordenadores son estúpidos". Sólo son capaces de llevar a cabo una serie de tareas ciega e ininterrumpidamente. Este hecho universal se aplica a todos los ordenadores, sin importar lo complejos y aparentemente formidables que puedan parecer. Si te convences a ti mismo firme y completamente de este sencillo atributo de todos los ordenadores, la programación te entrará de una manera natural e intuitiva. Si imaginas siquiera por un momento que el ordenador es inteligente, olvidarás que tienes que decirle absolutamente todo lo que debe hacer, incluso las cosas más sencillas que sea capaz de hacer, y es esta estúpida máquina la que aprende a costa de tus mejores esfuerzos.

Para ser más técnicos sobre el hecho central presentado antes, debemos analizar y describir completamente cualquier **labor** en una manera simple y lógica, y desglosar dicha labor en tareas, y luego en acciones, expresadas por **sentencias** usando las palabras que el ordenador puede entender y obedecer. Muchas de las equivocaciones -las pifias- que se encuentran con los ordenadores pueden rastrearse hasta detectar que en alguna parte ha fallado la aplicación de esta primera ley de la computación.

Este capítulo está dedicado en gran manera a ayudarte a pensar siguiendo **pequeños pasos lógicos.** El tamaño y la naturaleza de cada paso depende del lenguaje en que el ordenador es capaz de "hablar". El QL usa un dialecto de un famoso **lenguaje de alto nivel** llamado BASIC que corresponde a "Principiantes Todo-Propósito Simbólico Instrucción Código". En inglés, sale la palabra BASIC, pero no debe eso de "básico" considerar que implica que es un lenguaje en alguna manera inferior, o sólo para el uso de los principiantes ("beginners"). BASIC se usa para escribir muchas clases de programas serios, y no está en ninguna manera restringido a las aplicaciones simples. Llegarás a apreciar cuán amplias son las tareas que el BASIC es capaz de efectuar, a medida que progreses a lo largo de este libro.

La máquina

Para ver cómo el BASIC encaja dentro de la máquina, merece la pena saber un poquito sobre el ordenador. Por ejemplo, todos los microordenadores usan una "pastilla de silicio" donde están integrados circuitos electrónicos, y que se denomina normalmente **microprocesador.** Es el corazón de la máquina, y su cualidad más importante es que simplemente **comprende y ejecuta** instrucciones. Sin embargo, las instrucciones que el microprocesador comprende están escritas en un lenguaje denominado **código máquina,** que está muy lejos de ser similar al humano. Por eso decimos que el código máquina es un **lenguaje de bajo nivel.**

Con el fin de que el microprocesador ejecute instrucciones dadas en algo parecido al inglés, tales como las normales en BASIC, ha de estar dotada la máquina de **otro** programa que actúe de **intérprete** del programa que tú escribes. Esta interpretación es efectuada continuamente por el programa que reside en la máquina; y como tú no te das cuenta de lo que está sucediendo, decimos que esta "traducción" ocurre de una manera que es **transparente** para el usuario.

El Código Máquina no se describirá aquí, pero siempre debes recordar que teóricamente es posible producir una traducción desde cualquiera de los muchos lenguajes de alto nivel que existen hasta el código máquina. BASIC es simplemente uno de esos lenguajes, que en el presente es el más popular y uno de los más simples de aprender; pero no olvides nunca que en la máquina debe residir el **programa intérprete** que sirve de intermediario entre lo que tú escribes en el lenguaje BASIC y lo que el microprocesador es capaz de entender y ejecutar.

Vamos a comenzar aprendiendo a desglosar **tareas** en series de acciones más sencillas que puedan ser programadas como **sentencias** del BASIC.

Pensamiento lógico

A medida que aprendas a programar, te encontrarás con toda seguridad que **tu razonamiento** pasa más y más a ser un proceso preciso y analítico de la labor a efectuar.

Una de las mejores maneras de comenzar este proceso es intentar imaginarse que tendrías que explicar una sencilla actividad a un niño bastante estúpido pero muy ordenado y cumplidor. Un buen ejemplo es la labor de levantarse para ir a trabajar o a la escuela por las mañanas. Una de las razones de por qué es un buen ejemplo, es que probablemente nunca te hayas detenido a analizar las acciones que has de llevar a cabo para conseguir esa meta. Por ejemplo, ¿por dónde comienzas a explicar esa tarea? ¿Dirías que te levantas, desayunas, dices adiós a la familia y luego tomas el tren o el autobús? Si así fuera, nuestro chico cumplidor y "lógico" ¡podría imaginarse que te paseas durante todo el día en pijama! O pudiera pensar que vas a trabajar mientras estás dormido, dado que no le has explicado que primero tienes que despertarte.

Además, tendrías dificultades en explicarle cómo sabes cuando te despiertas al estar dormido, y así sucesivamente. Como puedes ver, **analizar** una tarea aparentemente sencilla puede llevar a un límite pormenorizado y molesto, y ¡todo depende mucho de lo que el niño comprende! Es esa exactamente la situación con los ordenadores. Tienes que saber exactamente el detalle al que tienes que llegar para explicar las acciones. En definitiva, lo que tienes que saber son las frases -las **sentencias**- que el ordenador comprende.

Un ejemplo de un problema

Vamos a suponer una tarea sencilla como ejemplo: ordenar tres números en orden ascendente. Los números son 9, 1 y 3. Puedes hacer eso sin pensarlo siquiera, pero ¿cómo explicarías las acciones que efectúas para conseguir tu objetivo?

Puedes suponer que el ordenador comprende **relaciones** tales como "es igual que", "es mayor que" y "es menor que". Además, que también comprende las palabras "canje" de dos objetos, y que sabe qué hacer ante frases **condicionales** de la forma "si...". Podrías así decirle que hiciera un CANJE de los dos primeros números SI el primero es MAYOR QUE el segundo. Como resultado de esa acción, obtendrías los números 1, 9, 3. El mismo tratamiento podría aplicarse luego a los dos números segundos, y con eso lograrías ordenarlos en **este caso particular.** ¿Cuántos otros conceptos tendrías que emplear para escribir esta serie de acciones en un caso general?

Probablemente llegarías a la conclusión de que necesitas como mínimo dos ideas primordiales sin que las hubieras considerado. Son las de **almacenamiento** e **identificación.** Hemos identificado, denominado, a los números que intervienen con las palabras "primero", "segundo" y "tercero", y hemos supuesto que podíamos depositarlos, almacenarlos, en alguna parte dentro del ordenador y no simplemente en el papel que tienes ante tus ojos.

Otras dos ideas fundamentales en computación son las de imponer, poner dentro **-entrada-** y la de exponer, poner fuera **-salida-** las cantidades con que hemos operado. Hemos supuesto en el ejemplo anterior, que el ordenador disponía de medios para ingresar los números y para expresar los resultados producidos en cada acción de la serie.

Como puedes ver, estos son algunos conceptos importantes que tendremos que elaborar antes de que podamos incluso comenzar a conversar con el ordenador.

Etiquetado y almacenado

Siempre que tratemos con una cantidad, una magnitud -una **variable**- en cualquier manera, la primera cuestión siempre será la de ¿**cómo se denomina y dónde se deposita**? No podemos incluso hablar entre nosotros sobre un **objeto** hasta que no tengamos un **nombre** para él, y a los ordenadores les pasa lo mismo. Una vez que una variable tiene su nombre identificativo, puede ser manipulada, alterada, y en general "operada" por el ordenador.

Otra importante actividad es la de crear espacio dentro del ordenador para el almacenamiento de una variable. El ordenador hace eso automáticamente. Siempre que "etiquetamos" algo, inmediatamente le reserva un espacio adecuado y debidamente señalado dentro de **la memoria** para depositar en él ese algo.

El término **variable** incluye una gran variedad de cosas diferentes desde números de diez cifras hasta letras y signos tales como el punto o la coma. Todos los **caracteres** mostrados en el teclado del QL pueden ser usados como **valor** de una variable, y expuestos en pantalla, o usados para otros propósitos en un programa de ordenador.

Cuando el ordenador reserva y adjudica un espacio de almacenamiento para una variable, es importante que prepare la cantidad correcta de espacio y lo adecúe a la variable. Para hacerlo tiene que saber si la variable considerada es para almacenar un número o simplemente una retahíla de caracteres cualesquiera. En BASIC, es la forma real del **nombre de la variable** la que le dice al ordenador el **tipo** de variable que es, i.e. la naturaleza o **índole** de los valores que puede tener.

Identificativos

La mayoría de los conceptos con que nos hemos topado en la programación de ordenadores tienen palabras técnicas especiales que son las usadas para describirlos. Por ejemplo, la palabra "cantidad" ha sido reemplazada por la palabra **variable.** Igualmente, la palabra "etiqueta" se sustituye habitualmente por **identificador,** o muchas veces simplemente por **nombre.** Decimos por tanto, que dar un "identificativo a una variable" es lo mismo que darle un nombre al lugar donde está depositada la cantidad correspondiente a esa variable.

Todos los ordenadores tienen reglas sobre cómo ha de formarse un identificador para una variable, y el QL no es ninguna excepción. Estas reglas ayudan a impedir que el ordenador se vea confundido. Por ejemplo, si se te permitiera llamar a una variable con el nombre 12.4, ¿cómo podría saber el ordenador si dicho 12.4 debiera tratarlo como un nombre o como un número?

El QL permite una serie de hasta 255 letras, cifras y el signo de subrayar (_) como identificador de una variable. Sin embargo, dicho identificador tiene que comenzar siempre con una letra para evitar confusión. Por ejemplo, puedes usar N o Número o NUMERO_3, etc., como identificador. No pueden usar _número, ni 212Número, porque la inicial de esos identificadores no es una letra.

Además, el QL no distingue entre letras mayúsculas y minúsculas para formar los identificadores, y los siguientes nombres son considerados iguales por el QL:

Cost cOST CoSt

Como regla general, merece la pena intentar dar nombres a las variables que tengan algún significado dentro del programa. Eso ayuda tanto para escribir como para comprender el programa. Por ejemplo, supongamos que intentas computar el precio de un artículo, dado el coste de los materiales, el de la mano de obra, los costes suplementarios y el beneficio. Te sería difícil recordar los significados de las variables si las llamaras A, B, C, D y demás. Es mucho mejor usar: PRECIO, MATERIALES, MANOBRA, GASTOS y BENEFICIO. Esos identificadores son autoexplicativos. Observa que los hemos abreviado un poco, ya que habitualmente merece la pena mantener los nombres con un número mínimo de símbolos, simplemente para teclear menos.

Almacenamiento

Cuando se usa un identificador, es importante que el ordenador reserve el almacenamiento en memoria de la clase correcta para los valores de esa variable, ya que una variable puede tener como valor tanto un número, como una colección de letras y otros símbolos, formando una serie. Hay por tanto, variables de dos **tipos: numéricas** y **literales** (las que "antiguamente" se denominaban cadenas, cuerdas, sartas de caracteres).

El ordenador distingue entre estas variables de diferente índole por el último símbolo del identificador de la variable. Si la variable es **literal**, es decir, sirve para contener un dato literal, debe usarse el signo dólar ($) como último símbolo del identificador. Con eso, el ordenador lo reconoce como literal y reserva la cantidad apropiada de memoria para almacenar el valor correspondiente. Por ejemplo, Nombre$ podría contener una retahíla de siete letras, y el ordenador automáticamente reservaría en memoria el espacio necesario para depositar esos siete símbolos. La ausencia de un $ implica que la variable es numérica, y el ordenador reserva el espacio adecuado para sus valores.

Hay otro sufijo que también puede usarse para variables **numerales:** es el signo porcentaje (%). Eso implica que los datos contenidos en la variable van a ser **enteros,** tal y como los describiremos más adelante.

Asignación

Tendrás que introducir algunas sentencias de programas, asegurarte que o bien has puesto en marcha el ordenador, o bien has pulsado el botón de "restaure", antes de comenzar esta sección. Para asegurarte que no alteras la naturaleza de la imagen en una forma inesperada, sigue cuidadosamente los ejemplos dados.

Cuando hayas ensayado unos cuantos, puedes experimentar por ti mismo.

El siguiente problema es **depositar** algo de información en el lugar reservado y "etiquetado" de acuerdo con nuestro identificador. Este proceso también tiene un nombre técnico. Se denomina **asignación**, ya que decimos que asigna un valor a una variable. Como explicamos antes, dicho valor no tiene por qué ser un número exclusivamente. Por ejemplo, una variable -un "lugar identificado"- puede tener depositado en él tu nombre y tu dirección. Tu nombre y tu dirección se menciona luego como el **valor** de esa variable, y muchas veces hablemos del **contenido** del lugar representado por dicha variable.

La manera de decir al ordenador en BASIC que reserve un lugar identificado, y deposite en él un valor dado, es algo así como "haga variable igual valor" o "sea variable igual valor", y se usa la palabra **clave** LET. Así por ejemplo:

```
LET A = 33.4
```

hará que el ordenador aparte algo de memoria, lo designe con el identificador A -la variable- y deposite en ese lugar el número **33.4** -el valor-. Para reclamar luego ese número, simplemente damos el nombre A, como veremos. Este ejemplo ha sido de **numerales**, pero con **literales** es bastante similar, aunque se requieren algunos símbolos adicionales.

El signo $ ya te lo hemos presentado. Antes de continuar, teclea el comando anterior (usando el teclado), y concluye el comando pulsando la tecla ENTER (que le indica algo así como "adentro"). No necesitas usar mayúsculas, aunque es práctica general usarlas, y así lo haremos en este libro. Para ahorrarte el tener pulsada permanentemente la tecla de **turno** -marcada SHIFT- pulsa la tecla de **enclavamiento de mayúsculas** -marcada CAPS LOCK- y con eso te asegurarás que siempre trabajas con mayúsculas.

LET es una palabra **reservada** en el lenguaje BASIC, y normalmente la denominamos **palabra clave.** Todas las palabras clave deben ir seguidas por un **espacio en blanco** -obtenido pulsando la barra espaciadora como te mostramos en el texto para evitar confusiones como veremos. Los otros espacios se incluyen para hacer que la imagen aparezca más legible, y puedes prescindir de ellos si quieres.

Si desearas depositar en memoria la palabra PEPE para usarla posteriormente, es importante que el ordenador no confunda la palabra PEPE con un identificador, un nombre de variable. Para impedir eso, el **dato literal** PEPE tiene que estar encerrado entre **comillas.** En el SuperBASIC del QL puedes usar tanto las comillas reales, o el **apóstrofe** (que es la comilla simple), aunque lo normalmente permitido en las otras máquinas es únicamente las comillas (dobles). Los siguientes son comandos válidos en SuperBASIC:

```
LET Nombre$="Pepe"
LET NOMBRE$='Pepe'
```

y ambos asignan a la misma variable la misma serie de letras. De nuevo, introduce esos comandos en el QL y conclúyelos con ENTER. Pulsa CAPS LOCK otra vez cuando quieras elegir letras minúsculas. Recuerda el espacio, el blanco, situado después de la palabra clave LET.

Recuerda también que el QL considera iguales las minúsculas y las mayúsculas **sólo para identificadores.** Por tanto, Nombre$ y NOMBRE$ son considerados por el QL como el mismo identificador, y por tanto el mismo lugar reservado de la memoria. Sin embargo, ese no es el caso para el **dato literal** que se almacena en dicho lugar. Un dato literal, entrecomillado, se deposita **literalmente**, 'al pie de la letra'. Por tanto, "Pepe" y "pepe" son datos literales completamente diferentes, porque tienen un símbolo diferente como inicial. Algunas veces, las propias **comillas** usadas como separadores o delimitadores de la serie de símbolos -y que no quedan depositadas como valor de la variable- se denominan también **literales**, ya que en la literatura y en informática todo lo que va entre las comillas se copia **letra a letra**, o mejor símbolo a símbolo en la memoria (y sin preocuparse de cuáles son esos símbolos).

Como comentario final sobre asignaciones, el SuperBASIC te permite que omitas la palabra LET si lo deseas. Es bastante habitual en BASIC y ahorra tiempo. Así el comando:

```
LET NOMBRE$="PEPE"
```

es el mismo que el comando:

```
NOMBRE$="PEPE"
```

pero omitir la palabra clave tal y como ocurre en la segunda sentencia, puede dar lugar a confusiones. Es crucial que comprendas que no es una frase diciéndote que NOMBRE$ y "PEPE" son lo mismo. Es un **comando** al ordenador para decirle que tome uno a uno los símbolos encerrados entre comillas y los deposite uno junto a otro en un recinto reservado de la memoria que tiene identificado mediante NOMBRE$. El signo igual aquí no es de hecho una igualdad ordinaria; es un mandato al ordenador para que asigne los caracteres de PEPE a la variable NOMBRE$, que es lo mismo que depositarlos en un lugar de la memoria etiquetado con NOMBRE$.

Si recuerdas el problema de reordenar tres números en orden ascendente, te acordarás de que identificábamos los tres números usando nombres bastante liosos, tales como primero, segundo y tercero. Ahora lo podemos hacer mucho mejor usando las reglas e ideas comentadas. Podríamos denominarlos por ejemplo A, B y C; y usar sentencias de asignación para hacer que dichas variables tuvieran como valor los números correspondientes. Veremos muy pronto cómo hacerlo.

Entrada y salida

La última sección ha resuelto el importante problema de permitirnos hacer referencias a cantidades variables mediante nombres que las identifican sin ambigüedad. La siguiente labor es ver cómo introducimos las cantidades con las que vamos a trabajar, y cómo exponemos los resultados del trabajo.

Hay muchos métodos de **imponer** (poner dentro) datos en el ordenador usando el lenguaje SuperBASIC, y se describen en los siguientes capítulos.

Por el momento, prescindimos de ingresar datos a través del teclado, y nos aprovechamos del método ya visto para dar valores a las variables que usamos en el programa: la asignación representada por la palabra clave LET.

Sí vamos a tratar la salida de datos, y en concreto el método más importante: **EXPOner** en pantalla lo que deseemos, y para eso usamos la palabra clave PRINT (que es una reliquia de los tiempos en que no había pantalla y todo se "imprimía" en papel).

Como explicamos en el Capítulo 1, la pantalla está dividida en tres áreas principales. Son las llamadas **ventanas** y a medida que das comandos por el teclado, los caracteres se reflejan en una ventana situada en la parte inferior de la pantalla, y puedes **editarlos** cuando desees. Si estás usando una TV, puede que te estés preguntando dónde ha ido una de las ventanas mencionadas, ya que sólo quedan dos en la pantalla. La respuesta es que dos de las ventanas están 'solapadas' o montadas una sobre otra, pero en realidad cada una ocupa toda la pantalla. Lo veremos pronto.

Si estás usando un monitor de video, entonces la sección izquierda de la pantalla estará coloreada en blanco, y la otra sección es de color rojo, al ponerse en marcha. Dicha sección es utilizada por el programa para proyectar sus datos de salida, y es donde la sentencia PRINT puede usarse para **exponer** el contenido de cualquier variable.

Como la mayoría de los comandos en BASIC; éste también es simple de usar y comprender. Puedes casi adivinar cómo usarlo. Si quieres que aparezca en pantalla el contenido de una variable tal como NOMBRE$, simplemente dile al ordenador que lo **exponga**, es decir:

```
PRINT NOMBRE$
```

Asegúrate que previamente has tecleado el comando de asignación para NOMBRE$ que ya comentamos, y termina el comando anterior pulsando la tecla ENTER, como siempre. De nuevo, como PRINT es una palabra clave, debe ir seguida de un espacio en blanco. Experimenta sin incluir dicho espacio en blanco, y verás que el ordenador parece que está comprobando algo durante un rato, 'en la retaguardia', y te sale diciendo que hay un error en alguna parte. Siempre que veas este mensaje de error de "nombre malo" (bad name) comprueba siempre primero si has incluido espacios después de las claves. Hay una lista de mensajes de error en el manual del QL, y cuando veas uno puedes usarla como ayuda para encontrar la equivocación cometida.

El ordenador efectúa una tremenda cantidad de trabajo en la retaguardia, para ejecutar el comando directo PRINT del ejemplo; pero no es de incumbencia del programador en BASIC. Cuando él manda PRINT NOMBRE$, simplemente espera que el contenido de NOMBRE$ sea exhibido en la pantalla. Si estás usando un monitor de video, después de dar el comando anterior lograrás que aparezca PEPE en la ventana de salida que es la situada en la parte derecha de la pantalla. Sim embargo, si estás usando una TV, la salida se expondrá en la esquina superior izquierda de la pantalla, y tendrá un **fondo** rojo. Eso te dice que los datos se están mostrando en la **ventana de salida** de la pantalla.

Un pequeño programa

Incluso aunque sólo hemos aprendido dos claves del BASIC, podemos escribir un pequeño programa que usa todas las ideas presentadas hasta ahora. Entre ellas incluimos el desglose del problema en pequeñas acciones que el ordenador puede comprender, los identificadores y las asignaciones. Para hacer eso, teclea lo siguiente:

```
LET A$="¡Este es el QL!"    (luego pulsa la tecla ENTER)
PRINT A$                    (luego pulsa la tecla ENTER)
```

Si tecleas los caracteres anteriores exactamente como aparecen, y pulsas la tecla ENTER después de cada línea, el resultado será la ejecución **inmediata** de estos **comandos.** Si olvidas el espacio después de la PRINT, ocurrirá un error de línea errónea. Sin embargo, si prescindes del espacio después de la LET, no se da ningún error, pero se expondrá un asterisco en lugar del contenido de A$. Esto es tu primer encuentro con la, realmente, **ciega logicidad** del ordenador, y señala cuán importante es para ti seguir las normas y decirle al ordenador precisamente lo que deseas en todo momento. Si prescindes del espacio después de la LET, el ordenador supone que deseas unir todas las palabras, y piensa que quieres depositar la frase entrecomillada en la variable denominada LETA$, ¡que es un identificador perfectamente válido!

Cuando pones en marcha el ordenador, todas las variables incluyendo LETA$ y A$, tienen como valor el literal **vacío,** y eso queda significado en imagen por el asterisco. Así, cuando el comando PRINT va a ser ejecutado, expone el literal vacío que hay depositado en A$.

Los espacios después de las claves son sólo realmente necesarios si las claves van seguidas de una letra, un número o el símbolo de subrayar. Sin embargo, no es mala idea, acostumbrarse a usar espacios para separar las palabras clave, a no ser que realmente estés agobiado por el espacio en memoria (los otros son espacios en "blanco").

El primer comando en el ejemplo anterior es una asignación que aloja los caracteres entre las comillas en un lugar de la memoria denominado A$. Cuando pulsas la tecla ENTER al final de esa línea es cuando el ordenador ejecuta dicha asignación; pero no antes ni después. El segundo comando simplemente le dice al ordenador que saque el contenido de la variable hasta el **canal de salida,** y en este caso ese contenido es el que aparecerá en la pantalla. Lo sacado aparecerá inmediatamente debajo de la última información mostrada, ya que el ordenador evita cuidadosamente la sobre-escritura. También ahora, el fondo en rojo te dice que se está exponiendo datos en la ventana de salida.

El simple ejemplo muestra unos cuandos conceptos diferentes. Primero, los **comandos** son ejecutados inmediatamente después de concluirlos pulsando la tecla ENTER. Ningún comando será ejecutado hasta que se pulse la tecla ENTER, y es una visión familiar observar a un principiante total sentado delante de la pantalla preguntándose por qué el ordenador no está haciendo nada, aunque en pantalla aparece un comando correcto, ¡simplemente porque no ha pulsado ENTER! La tecla ENTER simplemente le dice al ordenador que estás satisfecho con lo que le has mandado, y que lo lleve **adentro** para que sea ejecutado. Antes de pulses ENTER, puedes corregir cualquiera de los errores cometidos al teclear, tal y como se explicó en el capítulo 1.

Dado que las sentencias en BASIC son tan parecidas al inglés (y ya empieza a haber en el mercado las que son parecidas a otros lenguajes) te encontrarás que aprendes su significado de forma natural. Sin embargo, debes leer cuidadosamente la definición de cada palabra, dado que los lenguajes humanos no son lenguajes muy precisos, y sufren el problema de disponer de diversas interpretaciones posibles para una misma frase. Ese no es sin embargo, el caso con ningún lenguaje informático. Puedes escribir sentencias completamente claras e inequívocas en BASIC, y si sabes las reglas, el resultado de cualquier serie de sentencias es único y no puede argumentarse sobre él.

Un problema es que los programas grandes que contienen muchos centenares de **instrucciones** en BASIC, pueden ser demasiado complejos para poderlos seguir fácilmente y el resultado puede necesitar muchas horas de trabajo para comprenderlo. Te encontrarás que eso es cierto con programas que tú mismo has escrito, pero que no has examinado desde hace bastante tiempo. No es fácil observar un programa de ordenador y comprender ipso facto cómo funciona. Una manera de evitar la confusión es describir el problema -y por tanto el método de solución- usando una forma gráfica antes de escribir el programa. Tales diagramas gráficos se denominan **ordinogramas** -y también "diagramas de flujo"- y la mayoría de la gente los usa, o usa algo similar, con bastante naturalidad en su vida diaria.

Ordinogramas

Ahora que podemos hablar sobre los "objetos" que un programa trata, usando identificadores, dándoles valores e incluso exponiendo a voluntad esos valores en pantalla; estamos preparados para volver a la primera ley de la computación. Cualquier problema debe desglosarse en pequeñas acciones, cada una de las cuales ha de describirse usando palabras que el ordenador pueda comprender, y así sabe exactamente lo que hacer en cualquier momento. Sin embargo, para escribir la solución a un problema como un programa de ordenador, debemos aprender gran cantidad de cosas sobre programación, incluyendo el significado de todas las palabras que el lenguaje ofrece.

Para ahorrarte todo este trabajo de aprendizaje antes de que puedas incluso comenzar a entender la programación, usamos un método visual de desglosar el problema e ilustramos la serie de acciones y el **orden** en que han de realizarse usando un **ordino**grama. Un programa representado mediante un ordinograma no requiere ningún conocimiento de las palabras disponibles en un lenguaje particular, y debiera poderse **traducir** a cualquier lenguaje de cualquier máquina. Sin embargo, usaremos conceptos del SuperBASIC del QL siempre que sea posible en todo lo que sigue, para asegurar que son fáciles de convertir posteriormente al dialecto concreto del BASIC que estamos tratando.

Para mostrar los principios fundamentales, se ilustra un ordinograma muy simple en la Fig. 2.1. Con eso se te permite ver las diversas acciones que tendrían que llevarse a cabo para conseguir la solución del problema. En éste se detallan todas las acciones asociadas con el ordenador, y simplemente tienes que seguir **el curso** marcado por las **flechas.**

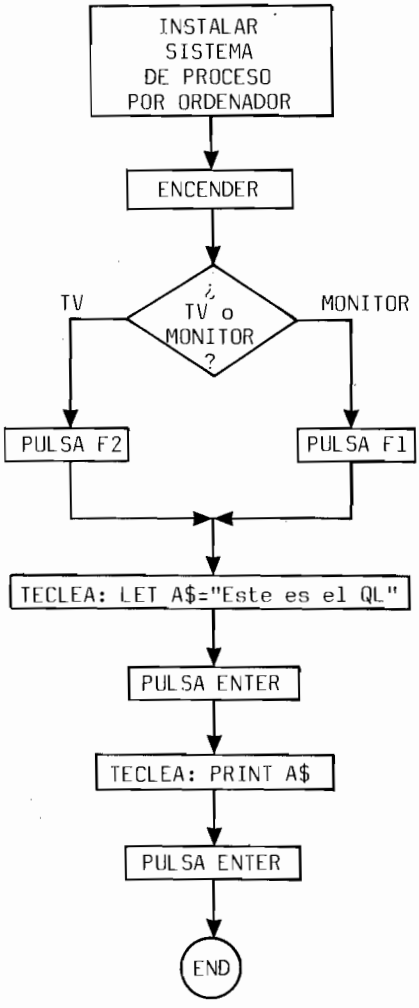


Fig. 2.1. Un ordinograma

Comienza diciéndote que enchufes el ordenador. La siguiente parte del diagrama te dice que conectes la alimentación. Eso son acciones a realizar, pero no son traducibles al QL: son para que el operador humano las lleve a cabo. El principio sin embargo es el mismo. Comprendes las frases usadas, y puedes traducirlas en acciones que tu puedes ejecutar. La siguiente acción, encerrada en un rombo, representa una **decisión.** En realidad efectúa una pregunta y sigue uno de los dos cursos de acción **alternativos** que parten del rombo, dependiendo del tipo de respuesta que se obtenga.

Si estás usando un receptor de TV, debes saber a estas alturas que tienes que pulsar la tecla rotulada F2 en el teclado para pasar a la siguiente acción. Si estás usando un monitor de video, F1 te llevará por el camino correcto. Esta capacidad que los humanos tienen para tomar decisiones,se refleja en BASIC por diversas palabras clave y te las presentaremos en capítulos ulteriores. Después de pulsar la tecla F correcta, estás preparado para teclear las sentencias del programa. Eso se efectúa exactamente como se indica, hasta que alcanzas el círculo de **finalice**, marcado END. Ninguna acción se toma en ese momento -por ahora- y simplemente muestra dónde termina el diagrama.

Este ordinograma introduce la mayoría de los conceptos que necesitas conocer sobre este método gráfico de representación: hay varios comandos normales o "recuadros de sentencia"; hay un "rombo para decisión", y hay un "círculo para final". En este caso, el curso de acción **fluye** desde arriba hacia abajo en el papel. Eso no es necesario, siempre y cuando recuerdes incluir flechas para indicar el curso de acción que deseas en cada momento.

Un ordinograma es un método simple y claro de representar incluso las **tareas** más complejas, y se recomienda firmemente como medio de agrupar tus ideas para conseguir un programa. Tristemente, a medida que uno se hace más eficiente en programación, a menudo pasa a despreciar la "faena" de producir un ordinograma y raramente verás uno completo para un programa de otra persona. La confusión resultante que esto provoca puede evitarse en parte si se documenta correctamente el programa mientras se está escribiendo. Usar identificadores con significado propio es de gran ayuda, pero el BASIC también te permite incluir nota y comentarios que sirvan de **memor**andum como veremos.

Por el momento, recuerda que un programa por sí mismo es difícil de leer y debiera haber siempre algún medio de explicarlo más, ya ea incorporando **memos** o representándolo por un diagrama. El de la fig. 2.1 contiene una gran cantidad de información sobre la manera en que usas el QL, y como sería repetitivo si lo incluyeramos en cada ordinograma, normalmente se omite; y además lo que importa son las **acciones** y los **datos** a manejar en el programa. Habitualmente escribiremos el ordinograma tal y como se muestra en la Fig. 2.2.

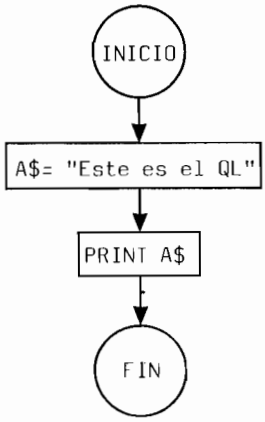


Fig. 2.2. Forma condensada de un diagrama de flujo

Bucles

Uno de los aspectos más útiles sobre un sistema de cómputo es que es capaz de efectuar acciones **repetitivas** precisamente, y sin que se llegue a aburrir. Por ejemplo, puedes haber reflejado una lista de nombres en una serie de variables, y descubrir posteriormente que has deletreado incorrectamente uno de los nombres. Supongamos que te gustaría cambiarlos en un momento dado. Si sólo hay cinco variables en la lista, podrías exponr sus valores en pantalla usando cinco comandos PRINT. Eso expondría los nombres en pantalla y mostraría cuál es el incorrecto. Luego podrías usar una sentencia de asignación mediante LET para corregir el nombre.

Si esta operación de exposición en pantalla tuviera que realizarse una y otra vez, puedes pensar en ella como una forma de dar vueltas y más vueltas, y decimos que deseas efectuar un **bucle de operaciones.** El origen de esta palabra y de la palabra similar **lazo,** también usada, se hará más clara cuando mostremos el ordinograma correspondiente para esa actividad **repetitiva.** Toda la operación puede que tarde apenas dos minutos, y no necesita ser mecanizada.

Supongamos sin embargo, que un programa ha producido un total de mil nombres, cada uno asignado como valor a una variable separada, y además que hay varios de ellos incorrectamente escritos, e incluso no sabes cuántos. Eso sí que pasa a ser un problema de computación, y tú con toda seguridad te aburrirías y además cometerías una cantidad enorme de errores si tuvieras que efectuar a mano la **tarea repetitiva** de exponerlos y corregirlos por ti mismo. Muy pronto veremos cómo un sencillo programa te ayuda a resolver el problema.

Denominando a las variables

Si recuerdas uno de los primeros conceptos presentados, sabrás que tenemos que decidir cómo rotular o identificar los lugares de memoria donde se depositan los datos, en este caso los nombres, para que el ordenador pueda efectuar alguna tarea sobre ellos. Recuerda que tratamos de nombres **-series de letras-** que no son números -cosas que se multiplican o suman- y que por tanto, debemos considerarlos como un dato **literal.** Eso significa que sus identificadores deben terminar con el signo $.

Hay un montón de métodos válidos para identificar esos datos. Podríamos usar los identificadores de literales:

A$, B$, C$, etc.

excepto que pronto nos quedarímos sin letras en el alfabeto. Podríamos recurrir a:

AA$, AB$, AC$... y así sucesivamente.

Sin embargo, eso nos da 676 variables diferentes, lo que todavía no es suficiente. (Las letras en el QL sólo son 26, ya que ¡la Ñ no existe!).

Recurrir a combinaciones de 3 letras sería necesario para poder conseguir los mil diferentes identificadores que necesitamos. Hay otros problemas asociados con esta manera de identificar, y en general verás que es esencial recurrir a **sufijos** numéricos para identificar datos cuando tienes una gran cantidad de ellos. Naturalmente también nosotros usamos en el habla normal números. Por ejemplo, los "números de teléfono" son en realidad "literales"; o si no prueba sumando el tuyo y el de tu primo a ver si sale el de tu abuelo.

Recurrir a números para nombrar cosas es bastante natural. En parte es debido a que tenemos asociaciones especiales con este tipo de identificación, y en parte porque queremos guardar una relación **secuencial** en los objetos a que hacen referencia. Por tanto, hablamos de primero, segundo, tercero... ya que el 1 va antes del 2 y éste antes del 3, y así sucesivamente.

La operación sobre un gran conjunto de identificadores se efectúa añadiendo **sufijos a un nombre común,** y la mayoría de los lenguajes de computación también permiten eso. En la vida normal, podríamos llamar a nuestros datos con la siguiente clase de identificadores:

A0$, A1$, A2$, A3$, A4$,....., A998$, A999$

Estas **variables** son completamente válidas en BASIC, ya que comienzan por una letra, sólo usan símbolos admitidos y terminan con un signo $. Podríamos trazar un ordinograma bastante fácilmente para el problema usando esos identificadores. Los números que aparecen en ellos se llaman **índices** o también **subscritos** (porque antes los escribíamos debajo, i.e. sub); pero debes mirarlos en este momento como un método de dar identificadores a unas variables, pero no pueden usarse en aritmética. Sin embargo, podemos ser capaces de usar la parte numérica de estos identificadores en operaciones aritméticas con sentencias BASIC, si resulta que encerramos los subíndices entre **paréntesis.** La colección de variables se escribiría entonces:

A$(0), A$(1), A$(2), ..., A$(998), A$(999)

Como puedes ver, incluso un simple objeto a tratar tal como un identificador, tiene que escribirse de acuerdo con una serie precisa de normas. Cada identificador debe comenzar con una letra, en este caso la A. Examinando estos "rótulos" podemos decir que corresponden a lugares para **literales** ya que terminan en el signo $, y como tienen subíndices entre paréntesis decimos que pertenecen a una misma **colección de variables** o a una variable **múltiple.** El signo $ tiene que ir antes del paréntesis de apertura. Son reglas que deben simplemente aprenderse y utilizarse.

Ya veremos que podremos realmente operar sobre los valores de los subíndices para **seleccionar** la variable del "colectivo" que nos interesa. Este tipo de variables que representan una colección de variables del mismo género se denominan **variables múltiples** o se dice que forman una 'ristra' de variables (también se llaman **tablas** y **matrices**). Volveremos a estudiarlas más detenidamente en un capítulo posterior.

El ordinograma del problema

Para continuar con el problema propuesto, supondremos además que el dato que ha sido mal deletreado es el de ANN, que debe corregirse obviamente porque se refería a ANA.

La figura 2.3 muestra un ordinograma que finalmente resolverá el problema. Las operaciones son repetitivas, y la primera acción es constatar si el primer miembro de la colección de variables contiene el valor ANN. Si así es, se ejecuta la sentencia de asignación y A$(0) se hace igual a ANA. Con eso habremos efectuado la corrección del nombre mal deletreado. Observa cómo se usa el signo = para hacer una pregunta en la condición previa para la decisión, pero dicho signo = se usa en la sentencia de asignación como un método de decirle al ordenador que haga una variable igual a una constante literal. Estas dos diferentes utilizaciones del mismo signo = se harán más claras a medida que prosigamos.

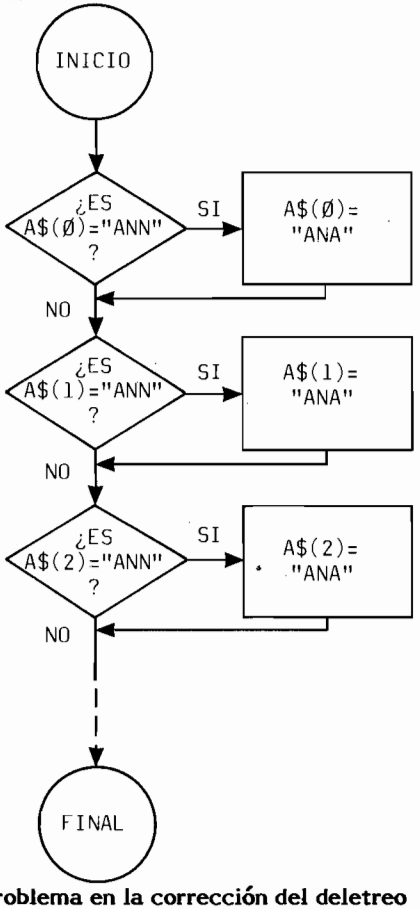


Fig. 2.3 Problema en la corrección del deletreo

La siguiente operación en el diagrama es la misma que la primera, pero actúa sobre la segunda variable de nuestra colección; y así sucesivamente. La línea de puntos en la parte inferior del diagrama implica que el proceso continúa y continúa **reiteradamente** hasta que esté completo. Eso obliga a 1000 operaciones de comparación y 1000 operaciones de asignación si escribieramos completamente el diagrama.

El diagrama sería extremadamente tedioso para convertirlo en un programa, y ciertamente no nos ahorraría ningún tiempo. Necesitamos una manera de mecanizar esta repetición de acciones iguales. Se efectúa mediante un **bucle,** tal y como se ilustra en la Fig. 2.4.

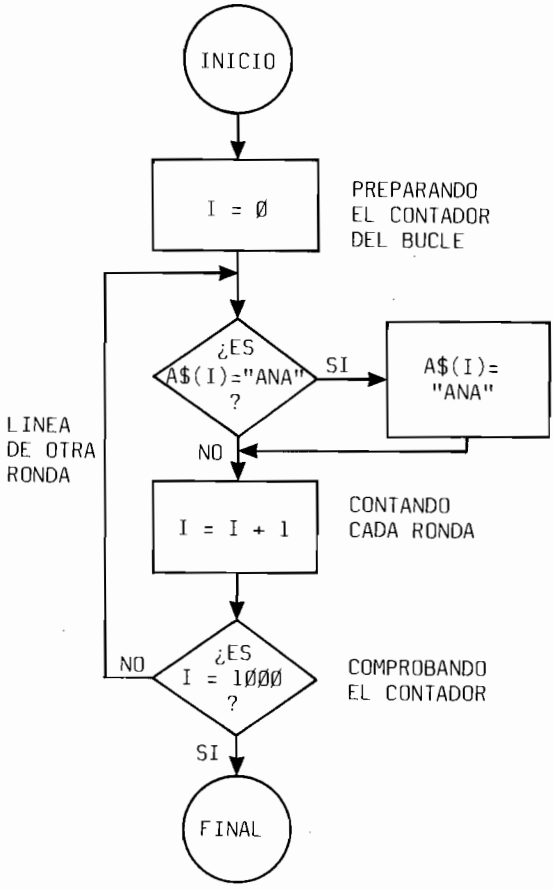


Fig. 2.4. Un bucle

Esta **rutina** sólo tiene cinco recuadros de sentencias, pero verás que sustituye los dos mil recuadros que hubieramos debido hacer en el último diagrama. Hay dos conceptos a comprender en la Fig. 2.4: primero, el concepto del **contador de rondas**, y el segundo, el propio **bucle.** El bucle en sí mismo consta de dos acciones de decisión, y una operación de suma; la sentencia:

```
I = I + 1
```

En algunas de las ocasiones también incluye la sentencia de asignación:

```
A$(I)="ANA"
```

pero sólo si la respuesta a la primera pregunta es SI. Imagina la situación cuando la respuesta a ambas decisiones que aparecen en la figura 2.4 fuera todas las veces NO. Si sigues las flechas del diagrama rápidamente verás que estás dando vueltas en círculo una y otra vez. Cada vez que se efectúa una vuelta -una ronda- el valor de la variable I -el contador de rondas- incrementa en una unidad. Por lo tanto, este contador I simplemente va acumulando el número de veces que se ejecutan las instrucciones que componen el bucle.

El programa procede a partir del principio en la forma siguiente. El contador de bucle I parte del valor cero, de acuerdo con la primera sentencia de asignación. Decimos que así se estipula el valor inicial. En el primer recuadro de decisiones, se efectúa una pregunta, o mejor dicho, se **comparan** dos valores. **Si** dichos valores coinciden, es decir: si se cumple la condición, hemos de pasar por la rama en que corregimos el error y hacemos que A$(I) -en este caso, A$(0)- sea sustituido por el valor correcto ANA. En los **demás** casos, es decir: cuando no se cumple la condición, iríamos por la rama del NO al efectuar la siguiente operación.

En ambos casos, la siguiente sentencia que se encuentra hace que se incremente en una unidad el contador del bucle. La hemos señalado como una forma muy habitual de sentencia de ordenador. Deliberadamente hemos dejado fuera la palabra clave usada LET para ilustrar la importancia de comprender el significado real del signo igual (=) en computación.

La sentencia:

```
I = I + 1
```

es un **comando** (no una ecuación matemática) y sería ilógico suponer que alguna vez pueden ser iguales I e I+1. El comando se usa para que el ordenador **haga** que el valor de I sea igual al valor que resulta de sumar la unidad a dicho valor de I. Por tanto, en esta primera pasada por el bucle, I queda cambiado de 0 a 1.

El siguiente recuadro vuelve a ser una decisión y pregunta si I ha alcanzado ya el valor de 1000. Como éste no es el caso, se toma la rama NO y a través de la línea de retroceso del bucle hacemos que **vaya** a efectuar la primera de las comprobaciones. Esta vez, el que se comprueba será A$(1) y de nuevo se efectuará la corrección si se cumple la condición. Se cumpla o no, la siguiente acción es incrementar I que pasa del valor 1 al valor 2, y se repite todo el proceso con este nuevo valor de I. Así, al ir incrementando I, se van comprobando todas las variables de la lista y ¡corrigiendo todas las que tengan "ANN" para ponerles "ANA"!

Después del tiempo correspondiente, le llegará su turno a A$(998) y luego a A$(999). Estas son las dos últimas en la lista, porque la lista comenzó en A$(0), y desde el 0 hasta el 999, ambos inclusive, hay 1000 números. Después de haber comprobado A$(999) el valor de I se ha incrementado y es de 1000. Por tanto, la segunda de las comparaciones que se efectúan, da un resultado cierto, ya que I=1000, y por tanto se toma la rama SI que nos lleva a la finalización del programa.

Toda la colección de datos literales ha sido tratada en muy corto intervalo de tiempo por un rápido programa de ordenador, y además el propio programa sólo se tarda en escribir un minuto o dos para cualquiera que esté familiarizado con el BASIC. Eso ahorra quizás horas de trabajo, si se hubiera seguido el método de la Fig. 2.3.

Con este ejemplo, describimos una simple **rutina** (que es un diminutivo de **ruta** y de **rodar** que es dar vueltas y vueltas que es una de las labores más **rutinarias** que puede haber). Como puedes ver, puede ser parte de un programa mayor, y cada acción de las que deben repetirse, debe especificarse y escribirse en términos precisos y claros usando palabras que el ordenador pueda comprender.

La labor final de este capítulo es regresar de nuevo al problema de reordenar números según el orden ascendente. La siguiente sección te muestra cómo llevar a cabo esta tarea usando un ordinograma.

Números en orden ascendente

El problema de reordenar tres números en orden ascendente es un caso especial de un importante problema en el tratamiento de grandes cantidades de información. Por ejemplo, una mirada a cualquier guía de teléfonos, destaca la necesidad de reordenar los datos que hay en ella según un orden, en este caso, el orden alfabético. Incluso aunque en la guía haya centenares de miles de abonados, sólo se tarda un minuto en encontrar cualquiera de ellos. Esa labor sería casi imposible si no estuvieran ordenados tal y como lo están.

Es una exigencia habitual que la información esté ordenada de alguna manera, y las facilidades de los programas para "bases de datos" tales como **archive**, permiten que se consulte información con facilidad. El ordenamiento, también llamado clasificación, puede efectuarse de varias maneras. El método empleado para el ordinograma de la Fig. 2.5 es un caso especial de los denominados **método de burbujas.**

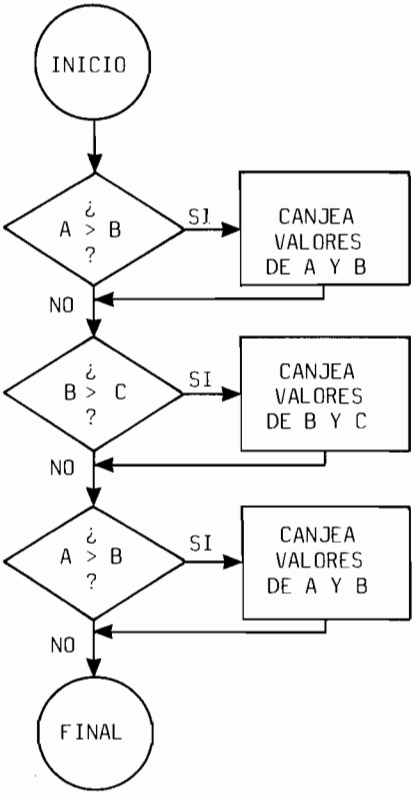


Fig. 2.5. Tres números en orden ascendente

Observa que en este ordinograma se ha utilizado un nuevo signo: el **mayor que** (>). Y por tanto, la expresión A > B que se pronuncia "A mayor que B" puede ser considerada como una **comparación** o como una operación de constatar **si** A es mayor que B o no. Es bastante obvio que se utilizará como **condición** previa para adoptar uno u otro curso de acción: una **decisión.**

A, B y C son identificadores, y sus valores cambiarán de acuerdo con las acciones establecidas en el diagrama de manera que queden finalmente en orden ascendente con el valor de A menor que el de B, y éste menor que el de C. Vamos a tomar un ejemplo con tres valores para ilustrar lo que sucede a medida que procedemos a lo largo del diagrama. Supongamos que inicialmente los números son:

A	B	C
9	3	1

Dichos números están precisamente en el orden contrario. La primera comparación en la Fig. 2.5 da como resultado la **certeza**, ya que efectivamente el valor de A es mayor que el de B. Por tanto, debemos intercambiar sus valores ya que debemos proseguir por la rama marcada con SI. En consecuencia, después de esa acción tendremos:

A	B	C
3	9	1

La siguiente acción es comparar el valor de B con el valor de C, y canjearlos si **B es mayor que C**; y no canjearlos en los demás casos. Por lo tanto, tendremos:

A	B	C
3	1	9

Ahora pasando al siguiente recuadro del diagrama, vemos que se nos pide comparar de nuevo los valores de A y B y canjearlos si A es mayor que B; lo que así sucede en este caso y por tanto efectuamos la operación de canje y terminamos con:

A	B	C
1	3	9

tal y como requeríamos.

Con tan pocos números no se nota muy bien, que los números mayores van sucesivamente adoptando posiciones más altas, hasta que después de todo el proceso terminan en el orden correcto. El símil es el de observar las **burbujas** en un vaso que van subiendo lentamente a la superficie, de ahí el nombre del método. El proceso también se denomina como ordenamiento por el **método de piedras**, ya que los números más "gordos" caen hacia el fondo como las piedras en un estanque.

Debieras ensayar con esta rutina con tus propios números, incluyendo quizás dos números que sean iguales y ver cómo quedan colocados después de efectuar el proceso. Observa que la Fig. 2.5 es un ejemplo de un proceso que podemos considerar como ya veremos, repetitivo, y es a lo que habría que recurrir si tuvieramos que ordenar mil números. En este caso hemos escrito comparación y acción para cada par separadamente. De hecho es posible usar un **bucle** similar a los ya vistos, y el capítulo 6 te muestra un caso como ejemplo, tanto para datos **numéricos** como **literales.**

Resumen

- Para tratar cantidades **variables** en el ordenador, debemos darle nombres que las **identifiquen.** Un identificador debe comenzar por una letra, pero puede contener letras y cifras y el signo de subrayar. Se pueden usar en un identificador hasta 255 caracteres. Es también provechoso que los identificadores tengan relación con el valor que contienen, y sean tan cortos como sea posible.

- Para dar un valor a una variable, es decir, para depositar un dato en el lugar de la memoria que la variable representa, usamos la llamada **sentencia de asignación.** Puede escribirse en la forma:

  LET A = 123

  o prescindir de la palabra clave LET y usar sólo

  A = 123

- Todas las palabras **clave** deben llevar un espacio, un "blanco" después de ellas a no ser que vayan seguidas de algo cuyo primer carácter no sea una letra ni una cifra ni el signo de subrayar (v.g. un paréntesis). En los demás casos, la palabra clave será considerada como siendo parte de un identificador formado por dicha palabra clave junto con todos los símbolos que aparezcan después de ella y hasta que aparezca el siguiente espacio o un carácter especial.

- Los datos **literales** deben estar encerrados entre **comillas** y serán considerados como simples **series de caracteres,** sin que puedan efectuarse operaciones aritméticas entre ellos (aunque haya cifras en ellos). Sus identificadores deben terminar con el signo $ para distinguir su especial naturaleza. Así, A$ = "Esta es una retahíla de 37 caracteres".

- Para **exponer** en pantalla el valor asignado a una variable, se usa la sentencia PRINT. Por ejemplo:

  PRINT A$

  haría que se mostrara en pantalla su valor:

  Esta es una retahíla de 37 caracteres

  pero observa que las comillas no forman parte del dato literal.

- Para ordenar según una regla un grupo de variables, pueden usarse subíndices numéricos después de un identificador común para todas ellas, pero los subíndices deben estar encerrados entre paréntesis. Las variables de esta clase se denominan **variables múltiples.** Una variable múltiple literal sería la colección de variables:

  NOMBRE$(1), NOMBRE$(2), ..., NOMBRE$(24), NOMBRE$(25)

  Cada una de estas variables **elementales** contiene un valor literal, y los números entre paréntesis -los subíndices- pueden venir dados por una variable, tal como el contador de un bucle.

- Los **bucles** son procesos en que se actúa reiteradamente sobre una colección de variables. El número de veces en que la serie de acciones que forman el bucle ha sido ejecutada queda reflejado por el valor de una variable concreta que se denomina **contador del bucle.**

- Los **ordinogramas** -los diagramas de flujo- dan un medio de describir en forma gráfica el método empleado para resolver un problema, para plantear un programa, que es más fácil de transmitir a otras personas. Si tú realmente no usas un ordinograma formal cuando estás programando, debieras como mínimo expresar las ideas principales de tu solución de esa manera. Te encontrarás que te ayuda a **estructurar** tus programas de manera más razonable y eficiente. También lleva a una metodología organizada para tus programas, y es de considerable ayuda cuando se están buscando las **pifias** o equivocaciones cometidas en la lógica del programa.

Capítulo Tres

Comenzando con SuperBASIC

En este capítulo veremos cómo se usan algunas de las más importantes palabras clave del BASIC, y examinaremos la habilidad del ordenador para actuar como una supercalculadora.

Uno de los conceptos centrales en computación es la idea de **programa almacenado.** En el último capítulo, se teclearon un par de líneas de programa y vímos cómo se ejecutaban inmediatamente: de ahí que las llamemos **comandos.** Veremos ahora cómo puede hacerse que las líneas de programa queden almacenadas para una ejecución posterior: de ahí que las llamemos **instrucciones.** Eso te permite construir programas largos que usan el lenguaje SuperBASIC con toda su potencia completa.

A medida que presentemos las palabras clave de las sentencias en BASIC, daremos ejemplos que te permitan comprender las ideas para la escritura y corrección de los programas, y a usar el teclado y la pantalla del QL con sus ventajas. Este capítulo sirve como presentación a las posibilidades especiales del QL, y será mencionado a lo largo del resto del libro.

Líneas de programa almacenado

El último capítulo presentó el programa como una **secuencia de instrucciones** para el ordenador, y se usaron ordinogramas para explicar algunas de las ideas principales en programación. Sin embargo, para construir un programa, debemos tener un método de escribir y conservar la secuencia completa de sentencias antes de obligar al ordenador a que ejecute el programa. Simplemente no es sensato ejecutar cada sentencia a medida que se escribe y antes de concluir el programa.

Uno de los puntos más cruciales es que las instrucciones del programa deben ejecutarse **sucesivamente.** Eso es muy parecido a ordenar un conjunto de números. Nosotros deseamos que las instrucciones se ejecuten desde la primera a la última, según un orden predefinido. Muchos lenguajes de computación, incluyendo el SuperBASIC, consiguen eso identificando cada línea del programa por un **número.** El ordenador, luego, simplemente ejecuta las instrucciones del programa manteniendo un **registro seguidor** con el número de línea que le toca ejecutar en cada momento. Y así es fácil.

Podemos dar a cada línea de programa un número simplemente tecleándolo antes de decirle la acción correspondiente. Cuando el ordenador ve un número al comienzo de una sentencia, sabe que no debe ejecutarlo inmediatamente, sino que debe depositar dicha línea en la memoria usando el número como rótulo identificativo.

No ejecutará la sentencia expresada en dicha línea hasta que se le mande hacerlo posteriormente mediante un verdadero comando. Por ejemplo, convertiríamos en programa las dos líneas mencionadas en el último capítulo si escribieramos:

```
10 A$ = "¡Este es el QL!"
20 PRINT A$
```

El ordenador automáticamente conserva dichas líneas en memoria y espera a que le mandes hacer algo con ellas. Se puede usar cualquier número para identificar las líneas de programa, siempre y cuando estén de acuerdo con el orden de ejecución. Ensaya tecleando esas líneas, recordando que todo se concluye siempre pulsando la tecla ENTER, y que hay que dejar un espacio en blanco como mínimo después de la palabra clave. Todos los otros espacios en blanco son superfluos, pero se te recomienda que los uses para mayor legibilidad. Si cometes algunas equivocaciones al teclear este programa, te encontrarás que el ordenador todavía está comprobando que la ortografía y la sintaxis de las frases empleadas son de acuerdo con las normas dadas.

Lo anterior es un ejemplo ridículo de un programa almacenado, pero quizás sea sólo el primero. El programa queda almacenado en la memoria, y aparecerá en pantalla con un fondo en color blanco para mostrarte que no está en la ventana de salida. Si tienes un monitor de video, aparecerá de todas formas en la ventana blanca. Observa cómo a medida que tecleas, los caracteres que componen las líneas son **repicados** en la parte inferior de la pantalla. Continuarán allí hasta que se pulse ENTER, con lo que se le dice **adentro** y el ordenador pasa a depositarlos en la memoria para usarlos posteriormente. Sólo en ese momento es cuando aparecen expuestos en la parte superior de la pantalla.

Edición (revisión, corrección de líneas)

Si cometes un error al teclear una línea y lo observas antes de pulsar ENTER, puedes alterarlo usando las reglas descritas en el Capítulo 1. Si no te percatas del error hasta que hayas mandado adentro la línea y está por tanto, expuesta en la parte superior de la pantalla, puedes alterarla volviendo a teclear dicha línea. Por ejemplo, si ahora tecleas:

```
10 A$ = "Hola"
```

y pulsas ENTER, la línea vieja numerada 10 será sustituida por esta nueva línea con el mismo número 10. La vieja versión ha desaparecido completamente y para siempre.

Otra importante faceta de la **edición** es la que te permite añadir más líneas al programa, en cualquier momento que lo desees. Por ejemplo, supongamos que quieres añadir una línea de programa que ha de ejecutarse después de la 10 y antes de la 20 que ya tienes: simplemente elige cualquier número entre el 10 y el 20 y úsalo para numerar dicha línea. Por ejemplo, teclea lo siguiente:

```
15 B$ = "Esta es la línea extra."
```

Cuando pulses ENTER, verás que la línea queda **insertada** en el lugar correcto que le corresponde en el programa.

Eso te muestra también una buena razón para numerar siempre los programas en **saltos** de 10. Eso te deja un montón de sitios entre las líneas para insertar más instrucciones.

Si deseas **suprimir** una línea, simplemente teclea el número de esa línea, pero sin escribir nada más. Ensaya tecleando:

```
15
```

Cuando pulses ENTER, el ordenador reconocerá que le has tecleado el número de línea 15, línea que ya tiene en memoria, y pasará a sustituir la línea en memoria con la que acabas de teclear. En ese momento, observa que en la tecleada no hay ningún comando, y por tanto interpreta que debe sustituir la vieja línea 15 con una línea vacía. Así, el efecto conseguido es quitar completamente la línea 15.

Para eliminar el programa existente en memoria y comenzar de nuevo, simplemente mándale que **vacíe** la memoria, que en BASIC se dice con la palabra clave NEW (que la deja como "nueva"). Pero no lo hagas todavía. Otra operación asociada con la **limpieza** de la memoria es la que usa la palabra clave CLEAR (que la deja en "claro"), ya que anula todos los valores de las variables que se hayan usado hasta el momento. Son órdenes que usaremos más adelante.

Estas operaciones dan un método rápido y simple de editar un programa. Veremos más adelante en este capítulo que hay otro, más inteligente para editar programas, es decir, para revisarlos, corregirlos y alterarlos.

Ejecutando el programa

La **ejecución** del programa por la máquina se llama vulgarmente pasar, correr, etc. La palabra clave en BASIC para determinar esta acción es RUN, que en inglés significa correr, rodar, etc. Ensaya tecleando el comando de **ejecución:**

```
RUN
```

Dado que no lo has precedido de número de línea, esta sentencia BASIC es considerada un comando y es ejecutada inmediatamente después de que pulses ENTER. Como consecuencia de dicho comando, el BASIC pasa a ejecutar el programa, es decir, a cumplimentar sucesivamente las instrucciones que contiene y comenzando por la primera que haya con el número más bajo. Si tienes el programa que nos sirve de ejemplo, en la pantalla deberá aparecer:

```
Hola
```

Dado que el programa ha sido almacenado según orden creciente de los números de línea, primero se ejecutará la 10 y luego la 20. Pero observa que todavía sigues teniendo el programa en memoria y lo puedes cambiar o volver a ejecutar a discreción. Si estás usando una TV, te encontrarás que la salida producida por este programa puede haber **escrito encima** de las líneas de programa que había en la pantalla. Sin embargo, el programa que tiene conservado en memoria permanece perfecto. Ensaya tecleando RUN de nuevo unas cuantas veces.

A medida que se ejecuta una y otra vez, la pantalla se va llenando con la palabra: Hola. Continúa haciendo que se ejecute hasta que tu pantalla de TV no muestre ninguna de las líneas de programa en absoluto, y esté completamente llena con la ventana de salida en fondo rojo. Si estás usando un monitor de video, puedes exponer tanto el propio programa como la salida que produce en la pantalla, y cada uno en su ventana correspondiente.

Listando el programa

Para hacer que **liste** las líneas del programa existente en memoria, y puedas verlo en la pantalla, usas otra clave del BASIC: el verbo LIST. Ensaya tecleando eso y observa el efecto.

Si tienes una TV, el programa reaparecerá aunque puede que lo haga mezclado con la información de salida del propio programa si estaba llena la pantalla. Si estás usando un monitor, el programa se repetirá de nuevo debajo de la última línea expuesta. Siempre puedes usar LIST para examinar el programa almacenado. Si el programa es demasiado largo como para encajar en la pantalla, puedes listar parte de él si lo deseas, como ya veremos.

Aritmética

Una de las labores más importantes que un ordenador puede efectuar es la de operaciones aritméticas. Posteriormente veremos lo notablemente potente que el QL es con los cálculos. Sin embargo, veamos primero cómo el ordenador efectúa simples operaciones usando sólo los **operadores** de suma, resta, multiplicación y división.

Para efectuar una adición o una sustracción, teclea las operaciones de manera similar a una calculadora. La única diferencia es que tienes que usar un comando para decirle al ordenador lo que quieres que **haga** con el resultado de la operación. Por ejemplo, que lo **exponga** en pantalla, y para ello basta decírselo con su palabra clave PRINT. Ensaya tecleando:

```
PRINT 123 + 456
```

Como no va precedida de número de línea es ejecutada inmediatamente, y la respuesta aparecerá en la ventana de salida por debajo de lo último expuesto -y puede que sea difícil de ver. Siempre puedes usar el ordenador de esta manera, ya que el BASIC reconoce los números ordinarios y los signos + y -. Ensaya tecleando:

```
PRINT 579 - 123
```

La respuesta 456 debe aparecer después de pulsar ENTER. Ensaya unas cuantas operaciones más, quizás usando números negativos y números con parte fraccionaria.

Todos estos cálculos darán como resultado una respuesta inmediata en la pantalla, ya que son sentencias dadas en forma de comando.

La multiplicación se efectúa como antes, pero el símbolo usado no es la **x** porque podría confundirse con la letra del alfabeto. El símbolo universal en BASIC para la multiplicación es el asterisco (*). Ensaya ahora tecleando:

```
PRINT 12 * 10
```

Aparecerá la respuesta 120. Ensaya unas cuantas operaciones más usando el asterisco y observa los resultados. También aquí, es muy similar a una calculadora.

Para dividir dos números, se usa la barra inclinada (/). Así teclea:

```
PRINT 12 / 10
```

y verás que aparece la respuesta 1.2 (se usa el punto y no la coma para separar la parte entera de la parte fraccionaria de un número -no en vano el QL es inglés).

Orden de evaluación

Cuando se combinan varias de las operaciones anteriores, hay que aprenderse algunas reglas adicionales. Por ejemplo, si quieres evaluar la siguiente **expresión**, o sea, la serie de cálculos:

```
12 + 10 * 8
```

tendrás que saber en qué orden van a ser efectuadas las diversas partes de este cálculo. Si tecleas eso en las más simples calculadoras, te encontrarás que la respuesta que dan es 176. Ensaya con el ordenador, diciéndole que exponga el resultado, y verás lo que da. La respuesta es 92. Eso no es debido a ninguna inexactitud en una u otra máquina, sino que es debido a que las dos máquinas efectúan los datos según diferentes **prioridades de operaciones.** La calculadora sumará primero 12 y 10 y el resultado lo multiplicará por 8; mientras que el ordenador multiplicará primero 10 por 8 y luego sumará el 12 al resultado.

Las reglas empleadas en BASIC es que todas las multiplicaciones y divisiones se efectúan primero de izquierda a derecha, y luego se efectúan las sumas y restas también de izquierda a derecha. Por ejemplo, en la expresión:

```
10 + 10 / 5 * 6
```

la división y la multiplicación se efectúan primeramente de izquierda a derecha; i.e. la división viene primero, y luego la multiplicación. La suma es de **menor prioridad** y es la efectuada en último lugar por el ordenador, con lo que se obtiene 22 como resultado. Puedes comprobar que estará en desacuerdo con la mayoría de las calculadoras. Además, si las operaciones de / y * se efectuaran de derecha a izquierda, sería 6 por 5 igual 30, y 10 dividido por 30 es 0,3333 que al sumarlo a 10 no daría el 22 que hemos dicho.

Es muy restrictivo tener que maniobrar con una expresión para asegurarse que se ejecuta correctamente, y además puede que quieras específicamente que la multiplicación se ejecute antes en el ejemplo anterior. Puedes cambiar el orden normal de ejecución si usas **paréntesis.** La regla que prevalece es que el interior de los paréntesis se evalúa antes de todo lo demás. Así, si escribieramos lo anterior como:

```
10 + 10 / (5 * 6)
```

se ejecutaría la multiplicación 5*6 antes antes que todo lo otro. El resto de operaciones seguirían las reglas establecidas, así que se divide 10 entre 30, y luego se le añade al resultado 10 para obtener 10,33333.

Como puedes ver, el orden en que se evalúa una expresión es importantísimo. Las normas anteriores son meramente un **convenio** al que todo el mundo debe adherirse.

Algunos ejemplos

Ensaya tecleando lo siguiente y trata de adivinar cuál va a ser la respuesta antes de pulsar la tecla ENTER, en cada ejemplo:

```
PRINT 15 - 3 * 3

PRINT 15 - 3 / 3

PRINT 3 / 3 - 3

PRINT 18 - 5 + 6 * 2 + 1

PRINT 100 * 0.5 / 25 + 20

PRINT 10 * (0.5 / 25) + 20
```

Las calculadoras dan diferentes resultados para cada una de las expresiones anteriores porque habitualmente utilizan **un acumulador** para ir reflejando la operación inmediatamente hecha después de introducida. Además, a no ser que sean programables, no pueden almacenar la serie de instrucciones. ¿Cuántos de los seis cálculos anteriores darían lo mismo en una simple calculadora?

Para usar estas operaciones aritméticas en programas, a menudo tendremos que usar variables para contener los números que vienen en la expresión, así como para los resultados. Por ejemplo, ensaya tecleando NEW para eliminar cualquier programa previo que hubiera en memoria, y luego teclea el siguiente programa:

```
10 A = 10
20 B = 20
30 C = 20
40 D = A + B * C
50 PRINT D
```

Recuerda que el único espacio que siempre debes incluir es el que va detrás de la palabra clave, en este caso PRINT, y que los identificativos de las variables pueden ser en mayúsculas o en minúsculas, y que el ordenador los considera como la misma variable. El verbo PRINT también puede ir en mayúsculas o en minúsculas, o en una mezcla de los dos. Si ensayas, te encontrarás que el ordenador siempre cambia las claves a mayúsculas al listar el programa.

Cuando quieras que el ordenador **ejecute** el programa, teclea RUN concluyendo el comando pulsando ENTER, con lo que aparecerá la respuesta 50. Recuerda que puedes volver a ejecutarlo una y otra vez. También puedes hacer que lo **liste** en cualquier momento, para ver en pantalla las líneas de programa.

Después de ejecutar el programa, los valores de las variables siguen conservados en sus lugares identificados de la memoria. Así que si das por ejemplo, el comando directo:

PRINT D

El valor de D, que era 50, volverá a exponerse en pantalla. Si repites este comando, aparecerá el 50 tantas veces como lo teclees, mostrando que el valor de la variable D no se ve afectado porque lo expongas en pantalla. También puedes examinar los valores de cualquier otra variable que haya sido utilizada. Es muy provechoso al intentar encontrar lo que ha sucedido en un programa complejo, que puede contener pifias. Simplemente haces que **pare el programa** si no se ha detenido por sí mismo, y haces que **exponga** el valor de la variable que te interese como posible pista. Ya veremos cómo haces que se detenga la ejecución de un programa más adelante. Los valores de las variables pueden ser **anulados** si das el comando CLEAR. Ensáyalo ahora, y luego intenta exponer el valor de D o de cualquier otra variable.

Te encontrarás que el ordenador no te expone simplemente el valor 0, sino que realmente te dice al mostrarte el **asterisco** que a dichas variables no se les ha asignado todavía ningún valor. (En realidad sí se le asignó pero quedó **cancelado** al haber dado el comando de limpieza CLEAR).

Literales

Recuerda que una retahíla de caracteres cualesquiera puede depositarse en el ordenador, usando en los identificadores un signo $ como último carácter. Es un dato **literal** por contraposición al **numeral,** y no se refiere a que los caracteres han de ser letras, pueden ser números; sino a que su índole o naturaleza no es numérica.

Además es posible usar el signo de operación & entre valores literales. Esta operación simplemente **empalma** o adosa un literal con otro formando una serie consecutiva. Se denomina también **concatenación.** Teclea por ejemplo NEW y luego el programa:

```
10 A$ = "HOLA"
20 B$ = "PEPE"
30 C$ = A$ & B$
40 PRINT C$
```

Cuando mandes que lo ejecute, las primeras dos instrucciones hacen que se asignen las constantes literales a las variables literales A$ y B$, y la tercera deposita el empalme de ambas en la variable C$, obtenida colocando sucesivamente los valores de A$ y B$. La instrucción PRINT te expondrá en pantalla por tanto el literal combinado "HOLA PEPE".

Si ahora tecleas:

```
30 C$ = B$ & A$
```

verás que lo que aparece es PEPEHOLA, mostrando que el orden sigue siendo igual de importante que siempre. Ese no es el caso para las operaciones aritméticas que son conmutativas como la suma o la multiplicación, pero no la resta o la división.

Una **serie de cifras** puede ser tratada como un **número**, o sea, un operando aritmético, o puede ser tratada como un **lítero**, o sea, un operando no aritmético, una simple **serie de caracteres.** Todo depende de cómo le digas al ordenador que debe considerarla. Por ejemplo, teclea:

```
10 A = 12
20 B = 15
30 A$ = "12"
40 B$ = "15"
```

y teclea RUN. Nada aparecerá en la pantalla, pero las cuatro asignaciones habrán sido hechas. Las líneas 10 y 20 ponen los **números** 12 y 15 en las variables numéricas A y B respectivamente. Las líneas 30 y 40 ponen los líteros 12 y 15 en las variables literales A$ y B$ respectivamente. Y esto último quiere decir que son tratados **literalmente,** lo que implica que los dos caracteres del literal 12 se almacenan en memoria, el 1 en una celdilla, y el 2 en la celdilla contigua de las dos ocupadas por A$. E igual con los caracteres 1 y 5 de B$. Y el ordenador en absoluto intenta entender lo que hay encerrado entre las comillas. No sucede así con los números 12 y 15 que son transformados a una notación interna adecuada para facilitar las operaciones aritméticas.

Ensaya ahora tecleando:

```
PRINT A + B
PRINT B + A
PRINT A$ & B$
PRINT B$ & A$
```

Estos cuatro comandos te darán tres respuestas diferentes. En los dos primeros, A y B son realmente **números** y la respuesta obviamente es 27. El tercer comando lo que hace es empalmar literales, y por tanto la respuesta conseguida es el literal "1215", aunque las comillas se quitan al exponerlo en pantalla. El cuarto comando por la misma razón dará "1512".

Las cifras que aparecen en las variables literales no son consideradas aritméticamente, son como cualquier otro símbolo del repertorio, y así intervienen al utilizar la operación de concatenación &; insistimos en que son series de caracteres que se tratan como si estuvieran **ensartados** unos a otros, formando una sola cadena.

En la mayoría de los ordenadores (y en los lenguajes más modernos, todavía mucho más) los literales que contienen sólo cifras como los anteriores, no serán reconocidos nunca como operandos válidos en operaciones aritméticas, incluso aunque aparentemente -sólo aparentemente- presentan información numérica. Por tanto, operaciones aritméticas entre literales hacen siempre que se produzca un error en la pantalla al ser ejecutadas (y en los lenguajes modernos simplemente al ser introducidas y antes de ser ejecutadas), que suele ser el de **discordancia de clase o tipo** de dato. Sin embargo, e **inexplicablemente** el QL está dotado de una facilidad especial que llaman **coerción** -que compensa esa imposibilidad lógica y presupone automáticamente que realmente quieres tratarlos como numerales y sumarlos (lo cual no es cierto en la mayoría de los casos, y es simplemente un error). Por tanto en el QL -sólo en el QL- puedes teclear la operación absurda:

```
PRINT A$ + B$
```

Aunque dará lugar a muchas confusiones, hay que reconocer que el QL se basa en que el signo de suma (+) es un **operador numérico** y por tanto ejecuta el comando convirtiendo el dato literal a numeral antes de efectuar la suma. La respuesta sería por tanto 27.

Tipos de datos

Hay dos principales **tipos de datos** -numerales y literales. Los literales son bastante sencillos, i.e. simples caracteres adosados para formar una cadena, que ocupan tantas celdillas de memoria como caracteres haya, por lo que no necesitan explicarse mucho más. Los números por otro lado, son un poco más complejos. Tenemos que saber la **banda** de números permitida y la **precisión** con que se representan, y la cantidad de celdillas que ocupan en memoria. Matemáticamente, no tiene ningún sentido intentar multiplicar números que tienen 15 posiciones fraccionarias en una máquina que sólo calcula con 5 ó 10 de ellas.

Los números en el QL se almacenan en memoria de dos formas diferentes, y se emplean dos notaciones. Todos los números hasta ahora se han supuesto, naturalmente, que están en la notación llamada de **coma flotante** (y estáte atento al lío de la coma usada por los hispanos y del punto usado por los anglófilos para separar la parte entera de la parte fraccionaria). Eso significa que la coma decimal, cuando es necesaria se sitúa en la posición correcta, y va cambiando de posición -"flota"- a medida que es necesario. El ordenador se preocupa de ello al igual que hacen la mayoría de las calculadoras.

La otra forma de almacenamiento se denomina de **número entero.** Los enteros nunca tienen permitido que aparezca en ellos una coma decimal, porque precisamente los llamamos enteros, i.e. sin parte fraccionaria (mejor que llamarla decimal). Pueden ser positivos o negativos, y su **banda** en el QL va desde -32768 (la cota inferior) hasta +32767 (la cota superior). (Ya reconocerás que con esa banda bastan dos celdillas de memoria para contener cualquier número entero). Como probablemente sabrás, cuando un número es positivo normalmente se omite el signo; pero si es negativo se coloca el signo - delante de él. Ese signo es el que se denomina **menos monádico,** o "menos unàrio" -porque sólo afecta a **un** operando.

El QL usa siempre ese convenio tanto para los números en coma flotante como para los números enteros. Las variables enteras son las que se distinguen dentro del ordenador adosando al final de su identificador el signo **porcentaje (%)**. Por ejemplo, NOMBRE% o D% o Esto_es_un_entero% serán siempre nombres de variables enteras. El signo % al final del nombre hace que el ordenador nos represente internamente de forma entera, ahorrando espacio e incrementando rapidez, y luego ya no permite que se usen para esos números la coma fraccionaria, y si lo haces automáticamente los redondeará al número entero más cercano.

La banda y la precisión de los números en coma flotante es un poco más complicada, y la trataremos a continuación.

Números en coma flotante

El QL habitualmente y si no se le especifica lo contrario -para omisiones- representa los números en coma flotante. Esta sección es bastante técnica, y si no estás interesado en operar con números muy grandes o muy pequeños te la puedes saltar; y volver a ella cuando la necesites.

El QL efectúa sus cálculos con 8 **cifras significativas** para los números, pero sólo expone los datos numéricos usando 7 cifras significativas, para lo que previamente redondea la última. Eso significa que por ejemplo, si se divide 10 entre 3, el QL mostrará como resultado 3.333333, aunque internamente y para nosotros ese número es 3,3333333. Es decir, la respuesta tiene 7 cifras significativas y está separada la parte entera de la parte fraccionaria mediante un **punto** (y los hispanos usamos para eso mismo la **coma**). En realidad, esa división da como cociente una fracción periódica pura, i.e. con los 3s llegando hasta ser infinitos, y cualquier clase de cálculo tiene que decidir la **precisión** con que quieres representar esa clase de números. Siete cifras es una precisión suficiente, y hay muy pocos cálculos que requieran más.

Es importante ser capaz de expresar y operar sobre números muy grandes. Para hacerlo más fácil para los humanos, se ha desarrollado un convenio especial, generalmente también utilizado en BASIC, se denomina **notación exponencial** o **científica.** Para ver cómo funciona, considera cómo escribirías el resultado del siguiente cálculo:

10.000 * 10.000 * 10.000

La respuesta debieras escribirla como...

1.000.000.000.000 (y los anglófilos 1,000,000,000,000)

y para ambos es un 1 seguido de doce ceros, que es el llamado billón (excepto para los USAmericanos, que para ellos es mil millones). Eso es muy complicado, y estamos propensos a cometer equivocaciones en la lectura y la escritura de esta clase de números. El problema se agudiza si el número es algo como:

9.436.894.607.987 (y los anglófilos 9,436,894,607,987)

Incluso decirlo no es nada fácil. Sería nueve billones, cuatrocientos treinta y seis mil, ochocientos noventa y cuatro millones, seiscientos siete mil, novecientos ochenta y siete.

Ambos de los números anteriores pueden expresarse mejor en otra notación, que emplea **potencias de 10.** Eso requiere que comprendas (o aprendas) la siguiente tabla:

10 elevado a la potencia 0 (10^0) = 1
10 elevado a la potencia 1 (10^1) = 10
10 elevado a la potencia 2 (10^2) = 100
10 elevado a la potencia 3 (10^3) = 1.000

y demás. **Elevar a una potencia** significa multiplicar el número por sí mismo tantas veces como indica el **exponente.** Por lo tanto, 10 elevado a la potencia 12 significa multiplicar 10 por sí mismo 12 veces; y ya sabes que el número de ceros después del 1 en el resultado es igual al exponente al que se eleva el número 10. Así, uno de los enormes números anteriores puede expresarse como:

1.000.000.000.000 = 10 elevado a la potencia 12

y puede escribirse en notación científica como la del BASIC, simplemente en la forma:

1E12

Eso significa 1 por 10 elevado a la potencia 12. El número 2.000.000.000.000 sería simplemente 2E12, que significa 2 por 10 elevado a la potencia 12.

Los números más complicados pueden también expresarse de esta forma. Por ejemplo, 2.269 pudiera expresarse como 2,269E3 (y los anglófilos lo invierten todo y dirían que 2,269 podría expresarse como 2.269E3, y así es en el BASIC).

El número tan largo y complicado anterior (9.436.894.607.987) puede expresarse como:

9,436894607987E12 (y en BASIC 9.436894807987E12)

Y tampoco es nada fácil de decir en la forma correcta, aunque se suele simplificar y se dice "nueve coma cuatro tres... y así hasta el siete final, y luego E12. Sin embargo, el QL no puede registrar números con esa **precisión** (y en raras ocasiones está justificado usar tantas cifras), y el número se **redondea** quedando como 9,436895E12.

Esta misma notación también es aprovechable y útil para almacenar números muy pequeños. Por ejemplo, el número 0,001 puede escribirse como:

$$\frac{1}{1000} = \frac{1}{1E3} = 1E\text{-}3$$

Observa que 1 dividido por 1E3 es lo mismo que 1E-3. Esa es una ley matemática (que ya conocías) y simplemente te la he recordado. Otro ejemplo de esta clase de números podía ser:

$$0{,}00000023456 = \frac{2{,}3456}{1E7} = 2{,}3456E\text{-}7$$

La regla es que el número de la parte izquierda de esta igualdad se cambia a 2,3456 (para que quede entre 1 y 10) desplazando la coma decimal las posiciones necesarias a la derecha. Ese es el origen del 1E7 del ejemplo, ya que la desplazamos 7 posiciones.

El QL, usando la notación científica anterior, puede almacenar números en coma flotante en la banda:

1E-615 hasta **1E615**

Esta **banda** es tan grande como para no significar prácticamente nada. Si esos dos números los escribieramos con todas las cifras, ocuparían aproximadamente la mitad de una página de este libro.

Como un ejemplo de cálculo, con números en coma flotante grandes y pequeños, ensaya tecleando:

PRINT .00075683 + 8.5E6

El resultado (y observa que debes poner punto donde habitualmente usas coma) muestra lo ilógica que es la operación, dado que añadimos un número muy pequeño a un número muy grande (algo así como dar 85 céntimos al que tiene miles de millones de pesetas). Ensaya tecleando:

PRINT .00075683 * 8.5E6

Aquí sí da un resultado razonable, y en general siempre tiene sentido multiplicar números grandes y pequeños. Veremos algunos ejemplos de cálculos con números en coma flotante más adelante, pero antes de que puedas usar el ordenador con todas sus posibilidades, hay otras cuantas facilidades muy importantes que debes aprovechar.

NUMerE y RENUMerE automáticamente

Estos dos **comandos** (y no pueden ser instrucciones) están provistas para ayudar en la escritura de programas. Cuando estás confeccionando un programa largo, es muy útil dejar que sea el ordenador el que genere **automáticamente** los números de línea que estás usando en el programa.

Si comienzas tu sesión de programación tecleando AUTO seguido de ENTER, entonces cada vez que teclees una línea y la concluyas pulsando ENTER, el ordenador te proporcionará un nuevo número de línea para la siguiente que vayas a introducir. El primer número será el 100 y los números sucesivos aparecerán con intervalos de 10, al igual que para los programas que hemos visto hasta ahora. También puedes alterar el número de comienzo y el salto, si tecleas dos números después de la clave AUTO. Los valores 100 y 10 son los denominados valores **prescritos** (para omisiones) del comando AUTO, porque si omites teclear otro valor, el comando AUTO adoptará esos números. El primer número tecleado después de AUTO es el número de comienzo, y el segundo es el salto o intervalo.

Por tanto:

AUTO 120, 20

comenzará con la línea número 120, y las sucesivas serán números en saltos de 20 en 20. Si dejas fuera el segundo de dichos **parámetros,** el salto automáticamente será de 10, que es el valor prescrito. Al igual que para otras palabras clave, AUTO para ser reconocida como tal debe ir seguida de un espacio en blanco.

Si de repente descubres que una de las líneas previamente tecleadas necesita corrección, o si has terminado con el programa, puedes **salirte** del modo AUTO tecleando CTRL y SPACE, que significa pulsar simultáneamente la tecla de **control** y la barra **espaciadora.** Pulsar ese par de teclas conjuntamente es lo que se denomina provocar una **ruptura** o brecha=BREAK, y también se usará para cortar la ejecución de un programa.

Ensaya entrando en el modo AUTO, y teclea varias veces PRINT seguida de la tecla ENTER. Con eso se ilustra adecuadamente este aspecto. Ahora ensaya con otro número de comienzo, y algún otro intervalo de salto. Un poco de práctica hará el uso del comando de **numere** bastante claro. Sal del modo AUTO, pero no borres las líneas de programa que has tecleado, ya que las aprovecharemos para la siguiente parte de esta sección.

Cuando corrijas un programa, te encontrarás con que insertas líneas extra que no corresponderán a la secuencia de números normales que estabas usando. Eso no importa en absoluto; pero puede incluso que te encuentres con que tienes que insertar más instrucciones entre algunas líneas y el intervalo entre los dos números de línea no es lo suficientemente amplio para tu objetivo. También hay otras muchas razones por las que puedes desear que el ordenador te **renumere** todo el programa, o parte de él, durante su desarrollo: el comando RENUM se usa para ese propósito.

Si tecleas simplemente RENUM seguido de ENTER, automáticamente todo el programa será renumerado a partir del número de línea 100 y en saltos de 10. Estos son también los valores prescritos para omisiones en el comando RENUM. Ensaya insertando algunas líneas extra entre las líneas del programa que ya tienes en memoria y teclea RENUM. Aparentemente no tiene ningún efecto sobre las líneas de programa mostradas ya en pantalla, pero desde luego habrá afectado al programa tal y como existe en la memoria. Para comprobarlo teclea el comando LIST. Con eso haces que la máquina **liste** el programa y lo muestre a partir de la última línea presentada, y "corriendo" la imagen anterior si es necesario. Puede que tengas algún problema intentando destacar el nuevo listado. Teclea de nuevo LIST y observa la pantalla cuidadosamene cuando pulsas ENTER.

Si deseas renumerar sólo una parte del programa, y deseas que dicha parte comience con un número de línea específico, y vaya en saltos determinados, entonces deberás especificar todos esos **parámetros** dentro del comando RENUM. La forma más general de dicho comando es:

RENUM A TO B;C,D

A es el número de la primera línea existente que va a ser renumerada, y B el de la última. C es el primer número de línea de la versión ya renumerada y D es el nuevo salto. Puedes omitir la condición **al B** (TO B), ya que el valor prescrito para omisiones será el último número de línea de la versión presente del programa. En otras palabras, renumerará todo el programa a partir de la línea número A y hasta el final del mismo. Si se omite D, el valor prescrito es 10. La nueva versión comenzará con el número de línea C. También observa que RENUM y TO son palabras clave del BASIC, y por tanto cada una requiere un espacio en blanco después de ella.

La única manera de ver cómo trabaja RENUM es practicar. Ensaya usando todos los parámetros del comando RENUM en el programa que tienes en este momento en memoria, e intentando predecir cuál será el resultado antes de pulsar ENTER -y luego observa si has acertado. Continúa practicando hasta que puedas siempre predecirlos correctamente. Es un comando muy útil y te encontrarás que lo usas más y más cuando estás escribiendo programas largos.

Liste y suprima líneas

El comando LIST puede usarse para que **liste** parte de un programa, añadiendo algunos números después de la clave para decir qué líneas quieres que liste. Por ejemplo, si tecleas:

```
LIST 10
```

sólo mostrará la línea 10. Igualmente,

```
LIST 10,50,60
```

solamente hará que se listen las líneas 10, 50 y 60. Ensaya con esto renumerando el programa para que comience a partir de la línea número 10 y vaya en saltos de 10, o introduce todo un nuevo conjunto de líneas para practicar.

Si tienes unas cuantas líneas de programa almacenadas, serás capaz de hacer que las **liste** si tecleas por ejemplo:

```
LIST 10 TO 50
```

Con eso haces que **liste** todas las líneas comprendidas en la **banda** de líneas que va del 10 **al** 50, ambas inclusive. Si no hay ninguna línea dentro de esa banda en tu programa, ensaya cambiando los parámetros anteriores para practicar con este comando.

Puedes omitir uno o ambos de dichos parámetros, y el valor prescrito (el menor o el mayor del programa) es el que será adoptado por el ordenador. Por ejemplo, LIST TO 50 considera la banda desde la primera línea del programa a la línea 50 inclusive, y

```
LIST 50 TO
```

se refiere a la banda desde la línea 50 al final del programa.

También puedes mezclar estas dos clases de parámetros en un mismo comando LIST. Por ejemplo:

LIST 10,30,60 TO 100

Con ese comando se listarán las líneas 10 y 30 y la banda de líneas 60 a 100, ambas inclusive. Ensáyalo por ti mismo y familiarízate con el el comando LIST -lo usarás frecuentemente.

Esa no es todavía la forma más general del comando LIST. La versión completa la explicaremos en un capítulo posterior, cuando veamos cómo hacer que el ordenador **liste** por impresora o cualquier otro periférico.

Además de hacer que liste una banda de líneas para examinarlas, puedes hacer que **tache** o suprima líneas del programa usando el comando DLINE (y la D es por "delete"=borrar, eliminar), especificando las líneas de forma similar a lo hecho en el comando LIST. Por ejemplo:

DLINE 30,50, 1000 TO 200

quitará de la memoria las líneas 30 y 50 y la banda de líneas 100 a 200. Eso te ahorra tener que ir tecleando cada número de línea y pulsando ENTER, que es otra forma ya conocida de eliminar líneas de programa.

Haciendo que trabaje

Otra posibilidad útil concierne a la sentencia RUN. Puede usarse sobre parte de un programa y no tienes porqué **ejecutar** todo el programa desde el principio. Si tecleas un número después de la clave RUN, harás que sólo se ejecute el programa a partir de ese número de línea. Por ejemplo, teclea NEW seguido de ENTER, con lo que borrarás el programa que estabas usando. Ahora teclea:

```
10 PRINT 10
20 PRINT 20
30 PRINT 30
```

Cuando mandas que se ejecute ese programa, verás que se exponen en líneas separadas los números 10, 20 y 30. Ahora, si das por ejemplo el comando

RUN 20

harás que el ordenador lo ejecute a partir de la línea 20 en adelante, saltándose por tanto la línea 10. Sólo se expondrán los últimos dos números. Ahora teclea:

RUN 30

y sólo aparecerá el 30.

RUN puede tener como parámetro una expresión numérica, así que por ejemplo:

```
RUN 5*6
```

hará que comience a ejecutar el programa a partir de la línea 30.

MEMOrandum

El Capítulo 2 presentó un método de elaborar y documentar un programa usando un diagrama de flujo. Eso permite al programador o a cualquier otro comprender la lógica de un programa dado, en una fecha posterior. Como mencionamos allí, otro método de explicar el funcionamiento de un programa es incluyendo comentarios dentro del propio programa. Si cada rutina principal del programa tiene un **título** para decirle al lector de qué se trata, eso ayudará tremendamente a la comprensión. Sin embargo, si tecleas simplemente algún texto en una línea de programa, el ordenador intentará interpretarlo como una instrucción del programa, y mostrará un mensaje de error.

Ahora bien, si una instrucción del programa comienza con la palabra clave REM, que es abreviatura de "REMARK"=comentario, nota, el ordenador no interpreta nada de lo que va detrás de esa línea, pero sí la conserva como **memorandum** y la mostrará en todos los listados. Eso te permite hacer observaciones en el programa en la forma que desees. Se te recomienda que lo hagas con libertad, ya que te encontrarás que al poco tiempo de hacerlo te olvidas de cómo funciona un programa. Los comentarios bien situados te ayudarán a poder leer el programa posteriormente. Los "memos" se usarán en los programas de este libro para ayudarte a comprender los ejemplos que te damos. El último programa podría haber tenido un comentario de la forma siguiente:

```
5 REM Este programa puede usarse para practicar con RUN
```

Debes teclearlo como una línea continua de texto. El ordenador no será capaz de reflejarla en una sola de sus líneas tampoco, sino que la partirá "fisicalmente" por algún sitio. También puedes añadir otra línea después de ésta de memorandum para separar un poco párrafos del listado. Por ejemplo:

```
7 REM
```

puede contribuir a una mayor legibilidad en un programa largo.

Bucles

El último capítulo presentó a los bucles en una forma gráfica, y esta sección presenta la sentencia del BASIC más simple que existe para bucles. Te ayudará a repetir acciones sin tener que volver a teclear RUN una y otra vez. La palabra clave es GOTO, que hace que el ordenador **vaya** a un determinado número de línea.

Al obedecer esta sentencia, el ordenador produce lo que llamamos una **bifurcación** en el curso del programa, ya que se salta la siguiente instrucción que teóricamente le tocaba, y "brinca" hasta el número de línea indicado en la instrucción. Para usar el ejemplo dado a continuación, tendrás que producir una ruptura o **corte** en la ejecución del programa, ya que si no continuará eternamente (o hasta que apagues). Esta operación de ruptura -BREAK- es accesible manteniendo pulsadas simultáneamente CTRL y la barra espaciadora.

Usa NEW para dejar la memoria como "nueva". Luego teclea el programa y ejecútalo. Puedes usar el comando AUTO para evitarte teclear cada número de línea, y provocar un corte cuando hayas acabado de escribirlo.

```
100 REM *** PREPARA CONTADOR BUCLE
110 N = 0
120 REM
130 REM *** BUCLE SE INICIA AQUI
140 REM *** EXPONE OTRO VALOR
150 REM *** DE LA TABLA DEL 12
160 REM
170 PRINT 12*N
180 PRINT
190 REM
200 REM *** SE INCREMENTA CONTADOR
210 REM
220 N = N + 1
230 REM
240 REM *** ESTE ES EL CONFIN DEL BUCLE
250 REM
260 GOTO 130
```

Observa que GOTO es escrito por el ordenador como GO TO, con un espacio. Tú puedes teclear GOTO sin el espacio intermedio para ahorrar tiempo, pero recuerda que al igual que con las demás claves del BASIC debe ir seguida de un espacio en blanco para que sea reconocida como tal.

Este programa saca la tabla de multiplicar por 12, pero para cualquier número hasta que sea demasiado grande como para ser manejado por el ordenador. Sin embargo, el proceso es comparativamente lento y tardará muchísimo tiempo en producir un **rebase** de la capacidad numérica, ya que el máximo posible es muy alto. No hay ningún método natural para "salirse del bucle" por lo que descansa en ti para que hagas que **pare** a la fuerza. La instrucción que le manda exponer nada, en la línea 180, simplemente provoca que saque una línea **vacía** para separar los números expuestos un poco más que si usáramos sólo una línea por número.

Como puedes ver, el programa tiene excesivas instrucciones de **memo**randum para dar al lector alguna idea de lo que sucede. No se te sugiere que uses tantas instrucciones REM vacías, pero ayudan a destacar las instrucciones, al igual que hacen los asteriscos que hemos incluido con ese propósito.

La instrucción GOTO puede ir seguida de una expresión numérica, y lo que es más conveniente de **una variable.**

Por ejemplo, GOTO 2*65 produciría la misma acción que la línea 260 del ejemplo. Igualmente, si la variable numérica RONDA se hubiera estipulado con el valor 130, la instrucción GOTO RONDA también funcionaría en esta ocasión.

GOTO también puede usarse como un **comando**, de ejecución inmediata. Por ejemplo, si has **cortado** la ejecución, y deseas que continúe, una manera de hacerlo **sin restituir** los valores de las variables a cero, (RUN hace CLEAR automáticamente) es hacer que se ejecute el programa usando el comando:

```
GOTO 130
```

El programa seguirá su ejecución con los valores que las variables tuvieran en el momento de la ruptura. De hecho, también puedes usar la palabra clave CONTINUE, para conseguir el mismo efecto de que **siga** la ejecución, y no necesitas saber exactamente en qué línea se detuvo. Además, en ciertas circunstancias al provocar un corte en la ejecución, puede que no sea posible usar GOTO para continuar exactamente donde se interrumpió. El comando CONTINUE sí que hace siempre que **siga** a partir del punto exacto, y en las circunstancias actuales en el momento que se te ocurrió la ruptura.

"Dar rondas y rondas" en un bucle, es una función muy importante en la programación, y veremos en el siguiente capítulo cómo en el SuperBASIC hay diferentes métodos para mandar que efectúe un bucle.

PAUSA

Un método de bajar la velocidad de ejecución de un programa, para ir viendo más claramente cómo es el curso del mismo, es decirle que **pause**. Esta instrucción acepta un parámetro numérico después de la clave, para indicarle la duración de la pausa. Dicho número mide el tiempo en unidades de 50-avos de segundo (20 milisegundos). Por lo tanto, especificar una pausa de 50 es otra forma de decir 1 segundo, una de 300 es de 6 segundos, y así sucesivamente. Ensaya añadiendo en el programa anterior la siguiente instrucción:

```
255 PAUSE 200
```

Verás cómo se retarda la exposición de los productos.

Pulsando cualquier tecla logras que dé la pausa por terminada y continúe con la ejecución. Ensáyalo durante una de las pausas de 4 segundos que has incluido.

Una instrucción de PAUSE sin un número hará que se detenga la ejecución permanentemente hasta que pulses cualquier tecla. Ensáyalo quitando el parámetro 200 de la instrucción de pausa, y observa cómo puedes usar el teclado para decidir cuándo el ordenador pase a dar la siguiente ronda del bucle.

Líneas multi-sentencia

Hasta ahora, cada línea de programa contenía solamente una sentencia. En ninguna parte hemos colocado dos sentencias dentro de la misma línea de programa. Hay ventajas en que podamos incluir más de una sentencia por línea, ya que así conseguimos un programa más compacto y breve en el listado. Para **separar** las instrucciones dadas en la misma línea, debe usarse un **dos-puntos (:)**, entre cada una de ellas, de manera que sepa cuándo termina una y cuándo comienza la otra. Por ejemplo, lo siguiente es una sentencia válida en BASIC, y puedes darla como **comando** de ejecución inmediata, o como **instrucción** a incluir en un programa real. Observa que el dos-puntos no es una palabra clave del BASIC, es un signo del lenguaje -pero no requiere espacio en blanco, ya que él mismo sirve de **separador.**

```
A=10:B=3:C=4:PRINT A*B*C
```

Aquí hay cuatro sentencias: tres de asignación y una de exposición de datos, dadas en una sola línea. Dicha línea hará que se asignen a A, B y C los valores dados y luego se exponga en pantalla su producto.

Como ejemplo de condensar listados, el programa de 17 líneas de la última sección, podría escribirse quitando las REM como:

```
10 N=0
20 PRINT 12*N:PRINT:N=N+1:GOTO 20
```

Observa que no hay ningún blanco después de la segunda PRINT. El dos-puntos es un método perfectamente válido de **separar** la clave PRINT del siguiente carácter que en los demás casos podría confundirse como siendo parte de un identificador que incluyera las letras de PRINT. Dos-puntos, recordarás, no puede usarse dentro de un identificador.

Este programa efectúa la misma tarea que el anterior. No tienes incluso que usar NEW antes de teclearlo. El programa usa números de línea que están colocados antes de los del último programa, y cuando tecleas RUN el ordenador comenzará a ejecutar el programa a partir del número de línea más pequeño que haya en el programa existente en su memoria. Sucede que ahora es la línea 10 de este ejemplo. Por tanto, el ordenador "entra en el lazo" especificado en la línea 20, y nunca se las arregla para llegar a la línea 100 que es donde comenzaba el otro programa. Para ejecutar este otro programa, puedes teclear RUN 100 o GOTO 100. Si tecleas GOTO 170, o RUN 170 entonces la variable N no será puesta a cero en la línea 110, y comenzará con el valor que ya tuviera como resultado de acciones anteriores. Ensaya todas estas combinaciones para ver lo que sucede.

Usando **dos-puntos** de esta manera, puede producir algunas líneas muy complicadas de leer, y es importante ser capaz de poderlas editar -revisar, corregir- sin tener que volver a teclear toda la línea. Eso se trata en la siguiente sección.

Edición: revisión, corrección

El QL está dotado de un método simple y potente de alterar líneas de programa, y evitarte la tediosa faena de tener que volver a teclearlas. Para entrar en el **modo** de **edición,** teclea la clave EDIT seguida del número de línea que deseas **revisar.** Eso hace que la línea solicitada sea traída desde la memoria hasta la ventana de comando en la parte inferior de la pantalla. Luego puedes alterar lo que desees usando las **teclas de flecha** izquierda y derecha en la manera habitual. Cuando has terminado, y sin importar dónde esté el cursor: pulsa simplemente ENTER, y el "editor" sustituirá la vieja versión de la línea que tiene en la memoria con la nueva versión que tú has alterado. Automáticamente se "sale" del modo de edición.

Ensaya a usar EDIT en la línea 20 de la última sección. Puedes cambiarla para que corresponda a la tabla del 13, por ejemplo. Para hacer eso, tecleas:

```
EDIT 20
```

La línea de comando mostrará la línea de programa solicitada -que es la llamada **vigente**- y ahora podrás mover el cursor hasta situarlo sobre el 2 del 12 en la primera instrucción PRINT, y cambiarlo para que sea 3. Pulsa ENTER y la línea mostrada en la pantalla cambiará de acuerdo con la revisión hecha.

Para editar una determinada **banda** de líneas, puedes especificar un intervalo después del primer parámetro numérico del comando EDIT. Si todavía tienes en la memoria el último programa largo del ejemplo, ensaya tecleando:

```
EDIT 100,10
```

Eso traerá la línea 100 para que puedas revisarla. Si no quieres cambios para ella, simplemente pulsa ENTER. Ahora no saldrás del modo edición como antes, sino que el "editor" añadirá 10 al número de línea vigente, y traerá a la línea de comando la que resulte. Cada vez que pulses ENTER, traerá la siguiente línea que corresponda. Si la siguiente línea no existe, será creada automáticamente, y puedes teclear en ella lo que desees. Para dejar de tratar con el "editor", provoca una **ruptura** -pulsando CTRL y la barra espaciadora- al igual que para AUTO.

Con esto se completa el primer conjunto de palabras clave del SuperBASIC, y de las **acciones** que con ellas consigues, usadas en el resto del libro. El siguiente capítulo explica cómo **ingresar** datos en un programa ("input") y como **exponer** datos obtenidos por el programa ("output"), y el material presentado en este capítulo será empleado a lo largo del siguiente para ayudarte a comprenderlas.

Resumen

- Las sentencias son conservadas por la máquina si van precedidas de un número de línea, pasando a ser **instrucciones** que forman las líneas de programa.

- Puede teclearse una nueva línea para sustituir una vieja simplemente dándole el mismo número de línea. El ordenador sustituye la línea vieja con la nueva. Lo puedes usar para corregir equivocaciones o para suprimir una línea concreta.

- Los programas pueden ser **ejecutados** usando el comando RUN seguido opcionalmente de un número de línea. LIST hará que muestre todas o parte de las líneas del programa en memoria.

- Se puede provocar una **ruptura** -BREAK- en la ejecución de un programa o de un comando si se pulsa simultáneamente la tecla de **control** (CTRL) y la barra **espaciadora.**

- DLINE se usa para **tachar** o eliminar ciertas líneas o bandas de líneas de un programa.

- NEW puede usarse para **vaciar** la memoria del programa existente; y CLEAR para **anular** ("limpiar") las asignaciones de valores hechas a todas las variables.

- Los operadores aritméticos * (multiplicación) y / (división) se evalúan antes de + (adición) y - (sustracción). Lo incluido entre paréntesis se evalúa siempre en primer lugar. En los demás casos, la evaluación procede de izquierda a derecha. El operador & se usa para empalmar (concatenar) datos literales.

- Las variables pueden contener literales ("series de caracteres"), números en coma flotante o enteros. Los identificadores de literales terminan en $, los de enteros en %. Todos los otros se supone que son variables numéricas en coma flotante.

- AUTO puede usarse para que el ordenador **numere** automáticamente durante el desarrollo del programa. RENUM se emplea para hacer que **renumere** líneas o bandas de líneas de un programa existente.

- REM se usa para colocar **memo**randums en un programa, y aparecen en los listados, y no tienen ningún otro efecto (más que ocupar memoria y aumentar la legibilidad).

- GOTO hace que el ordenador **vaya** a proseguir la ejecución de un programa a partir del número de línea mencionado en la sentencia.

- Se pueden incluir en una misma línea varias sentencias separadas por **dos-puntos.**

- EDIT seguido de un número de línea te pone en contacto con el "editor" que trabaja sobre la línea mencionada en el comando, o sobre la banda de líneas mencionada, y te permite **revisarlas** y alterarlas.

Capítulo Cuatro

Entrada/Salida

Una de las funciones más importantes de un ordenador es comunicar con los humanos. El hardware y el software requerido para eso es lo que se denomina el **nexo** o "interface hombre-máquina", abreviado en inglés I/O de Input/Output, que tienen significado claro de Poner dentro/Poner fuera. Los primitivos microordenadores tenían que usar conmutadores y pilotos luminosos para comunicarse con el usuario. No era una labor fácil escribir programas y obtener los resultados de esos programas con tales medios de **entrada/salida** tan rudimentarios. Hoy en día, el **teclado** y la **pantalla** son los medios habituales y standard.

Cualquier lenguaje de programación tiene que contener una serie bastante avanzada de instrucciones que le permita comunicar al usuario todas las ideas que pueden programarse en la máquina. Este capítulo está dedicado a describir las palabras y acciones del SuperBASIC que permiten utilizar la pantalla y el teclado de manera tan compleja como se desee. Un capítulo ulterior te mostrará cómo trazar figuras y dibujos de colores en la pantalla.

El QL también contiene un circuito electrónico que actúa como un **reloj** de cuarzo, y da años, meses, días, horas, minutos y segundos. Este capítulo describe cómo puede emplearse y tratarse mediante algunas sentencias BASIC para permitir la utilización del tiempo como dato de un programa.

Salida

El método de salida para todos los ejemplos de programas dados hasta ahora, ha sido mediante la sentencia de **exposición**, que tiene por clave PRINT. Nuestra siguiente labor es ampliar el uso de esta sentencia.

Cada vez que la hemos usado anteriormente, ha sacado una nueva línea de datos en pantalla. Es a menudo útil exponer los datos en la misma línea, quizás separados por sencillos espacios en blanco. También puede ser útil colocar los datos en columnas dentro de la pantalla o con elementos sucesivos contiguos uno a otro sin ningún espacio entre medias. Estos casos diferentes pueden producirse por algunos símbolos especiales que actúan como **separadores** de los datos que se van a exponer.

Para practicar con la sentencia de exposición PRINT, comienza tecleando NEW, seguido de ENTER, a no ser que acabes de ponerlo en marcha. Con eso vaciamos la memoria y dejamos el ordenador preparado para un nuevo programa.

Ahora teclea:

```
10 A = 123
20 B = 12.998
30 C = 5
40 D = -13.2
50 E$ = "La"
60 F$ = "Respuesta"
```

Teclea RUN y el programa será ejecutado. No se produce ninguna salida en pantalla, pero los valores quedarán asignados a las variables dadas. Podemos ahora experimentar con la sentencia PRINT en el modo inmediato -como **comando-** sin interferir con los valores anteriores.

Separadores

Está perfectamente admitido tener más de un dato después de la palabra PRINT, formando lo que se denomina una lista de datos a exponer. Es una regla que cada dato de la lista tiene que ir separado del siguiente por uno de cuatro símbolos separadores. Estos separadores son los siguientes:

, ; ! \

Si los datos están separados por una coma, se expondrán en zonas de 8 posiciones cada una. Ensaya tecleando lo siguiente:

PRINT A,B,C,D

El resultado será que los valores de esas variables se exponen en cuatro zonas de 8 posiciones cada una. Ensaya tecleando esa línea unas cuantas veces más, y la salida formará una tabla de cuatro columnas de números. Es muy útil para programas de tipo financiero, o presentación de cualquier otra clase de información. Para aumentar la separación entre los datos expuestos, se pueden usar dos comas para separarlos, y cada una de ellas hará que el siguiente aparezca 8 posiciones después del principio del anterior. Ensaya tecleando:

PRINT A,,B,C

unas cuantas veces para ver el efecto. Si añades otra coma entre B y C rebasas la capacidad de una línea si estás usando un receptor de TV.

Otro atributo útil de los separadores es que si tú terminas la lista a exponer con uno de ellos, su efecto se transfiere a la siguiente sentencia PRINT que se encuentre en el programa. Por ejemplo, si tecleas:

PRINT A,B,

y luego más adelante se encuentra con:

PRINT C,D

entonces incluso aunque los datos son expuestos por instrucciones separadas PRINT, la imagen obtenida será de los cuatro sobre una línea, separados por zonas. La coma al final de la primera sentencia dice al ordenador que continúe la exposición de datos como indica dicho separador. Ensaya tecleando eso, y luego ensaya lo mismo pero sin la coma al final de la primera sentencia PRINT.

Los datos separados por punto-y-coma se exponen uno inmediatamente detrás del otro sin ningún espacio en blanco entre medias. Así, tecleando lo siguiente:

PRINT A;B;C

dará 12312.9985, y tecleando:

PRINT E$;F$

dará:

LaRespuesta

con ningún blanco entre los caracteres. Esto es muy útil para exponer series compuestas de caracteres formadas a partir de otras, y te da control completo sobre la separación entre ellas. Si pones más de un punto-y-coma entre dos datos, actúa lo mismo que si usaras sólo uno. También aquí, si terminas la sentencia PRINT con un punto-y-coma, su efecto se transfiere a la siguiente PRINT, y el siguiente dato expuesto lo hará de forma contigua al más recientemente sacado. Ensaya tecleando:

PRINT E$;
PRINT F$

Eso da la misma imagen que si E$ y F$ se usaran en la misma sentencia PRINT separados por un punto-y-coma.

Cuando usas PRINT, ya habrás visto que pueden incluirse como dato a exponer números datos como **constantes** ordinarias. También puedes incluir constantes literales, entrecomilladas. Por ejemplo, ensaya:

PRINT E$;"Correcta";F$

Eso dará:

LaCorrectaRespuesta

Si quieres incluir espacios en blanco puedes conseguirlo haciendo que sean parte intrínseca de la constante literal, tal y como:

PRINT E$;" Correcta ";F$

Con lo que obtendrás un mensaje más fácilmente legible.

Ahora ensaya tecleando:

PRINT E$;" ";F$;" es ";A+B

Esta sentencia PRINT tiene cinco datos a exponer; forman una lista. El primero es E$, variable cuyo valor es "La"; la segunda es una constante literal formada por un solo carácter: el blanco o espacio; la tercera es F$ con "Respuesta"; y la cuarta es la constante literal " es ". El quinto dato es el resultado de un cálculo. El ordenador expone los datos reflejados en la lista, de izquierda a derecha, y -como ya has visto- evaluará cualquier expresión que aparezca de acuerdo con las reglas normales y exponiendo su resultado. La respuesta a este cálculo del ejemplo es 135.998, y ese es el dato que se expone a continuación del mensaje formado por los literales anteriores de la lista de datos a exponer. Eso te muestra lo aprovechable que estos aspectos más avanzados de la sentencia de exposición PRINT pueden ser.

El **signo de exclamación** es similar al punto-y-coma, excepto que en lugar de adosar los datos, inserta un único espacio en blanco para cada ! que se usa. Además, y es más importante, impide que un dato sea cortado por el medio cuando tiene que exponerse en posiciones cercanas al final de la línea, en las que no encaja. El signo ! hace que el dato se exponga en la siguiente línea, sin un espacio en blanco al principio. El signo ! actúa sobre el siguiente dato de una lista a exponer, y por tanto se coloca delante del dato sobre el que se quiere que actúe. Ensaya tecleando:

PRINT E$!F$

Como puedes ver, es un método conveniente de ahorrarte la necesidad de añadir un espacio entre E$ y F$, como teníamos que hacer anteriormente. Además, puede escribirse al final de una lista de datos a exponer, al igual que el punto-y-coma, y actuará sobre el primer dato que aparezca en la siguiente sentencia PRINT. Sin embargo, se requieren dos signos de admiración para producir un espacio en blanco. Ensaya con lo siguiente:

PRINT E$!!
PRINT F$

Ahora deja fuera uno de los signos de admiración y observa el efecto.

Para ver la otra función de este separador !, ensaya lo siguiente:

PRINT F$,F$,F$,F$!"Con esto se rebasará"

Los 4 F$ en la lista de datos llevan la imagen expuesta cerca del extremo de la línea. Si la constante literal que se manda exponer ahora estuviera separada por un punto-y-coma, quedaría dividida entre esta línea y la siguiente. La ! impide eso y hace que todo el literal pase a la siguiente línea. Ensaya sustituyendo el separador ! con el separador ; para ver la diferencia.

El símbolo **barra invertida** (\) se usa para forzar que los datos se expongan al principio de una nueva línea. Ensaya teclando:

PRINT E$\\F$

Cada carácter hace que se envíe a la pantalla un **avance de línea**, que es lo mismo que decir que se produce el paso a la línea siguiente por cada barra invertida que aparezca. También puedes usar una sentencia PRINT con la lista de datos a exponer "vacía", para conseguir líneas en blanco adicionales, como veremos en breve.

Puedes ahora añadir algunas líneas extra al programa almacenado en la memoria. Por ejemplo, se pueden sumar A y B para obtener una respuesta, y C y D para obtener otra. Estos resultados pueden exponerse en una forma agradable y pulcra usando la sentencia PRINT. Ensaya añadiendo las instrucciones:

```
70 PRINT "La suma de A y B da"
80 PRINT E$!F$!"como:"!A+B
90 PRINT
100 PRINT "La segunda suma da"
110 PRINT E$!F$!"como:"!C+D
```

Recuerda que cada signo separador en una sentencia PRINT tiene un significado preciso y predefinido. Observa en la línea 80, e imagínate el cabezal impresor de una máquina de escribir. La primera cosa que teclea es el contenido de E$. Luego, el separador ! le dice al cabezal que avance una posición, obteniéndose por tanto un espacio en blanco. Lo siguiente le dice que debe exponer el contenido de F$ seguido también de un espacio en blanco. Luego vienen los cinco caracteres que forman el literal entrecomillado, y luego otro espacio. En ese momento, observa que la lista de datos a exponer está terminada, y que no hay ningún otro signo separador a considerar, así que provoca un avance de línea, con lo que hace que pase al comienzo de la siguiente línea.

Intenta adivinar exactamente cuál será la imagen expuesta por el programa cuando se ejecute, y luego comprueba si has acertado. Ensaya añadiendo unas cuantas líneas en blanco más para "espaciar" los datos un poco más, y hacerlo más legible. Usa uno o más separadores \ en las líneas 70, 90 y 100, por ejemplo.

EN columna, fila determinadas

Las formas anteriores de PRINT sólo exponían datos de una manera consecutiva, y a partir de la esquina superior izquierda hacia la esquina inferior derecha, tal y como escribimos texto. A menudo es muy útil poder exponer un dato en una **posición** concreta de la pantalla que elijamos. Por ejemplo, cuando aprendamos sobre las funciones del reloj, veremos un programa que muestra la hora en la esquina superior izquierda de la pantalla, con los segundos marcándose. Eso requiere una sentencia PRINT que se ejecute continuamente, y que proyecte constantemente en la misma área de la pantalla la información. Todas las versiones vistas hasta ahora podían comenzar en el sitio correcto, pero harían que los datos aparecieran sucesivamente más abajo en la pantalla, y que no permanecieran en un mismo sitio.

La sentencia AT puede usarse para situar en una posición concreta de la pantalla el dato a exponer mediante la sentencia de exposición PRINT. Con eso se considera la pantalla como una retícula o **gradilla**, formada por **filas y columnas** para determinar la posición en que queremos sacar los caracteres.

Realmente, dicha gradilla difiere entre una TV y un monitor de video. La sentencia AT acepta dos **parámetros** numéricos, y puede incluirse en cualquier parte del programa antes de la instrucción PRINT que queremos controlar. Sin embargo, para mayor legibilidad del programa se sitúa normalmente en la misma línea, separada por un dos-puntos de la sentencia PRINT. Los parámetros numéricos en la instrucción AT reflejan respectivamente la **columna** y la **fila** a partir de la cual queremos exponer el dato. Así, por ejemplo,

```
AT 20,10:PRINT E$
```

expondría "La" a partir de la columna 20 de la línea número 10, contadas desde la esquina superior izquierda.

Ensaya con esta sentencia para percibir la extensión de la pantalla, e intenta exponer F$ de manera que termine justamente en la esquina superior derecha de la pantalla.

LIMPIE la pantalla

Otra sentencia muy útil para manejar la información que se expone en pantalla es la de palabra clave CLS, que es abreviatura de "clear screen", y cuando se ejecuta hace que se limpie la pantalla (se "aclare"). Eso es útil cuando no quieres que la imagen se deslice continuamente al ejecutar un bucle. Por ejemplo, teclea:

```
10 PRINT "Hola"
20 GOTO 10
```

Con eso haces que la imagen se **corra** continuamente hacia arriba. Ensaya añadiendo las siguientes líneas:

```
15 PAUSE 100
17 CLS
```

Ahora no se llenará la pantalla, ya que hacemos que la **limpie** antes de exponer "Hola". Sin embargo, el efecto no es muy bueno, y hay un método mejor de impedir que se deslice la imagen usando la instrucción AT. En un capítulo posterior veremos cómo limpiar diferentes partes de la pantalla con la sentencia CLS. Por el momento, simplemente la usaremos para comenzar con una pantalla en limpio. Teclea CLS para hacer que se limpie la pantalla cuando está llena, y lo necesita. Es útil particularmente antes de hacer que **liste** el programa.

Imposición de datos por teclado

Hasta ahora, todos los datos sobre los que opera el programa han sido aportados al mismo en forma de instrucciones de asignación. Veremos ahora un método de permitir al programa que **ingrese** los datos que tecleemos.

Esto permite al programa mantener una relación con el operador mientras se está ejecutando, y le permite asimismo controlar los valores de las variables usados por el programa para sus operaciones. La sentencia BASIC usada es INPUT, significando **poner dentro**, imponer. Esta sentencia INPUT va seguida de una lista de variables, de manera similar a la sentencia PRINT. Sin embargo, los valores de dichas variables no son sacados a la pantalla, sino que precisamente son **impuestos** de acuerdo con los datos tecleados. Como un simple ejemplo, teclea NEW para vaciar la memoria si hubiera algún programa almacenado en ella, y teclea las siguientes líneas de programa:

```
10 A=0.0001
20 INPUT B
30 PRINT A*B
```

La labor del programa es exponer el valor que resulta al multiplicar A por B. A tiene su valor asignado mediante la instrucción de la línea 10. Pero a B se le **impondrá** el valor que el operador teclee cuando se ejecute la línea 20. Ensaya con el programa y observa lo que pasa. Verás que la máquina se detiene como si se quedara pasmada, pero con el cursor parpadeando en la ventana de salida. Está esperando que introduzcas el valor que va a ser impuesto a la variable B. Puedes teclear en este momento el número que desees, y realmente nada se impondrá sobre la variable B hasta que no concluyas el dato introducido pulsando ENTER. Es el funcionamiento normal de la máquina y ya no debe ser una sorpresa para ti. Si cometes un error al teclear, o decides que quieres cambiar lo tecleado, puedes usar los métodos normales para corregir el dato introducido, pero antes de pulsar ENTER.

Cuando has tecleado el valor deseado para la variable B, y pulsado ENTER, la línea 30 pasa a ser ejecutada, y el valor de A*B expuesto en pantalla.

Debieras ejecutar este programa unas cuantas veces, introduciendo diferentes datos cada vez, para ver el efecto de usar números pequeños y grandes. Si tecleas un número muy pequeño, tal como 0.0000023, verás cómo el QL automáticamente pasa a exponer la respuesta en notación exponencial. También puedes ingresar números por teclado tales como 4.5E10, y observar el resultado. La única regla a recordar es que el tipo de dato que corresponde a B es un numeral de coma flotante, y para asegurar que el programa se ejecuta correctamente, debieras sólo teclear ese tipo de dato. Ensaya tecleando un **literal**, y observa lo que sucede; y prueba con un ¡literal formado por cifras pero entrecomillado para que lo considere como tal!

Si deseas imponer durante la ejecución un dato literal como valor de una variable, tendrás que usar una variable literal en la sentencia INPUT. Por ejemplo, altera el programa insertando las líneas:

```
25 INPUT A$
26 PRINT:PRINT
27 PRINT A$
```

La línea número 25 añade una sentencia INPUT para un dato literal. La línea 27 añade otra sentencia PRINT que expondrá el literal tecleado. Ensaya con el programa. El primer dato de entrada requerido es el mismo de antes, pero el segundo que se te pide es un literal (que será expuesto literalmente tal como lo teclees). Cuando se te pida el literal, teclea las siguientes palabras:

La respuesta a A*B es igual a:

Con eso añades un mensaje a la salida. Sin embargo, tendrás que teclear esas palabras cada vez que el programa se ejecute. Sería mejor incluir otra instrucción PRINT en el programa para que expusiera ese mensaje sin la necesidad de tener que ingresarlo por teclado cada vez.

Otro añadido útil para la pulcritud general de la información sacada podría provenir de emplear algunas instrucciones para el usuario en cuanto a lo que debe ingresar por teclado y cuándo. Eso puede hacerse mediante sentencias PRINT antes de cada sentencia INPUT explicando si lo que se le pide ahora introducir por teclado es un valor numeral o literal. Sin embargo, hay una manera más fácil de hacerlo, usando la propia instrucción INPUT. Realmente puedes incluir un **mensaje** dentro de la propia INPUT si lo deseas, y cada vez que se ejecute esa instrucción el mensaje será enviado a la pantalla y aparecerá justamente antes de pedir que teclees el dato. El mensaje -que es también llamado **misiva**- ha de ser un literal escrito inmediatamente detrás de la palabra clave INPUT. Tiene que terminarse con un signo separador tal como la coma o la admiración. Esnaya tecleando la siguiente línea:

```
20 INPUT "Teclea por favor el valor de B",B
```

Cuando ejecutes el programa, verás el efecto y lo diferente que es del uso general de la instrucción INPUT. No tienes que usar mensajes tan largos. Por ejemplo, ensaya con:

```
20 INPUT "B=",B
25 INPUT "A$=",A$
```

y verás lo fácil que es introducir datos de esta manera. Ensaya sustituyendo las comas anteriores con separadores como ! y ; para ver la diferencia.

Como un ejemplo de las ideas presentadas hasta ahora, veamos cómo usaríamos un programa para convertir moneda extranjera a libras esterlinas. El programa supondrá que conoces el "cambio" y que quieres convertir una cantidad dada de unidades monetarias en libras.

Para resolver este problema, tenemos que tomar como dato de entrada el nombre de la moneda extranjera, el cambio aplicado, y la cantidad cuyo equivalente queremos saber. La conversión se efectúa dividiendo esa cantidad por el cambio aplicado. Finalmente, debemos exponer el resultado obtenido. Esta breve explicación podría haberse escrito como un diagrama de flujo, pero es demasiado simple para el esfuerzo. Antes de tec'ear el siguiente ejemplo, manda NEW y luego CLS para "despejar" la memoria y la pantalla. Un posible programa para la conversión sería:

```
10 CLS
20 INPUT "Nombre de moneda es?"!MNOME$
30 INPUT "Cambio por libra es?"!TIPO
40 INPUT "Cantidad a convertir?"!MONTO
50 PRINT\\\\
60 PRINT "La conversion de moneda da:"
70 PRINT
80 PRINT MONTO/TIPO!"Libras por"!MONTO!MNOME$
```

Antes de que el programa trabaje, emplea un poco de tiempo determinando exactamente cómo será la salida del mismo.

La mayoría de las instrucciones son directas, pero tendrás que analizar cada símbolo sencillo de la línea 80, porque será un buen ejercicio para practicar con las ideas presentadas hasta ahora para la instrucción PRINT. Luego al ejecutarlo, comprueba que adivinaste correctamente, o analiza cuidadosamente cualquier error de apreciación que hubieras hecho. Recuerda que tendrás que escribir instrucciones como esa por ti mismo, si quieres programar.

La CLS en la primera línea de programa es una manera útil de mandar que **limpie** la pantalla. Las tres instrucciones de **imposición** de datos en el programa son las que recogen el nombre de la moneda, el cambio y la cantidad a convertir. El nombre de la moneda es un literal almacenado en MNOME$, y el valor impuesto es la serie de caracteres que teclees. Puedes elegir pesetas o francos, pero puedes igualmente teclear gatos y perros. El programa no aprecia ninguna diferencia, y seguirá fielmente tus instrucciones exponiendo en el resultado lo que aquí hayas escrito, al pie de la letra. Eso destaca aún más la ley fundamental de la computación: ningún programa de ordenador contiene más inteligencia de la que tú mismo le hayas imbuido. Eso fue declarado al comienzo del Capítulo 2 en una simple oración: **los ordenadores son estúpidos.** Sé cuidadoso en no suponer ninguna inteligencia en un programa distinta a la que el programador haya querido incluir en él.

Para proseguir con este ejemplo, podrías hacerlo más inteligente en el aspecto de reconocer un nombre actual para la moneda, simplemente conservando en una tabla los nombres de las monedas que aceptas. Luego cuando se impone el valor a MNOME$, el programa podría comparar lo tecleado con cada nombre que haya en la tabla y comprobar si por lo menos hay uno que concuerda. Si así no fuera, el ordenador te diría que has hecho un error, y te pediría que lo intentaras de nuevo. Esta clase de inteligencia subraya otro hecho bastante importante sobre los ordenadores: para hacer que sean más inteligentes, necesitas a menudo un montón de memoria donde imbuirle lo que quieres que aprenda.

La instrucción INPUT también puede usarse para **imponer** datos a diversas variables consecutivamente. Los nombres de las variables han de darse en la sentencia INPUT, y han de estar separados por signos separadores admitidos. Por ejemplo, el programa anterior podía haberse escrito de forma más compacta en la manera siguiente:

```
20 PRINT "Teclea nombre moneda,"
30 PRINT "Cambio por libra, y cantidad."
40 INPUT MNOME$,TIPO,MONTO
50 PRINT \\"Sacarlas"!MONTO/TIPO"Libras por tus"!MONTO!MNOME$
```

Es importante que teclees un tipo correcto de variable para cada uno de los datos a imponer, y que pulses ENTER **después de** cada dato introducido. Lo que se requiere teclear ahora es un valor literal seguido de dos valores numerales. Cualquier otra cosa puede dejar pasmado al programa, y hacer que aparezca un mensaje de error. Ensaya con el programa ingresando diferentes datos para ver cómo funciona y cuándo algo va mal.

Otras maneras de aportar datos

Otra importante manera de incorporar datos a un programa se consigue escribiéndolos incorporados en instrucciones del propio programa. Ya hemos visto ejemplos de eso. Muchos de los programas que hemos visto hasta ahora simplemente asignan datos a variables cuando se requieren. Hay sin embargo, una manera de cargar una larga **lista de datos** en el programa, y forzar a que el ordenador vaya usándolos a medida que procede. Es útil por ejemplo, para reflejar una serie de números sobre los que debe operarse durante la ejecución del programa. Esta clase de incorporación usa las palabras clave READ, DATA y RESTORE, como describimos a continuación.

Hay otras clases de ingreso de datos que pueden obtenerse empleando el teclado, y se usan a menudo para programar el ordenador para jugar. Permiten que sea percibida por el ordenador la pulsación de una tecla instantáneamente. Usan las palabras clave: INKEY$ y KEYROW, que presentaremos en breve.

APUNTE, DATO y REPUNTE

Son operaciones en BASIC que permiten incluir una lista de datos en el programa, accesibles durante la ejecución del mismo. La lista de datos se escribe con comas como separadores, usando sentencias de clave DATA. Puede haber cualquier cantidad de ellas, y colocarse en cualquier parte del programa. Por ejemplo:

```
10 DATA 1,2,3,4,5,6,7,8,9,10
20 DATA 11,12,13,14,15,16,17
```

Para hacer que el ordenador **apunte** estos datos como valores de las variables deseadas, se usa la sentencia READ (que es "leer"). Esta sentencia tiene una estructura similar a la INPUT. Por ejemplo:

```
30 READ A,B,C
```

La sentencia READ va seguida de identificadores de variables, separados por comas. Cuando se ejecuta, el ordenador busca el valor que corresponde en las sentencias DATA y apunta dicho **dato** como valor de las variables dadas en la instrucción. Los datos se van **tomando** y apuntando en el orden en que aparecen. Por tanto, se apunta en A el valor 1, en B el valor 2, y en C el valor 3. Cada vez que se ejecuta una sentencia READ, el ordenador toma los siguientes valores disponibles en la lista, y continúa recorriendo toda la lista de datos formada por las sentencias DATA, hasta que no quede ninguno. Aquí por ejemplo, hay 17 números y 3 variables a las que apuntar dichos números como valores. Eso significa que si se incluye la sentencia READ en un bucle, ejecutado una y otra vez, dispondríamos de 5 rondas del bucle, y entonces se apuntarían en A y B los dos datos últimos, 16 y 17 respectivamente, pero no dispondríamos de ninguno para apuntarlo a la variable C; por lo que el programa se detendría diciendo que no puede efectuar esa operación porque se ha quedado "sin datos". Ensaya añadiendo las siguientes líneas, y ejecutando el programa resultante.

```
25 CLS
40 PRINT A,B,C
50 GOTO 30
```

A medida que va apuntando los datos como valores de las variables, el ordenador mantiene la cuenta de los que ya ha utilizado, y va señalando el que corresponde para la siguiente mediante lo que se denomina un **puntero de datos** interno. Tú no puedes saber ni cambiar el valor de dicho puntero por ti mismo; todo lo que puedes hacer es que **repunte** hacia una línea determinada, si usas la instrucción RESTORE que le obliga a señalar el primero de los datos de la instrucción DATA que desees. Si el programa no funciona correctamente, ensaya tecleando RESTORE. El puntero de datos puede que no esté situado al principio de la lista de datos, cuando lo ejecutes.

Si haces que se **vaya** a la 30 después de ejecutar el último programa, deberás recordar que el ordenador no restaura los estados iniciales de las variables, y eso incluye también el llamado puntero de datos interno. El resultado sería el mismo error que detiene el programa. De hecho, tampoco el comando RUN restaura el estado inicial del puntero de datos - ensáyalo. Sin embargo, si tecleas simplemente el comando:

```
RESTORE
```

harás que automáticamente el ordenador **repunte** su puntero interno de datos al comienzo de la primera instrucción DATA, y luego puedes hacer que funcione el programa usando RUN o GOTO 30. Siempre debes comenzar un programa que emplee instrucciones READ y DATA con una instrucción RESTORE. Ahora inclúyela como:

```
27 RESTORE
```

Eso permitirá que se ejecute el programa usando RUN, desde el principio, y más de una vez. No puedes colocar la instrucción RESTORE más atrás en el programa, o hará que el puntero de datos **repunte** al principio, cada vez que se ejecute el bucle del programa, y sólo serán apuntados en las variables los tres primeros datos de la lista. El bucle nunca terminará. Ensáyalo para ver el efecto.

Si quisieras comenzar con la segunda instrucción DATA, puedes teclear:

```
RESTORE 20
```

La instrucción RESTORE acepta por tanto como parámetro un número de línea, y el valor prescrito para omisiones del parámetro es el número de línea de la primera instrucción DATA del programa. No hay ningún límite formal a la cantidad de instrucciones DATA que puedes escribir en un programa, ni tampoco ningún límite formal a la cantidad de datos que puedes incluir en una instrucción DATA. Sin embargo, debes mantener las listas de datos en un tamaño manejable. Puedes desglosarla en tantas instrucciones DATA como desees, y es más fácil utilizar la parte de la lista que desees usando la instrucción RESTORE para que repunte hacia el número de línea pertinente.

También pueden reflejarse y utilizarse datos literales como elementos de una lista DATA, pero han de ser apuntados a variables del mismo tipo, mediante la instrucción READ correspondiente.

Por ejemplo:

```
10 CLS
20 RESTORE
30 REM *** e.g. de DATOS numerales y literales
40 REM *** Apunta un literal y un numero
50 READ NOME$,NURO
60 REM *** Ahora expone el mensaje
70 PRINT "Nombre del numero"!NURO!"es"!NOME$
80 REM *** Bucle HASTA que
90 REM *** se queda sin datos.
100 GOTO 50
110 DATA "Maria",1,"Juan",2,"Jorge",3,"Ana",4
120 DATA "Jose",5,"Jaime",6,"Lourdes",7
```

Este programa extrae los datos de la instrucción DATA en parejas, una y otra vez, pero cada pareja está formada por un literal y un numeral. Es importante que el dato concuerde con el tipo de identificador que se usa en la instrucción READ, y sobre el que se apunta el dato **homólogo.**

Ejecuta el programa otra vez, para ver cómo **el puntero de datos** interno, mediante la instrucción RESTORE hace que **repunte** al comienzo de la lista. Si suprimes la línea 20, te encontrarás con que no puedes hacer que funcione el programa más de una vez.

Ingreso instantáneo por teclado

La instrucción INPUT espera a que pulses la tecla ENTER antes de pasar a ejecutar la siguiente instrucción. En juegos y otros programas donde hay un montón de tratamiento procesándose en la "retaguardia", haríamos el juego menos dinámico si lo detuviéramos cada vez que se pide introducir un dato por el jugador. Para resolver este problema, el QL ofrece dos **funciones** que permiten constatar la tecla que ha sido pulsada, sin necesidad de detener la ejecución del programa.

La palabra clave del SuperBASIC para la primera **función** es INKEY$, que permite **in**quirir, **in**dagar si hay o no una **tecla** (key) pulsada en ese momento, o pulsada previamente pero **pendiente** de ser tratada por el ordenador. Veremos en un capítulo ulterior cómo también se puede aplicar esa función para otros periféricos distintos del teclado; pero por el momento nos restringiremos a su utilización para ingresar un solo carácter del teclado, sin necesidad de esperar a que concluya pulsando ENTER.

INKEY$ en realidad es el identificador de una variable literal especial. Su valor es igual al del **carácter** correspondiente a la tecla que se está pulsando, cuando se hace referencia a la función INKEY$. Para ver cómo funciona, teclea el siguiente programa:

```
10 PRINT "Tecla pulsada ="!!
20 PRINT INKEY$
30 GOTO 10
```

Es un bucle que continuamente expone un mensaje seguido del valor **entregado** por la función INKEY$. Cuando no se está pulsando ninguna tecla, el valor de INKEY$ es el **literal nulo;** en los demás casos se expondrá el literal de un carácter correspondiente a la tecla pulsada en ese momento. Ensaya pulsando cualquier carácter del teclado. A medida que el programa pasa por la línea 20, se evalúa INKEY$ y se expone el valor literal correspondiente.

Como puedes ver, el programa se ejecuta ininterrumpidamente, y no se queda esperando en la línea 10 a que se pulse una tecla. Tendrás que provocar la ruptura en la ejecución para que se detenga.

Si tú quieres que el programa espere con la función INKEY$, debes especificar un **parámetro** para indicar la duración de la pausa. Dicho parámetro ha de ir entre paréntesis, después de la clave INKEY$. Si durante este tiempo de pausa se pulsa una tecla, se agota inmediatamente la pausa y continúa la ejecución. El número entre paréntesis es la cantidad de lapsos de 20 milisegundos (un quinto de segundo) que el programa esperará en INKEY$ antes de continuar con la siguiente instrucción. Si dicho número es 0, la pausa es 0, y la ejecución continúa inmediatamente al igual que hace para el valor prescrito para omisiones (sin parámetro). Si dicho número es -1, esperará por siempre a que se teclee algo. Da la instrucción siguiente:

```
20 PRINT INKEY$(100)
```

Con lo que harás que el ordenador espere durante dos segundos antes de que se canse y continúe con la ejecución del programa. Ensaya pulsando algunas teclas y observando el efecto. Ahora cambia la instrucción para que sea:

```
20 PRINT INKEY$(-1)
```

y observa la diferencia.

Hemos dicho que INKEY$ es una clase especial de identificador. Es el nombre de una variable que tiene un valor claramente definido en una situación dada. Es diferente digamos de la variable A o de la NOME$ que puede tener cualquier valor, y a la que puede asignársele ese valor en cualquier momento. INKEY$ es un ejemplo de una **función** del sistema. El número que sigue entre paréntesis es lo que se denomina un **argumento,** (o en este caso, mejor un **parámetro**) y se usa para matizar la acción efectuada de alguna manera claramente definida. Aquí corresponde al número de intervalos de 20 milisegundos que debe esperar el ordenador hasta que se pulse una tecla, o para continuar con la ejecución.

La mayoría de las funciones en el sentido matemático, tienen realmente como mínimo un **argumento,** sobre el que se calcula para obtener el resultado. La mayoría de los argumentos pueden ser **numerales** y no tienen porqué ser meramente **números.** Cuando se evalúa la función, se obtiene un resultado y decimos que **entrega** ese valor. Así diremos que INKEY$ es una función con un parámetro que especifica un tiempo de pausa, y que **entrega** un valor literal que corresponde a la primera tecla pulsada durante ese período de pausa.

Otra **función** para constatar la pulsación de teclas es la de clave KEYROW, que corresponde a una **fila** de teclas. El teclado del QL está dispuesto según una estructura rectangular de teclas con 8 **filas** (numeradas 0 a 7) y 8 columnas (numeradas 1,,2,4,8,16,32,64,128).

La función KEYROW tiene como parámetro entre paréntesis el número identificativo de una fila, y **entrega** como resultado el número de la columna correspondiente a cualquier tecla de esa fila que esté pulsada. El valor entregado es **cero** si no se está pulsando ninguna tecla en esa fila cuando se cita la función. Las filas y columnas están reproducidas en el manual del QL, en la lista de claves, y lo siguiente es un ejemplo:

```
10 PRINT "Columna num.="!KEYROW(5)
20 GOTO 10
```

Con eso se expondrán ceros hasta que se pulse una tecla de la fila número 5. Ensaya pulsando R, luego Y, luego I, luego O. Verás que se exponen respectivamente los números 16, 64, 4 y 128. Son los números de columna asociados con esas teclas que están todas en la fila número 5. Ensaya pulsando varias teclas a la vez y verás que el número entregado por la función es la suma de los números de columna pertinentes. Si comprendes los números binarios, te darás cuenta que los números de columna son potencias de dos, y eso te permite decir cuáles son las teclas pulsadas cuando hay más de una al mismo tiempo.

Observa que las filas y columnas no están de acuerdo con las filas y columnas tal y como aparecen en el propio teclado.

El reloj interno

El QL tiene un reloj interno que actúa como un cronómetro digital. El reloj está disponible tanto para estipular como para examinar el tiempo, usando algunas sentencias y funciones del BASIC. Esas son SDATE para **poner** la hora; ADATE para **ajustar** el reloj según un número prefijado de segundos, y las funciones DAY$ y DATE$ para ver el **día** y la **fecha** correspondientes. Ensaya limpiando la pantalla y tecleando los comandos:

```
PRINT DAY$
PRINT DATE$
```

Eso te dirá el día y la fecha según la marcha presente del reloj, y te mostrará el **formato** de los valores entregados por las funciones DAY$ y DATE$. Al igual que en la última sección, son identificadores especiales con definiciones predeterminadas. El valor entregado es un dato literal, o serie de caracteres que contienen el día y la fecha como una palabra. Así, la imagen muestra el momento en que se evaluaron las funciones DAY$ y DATE$.

Más adelante en este libro, veremos cómo podemos **segregar** partes de un literal. Eso nos permitirá usar parte de estas funciones.

SDATE es una **sentencia,** que se da como un **comando,** lo que significa que manda al ordenador llevar a cabo una acción. Se usa para **poner** la fecha (set date) y tiene que ir seguida de seis numerales -normalmente números- separados por comas, y de acuerdo con la siguiente información:

SDATE Año,Mes,Día,Hora,Minuto,Segundo

Un ejemplo de SDATE sería:

```
SDATE 1985,8,17,23,1,0
```

Con lo que se pondría el reloj en hora precisamente a las 11,01 de la tarde, del 17 de Agosto de 1985, tan pronto como se ejecutara el comando. Usalo para fijar la hora actual, y luego expón la función DATE$ para revisarla. Para ver cómo transcurre el tiempo, escribe el programa:

```
10 PRINT DATE$
20 GOTO 10
```

Provoca una ruptura para detener el programa.

La sentencia ADATE se usa para **ajustar**, alterar la fecha en unos cuantos segundos. Va seguida de un numeral -normalmente un número- que le dice al reloj que avance o retroceda ese número de segundos. Si es positivo, el reloj avanza, si es negativo, retrocede. Permite efectuar el ajuste fino. Por ejemplo, si mandas

```
ADATE 35
```

el reloj avanzará inmediatamente 35 segundos, y continuará. Si mandas

```
ADATE -300
```

retrocederá al instante 5 minutos, y continuará con la nueva hora.

La función DATE es obviamente **numérica**, y permite que la **fecha** sea depositada y recuperada en forma numérica (que ocupa menos memoria que la forma **litérica** DATE$) y permite ahorrar memoria cuando por alguna razón estás registrando el tiempo montones de veces. Ensaya tecleando:

```
PRINT DATE
```

Como puedes ver, el valor de esta función sí es un verdadero número. Puede convertirse a un literal más legible usando las funciones DATE$ o DAY$.

Ambas de estas funciones pueden aceptar además un argumento numeral cualquiera, y lo que hacen es pasarlo de dato numérico a su equivalente literal con el formato adecuado. Por ejemplo, teclea el siguiente programa, usando NEW y CLS si es necesario.

```
10 HORA = DATE
20 PRINT "La fecha numerica es:"!HORA
30 REM Ahora convertida a literal
40 HORA$ = DATE$(HORA)
50 PRINT "La forma normal de fecha es:"!HORA$
```

La línea 40 acepta el argumento -la variable numeral HORA- para la función del sistema DATE$ que entrega como resultado el valor literal equivalente según el formato de DATE$. No guarda ninguna relación con el reloj interno.

Para verlo claramente, teclea:

GOTO 40

unas cuantas veces. Verás que la hora mostrada permanece constante. Si ahora expones el valor de DATE$ por sí mismo, verás la hora oficial del sistema.

La función DAY$ opera exactamente de la misma manera, pero entrega como resultado la información del día que corresponde a su argumento numeral. Ensaya:

PRINT DAY$(HORA)

Con eso sacarás el día del valor de la variable numeral HORA.

Estas funciones y sentencias se usarán otra vez en ejemplos, y las encontrarás muy útiles en programas que han de ser ejecutados de acuerdo con una determinada fecha o día. La hora es un dato de entrada muy importante a un programa.

Entrada/Salida en la memoria

Hay otra clase de operaciones que pueden considerarse de entrada/salida, aunque realmente son **depositar/recuperar** datos de la memoria interna del propio QL.

Como probablemente sepas, el QL tiene una gran cantidad de memoria interna, que se emplea para conservar los **datos y programas,** tales como los que hemos estado ensayando antes. La memoria está organizada de una manera **reglada** y lógica, que podemos considerar como en **celdillas** contiguas pero distintas. Cada celdilla puede contener un **símbolo** que puede ser una cifra, una letra, un signo o cualquier cosa con significado especial. El QL tiene más de 131.000 **celdillas,** y en ellas se puede **meter** el símbolo que deseemos, o **mirar** el que hay dentro. Una cierta cantidad de celdillas está reservada para los caracteres que son proyectados en la pantalla, y alguna otra cantidad se usa para ser tratada internamente por el QL durante sus operaciones normales.

Los detalles de la estructura **reglada** de la memoria no son complejos pero caen más allá del ámbito de este libro, y lo que sí interesa son dos claves del BASIC: la función PEEK para **mirar** el contenido de una celdilla dada, y la instrucción POKE para **meter** en una celdilla determinada el símbolo que deseemos. Son sentencias muy potentes y muy relacionadas con el funcionamiento real de la máquina. Usándolas no se produce ningún daño en la propia máquina, pero se puede perder información o alterar el programa de forma extraña, si no se tiene cuidado y se poseen conocimientos.

Cada **celdilla** de la memoria en la máquina tiene su propia identificación -su propia **dirección**- que a menudo se escribe en un sistema de numeración especial, denominado **hexadecimal.** También se representa gráficamente mediante un **mapa de memoria,** como puedes ver en el Manual del QL. Esto también nos lleva más allá del objetivo de este libro, y se te aconseja leer un libro que explica cómo funciona la memoria antes de usar estas posibilidades.

La sentencia POKE realmente "recluye" cualquier dato que desees en la celdilla de memoria cuya dirección menciones en el comando, usando el sistema de numeración normal (en base 10, o "denario"). Su formato general es pues

POKE dirección,dato

Eso hará que el ordenador **meta** el dato especificado -que corresponde a un solo símbolo- en la celdilla marcada con esa dirección. Hay también versiones de esta sentencia que permiten afectar hasta 2 ó 4 celdillas consecutivaś a partir de la dirección dada, con lo que puedes meter más de un símbolo con una sola sentencia POKE.

PEEK es una función que tiene una dirección como argumento. Su forma general es:

PEEK (dirección)

El valor que entrega esta función es el dato -un solo símbolo- que está recluido en esa dirección. La palabra PEEK corresponde a examinar, mirar; y también se pueden en algunas versiones hacer que el ordenador entregue el valor **mirado** en hasta 2 ó 4 celdillas contiguas.

Otra sentencia que actúa sobre la estructura fundamental del funcionamiento de la máquina es la sentencia CALL que se usa para hacer que **ceda** el control a un programa escrito en el propio lenguaje del microprocesador: en **código máquina.** También es una potentísima facilidad, pero tendrás que consultar un libro sobre código máquina para apreciar su utilidad.

Resumen

- Una sentencia PRINT puede contener una lista de datos a exponer en pantalla, delimitados por **separadores.** El **punto-y-coma** asegura que los miembros de dicha lista se exponen sin espacios en blanco entre ellos. La **coma** hace que pase a la siguiente zona de exposición en pantalla que ocupa 8 posiciones. La **barra invertida** provoca un avance de línea siempre que se usa, y la **exclamación** es un espacio inteligentemente administrado que se usa para separar los datos por un solo blanco por cada admiración empleada, y además impide que sea dividido un dato en dos por estar colocado al final de una línea de pantalla.

- Cada sentencia PRINT encontrada comenzará a exponer sus datos en una nueva línea de pantalla, a no ser que haya un separador al final de la lista expuesta por la instrucción PRINT más reciente. Una PRINT por sí sola provoca un avance de línea.

- AT se usa para situar el cursor **en** una posición dada de la pantalla, para comenzar a exponer datos a partir de ella.

- CLS hace que se **limpie** la pantalla ("aclare").

- INPUT se utiliza para **imponer** a las variables, valores introducidos por el teclado. Se puede exponer un **mensaje** que explique el dato que se pide, si se coloca entre comillas después de la palabra clave INPUT.

- READ, DATA y RESTORE se usan para que el programa **apunte** un **dato** como valor de una variable, tomándolo de una lista de datos incorporada en el programa; y puede a voluntad hacerse que el ordenador **repunte** su puntero interno con el que marca los datos usados, hacia una instrucción dada.

- La función INKEY$ permite **inquirir** si hay alguna tecla pulsada, o pendiente de ser tratada en el teclado; y puede hacerse sin ninguna espera en la ejecución o con un período de espera determinado. La función KEYROW entrega el número de columna correspondiente a la tecla pulsada en ese momento, perteneciente a la **fila** mencionada como argumento de la función.

- El reloj interno del sistema puede ser puesto a la hora que se desee, ajustado y examinado su valor, usando las sentencias SDATE y ADATE y la función DATE$, respectivamente.

Capítulo Cinco

Bucles y Disyuntivas

Hemos visto cómo pueden establecerse bucles en un programa en BASIC usando la sentencia GOTO. Sin embargo, no se ha descrito ningún **método de salida** apropiadamente controlado por el programa. El bucle **con** contador del Capítulo 2 fue mostrado para usarlo en situaciones con límites **preconfinados.** Este capítulo muestra cómo establecer, entrar y salir de bucles en diversas maneras.

Una de las grandes potencias del ordenador es su habilidad para "tomar una decisión" ante una **disyuntiva** presentada. El QL tiene algunos comandos muy potentes para este aspecto del cómputo, y se describirán en este capítulo para permitirte producir algunos programas altamente complicados.

Verás que el curso a seguir depende del resultado de una **premisa lógica,** por lo que será necesario presentar algo sobre ello en este capítulo. También aquí, el QL dispone de construcciones complejas y lógicas con las sentencias adecuadas, y te encontrarás con un alto grado de flexibilidad en el lenguaje SuperBASIC para esta clase de acciones.

Bucles

Hasta ahora los bucles se han formado usando sentencias que incondicionalmente hacen que **vaya** a una instrucción determinada. Por ejemplo:

```
10 PRINT "Rondas por un bucle"
20 GOTO 10
```

Este bucle es ciego: no tiene **salida.** Y continuará hasta que provoques una **interrupción** de la ejecución. Incluso aunque incluyamos un bucle con contador, éste simplemente va llevando la cuenta pero no habíamos incluido ninguna manera de comprobar cuándo se ha alcanzado un valor particular, para salir del bucle.

Si simplemente se requiere ejecutar un bucle un **preconfinado** número de veces, digamos 10; puedes incluir un contador de bucle, digamos N, que incrementa en cada pasada o ronda que efectúe. Y luego mandar al ordenador que:

> Ejecute el bucle CON N = 1; luego añada a N el PASO establecido para saltar a OTRO valor (que normalmente es el 2), y ejecute otra vez el bucle. Y se le pide que continúe repitiendo hasta que el valor de N llegue AL 10, momento en que se saldrá del bucle.

Esta acción repetitiva es fácil de transferir a sentencias del BASIC, ya que es muy parecida... al inglés. En la sentencia equivalente del BASIC de este bucle preconfinado, se usan las palabras inglesas FOR ("CON"), TO ("AL"), STEP ("PASO") y NEXT ("OTRA") para conseguir la misma acción. Es la llamada sentencia FOR. El bucle requiere realmente dos sentencias, tal y como en este ejemplo:

```
FOR N = 1 TO 10 STEP 1

(Lista de sentencias que componen una ronda del bucle)

NEXT N
```

La sentencia encabezada por FOR es la **entrada** del bucle, y la sentencia NEXT N marca la **salida** del bucle. Las sentencias que sustituyen a la frase entre paréntesis son las que están dentro del bucle, y el ordenador interpreta esta estructura gramatical como entendiendo que dichas sentencias han de ejecutarse con valores de N variando del 1 al 10, en incrementos de 1. De hecho, 1 es el valor prescrito para omisiones de la **cláusula** STEP, por lo que puedes prescindir de ella en esta ocasión. Un ejemplo del uso de la sentencia FOR podría ser:

```
10 PRINT\\"Bucle comenzado."
20 FOR N=1 TO 10
30 PRINT N
40 NEXT N
50 PRINT\\"Bucle finalizado."
```

El bucle comienza en la línea 20 con el valor de N puesto a 1. Luego se ejecuta la única sentencia dentro del bucle, la línea 30, en que se expone el valor de N. La NEXT N le indica que dé **otra** vuelta si procede: en ese momento el ordenador verifica que N no ha llegado todavía **al 10** y vuelve a saltar a la línea 20 donde se le pide ahora que incremente el valor de N en una unidad, y dé otra pasada del bucle. Así continúa hasta que haya llegado al valor 10. En ese momento, al encontrar la sentencia NEXT, observa que el valor de N sobrepasaría al 10 si lo volviera a incrementar, con lo que finaliza la ejecución del bucle pasando a la sentencia que viene detrás de la 40. Esa podría ser la terminación del programa, pero ahora hay otra sentencia PRINT para informarte que el bucle ha finalizado.

Realmente no es necesario tener ninguna instrucción dentro del bucle. Por ejemplo, el siguiente bucle:

```
10 FOR I = 1 TO 1000
20 NEXT I
```

podría usarse para producir un "retardo" en el programa. Normalmente usarías la sentencia PAUSE, pero el retardo conseguido puede terminarse antes si se pulsa alguna tecla, lo que puede ser inconveniente en algunas circunstancias. Además, todo el bucle FOR puede colocarse en la misma línea separando las dos instrucciones por el signo **dos-puntos**, en la forma:

```
10 FOR I = 1 TO 1000:NEXT I
```

Veamos ahora alguna utilización simple de este método para establecer bucles **preconfinados.** El siguiente expone los primeros 12 valores de la tabla de multiplicar por 12:

```
10 FOR N = 1 TO 12
20 PRINT N*12
30 NEXT N
```

El bucle ejecuta la línea número 20 doce veces, exponiendo el número correspondiente. La imagen es simple y sin atractivo. Para mejorarla, podríamos añadir las palabras "Primera", "Segunda", etc. para indicar las sucesivas rondas que efectúa. Sin embargo, dichas palabras tienen que incluirse en el programa, para que pueda usarlas y exponerlas en pantalla. Podemos hacerlo incluyendo una sentencia DATA y una READ, como en el siguiente ejemplo:

```
10 CLS
20 RESTORE
30 REM - Bucle comenzado
40   FOR N=1 TO 12
50   REM - Apunta siguiente palabra
60   READ NUM$
70   PRINT NUM$!"Ronda","="!!12*N
80   NEXT N
90   REM - Bucle finalizado
100 REM - Ordinales requeridos
110 DATA "Primera","Segunda","Tercera"
120 DATA "Cuarta","Quinta","Sexta"
130 DATA "Septima","Octava","Novena"
140 DATA "Decima,"Undecima","Duodecima"
```

Las primeras dos sentencias dan al programa un claro despeje. Las sentencias REM explican el funcionamiento del programa. Debieras ser capaz de ver que ejecuta las sentencia dentro del bucle 12 veces. Esas son las líneas 50, 60 y 70. Observa cómo el propio bucle se destaca al **sangrarlo** un poco con respecto al número de línea. Este bucle expone la palabra correcta en cada pasada, seguida del valor pertinente de la tabla. Justamente cuando la lista de datos queda exhausta, el contador de bucle alcanza su límite, y por tanto finaliza. El resultado de salirse del bucle es el de ejecutar las sentencias que van detrás de la línea 80; pero como no son ejecutables en este ejemplo, el programa termina.

Hay todavía mejoras a efectuar sobre la imagen obtenida, tales como añadir los números que se están multiplicando en cada ronda. Ensaya usando EDIT 70 para revisar la línea 70 y que sea ahora:

```
70 PRINT NUM$!"Valor","="!!12 X"!N!!"="!12*N
```

Para analizar esta sentencia de exposición, debes prestar atención a cada uno de los caracteres de la lista de datos a exponer. Los primeros cuatro caracteres forman el identificador para cada palabra contenida en las sentencias DATA. El siguiente carácter es un separador (!) de un solo espacio. Luego viene un grupo de 5 caracteres (Valor) que forman una constante literal. Si examinas la lista DATA, verás que el número de caracteres del contenido de NUM$, más un espacio, más los 5 caracteres de la constante literal, suman más de 8 pero menos de 16, siempre.

Eso significa que las primeras dos zonas de 8 posiciones en la pantalla están ocupadas, y la coma separadora hace que se coloque siempre el signo = al comienzo de la tercera zona. Los siguientes dos espacios, seguido del grupo de cuatro caracteres (12 X), con su espacio (!) ocupan las siguientes 7 posiciones. Luego viene N seguido de dos separadores, y eso puede ocupar tres o cuatro posiciones dependiendo del valor de N. El signo = y el separador, seguido del valor de 12*N puede por tanto comenzar en una u otra de dos zonas en la pantalla, y es la causa de la falta de alineación en la última parte de la línea expuesta.

Esta explicación es complicada, aunque sólo usa los principios presentados hasta ahora. Merece la pena detenerse el tiempo suficiente como para asimilar esa información, y asegurar que la comprendes. La sentencia de exposición de datos PRINT es una de las más importantes del lenguaje y necesitarás usarla en manera más y más complicada a medida que aprendes a programar.

Generalmente en BASIC, la sentencia FOR no necesita usar enteros como contador de bucle: puede comenzar en cualquier valor, finalizar en cualquier valor, y dar un salto de cualquier valor. Esos valores pueden además ser positivos o negativos; o venir dados por variables o expresiones numéricas. Sin embargo, las primeras versiones del QL no te permitían incrementar el contador del bucle por debajo de cero.

Como un ejemplo de un bucle con contador no entero, te presentamos el siguiente programa que evalúa una expresión numérica que puede usarse para alguna clase de aplicación científica. También se incluyen sentencias para mostrarte lo rápido que el programa funciona. Debieras usar NEW antes de teclear este programa:

```
10 CLS
20 REM - Expone la hora de comienzo
30 PRINT "Hora inicial es"
40 N = 0
50   REM - El bucle comienza aqui
60   FOR Y = 2.5 TO 0 STEP -.5
70   N = N + 1
80   X =(1-X)/(10-Y)*8.75E8
90   PRINT "Valor num."!!N,"="!X
100  REM - Bucle finaliza aqui
110  NEXT Y
120 PRINT\\"Hora final es"!!DATE$
```

Este programa comienza con el contador en el valor 2,5 y resta 0,5 en cada ronda que efectúa, hasta que finaliza con el número 0. La variable N se usa para proporcionar el número correcto en el mensaje a exponer. Observa que N debe ser puesto a 0 al principio para asegurarse que está anulada para utilizarse dentro del bucle. Para comprobar tu máquina, ensaya cambiando el límite final del bucle, del valor 0 al valor -1 y observa el efecto. Si la salida oscila, y el bucle nunca finaliza, es que en tu máquina no se permite que la sentencia FOR emplee valores negativos como límite.

La sentencia FOR tiene otros atributos que te presentaremos después de la siguiente sección sobre "decisiones". También hay otras claves en SuperBASIC asociadas con **estructuras de bucle** que serán presentadas más adelante.

Disyuntivas

En los ordinogramas del Capítulo 2, se presentaron **condiciones** para hacer que el programa siguiera un curso diferente dependiendo del resultado de una **premisa comparativa** o alguna otra condición. En el lenguaje normal, usamos frases condicionales tales como **si... pues...**, para esta clase de situación. Podríamós decir 'si se cumple una determinada condición, entonces exponga el valor de una variable determinada'.

En inglés, las palabras claves son IF ("SI...") y THEN ("PUES, ENTONCES") -aunque lo habitual es suprimir el THEN- se usan para establecer las condiciones, y han sido las adoptadas para el BASIC. Como ejemplo del uso de estas palabras, recordarás que cuando te presentamos GOTO, los bucles que ensayamos simplemente continuaban por siempre hasta que se provocaba la **interrupción**. Usando premisas condicionales -encabezadas por IF- podemos establecer la condición para poder salir de un bucle. La forma más simple de condicionamiento de una sentencia, es la dada por:

IF condición es cierta THEN GOTO número de línea

Que indica que **si** se cumple la condición, **entonces vaya** al número de línea mencionado. Si la condición no se cumple, no se efectuará el comando y continuará la ejecución en la instrucción que venga detrás de la sentencia IF.

Por ejemplo:

```
10 CLS
20 N = 0
30 PRINT "Esta es la ronda numero"!N
40 IF N = 10 THEN GOTO 70
50 N = N + 1
60 GOTO 30
70 REM - El programa termina aqui
```

La primera sentencia después de la CLS marca el comienzo de un bucle **condicionado**, y con una variable que se fija inicialmente al valor cero. La sentencia 30 es la acción que se efectúa en el bucle, así como la 50 en que se incrementa el valor de la variable. La instrucción 40 es la que efectúa una **comprobación** de si N ha alcanzado ya el valor 10, lo que claramente no ocurre en la primera pasada. Si la condición fuera cierta, se ejecutaría la sentencia GOTO que forma la cláusula encabezada por THEN, y el programa continuaría en la línea 70, saliéndose del bucle. La condición es falsa al principio, por lo que se ignora la cláusula THEN, y el programa pasa a la línea siguiente que es la 50. La línea 60 manda incondicionalmente que vaya a la 30, con lo que se repite el proceso. Después de haberse dado **once** rondas, el valor de N alcanza el valor **diez**, con lo que la condición se convierte en cierta, y por tanto se ejecuta la orden de que vaya a la línea 70, y continúe allí (en este caso, acabando).

Es de suma importancia establecer el valor inicial del contador de bucle correctamente. Puede que haya sido una sorpresa para ti ver que este programa realmente expone 11 números, mientras que en la condición comprobada aparece el número 10. La razón es simplemente como puedes ver al ejecutar el programa, que se exponen los números 0 a 10.

Para que se expusieran 10 números, desde el 1 al 10, tendrías que haber dado a N el valor 1 en la línea 20. Para exponer los 10 números que van del 0 al 9, puedes simplemente cambiar el orden de las instrucciones en el programa. Por ejemplo, ensaya intercambiando las sentencias de las líneas 40 y 50 y ejecutando el programa. Eso hará que se exponga 0 como primer número, y el bucle finalizará al ser expuesto el 9. Es una lección muy importante y deberás prestar atención a los valores de comienzo y finalización de un bucle, y en general a la condición de entrada y de salida -para que cumplan exactamente lo que requieres.

La sentencia IF da un método de ejecutar una serie de instrucciones dependiendo de la certeza o falsedad de una condición dada. En los ejemplos anteriores, la sentencia GOTO que formaba la cláusula THEN, **sólo** se ejecutaba si la condición era cierta. Podría haberse colocado cualquier otra sentencia dentro de la cláusula THEN en la misma línea, e incluso varias sentencias separadas por dos-puntos. Todas ellas serían ejecutadas sólo si se cumple la condición. Además, la condición puede ser considerablemente más compleja que esta simple comprobación del valor de una variable.

El ordinograma de la Fig. 2.5 del Capítulo 2 puede ahora escribirse como un programa con cierta facilidad. El signo mayor que (>) se usa de la misma manera que el signo igual para formar **premisas condicionales** de determinadas sentencias. La Fig. 5.1 muestra el proceso usado para hacer que la máquina **canjee** los valores de dos variables.

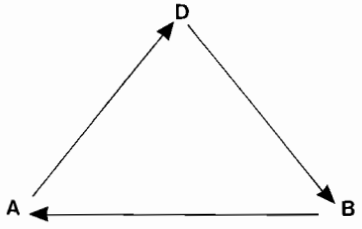


Fig. 5.1. Canjeando valores de variables

Las variables se muestran alrededor de los lados del triángulo para ilustrar el proceso. Para canjear los valores de A y B, primeramente tiene que depositarse el valor de A en alguna otra parte, en la variable D por ejemplo; y luego depositar el valor de B en A. Finalmente, se hace que B sea igual al valor depositado en D. Observa el diagrama de flujo de la Fig. 2.5 para ver la lógica subyacente en el programa. Se usan las mismas variables, y sus valores les serán impuestos durante la ejecución del programa:

```
10 CLS
20 PRINT "Teclea sucesivamente 3 numeros,"
30 PRINT "cada uno concluido con ENTER."
40 PRINT
50 INPUT A,B,C
60 REM - Si A>B canjee A y B
70 IF A>B THEN D=A:A=B:B=D
80 REM - Si B>C canjee B y C
```

```
90 IF B>C THEN D=B:B=C:C=D
100 REM - Si A>B canjee A y B
110 IF A>B THEN D=A:A=B:B=D
120 PRINT
130 PRINT "En orden ascendente son:"
140 PRINT A,B,C
```

Las líneas 70, 90 y 110 efectúan el intercambio de valores cuando es necesario, y los dejan tal como están en los demás casos. El resultado es pues que A, B y C contienen los números en el orden correcto. Si necesitas consultar la explicación del proceso, repasa el Capítulo 2. Ejecuta el programa unas cuantas veces, con diferentes números, incluyendo negativos, para ver lo que sucede.

Antes de explorar con mayor detalle la sentencia IF, veamos lo que puede añadirse al programa anterior para hacerlo que se ejecute una y otra vez hasta que decidas finalizar. Podríamos hacerlo, haciendo que el programa preguntara si deseas ejecutarlo de nuevo; y si no, terminar. Eso se consigue usando la sentencia INPUT para imponer la respuesta a una simple pregunta. Añade las siguientes líneas al programa:

```
150 PRINT
160 INPUT "Ejecuto otra vez?";ANS$
170 IF ANS$="si" THEN GOTO 10
180 PRINT\\"Gracias, ahora no ejecuto."
190 PRINT\\"y te digo ADIOS!"
```

La línea 160 te pide si deseas volver a ejecutar el programa, y deberás teclear una respuesta. La serie de caracteres que teclees será impuesta en la variable ANS$. La línea 170 comprueba luego el contenido de esa variable, efectuando una comparación con el literal "sí" que ha de estar en minúsculas. Si tecleaste eso, el programa comienza otra vez por el principio. **En** los **demás** casos termina con un mensaje. Observa que cuando se te hace la pregunta, si tu respuesta es cualquier otra cosa **-en demás-** el programa terminará. Ensaya con las que se te ocurran para ver el efecto.

Si deseas hacer el programa más inteligente, puedes añadir una línea que compruebe el "SI" después de la línea 170, y también vaya al comienzo del programa si la condición se cumple. Podrías incluso comprobar si se ha tecleando "NO" o "no" y en el caso en que no hubieras tecleado ninguna de ellas, el programa debería decirte que has cometido una equivocación y pedirte que volvieras a teclear tu respuesta. Es un ejercicio excelente ensayar a modificar el programa para que efectúe esas acciones. Un ejemplo posterior usará esas ideas, y entonces verás cómo conseguirlas si no tienes éxito ahora en tus intentos.

Esta suerte de interacción entre el usuario y la máquina es esencial al escribir programas para personas que no están interesadas en la programación, sino simplemente quieren que la máquina sea tan autoexplicativa como sea posible.

Relaciones BOOLEANAS

Para construir condiciones lógicas más complejas después de la clave IF, tendremos que examinar la manera de combinar condiciones lógicas simples tales como N=10 y A > B. Eso permitirá construir cualquier clase y matiz de condición, y hacer por tanto que la **decisión** sea más inteligente. Este tema es parte de un estudio denominado "álgebra booleana" en honor de George Boole, el matemático del siglo XIX que descubrió muchas de las leyes de la lógica que usamos hoy en la informática moderna.

En el Algebra Booleana, las **proposiciones** pueden tener dos valores: **cierto** (TRUE) y **falso** (FALSE). Así, la proposición:

la hierba es verde = TRUE

mientras que la:

2 > 15 = FALSE

Las proposiciones en la parte izquierda de estas igualdades son **entidades** lógicas, y al igual que para las variables numéricas, su valor (cierto y falso) puede ser expuesto y comprobado. Si fueramos a inventar un ordenador, que no trabajara sobre números, sino más bien sobre proposiciones lógicas, seríamos capaces de escribir:

```
PRINT la hierba es verde
```

Dado que las palabras no están contenidas entre comillas, esas palabras no serían expuestas **literalmente**, sino que el "valor" de la expresión sería mostrado como TRUE.

La sentencia siguiente:

```
PRINT 2 > 15
```

saldría como FALSE en la pantalla. Las entidades en la lista PRINT se denominan **expresiones Booleanas** para distinguirlas de las expresiones numerales o literales.

El equivalente en un ordenador numérico sería:

```
PRINT 12/3
```

y la pantalla mostraría el número 4 en lugar de exponer los caracteres 12/4.

En este ordenador lógico, los valores lógicos de las expresiones lógicas serían tratados igual que los valores numéricos y las proposiciones. Habría incluso algunos equivalentes a +, -, * y demás. Sea o no esta clase de ordenador de alguna utilidad, es una cuestión que cae fuera del ámbito de este libro. Sin embargo, el BASIC permite que se evalúen proposiciones lógicas en cuanto a su certeza o falsedad, de manera que puedan comprobarse condiciones complejas y emplearse en instrucciones tales como IF...THEN...

En SuperBASIC, una condición lógica o **premisa** sí tiene un valor CIERTO o FALSO como hemos presentado anteriormente.

Sin embargo, dichos valores no vienen dados por las claves TRUE y FALSE, sino por unos valores numéricos que son equivalentes. Por ejemplo, el ordenador da el valor 0 a toda expresión lógica falsa, y cualquier número distinto de cero a las expresiones TRUE. Así, ensaya lo siguiente:

```
PRINT 1 > 2
PRINT 2 > 1
```

La primera interpretará la expresión en la lista PRINT como una expresión Booleana a causa del signo > . Comprobará su certeza y en los demás casos observará que el FALSE, y por tanto hará PRINT 0. La segunda devolverá un número distinto de 0 ya que es TRUE.

Puedes "engañar" al ordenador en una sentencia IF sabiendo la manera en que funciona. Por ejemplo, ensaya:

```
IF 0 THEN PRINT "FALSO"
```

Eso no expondrá nada por que la condición después de la IF tiene un valor 0, y eso es interpretado como FALSE. Ahora ensaya con:

```
IF -15 THEN PRINT "CIERTO"
```

Eso expondrá CIERTO en la pantalla, ya que la condición IF es un número distinto de 0 y eso se interpreta como TRUE.

Para operar con sentencias de decisión, es esencial ser capaz de combinar condiciones, y hacerlas más complejas para incluirlas en la sentencia IF. Por ejemplo, el programa en la última sección podría haberse beneficiado de ser capaz de preguntar cosas tales como: '¿es ANS$ igual a SI o sí?'. La palabra importante es la conjunción O (en inglés OR) que se usa para combinar las dos posibilidades. Eso nos permitiría comprobar versiones en mayúsculas y en minúsculas de la respuesta dentro de la misma sentencia.

Hay varias operaciones lógicas que pueden combinarse para construir expresiones lógicas en SuperBASIC, y son las siguientes:

AND (Y SI), OR (O SI), XOR ("U" SI) y NOT (NO)

Esas **claves** se denominan **operadores Booleanos,** al igual que +, -, * y / se denominan operadores numéricos. Generalmente, estos operadores tienen el mismo significado en BASIC que en el lenguaje ordinario, pero son más precisos. El único que puede ser poco familiar es XOR ("U" SI). Todos se describen a continuación.

AND (conjunción copulativa **'y'** - yliación)
Se usa para combinar dos condiciones que **ambas** deben ser ciertas para que la expresión sea cierta. Por ejemplo:

1 > 0 AND 2 > 3 es falsa como expresión compuesta

dado que la segunda expresión componente no es cierta. Sin embargo, la siguiente condición compuesta es cierta:

12/4=3 AND 15 > 3

porque cada una de las expresiones Booleanas componentes tiene el valor TRUE.

Ensaya exponiendo el valor que toma la expresión combinada.

OR (conjunción aclarativa **'o'** - oliación)
Se usa para combinar dos expresiones, para dar una expresión Booleana que es cierta si **una u otra o ambas** de las condiciones componentes son ciertas. Por ejemplo,

```
1 > 0 OR 3 > 2
```

es cierta porque ambas condiciones son TRUE. También,

```
0 > 1 OR 15/5=3
```

son ciertas porque al menos una de las condiciones es TRUE. La siguiente sin embargo es falsa:

```
0 > 1 OR 18/3=16
```

porque ni una ni otra de las condiciones son ciertas. Otra vez, ensaya exponiendo los valores que resultan en la expresión lógica compuesta.

XOR (conjunción disyuntiva **"u"** - oleación)
Es el operador OR **exclusivo**, y es similar a él. Da una condición cierta si **una u otra** de las condiciones componentes es cierta, **pero no si ambas** lo son. Usando los mismos ejemplos que para el "o lógico **inclusivo**" (OR), compara los siguientes resultados:

```
1 > 0 XOR 3 > 2      que es falsa
0 > 1 XOR 15/5=3     que es cierta
0 > 1 XOR 18/3=16    que es falsa
```

Concuerda con el "o lógico inclusivo" excepto en la primera expresión, dado que **ambas** partes de la condición son ciertas, lo que excluye en este caso un resultado cierto.

NOT (negación)
No se usa para combinar dos expresiones, sino que puede usarse para cambiar la certeza o falsedad de una condición lógica. Por ejemplo:

```
NOT 2 > 3
```

es lo mismo que decir que "**no** es 2 mayor que 3", lo que obviamente es cierto. Mira a ver si puedes predecir el resultado de exponer el valor de esa expresión, y luego comprueba si has acertado.

Veremos estos operadores Booleanos en ejemplos posteriormente.

Relaciones

El símbolo > se denomina un **operador de relación,** y es uno de los que se usan para **comparar** datos literales y numerales. Hay seis operadores de relación en el repertorio del SuperBASIC:

Mayor que	>
Menor que	<
Mayor o igual que	>=
Menor o igual que	<=
Igual a	=
Distinto de (no igual a)	<>

Se usan todos de la misma manera que el signo > , y habitualmente es posible escribir las comparaciones usándolos de diversas maneras. Por ejemplo, si A es mayor que B, también es B menor que A. La expresión A > B podría por tanto, escribirse como B > A.

También es perfectamente correcto -aunque muy lioso- escribir algo así como:

```
A>=B AND NOT A=B
```

Tendrás que inspeccionarlo bastante cuidadosamente, y pensar bien ambas expresiones y la expresión compuesta resultante, cuando no estés familiarizado con este tipo de lógica. Ensaya a buscar otras expresiones que sean equivalentes a esta. Encontrarás que ayuda -la mayor parte de las veces- expresar estas relaciones con el lenguaje que empleas normamente.

Un ejemplo de programa

Esta sección describe un simple juego que comprobará y mejorará tu tiempo de reacción al usar las teclas numéricas. El objeto del juego es pulsar la tecla correspondiente a la cifra mostrada en la pantalla, dentro de un cierto límite de tiempo. El programa usa la función INKEY$, para **inquirir** la tecla pulsada y varias expresiones Booleanas.

El programa comienza mostrando un mensaje para decirte lo que hacer. Luego has de teclear un número que el ordenador usa para determinar lo rápido que ha de ser el juego. Las cifras permitidas son del 0 al 9. Cuanto mayor sea el número que elijas, más lento será el juego. Después de elegir la rapidez, otro mensaje te dice lo que hacer luego. Se te pide que pulses cualquier tecla para comenzar el juego. El ordenador elige después un número al **azar** y lo muestra en la pantalla. Si te las arreglas para pulsar la tecla rápidamente, el ordenador te dice que has ganado. El tiempo máximo que se te concede es poco menos de un segundo. Lo más rápido posible es probablemente demasiado rápido a no ser que seas muy afortunado. Cuando comprendas el programa, "debes" modificarlo a tu gusto.

Después de haber tecleado el programa, te daremos algunas instrucciones para que lo **guardes** en un cartucho de cinta para usarlo posteriormente.

Hay una función usada en este ejemplo con la que no nos hemos topado antes. Es la que produce números al azar. La función RND(N) proporcionará un número aleatorio (en inglés "**rand**om") entre 0 y N; y la describiremos completamente en el Capítulo 7.

El programa deberá teclearse con gran cuidado para incluir cada uno de sus detalles. Cuando teclees, cometerás errores y tendrás que corregirlos. Eso te dará experiencia útil al usar el QL para programas que son demasiado largos como para caber en una sola plana de pantalla. Usa la función EDIT para revisar cada línea, en aquéllas que sean demasiado largas, para evitar volver a teclearlas por causa de un solo error; y usa el comando para que liste partes del programa cuando hayas acabado. Recuerda que puedes hacer que lo liste por secciones. Además, si el listado se sale y amenaza con correr la imagen, quitando las líneas que quieres inspeccionar, siempre puedes detenerlo provocando una inter**rupción.** Observa que por hacerlo más compacto, hemos usado el mínimo número de espacios en blanco.

Cuando hagas que el ordenador ejecute el programa, puede contener todavía algunos de los errores cometidos al teclear. Oblígale a que **edite** las líneas que el QL detecta erróneas, y compáralas con el listado del libro. Revísalas según sea necesario:

```
10 CLS
20 REM *** EL JUEGO DE REACCION***
30 PRINT"   LO RAPIDO QUE ERES"
40 PRINT"ESTE JUEGO TE LO MOSTRARA"
50 PRINT
60 PRINT"ELIGE UN NIVEL DE RAPIDEZ 0-9"
70 PRINT"0 ES RAPIDO, 9 ES LENTO"
80 REM
90 REM INGRESAR NIVEL DE RAPIDEZ
100 REM
110 PRINT:INPUT"RAPIDEZ?",SPD%
120 IF SPD%<0 OR SPD%>9 THEN
    PRINT"TECLEA UN NUMERO DEL
    0 AL 9":GOTO 110
130 CLS
140 PRINT"TU NIVEL DE RAPIDES ES"!SPD%
150 PRINT
160 PRINT"SE TE MOSTRARA"
170 PRINT"      UNA CIFRA"
180 PRINT"EL JUEGO ES PULSAR"
190 PRINT"ESA TECLA RAPIDAMENTE!"
200 PRINT
210 PRINT"PULSA CUALQUIER TECLA PARA JUGAR"
220 REM
230 REM ESPERAR A QUE PULSE TECLA
240 REM
250 PAUSE -1
260 REM
270 REM ELEGIDO NUMERO AL AZAR
280 REM
290 RN%=RND(9)
```

```
300 PRINT
310 PRINT"PULSA ESA TECLA:"
320 PRINT:PRINT,RN%,"AHORA!!"
330 REM
340 REM A ESPERAR UN POCO DE TIEMPO
350 REM HASTA QUE PULSE ESA TECLA
360 REM
370 KEY$=INKEY$(1+SPD%*5)
380 REM
390 REM PIERDES SI NO ES UNA CIFRA
400 REM
410 IF KEY$("0" OR KEY$)"9" THEN
    GOTO 500
420 VAL=KEY$
430 REM
440 REM COMPROBAR SI TECLA CORRECTA
450 REM
460 IF VAL=RN% THEN GOTO 520
470 REM
480 REM MENSAJE ACIERTO O FALLO
490 REM
500 PRINT:PRINT"MALA SUERTE - PIERDES"
510 GOTO 560
520 PRINT"BIEN HECHO!!  GANAS"
530 REM
540 REM VER SI OTRO JUEGO
550 REM
560 PRINT
570 INPUT"OTRA JUGADA?",ANS$
580 IF ANS$="SI" OR ANS$"si"
    OR ANS$="S" OF ANS$="s"
    THEN GOTO 10
590 IF ANS$()"NO" AND ANS$()"no"
    THEN GOTO 570
600 PRINT:PRINT"         ADIOS!!!"
```

La primera sección de este programa es una serie de exposiciones en pantalla de mensajes, y un ingreso del dato de rapidez impuesto como valor de SPD%. Es una variable entera, y aunque se introduzca un número no entero, será redondeado. La línea 120 comprueba si el jugador ha tecleado un número entre 0 y 9; si no es así, muestra un mensaje para avisar al jugador que debe teclear una cifra válida. Eso se repite hasta que lo logra. La siguiente sección es otra vez de exposición de mensajes, para hacer el juego interactivo y autoexplicativo.

La línea 250 hace que se produzca una **pausa** usando la sentencia con un número tal que hace que la pausa se prolongue hasta que se pulse una tecla. Cuando así ocurre, se asigna a RN% un número aleatorio entre 0 y 9, ambos inclusive. La función RND no produce un entero, y RN% hace que se redondee cuando es depositado en esa variable. Luego se expone el número, y el jugador intenta pulsar esa tecla antes de que se agote el tiempo especificado por el argumento de la función INKEY$. Cualquiera que sea el carácter pulsado, que es el valor devuelto por la función INKEY$, queda depositado en la variable KEY$ para uso posterior. El argumento en INKEY$ es una expresión formada a partir de SPD%, de manera que cuanto mayor sea, mayor será la demora permitida. Puedes alargarla incluso si multiplicas SPD% por un número mayor de 5.

Si no se ha pulsado una tecla antes de que se agote la pausa permitida, la función INKEY$ entregará como resultado el carácter **nulo**. La línea 410 comprueba si lo tecleado es una cifra entre 0 y 9. Verás que existen exactamente los mismos operadores de relación para literales que los vistos para numerales. Lo explicaremos completamente en el Capítulo 6.

Si en KEY$ no hay un número entre 0 y 9, hacemos que el programa se vaya a la línea 500 y diga que has perdido. Cuando sí es un número, el **concepto de coerción** es usado automáticamente por el ordenador para cambiar ese literal a un numeral. La línea 420 asigna a una variable numérica el valor litérico de la variable literal KEY$. El QL cambia automáticamente un dato literal a un dato numeral, cuando se le pide, como en este caso (y cuando lo puede hacer sin provocar un error). La línea 460 comprueba luego si el número ha sido el correcto, comparándolo con el número aleatorio contenido en RN%. Si sí es correcto, aparece un mensaje de ganador; en los demás casos, pierdes.

En uno u otro caso se alcanza la línea 570, donde se te pregunta si deseas jugar de nuevo. La línea 580 comprueba cuatro maneras de responder afirmativamente. Puedes teclear la palabra SI o simplemente la letra S en mayúsculas o en minúsculas. La siguiente línea comprueba que tu respuesta no contiene la palabra NO ni en minúsculas ni en mayúsculas. Si la contuviera, el programa terminaría; en los demás casos al jugador se le vuelve a repetir la pregunta.

Las sentencias IF asociadas con las preguntas y respuestas, con el diálogo entre el hombre y la máquina, están construidas para atrapar cualquier posible error cometido al introducir datos.

Este programa da una cantidad justa de práctica sobre el trabajo hecho hasta ahora, y deberás intentar comprenderlo y usar algunas de sus ideas en tus propios programas.

Si deseas hacer que el ordenador **guarde** este programa en sus almacenamientos de cinta, la siguiente sección te dice cómo hacerlo. El Capítulo 9 describe el uso de las **"microductoras"** de cinta con más detalle.

Guardando el programa

Si esta es la primera vez que has usado o almacenado un programa, debieras consultar el manual QL y observar las precauciones que indica con gran cuidado. Recuerda que los cartuchos contienen cinta magnética, y que la información en ellos puede ser borrada por cualquier campo magnético que interfiera. Una manera favorita de perder datos es llevar los cartuchos cerca de un artefacto que oculta una magneto -por ejemplo, altavoces en radio-transistores, o botones magnéticos para retener papeles de los que uno encuentra en los despachos de la gente.

Habiendo leído las precauciones, si tienes una cinta **virgen** -que nunca ha sido usada antes- debes primero **formatearla** dando el comando FORMAT. Quítalo de su funda plástica externa y colócalo en la ranura de la "microductora" izquierda.

Teclea exactamente lo siguiente:

FORMAT MDV1_PROGS1

No olvides el carácter **guión de subrayar** (_). Este comando hará que el QL **formatee,** dé forma a la cinta para que puedan almacenarse programas y datos en ella; y la pondrá como **título** el de PROGS1 para identificación posterior. No efectúes esta operación con una cinta que ya tenga información guardada en ella, ya que la destruirás y la dejarás como nueva.

El siguiente paso es hacer que **guarde** el programa en la cinta como un **fichero** con un determinado nombre o título. Para que lo **guarde,** tienes que darle la clave SAVE (ahorrar, poner a salvo) y puedes darle el nombre que desees; así que con el siguiente comando

SAVE MDV1_GAME

le asignas el nombre GAME (juego). Para ver que ha quedado grabado en la cinta, puedes inspeccionar el **directorio** de esa cinta, que es donde se mantienen registrados todos los ficheros que hay en ella. Teclea lo siguiente:

DIR MDV1_

Eso hará que te muestre un mensaje concerniente a la cantidad de **sectores** libres/disponibles, seguido del título del fichero GAME. Es buena práctica y puede evitar desastres, guardar más de una copia de cada fichero. Coge una segunda cinta y repite el proceso anterior. Eso te dará una copia principal que debieras usar todas las veces, y una **copia de respaldo** que deberá comprobarse pero sólo emplearse cuando falla la copia principal.

Para recuperar el programa de la cinta y hacer que se **cargue** en la memoria del QL, teclea lo siguiente:

LOAD MDV1_GAME

Cuando el cursor regresa a la pantalla, puedes hacer que ejecute el programa incluso aunque la cinta esté todavía en marcha. Sin embargo, no quites el cartucho hasta que esté parada.

Facilidades en SuperBASIC

La mayoría de estas facilidades que hemos visto hasta ahora son standard en la mayoría de los dialectos del BASIC. El resto de este capítulo -y mucho del resto del libro- describe facilidades que son más características y peculiares del QL y del SuperBASIC. Eso no quiere decir que otros ordenadores no dispongan de esas facilidades, sino que son un poco menos comunes.

Ahora describiremos las maneras en que se estipulan normalmente los bucles y decisiones (mejor llamados cruces) en SuperBASIC, y veremos cómo así se fomenta el arte de confección de programas que se denomina **programación estructurada.**

La "programación estructurada" es el método de redactar un programa de una manera convenida y fácilmente legible. Entraña por ejemplo, escribir las acciones principales a efectuar como un conjunto de **módulos** fácilmente modificables. Una ventaja es que el programa es acorde con una clase de estructura conocida, y eso ayuda a que sea **portable** o transferible a otros lenguajes, a otras máquinas y a otras personas. Si adoptas la costumbre de redactar tus programas de una manera coherente y estructurada, serán pulcros y nítidos, fáciles de leer, fáciles de transferir a otras máquinas y fáciles de modificar en fechas ulteriores. Esas son ventajas importantes y muy raras veces los programas reales confeccionados así son más largos de lo que serían si los escribieras simplemente como un bloque sin ninguna estructura de frases codificadas para el ordenador.

La sentencia REPite

La sentencia REP es la más simple de las construcciones estructuradas en SuperBASIC. Sustituye a la sentencia GOTO en estructuras de bucle. Usa un **rótulo** o **identificador de bucle** (en inglés 'label'=etiqueta, marbete) para determinar los límites del bucle, en lugar del número de línea habitual. Las reglas normales para identificadores se aplican también a este rótulo. En una sentencia REP, el comienzo del bucle sería de la forma:

```
REP bucle1
```

siendo bucle1 el identificador o rótulo. Han de usarse otras palabras clave del SuperBASIC precediendo a ese mismo identificador, para marcar dónde termina el bucle o para indicar que se **salga** del bucle. EXIT (salida) es de hecho una palabra clave en SuperBASIC, y cuando va seguida del rótulo identificativo de un bucle, hace que el programa deje de dar vueltas y se salga. A menudo se usará con una sentencia IF. La instrucción REP tiene dos formas, mencionadas normalmente como la "larga" y la "corta".

Un bucle con REP corta permite completar la estructura de bucle dentro de una sola línea, usando dos-puntos para separar las sentencias que lo componen. La siguiente muestra un bucle típico con REP corta:

```
10 N=0
20 REP LOOP1:N=N+1:PRINT N:
   IF N=1 THEN EXIT LOOP1
```

(Observa que cuando tecleas eso, el QL cambia la abreviatura REP en la palabra completa REPeat; y que normalmente a los bucles en inglés se les llama "loop" que es más bien lazo, gaza, rizo).

Si examinas esta sentencia múltiple, verás que hay tres comandos antes de la instrucción IF, incluyendo el REP. Luego viene la propia sentencia IF en la que se condiciona un quinto comando, que es el de **salga** del bucle.

Todo en la línea 20 está considerado como perteneciente al bucle identificado como LOOP1, y se ejecutarán una y otra vez hasta que se cumplimente el comando EXIT que corresponde al identificador de bucle correcto. Aquí sólo sucederá cuando N haya llegado a la cuenta de 10. En ese momento, la relación lógica de la sentencia IF será evaluada como TRUE, y el comando EXIT de la cláusula THEN hará que el programa se salga del bucle, y termine.

El comando EXIT sólo se enlaza con el correspondiente REP a través del identificador de bucle. Si no lo mencionaras con EXIT, el ordenador te diría que hay un error cuando escribieras la línea; pero si pusieras otro rótulo en lugar de ese, el ordenador lo aceptaría en el programa, pero daría el error cuando fuera a ejecutarse. Además, todavía puedes usar un comando GOTO en la cláusula THEN para hacer que se vaya del bucle, si lo deseas. Eso es una clase de estructura menos común, ya que el uso de EXIT con el identificador de bucle siempre es una manera mejor de hacer que el programa en algún sentido esté autodocumentado.

Un comentario adicional que concierne a la sentencia IF. Cualquier comando escrito después de la clave THEN, está considerado como parte de la propia sentencia IF y no como parte del bucle. Así, si trasladas el comando PRINT de su posición presente hasta detrás de la clave EXIT, usando un dos-puntos, solamente se ejecutará cuando N llega a 10. Ensáyalo y verás. Usa EDIT para hacer que revise esa línea; no necesitas volver a teclearla completamente para alterarla.

Se pueden colocar comandos antes de la REP en la línea 20 y no se considerarán incluidos en el bucle. El REP es la "cabecera" de LOOP1, aparezca donde sea. Intenta cambiando el programa para que sea:

```
10 N=0:PRINT"COMIENZO":REP LOOP1:N=N+1:
   PRINT N:IF N=10 THEN EXIT LOOP1
```

El bucle comienza ahora en el tercer comando de la línea 20, y no se necesita ninguna otra línea. Esta clase de condensación es útil para pequeñas rutinas tales como ésta, que efectúan una acción muy limitada. Esta simple instrucción de un programa ocuparía seis líneas de programa si se desarrollara completamente. Sin embargo, no siempre es buena idea condensar todo para ahorrar espacio, ya que un programa escrito de esa manera puede ser confuso y difícil de leer y corregir. Tendrás que encontrar el equilibrio mediante la experiencia.

La forma larga de REPite contiene instrucciones que están distribuidas sobre varias líneas contiguas, y el final del bucle queda identificado por el comando END REP, seguido del identificador de bucle. Se puede hacer que se salga del bucle mediante EXIT como antes. El siguiente es el mismo ejemplo, pero usando END REP:

```
10 N=0
20 REP LOOP1
30 N=N+1:PRINT N
40 IF N=10 THEN EXIT LOOP1
50 END REP LOOP1
```

Esto permite que una sucesión de líneas de programa sea ejecutada continuamente hasta que pase a ejecutarse el comando EXIT. Permite que se formen construcciones sintácticas más avanzadas y de manera más legible que usando bucles cortos.

La filosofía REP es usar rótulos (o etiquetas) para las cabeceras de bucle en lugar de usar los habituales números de línea. Eso permite emplear una palabra significativa para designar una **rutina** dada y no solamente un número. Ahorra también el tener que desarrollar los números de línea mientras se está escribiendo el programa, y te ayuda a ver el ámbito del bucle cuando estás trabajando. Si estás acostumbrado a otros BASIC, rápidamente verás la ventaja de estas estructuras rotuladas.

El siguiente programa está autodocumentado. Léelo cuidadosamente antes de continuar:

```
10 CLS
20 REP INGRESAR_NUMERO
30 INPUT"TECLEA UN NUMERO
   DEL 100 AL 200",X
40 IF X)=100 AND X(=200
   THEN EXIT INGRESAR_NUMERO
50 PRINT"SIGUE LAS INSTRUCCIONES!"
60 GOTO 30
70 END REP INGRESAR_NUMERO
80 PRINT"TU NUMERO HA SIDO:"!X
```

El bucle llamado INGRESAR_NUMERO se repite hasta que el usuario introduce un número válido. Observa que no puedes usar un espacio en blanco dentro de un identificador, y que para conseguir legibilidad está permitido usar el guión de subrayado. El identificador pretende que sea significativo y que dé alguna indicación de lo que representa o sucede.

El bucle comienza en la línea 20, e incluye todas las instrucciones hasta la línea 70 que marca el final del bucle. Cuando el número impuesto por teclado está dentro de la banda, tal y como se compara en la premisa IF, se ejecuta el comando EXIT que hace que la ejecución **salga** del bucle y continúe en la instrucción que va detrás de la que marca el final del bucle. Experimenta ejecutando el programa.

También puede usarse la clave NEXT con el identificador de un bucle REP, y tiene el efecto de provocar **otra** ronda del bucle, lo que es similar a END REP. Puede usarse dentro de una sentencia IF, tal y como se muestra en el siguiente ejemplo:

```
10 CLS:N=0
20 REP LOOP
30 N=N+1
40 IF N(5 THEN PRINT"N(5":NEXT LOOP
50 PRINT
60 PRINT "N)=5"
70 IF N=10 THEN EXIT LOOP
80 END REP LOOP
```

El bucle comienza en la línea 20 con 0 como valor de N. Dentro de él se incrementa N y luego se examina para ver si es menor que 5; **entonces** se ejecuta la cláusula THEN, y luego se pasa a **otra** ronda del bucle (NEXT LOOP). Es decir, se hace que el programa vaya al comienzo del bucle en la línea 20. En los **demás** casos -cuando N es mayor o igual a 5- se ejecuta la línea 50 y siguientes hasta alcanzar la 80 que marca el final del bucle, y por tanto, que se ejecute otra ronda.

Así se continúa hasta que N alcanza el valor 10, con lo que en la instrucción 70 hacemos que se salga del bucle. Un EXIT, hace por tanto que continúe la ejecución en la instrucción posterior a la END REP, mientras que NEXT fuerza a cortar la ejecución del bucle y pasar a **otra** ronda del mismo. Puede ser muy útil, como el ejemplo muestra, para ejecutar partes de un bucle bajo una condición, y otra parte bajo otra condición.

PARE, SIGA, REINTENTE

Hay otra manera de terminar un bucle usando la sentencia STOP. Es similar a provocar una interrupción (BREAK), pero se incluye dentro del programa. Cuando el programa encuentra un **pare**, cesa la ejecución y regresa al modo comando. Eso puede usarse de diversas maneras para ayudar a **depurar** un programa. Además, puedes hacer que siga, mandando CONTINUE después de un STOP, y el programa lo hará a partir de la siguiente línea. Ensaya añadiendo STOP en una línea numerada 55 en el penúltimo programa visto. Cuando introduces un número fuera de la banda exigida, el programa te dirá que sigas las instrucciones, y luego para. Puedes decir que CONTINUE en ese momento, y harás que siga el programa como si nada hubiera sucedido. Si cambias algo en el programa, entonces no puedes normalmente continuar de esa manera. Quita la línea 55 otra vez para hacer que le programa se ejecute normalmente.

El REINTENTO, de clave RETRY, se usa para hacer que continúe a partir del punto donde ocurrió un error, volviendo a intentar la ejecución de la última línea cumplimentada para ver si aparece el error otra vez. Eso puede ayudarte a quitar la pifia. Para verlo, haz que ejecute el programa y cuando te pide el número, teclea algunas letras del alfabeto. Al pulsar ENTER, el programa pasa al modo comando y te dice que hay un error en la expresión de la línea 30. Ahora teclea RETRY y el programa volverá a ejecutar esa misma línea. Esta vez ingresa números y el programa funcionará correctamente.

Ampliaciones a la sentencia FOR

SuperBASIC también te permite una forma corta para la sentencia FOR, junto con una construcción estructural larga usando una instrucción END FOR, de la misma categoría que en los bucles estipulados mediante sentencias REP.

La forma corta de la instrucción FOR es la siguiente:

FOR X=1 TO J STEPK: sentencias

Observa que no exige la sentencia NEXT. Simplemente ejecutará los comandos después de los dos-puntos hasta que la variable X de control del bucle llegue a su valor límite final. Es una forma muy conveniente de condensar un simple bucle en una sola línea de programa.

También aquí, no debiera usarse como un sustituto para desarrollar un bucle FOR completo como líneas separadas, pero es útil efectuar una tarea aislada y completa de características simples. Por ejemplo, puede que quieras un cartel particular rodeado de asteriscos. Esta sería una buena manera de hacerlo:

```
10 CLS
20 FOR I=1 TO 22:PRINT"*";
30 PRINT
40 PRINT"* ESTE ES UN CARTEL *"
50 FOR I=1 TO 22:PRINT"*";
```

Esta clase de manejo de imágenes puede ser aprovechada para hacer que parezcan más interesantes, impresos que en otros casos resultarían aburridos. Además, ayuda a destacar áreas diferentes cuando no quieres usar colores o gráficos.

La forma corta de la sentencia FOR es útil para labores simples que efectúan bucles; mientras que la forma larga que describiremos ahora permite producir algunas de las más complejas estructuras de bucle.

Como vímos antes, la sentencia FOR puede usar NEXT para definir la finalización de un bucle FOR. Podemos usar también las claves END FOR para obtener exactamente la misma indicación de final.

Por ejemplo, los siguientes dos programas tienen efectos idénticos:

```
10 FOR I=1 TO 50
20 PRINT I
30 NEXT I

10 FOR I=1 TO 50
20 PRINT I
30 END FOR I
```

La diferencia entre NEXT y END FOR surge al usar el comando EXIT que presentamos con los bucles REP. Debieras recordar que cuando se ejecuta EXIT, se hace que inmediatamente el programa salga del bucle y pase a ejecutar la instrucción que sigue a la END REP. Eso es lo que diferencia a las sentencias END FOR y NEXT. Si hay una instrucción NEXT y una END FOR dentro de un bucle FOR, la EXIT hará que sea ejecutada la instrucción inmediatamente detrás de la END FOR.

El siguiente programa muestra cómo puede aprovecharse eso:

```
10 CLS
20 FOR COUNT=1 TO 5
30 INPUT"DESEAS QUE PARE?",ANS$
40 IF ANS$="SI" THEN EXIT COUNT
50 NEXT COUNT
60 PRINT"CONTADOR NO PARADO"
70 END FOR COUNT
80 PRINT "PROGRAMA FINALIZA"
```

El bucle FOR tiene dos finalizaciones: una NEXT y una END FOR. El EXIT en la línea 40 sólo usa la END FOR. Si tu respuesta ha sido SI en la línea 30, la premisa encabezada por IF es cierta, y se ejecuta el comando EXIT COUNT. Eso hace que se salga del bucle controlado por COUNT, y por tanto se pase a ejecutar la línea 80, con lo que aparecerá el mensaje PROGRAMA TERMINA. Si respondes con algo distinto a SI, la premisa es falsa y pasa a ejecutarse la línea 50 que indica **otra** ronda por lo que se regresa a la cabecera del bucle en la líne 20. Si continúas respondiendo de esa manera, llegará el momento en que la variable COUNT llegue al valor límite 5, con lo que finalizará el bucle y se pasará a ejecutar la instrucción situada inmediatamente detrás de la NEXT.

Deberás recordar que EXIT hace que se **salga** del bucle, pasándose a ejecutar la instrucción siguiente a la END FOR; y que al finalizar la cuenta del bucle hace que se ejecute la sentencia detrás de la NEXT. Estos hechos te ayudarán a determinar cómo trabajan las diferentes combinaciones de sentencias.

Intenta **canjear** las líneas 50 y 70. Esta vez, cuando contestes SI y se encuentre el EXIT en la línea 40, la instrucción después de la END FOR será la línea 60, con lo que verás aparecer el mensaje. La instrucción NEXT se encuentra a continuación en la línea 70, por lo que se incrementa el valor de la variable del bucle y se pasa a ejecutar **otra** vez la línea 30. Ese proceso se repite luego hasta que se alcanza la cuenta final de 5 o hasta que se responde con algo distinto de SI. Si la cuenta finaliza, se ejecuta la instrucción en la línea 70 que va detrás de la NEXT, y el programa termina con el mensaje de finalización. Si no respondes SI, la instrucción que va detrás de la END FOR y que está ahora en la línea 50 es la que se ejecuta. Se expone el mensaje en la línea 60, y la NEXT de la línea 70 hace que continúe la cuenta. No hay manera de salir del programa hasta que se alcanza el límite del contador.

Normalmente, la NEXT va antes de la END FOR y la serie de instrucciones comprendidas entre ellas es lo que se conoce como **epílogo del bucle.** Sólo se ejecuta si no hacemos que se **salga** del bucle mediante EXIT y dejamos que termine normalmente cuando el contador llega al límite.

Bucles anidados

Todas las estructuras de bucle en el QL está permitido que sean "incrustadas" una dentro de otra, que es lo que denominamos **anidamiento.** Un ejemplo de dos bucles FOR anidados sería:

```
10 CLS
20 FOR OUTER=1 TO 3
30 PRINT"   ***   RONDA EXTERNA   ***"
40   FOR INNER=1 TO 5
50   PRINT "RONDA INTERNA"
60   NEXT INNER
70 NEXT OUTER
```

Examinemos las parejas FOR NEXT que definen los bucles. El bucle **externo** (outer) ha de ejecutarse 3 veces, y en cada una de las rondas hace que el bucle **interno** (inner) sea ejecutado 5 veces. Eso se indica en los mensajes expuestos. El bucle externo comienza con la variable OUTER igual a 1. La línea 30 muestra luego que ha comenzado una ronda de este bucle, y el programa entra en el bucle interno mediante la línea número 40. Este bucle usa INNER como variable contadora, y a medida que efectúa las rondas, expone un mensaje en pantalla. Cuando INNER alcanza el final de su cuenta, se ejecuta la instrucción después de la NEXT INNER, y vuelve a efectuarse otra ronda del bucle externo. El proceso se repite por tanto hasta que ambos contadores de bucle alcanzan sus valores límites.

Podemos incluso anidar más bucles dentro del bucle interno. La regla más importante a observar es que siempre debes encajar totalmente un bucle dentro de otro para que el anidamiento sea correcto. No permitas que ninguna de las instrucciones de un bucle caigan fuera del otro de manera que uno cruce las instrucciones que marcan el comienzo y el final del otro. Por ejemplo, no está permitido intercambiar las líneas 60 y 70 de este ejemplo.

Es bastante normal que se incluyan instrucciones adicionales dentro de los bucles internos y externos. Intenta añadir una instrucción PRINT entre las líneas 60 y 70, y verás que lo expondrá cada vez que el programa sale del bucle interno.

Las instrucciones SI... y la marca de final de la alternativa

La sentencia IF presentada hasta ahora, sólo ha sido usada en su forma corta, por la que si la condición es cierta **entonces** se pasa a ejecutar todos los comandos después de la clave THEN siempre y cuando estén en la misma línea de programa y separados por dos-puntos. La sentencia IF también está disponible en dos formas largas. Una de ellas usa END para permitir que sean ejecutadas varias líneas consecutivas de programa colocadas entre las claves THEN hasta la END IF. La forma general de esta instrucción es pues:

SI condición THEN
instrucciones
END IF

La primera instrucción establece la condición que, de ser TRUE (cierta), hará que **entonces** se ejecuten todas las instrucciones que van detrás de THEN y hasta llegar a la END IF. Si es FALSE (falsa), el programa continúa con la instrucción que venga inmediatamente detrás de la que marca el "FIN SI".

Por ejemplo, veamos el problema de hacer que se ejecuten una o dos rutinas dependiendo de algún resultado. El siguiente programa, o bien hace que se **borre** la pantalla y se **exponga** un mensaje en su centro; o bien hace que se **liste** el propio programa en la pantalla y te informe que ese es el programa "corriente". También comprueba que has seguido las instrucciones correctamente y repite la acción hasta que lo hagas. Observa cómo se ha usado REP para sustituir un GOTO. De hecho, es posible trabajar sin usar GOTO en absoluto a partir de ahora.

```
10 REP ESTE_PROGRAMA
20 INPUT"teclea s o n",A$
30 IF A$="s" OR A$="S" THEN
40 CLS
50 AT 5,10: PRINT "ESTE PROGRAMA
   DEMUESTRA END IF"
60 EXIT ESTE_PROGRAMA
70 END IF
80 IF A$="n" OR A$="N" THEN
90 CLS:LIST
100 AT 0,0:PRINT"ESTE ES EL
    PROGRAMA CORRIENTE"
110 EXIT ESTE_PROGRAMA
120 END IF
130 PRINT"NO HAS SEGUIDO LAS
    INSTRUCCIONES'
140 END REP ESTE_PROGRAMA
150 REM ESTE ES EL FINAL DEL
    PROGRAMA
```

Es un ejemplo de anidamiento. Decimos que las instrucciones IF están **anidadas** dentro de un bucle, marcado por las instrucciones REP y END REP. Para leer este programa correctamente examina las instrucciones REP y END REP para ver el **ámbito** del bucle principal. Luego encuentra las instrucciones IF y END IF para obtener alguna idea de la estructura modular del programa dentro del bucle. Cuando las hayas localizado, analiza cada una separadamente. Ese es el enfoque normal para leer programas estructurados.

Todo entre la línea 30 y la línea 70 será ejecutado por venir después de la clave THEN si la condición es cierta. Lo mismo ocurre para las líneas entre la 80 y la 120 para la segunda estructura IF. Si ninguna de estas cláusulas THEN es ejecutada, significa que el usuario del programa ha tecleado una letra no válida, y el programa vuelve a dar otra ronda y a repetir las acciones hasta que uno de dichos bloques THEN sea ejecutado. Cada cláusula THEN contiene un comando EXIT. Ensaya con el programa, pulsando teclas válidas y no válidas para ver el efecto.

EN las DEMAS situaciones

La instrucción IF tiene otra forma larga que aprovecha la palabra clave ELSE. Da a la instrucción IF más potencia para elegir entre dos alternativas. La estructura completa es de la forma:

IF condición THEN	SI condición cierta ENTONCES
instrucciones	instrucciones
ELSE	EN DEMAS (cuando no es cierta)
instrucciones	instrucciones
END IF	FIN SI

Como el equivalente hispano de eso implica, las instrucciones entre THEN y ELSE se ejecutan si la condición es cierta, y luego la ejecución pega un salto hasta la línea que va detrás de la que marca el "FIN SI". Si la condición es falsa, son las instrucciones que van detrás de la clave ELSE las que se ejecutan, y a continuación se prosigue con las líneas que van detrás de la que marca el END IF.

El siguiente es un ejemplo de esta estructura:

```
10 REP COMPROBAR_LETRA
20 INPUT"TECLEA LA LETRA K";A$
30 IF A$="K" THEN
40 CLS
50 PRINT"LETRA CORRECTA"
60 EXIT COMPROBAR_LETRA
70 ELSE
80 PRINT"LETRA ERRONEA"
90 END IF
100 END REP COMPROBAR_LETRA
```

También aquí cuando veas la REP a la cabecera del programa, busca la instrucción END REP que delimita el ámbito del bucle. Luego mira la instrucción IF y busca la END IF y/o la ELSE. Eso te ayudará a apreciar los módulos que están incluidos en el programa, y de ahí su estructura.

Si la condición después de IF es cierta, se ejecutan las líneas 40 y 60 que hace que se salga del bucle. Si la condición es falsa, se ejecutan las instrucciones que forman la cláusula ELSE y luego viene la marca de END IF, que no hace que finalice el bucle. Por lo tanto, se ejecuta una y otra vez hasta que se haya tecleado la letra correcta.

Debieras tomar un descanso en este momento, y ensayar un ejercicio. Mira a ver si puedes volver a escribir el programa del tiempo de reacción presentado antes de una nueva forma usando el enfoque más estructural descrito anteriormente. En particular, intenta quitar todas las instrucciones GOTO y sustituirlas con instrucciones REP.

Sentencias condicionales anidadas

El uso de las instrucciones IF puede ampliarse aún más mediante el **anidamiento.** Eso da algunas de las más complejas estructuras en SuperBASIC y te mostramos un ejemplo que ha de ser examinado muy cuidadosamente:

```
10 CLS
20 PRINT"TECLEA 3 NUMEROS DIFERENTES."
30 INPUT"CONCLUYE CADA UNO CON ENTER",A,B,C
40 IF A>B THEN
50   IF B>C THEN
60   PRINT
70   PRINT "ORDEN DESCENDENTE"
80   ELSE
90   PRINT"ORDEN MEZCLADO"
```

```
100  END IF
110 ELSE
120  IF C>B THEN
130  PRINT "ORDEN ASCENDENTE"
140  ELSE
150  PRINT"ORDEN MEZCLADO"
160 END IF
```

Los **sangrados** (indentados, decalados) en las líneas anteriores los hacemos para ayudarte a seguir la lógica de este programa. Cuando lo ejecutes, sigue las instrucciones exactamente: no teclees dos o tres números iguales. Sin embargo, ensaya números según todas las clases diferentes de ordenamiento.

La instrucción IF en la línea 40 tiene su ELSE en la línea 110 y su END IF en la 160. Tiene dos instrucciones separadas IF anidadas dentro de ella. La primera de ella comprueba si B > C cuando se ha verificado que A > B por la instrucción IF **externa.** Si B > C, debes convencerte a ti mismo que los números están ineludiblemente en orden descendente. Si no es así, el orden es un puro desorden.

Si B > A, no se ejecuta la cláusula THEN comenzada en la línea 40, y el programa salta al ELSE de la línea 110. Si ahora es C > B, es que los números están en orden ascendente. En el caso contrario el orden es irregular.

Observa que al igual que para las instrucciones FOR, los programas no funcionan adecuadamenta a no ser que el anidamiento sea completo y que no haya "solape" de ámbitos. No permitas que parte de un bloque IF se cruce con otro. Puedes anidar instrucciones IF hasta el nivel de profundidad que desees, siempre y cuando cada uno esté completamente incluido dentro del más externo.

A menudo es mejor, desde el punto de vista de la escritura de un programa, comprobar todas las posibilidades de un conjunto dado de condiciones mediante instrucciones IF separadas, en lugar de usar la estructura anidada presentada. Esto ayudará a hacer el programa más legible y mejor estructurado. Sin embargo, algunos programas muy pulcros y condensados pueden escribirse mediante el anidamiento de las instrucciones IF. Deberás prestar cuidadosa atención a la manera exacta en que se enlazan las cláusulas THEN y ELSE y la marca END IF con la instrucción IF correspondiente. Es fácil para el programa -perdón, para el programador- estar confundido.

Bucles selectivos

La última serie de palabras clave de este capítulo son las que conciernen a la ejecución de un conjunto de bloques en un programa, dependiendo de los diversos resultados de una **decisión.** Se habla entonces mejor de **selección,** y las palabras normales del BASIC para esta operación selectiva son las de ON variable GOSUB... El QL admite estas claves por razones de compatibilidad con otros lenguajes, pero dispone de otras que son más potentes y más estructuradas. Estas son las instrucciones SELeccione (SELECT) y la normal END SEL que le acompaña. La primera sección describe la instrucción ON numeral GOTO número de línea.

SEGUN numeral VAYA...

Cuando se ejecuta GOTO, conmuta el tratamiento para que pase a ejecutarse una determinada línea del programa. ON GOTO permite que dicho **selector** se dirija hacia uno de una serie de números de línea, dependiendo del valor de una expresión numérica. La forma general de la sentencia es:

ON expresión numérica GOTO a,b,c,...

siendo a,b,c,... **números de línea** que están separados entre sí por comas. La expresión numérica debe tener un valor entre 1 y la cantidad de números de línea que aparecen en la lista después de la clave GOTO. El valor de la expresión numérica determina el "puesto" en la lista, contando a partir de la izquierda y hacia la derecha, del número de línea que corresponde ser ejecutado. Por ejemplo:

ON 3 GOTO 400,50,55,4000,500

hará que vaya a la línea número 55 según el valor 3. Esta es una manera muy complicada de decir vaya a la 55, pero ilustra el tema. Un ejemplo más práctico pudiera ser:

ON N GOTO 20,50,100,500,2000

Ese es el método de transferir la ejecución a una determinada línea del programa **según** el valor que tenga la variable N. Esta instrucción puede estar incluida en un bucle o puede ser un método para efectuar una entre 5 acciones posibles, por ejemplo para elegir entre lo que llamamos un **menú** que pide al usuario teclee un número según la labor que desea sea ejecutada por el programa. Veremos un simple ejemplo de esto en la siguiente sección.

Es importante que la expresión numérica tenga su valor dentro de los límites válidos, ya que en los demás casos dará un error. No tiene que ser un entero, ya que automáticamente será redondeado.

Estructura selectiva entre varias opciones

El método más estructurado de efectuar una labor **selectiva** como la presentada en la sección anterior es mediante la instrucción SEL. En común con las otras estructuras en SuperBASIC dispone de una forma corta y de una larga, usándola ésta última la marca END SEL para indicar el final de esta labor.

La forma breve es como sigue:

SEL ON X = A TO B: instrucciones

Las instrucciones después de los dos-puntos sólo se efectúan si X está dentro de la banda A al B, ambos inclusive.

La forma corta es muy útil para sustituir una forma bastante complicada de la instrucción IF. Por ejemplo, en un ejemplo anterior teníamos que comprobar si un dato introducido por el usuario estaba dentro de una cierta banda. Si el dato se imponía en la variable X y las cotas válidas eran 100 y 200, inclusive, escribíamos:

IF 100 < =X AND X< =200 THEN sentencias

Eso puede escribirse más fácilmente usando la instrucción SEL tal y como sigue:

SEL ON X=100 TO 200: sentencias

Los comandos, todos en la misma línea de programa separados por dos-puntos, se ejecutan sólo si X está dentro de la banda 100 a 200, inclusive. Los valores de X se redondean a enteros, y eso significa que por ejemplo, 99.5 y 200.49 se consideran como dentro de la banda. Similarmente, el ordenador redondea las cotas cuando no son enteras. Por ejemplo, la siguiente es idéntica a la anterior:

SEL ON X=99.6 TO 200.39: sentencias

Aunque realmente sólo debiera usarse con enteros.

La forma larga de **selección** se usa para ejecutar **uno entre varios** bloques, dependiendo -según- el valor de una variable al igual que ON GOTO. En este caso, la primera línea del bloque selectivo es de la forma:

SEL ON variable

Esto establece la variable que será usada en las siguientes líneas de selección en la forma:

```
ON variable = A TO B
instrucciones
ON variable = C TO D
instrucciones
ON variable = REMAINDER          = RESTO
instrucciones
END SEL
```

Eso constituye una **lista de selección.** El bloque de instrucciones después de la primera ON se ejecuta sólo si la variable está dentro de la variable inclusive acotada por A y B, que deben ser expresiones numéricas. Si se ejecuta este bloque de instrucciones, el programa salta directamente a la línea que va detrás de la que marca END SEL. Cada bloque de instrucciones se termina o bien por venir la siguiente instrucción ON, o bien por la instrucción END SEL. El último miembro de la lista de selección usada antes, emplea la palabra clave opcional REMAINDER, para indicar que si ninguna de las condiciones previas se ha cumplido, es este otro bloque de instrucciones el que se ejecuta, cuando así lo requieras.

Un ejemplo de esta clase de estructura sería el de permitir al usuario que tecleara un número para elegir una opción particular de un menú que aparece en pantalla. Como un caso simple de eso, considera un programa que exponga uno entre tres nombres dependiendo de cuál es el número elegido en el menú.

Incluso aunque es un poquito largo, funciona perfectamente y es fácil de leer su estructura teniendo en cuenta la instrucción SELeccione SEGUN.

```
10 CLS
20 REP PROG
30 REM - EXPONER MENU
40 PRINT"ESCOJA UN NUMERO SEGUN"
50 PRINT\\1) PRINT "MARIA"
60 PRINT\\2) PRINT "JORGE"
70 PRINT\\3) PRINT "JUAN"
80 PRINT\"(99) SALGA PROGRAMA"
90 PRINT
100 REM - REQUERIR OPCION
110 INPUT"NUMERO?",CHOICE
120 PRINT:PRINT
130 REM - SELECCIONAR NOMBRE O SALIDA
140 SEL ON CHOICE
150 ON CHOICE=1
160 PRINT"NOMBRE ESCOGIDO ES MARIA"
170 ON CHOICE=2
180 PRINT"NOMBRE ESCOGIDO ES JORGE"
190 ON CHOICE=3
200 PRINT"NOMBRE ESCOGIDO ES JUAN"
210 ON CHOICE=99
220 PRINT"GRACIAS - ADIOS!"
230 REM - ELEGIDO SALIDA DE PROGRAMA
240 EXIT PROG
250 REM - ATRAPAR ERRORES
260 ON CHOICE=REMAINDER
270 PRINT"ESCOJA DE NUEVO!!"
280 GOTO 110
290 END SEL
300 REM - SUBIR IMAGEN
310 PRINT\\\\
320 END REP PROG
```

Las líneas 50 a 70 usan apóstrofes (') para delimitar los literales porque las comillas (") forman parte de los mensajes a exponer.

Observa que este programa da un ejemplo de una estructura selectiva anidada dentro de una estructura repetitiva. El anidamiento de bloques de esta manera es lo suficientemente natural como para que no exija pensamientos especiales. Observa también que hay tendencia a incluir una instrucción GOTO 110 como línea 280: es un buen ejercicio que analices la necesidad o no necesidad de incluirla, o de modificar la estructura repetitiva. Las instrucciones REM incluidas en el ejemplo te permiten repasar el programa con el cuidado que precises.

Puedes ver cómo el uso de la opción REMAINDER -para elegir en el **resto** de los casos- simplifica el problema de "atrapar" los datos tecleados incorrectamente. Compara eso con el mismo problema en el ejemplo anterior que usaba instrucciones condicionales IF. Imagina además cómo podrías describir la labor efectuada por este programa usando esas instrucciones.

Aprenderías un montón volviendo a escribir el programa usando instrucciones condicionales y saltos GOTO para llevarla a cabo. Si así lo haces, debieras ensayar a estructurar el programa escribiendo cada tarea como un módulo separado colocado al final del programa y elegido mediante SI... VAYA...

Resumen

- FOR STEP... NEXT se utiliza para establecer bucles **preconfinados**, que han de ser repetidos un número determinado de veces. END FOR puede usarse para terminar el bucle FOR inmediatamente que se encuentre el comando EXIT. Un FOR corto puede colocarse en una sola línea, seguida de una serie de sentencias. No se exige NEXT en ese caso.

- IF THEN se usa para ejecutar un bloque de sentencias dependiendo de la certeza de una condición lógica. Un bloque de sentencias puede terminarse por un END IF. ELSE permite que un bloque sea ejecutado si la condición IF es falsa.

- Las sentencias Booleanas establecen condiciones lógicas, cuyo valor puede ser cierto (TRUE), arrojando un valor distinto de 0; o FALSE, representado por 0. Las conjunciones AND, OR y XOR, y la negación NOT son operadores usados para construir condiciones Booleanas compuestas.

- REP se usa para presentar bucles que terminan con END REP, y de los que puede salirse usando EXIT.

- STOP, CONTINUE y RETRY se usan para hacer que pare, y luego siga bien con la siguiente instrucción, o bien **reintentando** la última que se ejecutó.

- Incluir bucles e instrucciones condicionales una dentro de otra, se denomina **anidamiento.** El ámbito de un anidamiento no puede "solaparse" con el ámbito de otro anidamiento externo.

- SEL, END SEL y REMAINDER se usan para establecer una **selección** entre una serie de bloques de instrucciones **según** el valor de una variable determinada.

Capítulo Seis

Series y Tablas

Este capítulo trata dos conceptos que previamente han sido introducidos **series** (strings) y **tablas** (arrays).

Los datos **literales** son aquellos que el ordenador considera y trata como meras "cadenas de caracteres" y podemos operar con ellos usando funciones especiales y sentencias como las descritas en este capítulo. También pueden organizarse en forma de datos **colectivos,** al igual que los numerales que vímos en el Capítulo 2, y este capítulo define la manera en que estas **variables múltiples,** también llamadas **subindicadas,** se definen y tratan en SuperBASIC.

Este capítulo presenta además un **código** numérico que es usado para todos y cada uno de los **caracteres** del repertorio, y muestra cómo comparar literales usando las mismas relaciones lógicas que ya vímos para numerales.

Ordenamiento y almacenamiento de literales

Todo en un ordenador se almacena numéricamente. Hemos visto que incluso la certeza de una sentencia Booleana es almacenada numéricamente, i.e., TRUE (CIERTO) se almacena como un número distinto de cero, y FALSE como cero. Lo mismo ocurre con los caracteres reflejados en las teclas del teclado. Cada carácter se almacena como un número de acuerdo con un convenio internacional denominado ASCII. Son las siglas de American Standard Code for Information Interchange.

Por tanto, un literal -una cadena de caracteres- queda almacenada como una secuencia de números, en que cada uno se corresponde biunívocamente con cada uno de los caracteres del literal. Esa es una de las razones por las que tienes que identificar una celdilla de memoria por un identificador que automáticamente designe si corresponde o no a un literal. Si no lo hicieramos así, los literales constituidos por cifras se confundirían con los números que también están constituidos por cifras pero que tienen una naturaleza o índole completamente distinta.

La tabla de **códigos** ASCII está dada en el manual del QL, y deberás consultarla en relación con lo siguiente.

Para ver cómo funciona la numeración de los caracteres, se dispone de dos funciones que te permiten mostrar el número de código que realmente corresponde a cada carácter, y viceversa. Esas funciones se denominan CODE y CHR$ (abreviatura de character).

CODE acepta un literal como argumento y entrega el código ASCII que corresponde al primer carácter de dicho literal. Por ejemplo, teclea:

PRINT CODE("A")

Eso devolverá el código ASCII de número 65. Ahora ensaya:

PRINT CODE("ABCD")

Eso dará la misma respuesta, ya que sólo el código del primer carácter del literal es el que se expone. Para ver los códigos de los otros, puedes teclear:

PRINT CODE("B),CODE("C"),CODE("D")

Eso te mostrará que dichos caracteres tienen códigos que están en orden ascendente. Para convertir un código numérico a su carácter correspondiente, puedes usar la función CHR$. El signo $ en ese identificador significa que la función tiene un valor literal. Ensaya tecleando:

PRINT CHR$(65)

y verás que se expone la letra A, confirmando que el número 65 es el código ASCII para la letra A en **caja** mayúscula. Puedes usar esta función para descubrir el código ASCII de cada carácter. Debieras usar CLS, y luego teclear el siguiente comando, analizando cuidadosamente el resultado.

FOR I=32 TO 127:PRINT CHR$(I);

Simplemente expone el **repertorio** completo de caracteres que tu QL puede mostrar.

Como puedes ver, los números de código van desde el 32 hasta el 127. Otros números de código producen caracteres **no-visivos** que son empleados por el sistema para otros menesteres. Debieras ser capaz de ver que hay un **espacio** en blanco al comienzo del conjunto expuesto: es el primer carácter del repertorio y tiene el 32 como código ASCII. El resto de los caracteres comienza con el signo de exclamación; luego pasa a través de otros signos y de los dígitos del 0 al 9. Vienen luego algunos símbolos más, seguidos de las letras en caja mayúscula. Más símbolos van seguidos de las letras en caja minúscula, y finalmente los últimos pocos símbolos. El repertorio termina con el número 127 que es el signo de derechos reservados (copyright) que no es standard y sí especial del QL.

En la mayoría de los BASIC este código numérico es el usado para ordenar los datos literales usando las relaciones ya conocidas de:

> >= < <=

Normalmente, si un carácter tiene un código ASCII inferior al de un segundo carácter, el primero se define como menor que el segundo, usando la relación <. Esta clase de comparación se toma como base en la labor de ordenamiento de listas de palabras alfabéticamente. Al igual que en una guía telefónica, las palabras se ordenan normalmente por la inicial, luego por la segunda letra, y así sucesivamente.

Sin embargo, eso causa confusión si se usa el código ASCII, porque -si miras al conjunto de caracteres en la pantalla- las letras en caja minúscula son todas posteriores en el repertorio a las letras en caja mayúscula. Eso significa que una lista almacenada de acuerdo con el código ASCII, pondría la palabra "manzana" después de la "PERA", dado que el código ASCII para la "P" es inferior a la de la "m".

El QL resuelve este problema definiendo el ordenamiento de los caracteres literales, bastante diferente a lo que corresponde según el código ASCII. Si usas cualquiera de las cuatro relaciones standard, el ordenamiento real de las letras es el siguiente:

AaBbCcDdEeFfGgHh, etc.

Eso arregla la situación y permite el ordenamiento alfabético apropiado sin tener en cuenta la **caja** mayúscula o minúscula. Si quieres ordenar de acuerdo con el código ASCII, puedes usar la función CODE para convertir los caracteres, efectuar la labor de ordenamiento, y luego volver a reconvertirlos usando la función CHR$.

El ordenamiento real de los caracteres de acuerdo con este nuevo método de comparación no está establecido claramente en las primeras versiones del manual QL. Veremos en breve, con un ejemplo, el repertorio correctamente ordenado y cómo se relaciona con el ordenamiento ASCII.

Así se explica cómo los cuatro operadores de relación standard anteriores operan sobre los literales, pero hay otros tres operadores más para literales, que son:

```
=       <>      ==
```

El primero es el igual normal, y los dos literales deben ser idénticos en todo, incluyendo la caja -mayúscula o minúscula- para satisfacer esa relación. El segundo es el no igual, o distinto de, y cualquier diferencia satisfará esa relación, incluyendo una diferencia de caja. El tercero se llama **equivalente** o bien **casi igual.** Dos literales son equivalentes si tienen las mismas letras y símbolos, pero una o más de las letras son de diferente caja. Para comprobar ejemplos por ti mismo, simplemente expón el resultado de la relación. Un resultado cero significa que es **falsa,** un resultado distinto de cero significa **cierta.** Consulta de nuevo las relaciones Booleanas si no estás seguro de este aspecto. Los siguientes son ejemplos de estas relaciones:

```
"ABC"   =    "aBC"    es FALSE
"Pepe"  ==   "PEPE"   es TRUE
"Pepe"  ==   "Pepa"   es FALSE
"PEPE"  <>   "pepe"   es TRUE
```

Con el fin de operar sobre literales -series de caracteres- es importante ser capaz de almacenarlos de manera consecutiva numéricamente. Eso se presentó en el Capítulo 2, que debieras consultar ahora para recordar la sección que trata de tablas. Te presentamos ahora las tablas tanto para literales como para numerales.

Tablas (Arrays)

Una tabla es una **colección de variables** en la que puede señalarse cada elemento de la colección mediante **índices,** que son numerales normalmente a partir de uno o de cero y hasta un número máximo que depende de cuántos valores componen la tabla.

Supongamos que deseas almacenar 100 valores numéricos como una **variable múltiple.** Primero debes elegir un identificador para esa colección. Usaremos VALUE en este ejemplo. Este identificador no puede usarse nada más que para representar los valores que lo componen, y para ello añadimos como **sufijo** un número del 1 al 100 para señalar cada uno de los 100 valores que deseamos almacenar. Ese sufijo se adosa al identificador usando paréntesis. Así las 100 variables que componen la colección, una para cada valor almacenado, se llamarían:

VALUE(1), VALUE(2), ..., VALUE(100)

Los sufijos numerales entre paréntesis pueden venir dados por el contenido de una variable o por una expresión numérica, y cada uno de los elementos de la tabla anterior se trata justamente como cualquier identificador normal tales como X o CUENTA. Por ejemplo, PRINT VALUE(65) hará que se exponga el valor correspondiente a la 65-ésima variable de la tabla.

Con el fin de usar cualquier tabla, hemos de hacer que la máquina **ocupe** el espacio correspondiente de la memoria. La mayoría de los dialectos de BASIC tienen una sentencia que efectúa esa labor. Significa que tienes que saber cuántos valores vas a almacenar, o dar el máximo de los que puedas necesitar. No debes usar un número demasiado alto, ya que eso ocuparía demasiado espacio en memoria y puede restringir el reservado para el propio programa.

La sentencia DIM (abreviatura de dimensión) es la que se usa para reservar el espacio necesario para una tabla. Su forma general es:

DIM identificador de tabla (número de elementos)

Esta sentencia define el **índice** máximo que puede tener un elemento de la tabla, comenzándolos a numerar a partir de cero. Así por ejemplo:

DIM VALUE(100)

permitiría que la tabla VALUE estuviera compuesta por los elementos:

VALUE(0), VALUE(1),..., VALUE(100)

que realmente son **101 elementos,** pero es por ahora conveniente olvidar el elemento cuyo sufijo es **cero.** Es útil en algunas aplicaciones.

Si olvidas usar DIM en un programa, se provocará un error a menos que tú mismo te restrinjas a índices del 0 al 10. Sin embargo, también con esos es buena práctica usar DIM para mayor claridad y así lo haremos en este libro. También se producirá un error si el sufijo índice del elemento sobrepasa el máximo especificado en una sentencia DIM.

Veamos ahora cómo rellenar los valores de una tabla, y cómo exponerlos usando un bucle. Mostraremos dos de los muchos métodos de asignar los valores de una tabla:

```
10 DIM VALUE(100)
20 FOR I=1 TO 100
30 VALUE(I)=I*I
40 NEXT I
```

```
10 DIM VALUE(100)
20 FOR I=1 TO 100
30 INPUT"TECLEA UN NUMERO",VALUE(1)
40 IF VALUE(I) = -1 THEN EXIT I
50 END FOR I
```

El primero de ellos emplea asignaciones para rellenar la tabla. En el segundo se ingresan los números desde el teclado, y permite que se detenga el bucle cuando se teclea -1, si no se quiere rellenar toda la tabla. Una sentencia FOR puede usarse para exponer los valores de los elementos de la tabla.

```
FOR I=1 TO 100:PRINT VALUE(I);
```

Esto sólo puedes hacerlo sin la sentencia DIM, porque justamente acabas de ejecutar un programa que dimensiona la tabla. La instrucción DIM también **anula** los valores de la tabla. Ensaya usando DIM VALUE(100) e inmediatamente exponiendo los valores de la tabla como antes: verás que todos los elementos están puestos a 0.

Las tablas pueden tener **más de un subíndice,** y la sentencia DIM debe dar el valor máximo para cada uno de ellos. Son las tablas llamadas multidimensionales, y mostramos un ejemplo con una de 3 dimensiones.

```
10 DIM A(12,34,20)
20 FOR DIR1=1 TO 12
30 FOR DIM2=1 TO 34
40 FOR DIM3=1 TO 20
50 A(DIM1,DIM2,DIM3)
   +DIM1+DIM2+DIM3
60 NEXT DIM3
70 NEXT DIM2
80 NEXT DIM1
```

La primera instrucción hace que se **ocupe** el espacio para una tabla que tenga 12 como valor máximo en la primera dimensión, 34 en la segunda y 20 en la tercera. El programa usa luego 3 bucles FOR anidados para rellenar la tabla con valores en que cada uno es la suma de los índices que designan a ese elemento. Encontrarás que los bucles anidados son estructuras muy comunes para la operación con tablas multidimensionales.

La función de clave DIMN está disponible para constatar los valores dados para las dimensiones de una tabla previamente dimensionada. Puede ser útil en un programa para comprobar que no se va a sobrepasar el máximo valor en ninguna de las dimensiones. DIMN tiene la siguiente forma:

DIMN (identificador de tabla, dimensión-ésima)

Por ejemplo, si el programa desea comprobar el valor de la segunda dimensión de la tabla A, evaluaríamos la función:

DIMN(A,2)

y en el caso anterior el valor resultante sería 34. Si el número correspondiente a la dimensión queda fuera del argumento, está prescrito que se adopte para omisiones la primera de las dimensiones. Por lo tanto, la función DIMN(A) devolvería en este caso el número 12.

También pueden almacenarse datos literales en tablas cuyos identificadores terminen con el signo $. La única diferencia está en la sentencia DIM, que debe tener como parámetro un número extra que le diga al ordenador el número máximo de caracteres que va a formar cada uno de los elementos literales. Este número extra es siempre el correspondiente a la última dimensión dada. La función DIMN actúa sobre tablas literales exactamente igual que para las numerales. La última dimensión corresponde a la **longitud** del literal y es también accesible usando DIMN, pero deberás comprobar en tu máquina lo que sucede, porque puedes encontrarte con que añade uno al valor correcto para dimensiones comprendidas entre 1 y 9.

El siguiente ejemplo rellena una tabla literal con 20 elementos, y cada uno de ellos con hasta 100 caracteres.

```
10 DIM NAMADD$(20,200)
20 FOR K=1 TO 20
30 PRINT"TECLEA NOMBRE Y DIRECCION"
40 INPUT"100 CARACTERES MAXIMO:"!NAMADD$(K)
50 NEXT K
```

El programa hace que primero se **ocupe** espacio para una tabla literal llamada NAMADD$; de manera que pueda contenerse en él 20 elementos, cada uno con una longitud máxima de 100 caracteres. Luego se ejecuta un bucle FOR para imponer los valores de los 20 nombres y direcciones. Si ingresas más de 100 caracteres para cualquiera de ellos, el programa simplemente almacena los primeros 100, y **desecha** cualquier otro extra. Si se introducen menos, hará que el dato literal sea rellenado con espacios en blanco después de los caracteres que tú hayas introducido.

Como ejemplo de las ideas de este capítulo y para practicar algunos de los conceptos de capítulos anteriores, podemos volver ahora al problema de ordenar literales usando comparaciones.

Un ejemplo de ordenamiento de literales

Esto proporciona un modelo excelente de la utilización de bucles anidados FOR y de tablas, así como de un programa general para efectuar el llamado **ordenamiento por burbujas** que se mencionó en el Capítulo 2. Debes consultar el diagrama de flujo sobre ordenamiento de números en orden ascendente, del Capítulo 2. Haremos ahora eso para los 96 códigos ASCII, y los ordenaremos de acuerdo con la clase que usa el QL.

Para ver cómo funciona, piensa que la primera parte de la rutina de ordenamiento de 3 números, simplemente 'burbujeaba' el número más grande hacia el extremo derecho del conjunto. En otras palabras, si los números eran:

6 1 3

el primer paso era intercambiar el 6 con el 1, y luego con el 3 para obtener:

1 3 6

El siguiente paso era dejar el 6 solo, ya que había adoptado su posición correcta y trabajar con los otros efectuando la misma operación de nuevo. Esta vez, no se necesita hacer el **canje** ("swop" en inglés) y el proceso está completo.

Si tienes más de tres números, el proceso es idéntico pero se necesitan más pasadas. La primera labor es bombear el número más grande que haya en el conjunto hasta la parte superior. Por ejemplo, observa el conjunto de los siguientes 6 números:

1, 6, 2, 34, 0, 4

En la primera pasada, no se canjean el 1 y el 6 porque el 1 es menor que el 6. Sí se canjean el 6 y el 2, y los números se convierten en:

1, 2, 6, 34, 0, 4

Luego no se canjean el 6 y el 34, pero sí el 34 y el 0; y luego el 34 y el 4. Con eso se consigue:

1, 2, 6, 0, 4, 34

y ya se puede olvidar el 34 porque ha llegado a su destino final. Los 5 números de la izquierda son los que se tratan ahora de la misma manera, consiguiendo que su número más grande -el 6- suba como una burbuja hasta colocarse por debajo del 34, luego el 4 que ya está en su sitio y permanece; y así sucesivamente. Hay dos bucles en proceso en esta labor. Uno de ellos mantiene la cuenta del tamaño del conjunto de números que hay a la izquierda; y el otro es un bucle que opera sobre este conjunto reducido bombeando el número más grande hasta la parte superior. El siguiente programa ordenará todos los 96 caracteres según el orden ascendente, de acuerdo con las comparaciones >:

```
10 CLS:DIM A$(96,1)
20 REM - PONE LOS CARACTERES EN A$
30 FOR I=1 TO 96:A$(I)=CHR$(I+31)
40 REM - BUCLE EXTERNO
50 FOR COUNT=1 TO 96
60 REM - BUCLE INTERNO
70 FOR SWOP=1 TO 96-COUNT
80 IF A$(SWOP)>A$(SWOP+1) THEN
90 REM - CANJE DE POSICIONES
100 D$=A$(SWOP)
110 A$(SWOP)=A$(SWOP+1)
120 A$(SWOP+1)=D$
130 END IF
```

```
140 END FOR SWOP
150 END FOR COUNT
```

La primera línea borra la pantalla y ocupa el espacio para la tabla A$ que tiene 96 elementos de un carácter cada uno. La línea 30 deposita los 96 caracteres en la tabla A$ usando un bucle corto FOR y la función CHR$ con argumentos desde 32 hasta 127. Luego comienza el bucle externo y cuenta del 1 al 96 a medida que se ejecuta.

Para la primera de las rondas, COUNT es igual a 1, y el bucle interno cuenta del 1 al 95, en la línea 70. Por cada una de estas 95 pasadas, el procedimiento de burbuja funciona para mover el carácter más alto hasta la posición superior haciendo canjes cuando es necesario. Para este valor de COUNT que es 1, la última comparación en la línea 80 se efectúa entre A$(95) y A$(96). Eso muestra que se ha cubierto el conjunto completo de 96 caracteres, almacenado en A$(1) hasta A$(96).

El bucle interno finaliza en la línea 140 cuando se haya efectuado la comparación final y el posible canje necesario. El bucle externo luego incrementa su contador a 2, y sólo están involucrados esta vez en el bucle interno desde el dato A$(1) hasta el A$(95). El proceso continúa hasta que todo el ordenamiento sea completo.

No te alarmes ante el tiempo que tarda en ejecutarse el programa. Tiene 96 elementos para bombear hacia el final. La primera pasada del bucle externo causa que el bucle interno examine todos los 96 caracteres, y lleve el mayor de ellos hasta lo más alto. Luego el bucle interno examina los restantes 95 caracteres; luego los 94; y así hasta acabar. Esa da un total de:

96 + 95 + 94 + ... + 1

rondas del bucle interno. Y ya sabes que eso es... bastante.

Mira a ver si puedes teclear un bucle FOR de ejecución inmediata que te dé la respuesta a esa suma, y te diga por tanto cuántos bucles se efectúan. Tienes que comenzar por hacer una variable igual a 0, y usando la variable dentro del bucle para **acumular** las sumas parciales. Suerte.

Las líneas de programa que te mostramos ahora, expondrán los 96 caracteres de dos maneras diferentes. Primero, en el orden especial de los códigos de carácter producidos por el programa anterior. En segundo lugar, el mismo repertorio del código ASCII para que puedas observar la diferencia. Si ya has hecho que el programa anterior se ejecute, la tabla A$ estará ordenada. Así que teclea sólo las siguientes líneas, usa CLS para borrar pantalla y teclea RUN 160 para ir directamente a esta sección del programa.

```
160 REM - AHORA EXPONE LOS DOS CONJUNTOS
170 PRINT "EL ORDEN EN COMPARACIONES ES:"
180 PRINT
190 FOR I=1 TO 96:PRINT A(I);
200 PRINT:PRINT
210 PRINT"EL ORDEN EN CODIGO ASCII ES:"
220 PRINT
230 FOR I=32 TO 127:PRINT CHR$(I);
```

Verás que en lo que concierne a las comparaciones , el signo **punto** es el carácter de orden inferior, mientras que en el código ASCII es el **espacio** en blanco el que está en el fondo del repertorio. Puedes ver también que todos los números están por debajo de todas letras en ambos casos, y que siguen el orden numérico correcto. Observarás que las letras de caja mayúscula y minúscula están **intercaladas** en el caso de la comparación, como uno podía esperar de observar la guía telefónica.

Operando con literales

Para ser capaz de **segregar** o de **sustituir** parte de un literal, usamos las sentencias disponibles en el SuperBASIC para el 'troceado de literales', con los que sacamos **lonchas** ("slices"). Eso te permitirá por ejemplo extraer sólo el mes de DATE$. Hasta ahora estaba incrustado dentro del literal y sólo era accesible para exponerlo junto con todo lo demás que hay en **fecha$**. Otra aplicación sería hacer que la imposición de datos por teclado fuera más inteligente. Cuando se le pide al operador que teclee SI o NO, podríamos simplemente extraer la primera letra de la respuesta y siempre y cuando fuera correcta, hacer que prosiguiera el programa sin tener que pedirle una respuesta más exacta a una pregunta tan simple.

El "troceado de literales" como se denomina en SuperBASIC, se efectúa de una manera muy diferente a como se hace en los otros BASIC, y cualquiera acostumbrado a ellas, verá las ventajas del método del QL bastante rápidamente.

La reserva de lugar en la memoria para los literales en un sistema de cómputo, puede tratarse de dos maneras distintas. Muchas máquinas simplemente permiten que se aparte una cantidad **prefijada** de celdillas de memoria cada vez que el ordenador se encuentra con un identificador literal en un programa. Eso bien es un desperdicio de espacio, particularmente usando tablas; o bien es insuficiente para literales realmente largos. El QL sin embargo, permite apartar una cantidad de espacio **adecuada** a la longitud de cada literal. Cuando el programa se encuentra con una sentencia de asignación para un dato literal, sólo reserva el espacio suficiente para poder cumplimentar esa asignación. Si posteriormente tropieza con una asignación más larga, incrementará el tamaño de acuerdo con ello. La reserva de espacio no se reduce sin embargo cuando se encuentra con un dato de menor longitud.

Este "dinamismo" en la reserva de espacio en memoria no es efectivo para tablas de literales, y es por eso por lo que debes especificar la longitud de los literales que componen la tabla como última dimensión dada en la sentencia DIM. Por tanto, todos los miembros de una variable múltiple literal han de ocupar la misma longitud de espacio en memoria.

Este reparto del espacio en memoria para literales es muy importante para tratar correctamente con ellos, y poder segregar porciones del mismo, i.e. sub-literales.

Para describir el troceado de literales adoptaremos el siguiente ejemplo y mostraremos cómo extraer y cambiar partes de él. El escogido viene dado por una sentencia de asignación que deberás teclear después de usar NEW y CLS.

es la siguiente:

```
TEST$ = "HOLAPANCHOLOPEZ"
```

Con eso apartamos 15 celdillas consecutivas de memoria que quedan identificadas con el nombre TEST$ y almacenamos en ella **literalmente,** letra a letra, las mostradas. Las posiciones en el literal se consideran numeradas de izquierda a derecha. Puedes segregar algunas de ellas si especificas las **cotas** inicial y final, entre paréntesis y después del nombre del literal. Por ejemplo, ensaya lo siguiente:

```
PRINT TEST$(5 TO 10)
```

Eso expondrá el **subliteral** formado por los caracteres del 5º al 10º en TEST$. El literal TEST$ no se ve afectado por esta acción, como puedes ver si sacas su contenido. Esta es la técnica que denominamos "troceado de literales". Por ejemplo, ensaya las asignaciones de subliterales:

```
T1$=TEST$(1 TO 4)
T2$=TEST$(5 TO 10)
T3$=TEST$(11 TO 15)
```

Y luego teclea lo siguiente:

```
PRINT T1$!T2$!T3$
```

Eso te mostrará que has segregado 3 subliterales del literal TEST$. Además, no ha sido necesario apartar ningún otro espacio en memoria para ellos, sino que simplemente son porciones del espacio que éste ocupa.

Para extraer un solo carácter del literal, usamos un único número para indicar la posición de dicho carácter. Por ejemplo:

```
PRINT TEST$(5)
```

nos daría la letra P.

También parte de un literal puede ser **sustituida** mediante esta técnica de troceado, usando el subliteral como variable a la que se asigna el dato. Por ejemplo:

```
TEST$(5 TO 10)="ARTURO"
```

Haría que se sustituyera el subliteral PANCHO con ARTURO. Tienes ahora completa libertad sobre el contenido y el manejo de un literal, excepto en una consideración importante -el espacio ocupado en memoria- tienes que ser cuidadoso en no reemplazar parte del literal con más de lo que el literal completo puede aceptar. Justamente recuerda que la última asignación de valor para el literal total es la que determina su longitud **corriente.** Las sustituciones de parte del literal no afectan esta reserva de memoria, y queda a tu discreción el garantizar que las sustituciones hechas no sobrepasan la longitud total. Por ejemplo, TEST$ tiene 15 caracteres: ninguna asignación puede hacer referencia a un elemento con posición 16 o mayor.

Si deseas establecer un literal de una cierta longitud para posteriormente operar con él, puedes comenzar rellenándolo con espacios en blanco. Puede hacerse así:

```
STRING$ = "          "
```

Sin embargo, es una labor difícil para literales largos. Podrías usar un bucle FOR para rellenar el literal N$ con 50 espacios en blanco, en la forma:

```
N$=" ":FOR I=0 TO 49:N$=N$&" "
```

Sin embargo, se dispone de una función de clave FILL$ que permite que un literal sea **rellenado** con cualquier número de caracteres. Tiene la forma:

```
FILL$( "A" , X )
```

o bien:

```
FILL$( "AB" , X )
```

La primera entrega un literal de longitud X, rellenado con Aes. La segunda entrega un literal de longitud X rellenado con repeticiones de ABes. Si X es impar, el último carácter del literal será una A.

Esta función puede usarse para exponer líneas de caracteres en la pantalla. Por ejemplo:

```
PRINT FILL$("*",30)
```

coloca una línea de 30 asteriscos en la pantalla.

Para reservar 50 posiciones en N$ para sustitución posterior con subliterales, usa:

```
N$ = FILL$(" ",50)
```

Para entresacar justamente el mes en una fecha dada por DATE$, exponlo primero y observa dónde cae el mes:

```
PRINT DATE$
```

DATE$ está definida de tal manera que los primeros cuatro caracteres dan el año. Luego viene un blanco, luego 3 caracteres para el mes, luego un blanco y lo demás. La longitud de DATE$ y las posiciones de la información almacenada en esa variable son constantes para permitirte que segregues porciones a voluntad. Sin embargo, debieras recordar que DATE$ puede aceptar un argumento que es elaborado para convertir una fecha numérica en un literal. No es posible por tanto, usar las cotas del subliteral a extraer directamente con la propia función DATE$, ya que se confundiría con un argumento numérico de fecha. Por tanto, debe asignarse el contenido de DATE$ a otro literal primeramente, y luego segregar de él. Se puede hacer como sigue:

```
10 CLS
20 PRINT"EL PRESENTE MES ES:"!!
30 K$=DATE$
40 PRINT K$(6 TO 8)
```

También puede usarse esto para cronometrar una tarea concreta extrayendo los minutos y segundos antes y después de su ejecución. Lo siguiente te muestra cómo:

```
10 CLS
20 K$=DATE$
30 PRINT"MINS Y SEGS:"!K$(16 TO 20)
40 FOR I= 1 TO 10000:N=1
50 PRINT
60 K$=DATE$
70 PRINT"MINS Y SEGS:"!K$(16 TO 20)
```

El tiempo es corto y por tanto no es muy exacto, pero intenta añadirlo por ejemplo al ordenamiento por burbujas de 96 caracteres anteriores, y sí te saldrá una medida exacta.

Coerción

Con el fin de usar el tiempo en un programa, será necesario a menudo ser capaz de tratar los números en el literal DATE$ como **cantidades numéricas** en lugar de meramente **cadenas de caracteres.** Ahí es donde el concepto llamado en el QL de **coerción** aparece. (Y coerción es la acción de obligar por la fuerza a algo). A causa de este concepto, en el QL puedes convertir un literal formado por cifras a un numeral, mediante una sentencia sencilla de asignación. Supongamos que el literal "12345" está contenido en K$ (ocupando 5 posiciones en memoria). Entonces, en la siguiente sentencia:

K = K$

el QL observa que hay una **discordancia** en ambos miembros de la igualdad. A la derecha tenemos un literal y a la izquierda tenemos un numeral en punto flotante (sin signo $). Intentará interpretar en lo posible lo que pretendes, y así examinará el literal para ver si puede ser considerado realmente como un número. Si lo es, lo asigna como valor de K. Esta es una facultad del SuperBASIC y normalmente otros BASIC tienen en lugar de ello funciones especiales para la conversión.

El concepto de coerción también funciona en el sentido inverso. Por ejemplo:

K$ = 8504

almacenará en el literal la "cadena de cifras". Y puedes segregar partes de él para ver que ha funcionado. Ensaya por ejemplo:

PRINT K$(2 TO 3)

que expondrá el subliteral 50. Si intentas trocear un numeral en coma flotante, verás que produces un error.

Tienes que ser muy cuidadoso al usar números grandes en estas asignaciones. Por ejemplo, ensaya:

```
K$ = 27364802
```

Intenta predecir lo que expondrá la siguiente sentencia:

```
PRINT K$(2 TO 5)
```

Puede que esperaras 7364. Ensáyalo y verás que la respuesta realmente es .736. Para ver por qué sucede eso, intenta exponer el valor de K$. Antes de que actuara la **coerción**, se evalúa cualquier expresión numérica que hubiera en la parte derecha de la asignación. Eso incluye el redondeo y la conversión a la notación científica (con E) si es necesario. Ensaya segregando diversos subliterales de K$ para dominar la situación.

La coerción también trabaja al evaluar las versiones literales de expresiones numerales. Por ejemplo, si N$="10" y M$="20", entonces:

```
PRINT N$*M$
```

expondría 200 como resultado. Siempre que el SuperBASIC ve una operación numérica que tiene variables literales en lugar de numerales, intentará convertir la **índole** de los operandos para cumplimentar lo que se le manda. No siempre será capaz de hacerlo sin embargo, particularmente si uno de los literales contiene letras u otros signos no interpretables como numéricos.

Otras funciones literales

Hay dos claves más que pueden usarse para operar con literales. Son las de LEN (de legth=longitud) que entrega la **largura** de un literal, y la INSTR (in string=dentro de la cadena) que determina si un literal dado está contenido -**pertenece a**- otro literal. La función LEN se usa como sigue:

```
LEN(K$)
```

y el resultado entregado es el número de caracteres que forman el literal K$; por lo que puede emplearse para garantizar que cualquier operación con subliterales de él es válida antes de ejecutarla.

La clave INSTR es en realidad un **operador** que actúa sobre dos literales operandos de la siguiente manera:

```
A$ INSTR B$
```

El resultado de la operación será 0 -que puede interpretarse como falso- si A$ no forma parte de B$. En cambio, si A$ pertenece a B$, el resultado es un número distinto de cero que refleja la posición del primer carácter de A$ que corresponde al primero de B$. El resultado puede interpretarse como TRUE (cierto).

Por ejemplo:

```
K = "CARA" INSTR "CARADURA
```

haría que se asignara a K el valor 1; mientras que la siguiente:

```
K = "CARADURA" INSTR "CARA"
```

daría como resultado de la operación el número 0.

Una aplicación de esto sería, por ejemplo, ver cuántas veces una ciudad concreta aparece en una lista de direcciones en forma de tabla. Supongamos que la tabla de 100 elementos ADD$ contiene dichas direcciones y que estamos examinando la cantidad de veces que aparece la ciudad Bollullos. El siguiente programa nos daría la respuesta:

```
100 COUNT=0
110 FOR I=1 TO 100
120 IF "Bollullos"INSTR ADD$(I)
    THEN COUNT=COUNT+1
130 NEXT I
140 PRINT"Bollullos aparece"!COUNT!"veces"
```

Se supone que este ejemplo es parte de un programa mayor en el que aparece ADD$, con su correspondiente instrucción DIM y los datos en ella.

Esta rutina nos proporciona un ejemplo de la importancia de usar correctamente las relaciones. En este caso mejor sería usar la relación **casi igual** ==, ya que con ella no importaría si el nombre de la ciudad estaba escrito en mayúsculas o en minúsculas, siempre y cuando no estuviera mal deletreado. Usando INSTR nos daría falso cuando estuvieran escritos de distinta caja, mayúscula o minúscula.

No es fácil diseñar un programa de ordenador que efectúe la actividad inteligente de adivinar la ciudad correcta a partir de una versión mal escrita. El programa tendría que saber todas las ciudades del país en cuestión o tener algún otro método de examinar el nombre inteligentemente.

Otro uso de la relación == sería la de comprobar la respuesta del operador a preguntas que exigen sí o no. Previamente habíamos tenido que comprobar un conjunto complicado de comparaciones específicas para reconocer SI, sí, o cualquier cosa entre medias. Eso puede usarse inmediatamente utilizando la relación **casi igual** ==.

Resumen

- El código numérico ASCII es el usado para almacenar caracteres. La función CODE se usa para hallar el código de un carácter, y la CHR$ acepta un código y entrega el correspondiente carácter literal.

- Las relaciones de orden tales como no usan el ordenamiento ASCII: usan uno más útil para la ordenación alfabética, sin mirar la caja mayúscula o minúscula.

- Una tabla es una colección de variables subindicadas. Los identificadores de las tablas deben terminar en $ para literales y en % para enteros. La sentencia DIM se emplea para ocupar el espacio adecuado en la memoria para la tabla. La función DIMN se usa para examinar la dimensión de una tabla, si se requiere.

- Un literal tiene asignado en memoria una cantidad de espacio igual a sus posiciones, a su longitud. Puede incrementarse asignándole valores literales de mayor longitud.

- Un literal puede **trocearse** para segregar o sustituir un subliteral de él.

- La **coerción** asegura que una asignación entre una variable literal y un numeral tendrá éxito, siempre que sea factible. Puede emplearse para cambiar la índole de un dato de literal a numeral, o viceversa.

- La función FILL$ se usa para producir un literal que contiene un carácter o una pareja de caracteres repetidos hasta una longitud dada.

- La función LEN dice la **longitud** de un literal, y el operador INSTR permite determinar si un subliteral dado pertenece a un literal principal, y la posición en que se encuentra dentro de él.

Capítulo Siete

Cálculos y Funciones Standard

El QL dispone de una amplia gama de funciones científicas para cálculos complejos. El uso completo de estas funciones requiere una comprensión de matemáticas que sobrepasa el alcance de este libro. Si no necesitas estas fórmulas matemáticas, debieras saltarte este capítulo por el momento. Siempre puedes consultarlo en una fecha ulterior.

Se supone en lo que sigue que estás familiarizado en lo general con las matemáticas de las funciones y operaciones involucradas. El capítulo está diseñado para mostrarte cómo usar las funciones y operadores del SuperBASIC, y pretende darte bastantes ejemplos para que las uses con exactitud. Los primeros manuales del QL contenían muchos errores y este capítulo te provee las versiones correctas de las sentencias en los manuales sin hacer referencia necesariamente a aquellas equivocaciones específicas. Se te aconseja usar los ejemplos demostrados a continuación como una guía y ensayar las funciones y operadores por ti mismo antes de confiar en lo que los manuales establecen.

Cálculos en SuperBASIC

Los cálculos en BASIC se efectúan normalmente usando expresiones que contienen operadores y funciones. Nos hemos encontrado ya los operadores numéricos principales de suma, resta, multiplicación y división y algunas de las funciones incluidas en el SuperBASIC. Las siguientes secciones te presentan todas las otras.

Como ya hemos visto, hay dos clases de variables numéricas: **enteras** y de **coma flotante.** Debieras recordar que una variable entera redondea cualquier número que se le asigne, y lo almacena como un número exacto. Un problema es que el rango de tales números es comparativamente pequeño. Las variables en coma flotante pueden contener números mayores, pero no necesariamente almacenan los enteros con exactitud. Por ejemplo, un cálculo en coma flotante puede dar como resultado el número 2.999999 como respuesta, y en lo que al cálculo concierne, eso es el mismo que el número 3. Sin embargo, si tu programa está comprobando un 3, puede incluir una sentencia que diga algo así como:

```
IF X=3 THEN...
```

La condición en una sentencia como esa no se satisfará si X tiene el valor 2.999999.

Debes tener en mente que no es buena idea confiar en un cálculo de coma flotante para dar una respuesta exacta.

Una forma aritmética provechosa del doble signo igual (==) se presenta en la última sección para permitirte algo de flexibilidad en este aspecto, pero a pesar de ello debes tener cuidado con estas situaciones.

Los cálculos entre enteros sí producen números exactos, pero redondean a incluso más grandes inexactitudes. Por ejemplo, los siguientes cálculos todos dan 5 como la misma respuesta:

```
X% = 3.99 + 1.15
X% = 3.99 + 1.45
X% = 10.33/2
```

Usando el signo % en el identificador hace que el ordenador redondee para asegurar que un entero se almacena en la variable X%.

Funciones numéricas generales

En general, la forma de estas funciones es:

Identificador(argumento)

donde **argumento** puede ser una expresión numérica que por sí misma contiene funciones. De hecho, en la mayoría de los sitios donde se requiere un número en una función, en un operador o en una sentencia, el SuperBASIC permite darlo como una expresión numérica o el contenido de una variable.

Las reglas de evaluación de las expresiones aritméticas, usando los cuatro operadores aritméticos simples ya ha sido dada. La regla predominante es que los paréntesis, incluyendo aquéllos que encierran el argumento de una función, son evaluados en primer lugar. Luego viene la evaluación de la función, seguida de los operadores aritméticos. En lo que sigue daremos ejemplos de esto.

Las funciones y operadores que presentamos son solamente útiles si comprendes la matemática que esconden. Como tales, están claramente definidas aquí, pero se da poca explicación en cuanto a su significado. Usalas para referencia, y para corregir y respaldar el manual QL cuando sea necesario.

SQRT (Square Root)

Esta es la función normal **raíz cuadrada** y debe tener un argumento mayor o igual a cero. La raíz cuadrada de un número **X**, es un número que cuando se multiplica por sí mismo da X. Todos los números positivos distintos de cero tienen dos raíces cuadradas -de igual magnitud pero de signo opuesto. El QL da justamente la raíz cuadrada positiva de un número y se usa en la manera siguiente:

```
PRINT SQRT(4)
```

Eso dará el número 2. Cualquier expresión numérica puede colocarse entre los paréntesis detrás del identificador de función.

Por ejemplo:

```
SQRT(SQRT(27*3))
```

Entregará como resultado el número 3. El contenido del paréntesis más interno será evaluado en primer lugar, produciendo 81. Luego vendrá la función SQRT más interna que resultará en 9. La SQRT externa nos dará finalmente el 3.

Todas las funciones presentadas satisfarán las mismas reglas, y puedes incluir funciones dentro de funciones tanto como desees.

INT (Integer)
Esta función entrega el **entero** más inferior al valor de su argumento, a no ser que le argumento ya sea un entero, en cuyo caso da como resultado ese entero sin cambio. Por ejemplo:

```
PRINT INT(4,9)      da 4
PRINT INT(4,2)      da 4
PRINT INT(4)        da 4
PRINT INT(-4,9)     da -5
PRINT INT(-4,2)     da -5
PRINT INT(-4)       da -4
```

Observa que el **próximo más inferior** significa el más cercano a menos infinito, y no necesariamente el más cercano a cero.

DIV (Division)
Este es el equivalente entero de la división (/) en coma flotante. Es un operador, y se usa entre dos expresiones numéricas enteras y debe ir seguida de un espacio en blanco, a diferencia de la /. Este operador no deberá usarse entre valores de coma flotante. La definición del operador entre dos expresiones enteras que evaluadas dan X e Y, es como sigue:

```
X DIV Y = INT(X/Y)    donde X e Y son numerales enteros
```

No debes usar expresiones en coma flotante para X e Y, ya que puede que realmente no contengan enteros.

Ejemplos del uso de DIV son:

```
PRINT 3 DIV 2 da      INT(1.5) = 1
PRINT 3 DIV -2 da     INT(3/-2) = -2
```

ABS (Absolute)
Entrega la **magnitud** o valor absoluto de una expresión en coma flotante, i.e. entrega el mismo número si es positivo y sustituye el - con + si el número es negativo.

```
PRINT ABS(33,156)     da 33.156
PRINT ABS(-125.7)     da 125.7
```

MOD (Module)
Este es un operador que sólo actúa como entero, y se refiere a las congruencias **módulo** un entero. Al igual que para DIV, no debe actuar entre expresiones de coma flotante. Da el **resto** en una división entera. Su definición es la siguiente:

X MOD Y = X - (X DIV Y)*Y

que podemos ampliarlo aún más diciendo que equivale a:

X MOD Y = X (INT(X/Y))*Y

Por ejemplo:

```
PRINT 5 MOD 3      da 2
PRINT 5 MOD -3     da -1
PRINT -5 MOD 3     da 1
PRINT -5 MOD -3    da -2
```

Observa cómo la posición del número negativo en los argumentos de este operador es muy importante.

PI
Esta palabra clave entregará el valor de **pi** con 7 cifras significativas. Así:

PRINT PI da 3.141593

DEG (Degrees)
Esta función convierte el valor de una expresión numérica dada en radianes a sus correspondientes **grados.** Por ejemplo:

```
PRINT DEG(2*PI)      da 360
PRINT DEG(3.141593)  da 180
PRINT DEG(8.772)     da 502.5986
PRINT DEG(-3*PI)     da -540
```

RAD (Radians)
Es la opuesta de la última función, y convierte grados a **radianes** con 7 cifras significativas. Por ejemplo:

PRINT RAD(360) da 6.283185 que es igual a 2 pi.

SIN COS TAN y COT
Estas son las funciones trigonométricas habituales, **seno, coseno, tangente** y **cotangente** y son evaluadas con 7 cifras significativas. Todos los argumentos están en radianes, por lo que los ángulos en grados deben cambiarse a radianes antes de usar las funciones trigonométricas. La banda de valores para el argumento es bastante grande, sienod de -6000 a +6000 para SIN y COS; y de -3000 a +3000 para TAN y COT.

Al usar la constante funcional PI, encontrarás que debido a la inexactitud en su evaluación, estas funciones no darán respuestas exactas. Por ejemplo, ensaya lo siguiente:

```
PRINT TAN(PI/2)
PRINT SIN(PI)
```

En la primera dará una respuesta grande pero finita, y la segunda un valor pequeño pero distinto de cero.

ATAN y ACOT
Son las funciones trigonométricas inversas **ArcoTANgente** y **ArcoCOTangente.** No hay límite formal para los argumentos de estas funciones. El argumento no es desde luego un ángulo tal y como parece indicar el manual del QL, y no está expresado en grados, radianes o ninguna otra dimensión. Por ejemplo:

PRINT DEG(ATAN(1)) da 45 grados.

EXP (Exponential)
Es la función **exponencial** que da potencias de la constante neperiana **e**=2.718282. Su banda es de -500 a +500. Por ejemplo:

PRINT EXP(1) da 2.718282
PRINT EXP(0) da 1

LOG10 y LN (Logarithm and Logarithm Natural)
Son las habituales funciones **logarítmicas,** con respecto a la base 10 y a la base e, respectivamente. Sus argumentos deben ser positivos y distintos de cero. Por ejemplo:

PRINT LN(EXP(13.89)) da 13.89

RND y RANDOMISE
Estas palabras clave guardan relación con la generación de números pseudo-aleatorios que bien pueden comenzar a partir del mismo número cada vez o pueden usarse para obtener series con un comienzo impredecible. RND (RaNDom=Suerte, fortuíto) es la función que entrega el número **aleatorio,** y RANDOMISE es una sentencia que puede usarse para establecer el **gérmen** a partir del cual se genera la serie de números aleatorios. RND puede usarse con 2 argumentos como máximo para especificar la banda de números aleatorios producidos.

Si RND no tiene ningún argumento, entregará un número pseudoaleatorio en **coma flotante** en la banda de 0 a 1, pero no incluyendo ni el 0 ni el 1.

Para producir un número dentro de una banda concreta, un único argumento producirá un **entero** entre 0 y dicho argumento inclusive. Así:

RND(13)

producirá un entero en la banda 0 a 13, ambos inclusive.

Un segundo argumento, después de la clave TO, dará una banda de enteros con esas **cotas**, ambas inclusive. Así:

RND(13 TO 135)

entregará un entero del 13 al 135, ambos inclusive.

RANDOMISE se usa para establecer la **semilla** usada por el generador de números aleatorios y hacer que lo haga de una manera predecible. Si RANDOMISE se ejecuta antes de RND, entonces se producirá la misma serie de números al usar la función RND. Por ejemplo, el siguiente programa, dará siempre el mismo par de números en la pantalla:

```
10 RANDOMISE
20 PRINT RND(9)!RND(9)
30 RANDOMISE
40 PRINT RND(9)!RND(9)
50 GOTO 10
```

La primera sentencia arranca el generador de números aleatorios a partir de un número prefijado internamente que nunca varía. La siguiente sentencia extrae los primeros dos números generados y los expone en pantalla. La líne 30 vuelve luego a reestablecer la semilla del generador y la siguiente sentencia recibe por tanto exactamente los mismos números aleatorios que antes.

Se puede imponer un diferente gérmen para el generador, mediante un parámetro detrás de la clave RANDOMISE: Así:

RANDOMISE(12)

hará que el generador use el 12 como semilla. Eso puede ser útil si dos rutinas separadas están usando bloques de números aleatorios dentro de la misma banda. Usando el comando RANDOMISE con diferentes parámetro asegurará que esos bloques son diferentes uno del otro, pero ambos repetibles. Si no se usa RANDOMISE en esta situación, los bloques de números ciertamente serán diferentes, pero no repetibles.

Operadores y relaciones

Nos hemos encontrado con varios operadores y relaciones para las funciones numéricas y lógicas del QL, y esta sección presenta el resto del conjunto disponible.

El QL permite que se usen operadores **calibrados en binario** para usarse entre números en base 10. El SuperBASIC convierte estos números del sistema decimal a su **equivalente binario,** y actúa sobre ellos "bit a bit" usando el operador especificado, y convirtiendo el resultado obtenido a su equivalente en base 10.

Se dispone también de un operador para elevar a una potencia, y de una relación para comparar números que son aproximadamente iguales.

Operadores binarios

Los cuatro operadores para números binarios (o binarizados) disponibles en el QL no usan las equivalentes palabras **claves** logicales: usan los siguientes símbolos

AND &&	**(Y Lógico Inclusivo)**
OR ||	**(O Lógico Inclusivo)**
XOR ^^	**(O Lógico Exclusivo)**
NOT ~~	**(NEGación)**

Se usan de la misma manera que los operadores lógicos, como puede verse por los siguientes ejemplos:

PRINT 125 && 100 da 100
PRINT 125 || 100 da 125

Símbolos numéricos adicionales

La precisión con que el QL almacena los números en coma flotante significa que incluso las más ligeras inexactitudes en la evaluación harán que dos números no sean iguales en el sentido más estricto. Si una rutina requiere que dos números sean considerados como iguales si concuerdan en la precisión de una parte en millón hasta la séptima cifra decimal, entonces se puede usar la relación **casi igual** (==). Por ejemplo:

```
PRINT 1.1354761 == 1.1354762
```

dará una respuesta distinta de cero porque esa relación es TRUE (cierta). Sin embargo, la siguiente da 0 para mostrar que es FALSE:

```
PRINT 1.1354761 = 1.1354762
```

Es muy útil al evaluar series infinitas con esa exactitud.

Otra operación numérica disponible es la **elevación a una potencia.** Está simbolizada por el signo **copete** (^). Así por tanto:

```
PRINT 3^2        da 9
PRINT 3.2^1.9    da 9.115594
```

En las reglas para la prioridad en la evaluación de operaciones, el signo ^ tiene una prioridad mayor que la multiplicación y la división. Así, la siguiente expresión:

```
5*2^4/10
```

Será evaluada en el siguiente orden:

```
2^4 = 16
5*16 = 80
80/10 = 8
```

Hay otras dos funciones numéricas mencionadas en los manuales del QL, pero que puede que no trabajen en las primeras máquinas. Esas son la **elevación entera a una potencia** y la función **signo** (SGN).

La elevación entera a una potencia tiene la siguiente forma:

A(^)B

y actúa solamente sobre valores enteros.

La función SGN debiera entregar -1 si el argumento es negativo, 0 si es cero, y +1 si es positivo. Su forma es:

SGN(A)

Con éstas se completa la descripción de las funciones numéricas disponibles en el QL. El siguiente capítulo tratará con los gráficos y con las facilidades para manejo de la pantalla del SuperBASIC.

Resumen

- SQRT calcula la raíz cuadrada de un numeral.
- INT, DIV, MOD actúan sobre enteros para dar el entero más próximo inferior, cociente entero y resto de la división entera, respectivamente.
- ABS da el valor absoluto de una expresión numérica.
- PI da el valor de **pi**=3,141593.
- DEG y RAD cambian radianes a grados (degrees) y grados a radianes, respectivamente.
- SIN, COS, TAN y COT son las funciones trigonométricas seno, coseno, tangente y cotangente y actúan sobre ángulos dados en radianes.
- ATAN y ACOT son las funciones trigonométricas inversas.
- EXP, LN y LOG10 dan la exponenciación, los logaritmos naturales y los logaritmos en base 10 respectivamente.
- RND y RANDOMISE se usan para producir números aleatorios.
- AND, OR, XOR y NOT están disponibles para números calibrados en binario.
- ^ eleva un numeral a la potencia de otro.

Capítulo Ocho

Gráficos y Sonidos

Este capítulo concierne a la habilidad del QL para mostrar gráficos en colores de alta resolución. En otras palabras, puede mostrar figuras generales con líneas y curvas con un alto grado de exactitud. Eso puede apreciarse fácilmente ejecutando los paquetes de programas Psion que vienen con el ordenador. En particular, el programa EASEL usa la pantalla en todo su poderío, y debieras ensayar con ese programa antes de empezar a programar gráficos por ti mismo. EASEL (que se refiere a algo así como a un trípode **teodolito)** te dará una buena idea de las capacidades de la máquina, así como te suministrará algunas ideas útiles para tus propios programas.

Gráficos

La palabra **gráficos** se aplica generalmente a la imagen en pantalla. Hasta ahora sólo hemos usado los **grafismos** simples que son las letras, cifras y signos del repertorio. No hemos intentado ajustar los colores de la imagen, o exponer figuras generales en la pantalla.

Si has practicado con los paquetes de programas Psion de la máquina, te estarás preguntando cómo se pueden producir efectos visuales tan notables. Todos esos efectos son accesibles desde SuperBASIC.

La naturaleza exacta de los gráficos que puedes mostrar usando el QL dependen de si tienes una TV o un monitor de video. Algunos de los efectos más apabullantes sólo pueden ser apreciados en un monitor, donde la resolución es mucho más elevada.

La pantalla se divide en motas separadas o **elementos pictóricos,** que se demoninan "pixels" para abreviar. Usando SuperBASIC puedes trazar cada pixel separadamente, e incluso elegir su color.

Con el fin de elegir una **mota** individualmente, tenemos que tener un sistema de numeración, o de etiquetado consistente para ellas. Usamos coordenada rectangulares para especificar una mota concreta y lo describiremos en breve.

Hay dos modos de imagen posibles, denominados **modo de alta resolución** y **modo de baja resolución.** En el modo de alta resolución se pueden pintar líneas y puntos muy finos en la pantalla, pero la capacidad de colores queda restringida. En el modo de baja resolución, las líneas y los puntos son más gruesos, pero entonces hay muchos más colores. El modo que elijas depende de la aplicación particular.

Finalmente, no tienes por qué trazar figuras geométricas especificando cada mota de ellas separadamente. El QL dispone de muchas facilidades especiales de dibujo para ayudarte a crear figuras especificando parámetros numéricos en sentencias de gráficos.

La pantalla del QL

Para ver las dos maneras diferentes de exponer la imagen en la pantalla, simplemente elige la tecla funcional errónea después de restaurar las condiciones iniciales de la máquina. Si estás usando una TV para salida de información, restaura la máquina y en lugar de elegir TV cuando se te pregunta, pulsa F1.

Como vímos en el Capítulo 1, el resultado son **tres ventanas** en tu TV. La ventana blanca de la izquierda es para listados; la roja de la derecha es para exponer información de salida; y la de la parte inferior es la de **comando** donde se muestran las sentencias impuestas a través del teclado. Manteniendo pulsada una tecla de letra, mostrará la pequeña versión del modo de salida PRINT del QL. Este modo de imagen se denomina de **alta resolución** ya que la imagen puede mostrar trazos más finos y por tanto acumular más información en ella. Sin embargo, como puedes ver, si no dispones de un monitor, los caracteres desaparecen por fuera de los bordes de la pantalla cuando estás en el modo monitor.

Si tienes un monitor de color, debieras ensayar pulsando F2 después de **restaurar**, y ver la resolución usada normalmente con televisores.

Si tienes una TV, es posible usar el modo de alta resolución sin perder parte del cuadro por los bordes. Hay un comando especial que puedes usar para entrar en uno u otro de los modos. Es el comando MODE, como veremos.

Aprieta el botón RESET, y efectúa tu elección correcta para F1 o F2.

La pantalla completa está dividida en 512 pixels a lo ancho por 256 pixels verticalmente hacia abajo. Estos pixels se unen estrechamente en la pantalla para producir el efecto de una hoja de papel alisada. De hecho, el término técnico en el QL para el **fondo** de la imagen es el de **papel** (PAPER). Escribir en este papel se efectúa con **tinta** (INK), y el recuadro del papel en que aparece lo escrito es lo que se denomina **ventana** (WINDOW). Veremos que además podemos rodear de un color determinado el **borde** (BORDER) de dicho papel. Una TV perderá alguno de los elementos pictóricos por fuera de la parte inferior y la parte derecha, reduciéndose así el número total efectivo que puede manejarse en la pantalla.

Cuando pones en marcha el ordenador y pulsas F1 o F2, los 512 x 256 pixels están siendo usados de una manera específica según dispongas de una TV o de un monitor. Sin embargo, debes tener presente que la imagen todavía está compuesta de 512 x 256 motas, y que el programa interno simplemente la está usando de una manera especial. Tú tienes control completo sobre las características de la imagen en un programa, y veremos cómo usar esos **atributos** a lo largo de este capítulo.

Si tienes una TV, teclea:

MODE 4 o bien: **MODE 512**

(Son equivalentes).

Eso cambiará el **modo** a alta resolución, y reducirá el número de colores disponibles de 8 a 4. La razón para eso es que los gráficos en alta resolución exigen una gran cantidad de memoria. Si fomentas una de la características, pierdes parte de la otra para compensar, ya que la memoria disponible está prefijada. Teclea algunas letras, y verás que aparecen del mismo tamaño que cuando ensayaste a pulsar F1 hace un minuto para ver la imagen del monitor después de RESET. Esta vez, sin embargo, el comienzo de la exposición en pantalla está ajustado para asegurar que no se pierde ningún carácter por los extremos. Eso se consigue reduciendo el número de caracteres permitidos en cada línea de pantalla para las actividades normales.

Como puedes ver, el modo de alta resolución no es muy satisfactorio con un aparato de TV, y tendrás que ajustar los controles del mismo, incluyendo la sintonía para producir el mejor efecto. Las TVs no son buenas para mostrar finos detalles.

Ahora teclea:

MODE 8 o bien: **MODE 256**

y la imagen regresará al estado normal para una TV. Puedes cambiar de modo en tus programas como desees.

La pantalla está dividida en una **gradilla**, y cada malla de ella -cada pixel- está numerada de acuerdo con su distancia tomada desde el extremo izquierdo de la ventana de salida, y desde el margen superior. El pixel numerado 0,0 es pues el cero desde la izquierda, y el cero desde la parte superior de la ventana, respectivamente. Igualmente, el pixel numerado 52,100 está situado 52 puntos contados desde la izquierda y 100 desde el margen superior. La siguiente sección te muestra cómo usar la sentencia BLOCK para pintar motas individuales, así como **bloques** rectangulares de ellas. Recuerda que en una TV la ventana de salida ocupa toda la pantalla; y en un monitor, en el modo normal sólo cubre la parte derecha de la misma.

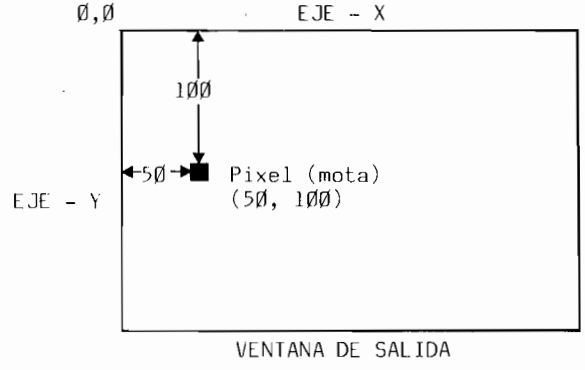


Fig. 8.1. Sistema para localización de motas

Bloque y Color

La sentencia BLOCK ilustra todas las facetas del sistema de localización de un punto en la pantalla, y lo usaremos para presentar esos conceptos.

La Fig. 8.1 muestra una ventana de salida con el sistema de **coordenadas** para identificar cada punto de la pantalla. Los pares de números mostrados en la Fig. 8.1 son las **coordenadas.** Por convenio, el eje horizontal de izquierda a derecha se denomina el eje X, y el eje vertical de arriba hacia abajo se denomina el eje Y.

La mota mostrada es una pequeña 'pinta' de color separada 50 puntos a partir de la izquierda, y 100 a partir del margen superior. Esta mota (o pixel) queda así marcada con las coordenadas 50,100. El primer número es la llamada coordenada X, y el segundo la coordenada Y. Los paréntesis no se usan en SuperBASIC, pero es el convenio para escribir esta identificación.

La sentencia BLOCK se utiliza para cambiar el **color** de una mota, y dicha mota sólo se destacará si tiene un color diferente del color del papel sobre el que se proyecta. El color normal del papel en la pantalla al encender para una TV, es el blanco. Y si una determinada mota se **colorea** de cualquier otro color, será claramente perceptible visualmente.

BLOCK es una sentencia en SuperBASIC que acepta unos cuantos **parámetros** para decirle al ordenador dónde tiene que situar un bloque rectangular de una o más motas, y con qué color ha de proyectarlas. Antes de ver esta sentencia BLOCK en funcionamiento, debemos definir la manera en que se especifica el color en el QL.

El **color** exige un máximo de tres parámetros para su definición completa, aunque puede especificarse por un número compuesto tal y como se describe en el manual del QL. Comenzaremos tratando con los colores sólidos y pasaremos a los **efectos de trama** más adelante. Por el momento, la siguiente tabla se puede usar para especificar el color con que se rellena una determinada mota o elemento pictórico (pixel):

Número	MODE 4 (Alta res)	MODE 8 (Baja res)
0	Negro	Negro
1	Negro	Azul
2	Rojo	Rojo
3	Rojo	Magenta
4	Verde	Verde
5	Verde	Ciano
6	Blanco	Amarillo
7	Blanco	Blanco

Como puedes ver, el modo de más alta resolución dispone de menos colores, y sólo hay dos números posibles para cada color.

Ahora podemos usar la sentencia BLOCK. La forma general de ella es la siguiente:

BLOCK anchura,altura,horizontal,vertical,color

Esta sentencia pinta un rectángulo en la pantalla con los primeros dos parámetros especificando su anchura y altura en **puntos** gráficos. El tercero y cuarto especifican las coordenadas de la esquina superior izquierda de dicho rectángulo, y luego viene el parámetro para el color de acuerdo con la tabla anterior. Ensaya tecleando lo siguiente:

BLOCK 100,5,8,20,4

Eso dibujará un rectángulo verde de 100 puntos de ancho y 5 de alto, y con la esquina superior izquierda situada en las coordenadas (8,20) de la ventana de salida. Como ejercicio aprovechable, toma la medida aproximada de este rectángulo y luego cambia el modo y vuelve a teclear el comando BLOCK como antes, para comprobar que el tamaño del rectángulo permanece el mismo y es por tanto independiente del modo. Luego restaura y cambia a monitor si estás usando una TV, o viceversa en caso contrario. Repite lo anterior, incluso aunque no puedas verlo en la línea de comando. También aquí, el rectángulo es del mismo tamaño y está en la misma posición dentro de la ventana de salida. Eso hace patente que no importa el modo usado, la anchura medida en puntos gráficos es constante -y necesitarás recordarlo para todo lo que sigue.

Vuelve a restaurar y regresa a tu modo normal. El siguiente programa muestra todos los colores disponibles en tu TV o en tu monitor:

```
10 FOR I=0 TO 7
20 BLOCK 20,20,10+I*30,50,I
30 NEXT I
```

Los rectángulos pintados son de altura y anchura de 20 puntos, y su color cambia del 0 al 7. Recuerda que el 7 es el blanco, y puede no ser perceptible contra ese fondo. Debieras usar ahora MODE para cambiar al otro modo de resolución, y RUN el programa otra vez. Eso producirá los mismos rectángulos, pero mostrará cómo cambia el color con el valor del parámetro pertinente.

La proyección de las motas o pixels individualmente, sólo puede conseguirse en el modo de alta resolución, y ese no funciona bien en una TV. Sin embargo, si ajustas tu aparato cuidadosamente, puede que seas capaz de ver una cambiando al MODE 4, y tecleando lo siguiente:

BLOCK 1,1,200,100,4

Será verde, y estará en el centro de la pantalla. Es un diminuto bloque de un punto de ancho y un punto de alto. Si ahora pasas al modo 8, que es de baja resolución, todavía puedes continuar especificando motas sencillas verticalmente, pero la resolución horizontal disminuye, y sólo puedes usar motas que ocupan un par de puntos gráficos. Especificando una anchura de 2 o de 3, hará que aparezca el mismo bloque de 2 puntos de ancho. Similarmente, una anchura de 0 o de 1, hará que no aparezca ninguna mota en absoluto. Para ver que la resolución vertical permanece la misma en uno y otro modo, ensaya lo siguiente en el modo 8:

BLOCK 2,1,200,100,4

La mota ocupando un par de puntos gráficos debiera parecer en la mayoría de las TVs si están apropiadamente sintonizadas y ajustadas. Estará en el mismo lugar que la última vez, y será de dos veces el tamaño horizontalmente. Si ahora ensayas:

```
BLOCK 2,2,250,100,4
```

serás capaz de apreciar la diferencia entre esos dos bloques, dependiendo de tu televisión. El segundo bloque es el doble de alto que el primero.

Debido a la línea de comando en la parte inferior, y al hecho de que las TVs no muestran tanta información en los extremos de la pantalla, no serás capaz de manejar los 512 x 256 pixels completos, en una TV. La anchura total de la pantalla sólo está disponible en un monitor.

Observa que 0,0 en los primeros dos parámetros de la sentencia BLOCK dará un rectángulo de tamaño 0, y en el modo de baja resolución es lo mismo que se especifique 1 o que se especifique 0 para esos parámetros.

COLOR

Los otros parámetros de color te permiten cambiar el bloque proyectado para incluir **efectos de trama** entre colores que contrastan. Esos efectos no aparecen apropiadamente en una TV, pero todavía puedes practicar con ellos.

El color se especifica en general por tres parámetros, pero hasta ahora sólo hemos usado un parámetro y los otros han adoptado automáticamente los valores prescritos para omisiones. Un segundo parámetro elegiría un color de contraste para formar un modelo de trama, y el tercer número elige la forma de la trama que va a usarse. La trama se forma considerando grupos de cuatro pixels (2 x 2) y la forma de las tramas se muestra en la Fig. 8.2, de acuerdo con los valores del parámetro pertinente. Así los valores 2,4,2 para los parámetros de color darían un color principal rojo (2) en contraste con el verde (4) y combinados para formar una trama vertical a barras.

Sería un buen ejercicio escribir un programa que recorriera todas las formas de tramas y de colores disponibles para un tamaño prefijado de rectángulo, usando tres bucles anidados FOR para cambiar los parámetros de color. Hay una solución en el manual del QL que no usa bucles anidados FOR.

Estos parámetros de color son accesibles siempre que se especifica color como argumento de una función o parámetro de una sentencia.

Valor Parámetro	Nombre 'modelo'	Efecto de trama
0	Una sola mota de contraste por bloque de 4	
1	Barras horizontales	
2	Barras verticales	
3	Escaques	

Claves: Mota de color principal
Mota de color de contraste

Fig. 8.2. Modelos de tramas

Papel, Tinta, Reborde

Como hemos mencionado, el PAPER es el **fondo** -el papel- sobre el que se efectúa la escritura o el dibujado. La INK es la **tinta** usada para escribir o dibujar, y cada WINDOW -cada ventana- tiene un BORDER que actúa de **'marco'**. Todas estas son palabras clave que pueden aceptar parámetros para especificar sus características.

Ensaya tecleando:

```
PAPER 4:INK 7
```

Eso hará cambiar el color del PAPEL a verde, y el de TINTA a blanco. No sucede nada hasta que se efectúe un borrado de pantalla para forzar a que los colores corrientes alteren la ventana de salida. Ensaya ahora CLS, seguido de una sentencia PRINT con algún texto a exponer. Eso producirá una escritura en blanco sobre un fondo verde. Tanto PAPER como INK aceptan los parámetros de color especificados anteriormente. Incluso en una TV deberás ser capaz de apreciar el siguiente PAPEL 'tramado'. Ensaya tecleando:

```
PAPER 2,7
```

El valor prescrito para omisiones en el código de trama es 3, que produce el tablero típico de ajedrez. Ensaya añadiendo un tercer parámetro a esta sentencia, y observa el efecto de trama que aparecerá en la TV. Ahora ensaya diferentes parámetros de tinta.

Algunos colores de papel interfieren fuertemente con los colores de tinta; aprenderás muy pronto mediante ensayo y error cuáles son los mejores.

Cauces y Ventanas

Hasta ahora nos hemos restringido a las ventanas prescritas. Estas ventanas también pueden ser cambiadas en tamaño y en color, y más de tres ventanas pueden aparecer al mismo tiempo en pantalla.

Con el fin de controlar cualquier 'entidad' en un ordenador, debe ser **identificada.** La identificación de las ventanas en pantalla es mediante un número, y a cada ventana se le da un **número de cauce** (también llamado de canal) separado, y precedido del signo diésis (#). Los números de cauce para las ventanas standard son:

ventana de comando: #0
ventana de salida: #1
ventana de listado: #2

Estos identificativos de cauce pueden especificarse en las sentencias de gráficos mencionadas anteriormente, usándolos como primer parámetro de ellas. Así, teclea:

```
PAPER #0,2
```

Eso cambiará el color de la ventana de comandos. Cada introducción a través del teclado aparecerá ahora expuesta con tinta verde (el color de tinta prescrito para omisiones) sobre un fondo rojo. INK y BLOCK pueden aceptar un número de cauce de esa misma manera, y por tanto es posible la libertad completa de uso de la pantalla.

Se pueden definir en la pantalla nuevas ventanas de cualquier tamaño y asignarle sus propios números de cauce. Además, sentencias tales como PRINT pueden también aceptar un número de cauce y así un programa puede mantener varias ventanas simultáneamente y usar cada una de ellas para una función diferente; de manera similar a lo que hace el propio QL.

Para **abrir** una nueva ventana con un número de cauce determinado, debe usarse el comando OPEN. Es un comando muy complejo y puede afectar la manera en que funcionan lo procesos fundamentales del QL. Nos toparemos con él de nuevo en el siguiente capítulo. Por ahora lo usaremos para afectar a la pantalla, y para efectuar esta acción acepta una serie de parámetros que indican el tamaño de la ventana y la posición en la pantalla. La forma general de este comando es:

```
OPEN #N, SCR_ANCHURAXALTURAaHORIZXVERT
```

El primer parámetro es el número de cauce que identifica a esa ventana que deseas abrir. La siguiente parte de la sentencia es un identificador de un **dispositivo** concreto ('fisical') que el QL reconoce -en este caso la **pantalla-** (en inglés SCReen), y está preceptuado como SCR_. Luego vienen las características físicas de la ventana, tal y como se muestran, con algunas letras actuando como separadores.

Por ejemplo, teclea lo siguiente:

OPEN #9, SCR_50X50a100X100

Eso abrirá un nuevo cauce hacia la pantalla donde aparecerá como una ventana de 50 puntos de anchura y de altura, y con su esquina superior izquierda situada en las coordenadas (100,100).

Ahora teclea CLS #9 para que se haga visible la nueva ventana. Observa que la mayoría de las sentencias de entrada y salida que nos hemos encontrado hasta ahora funcionarán igualmente si se les especifica el número de cauce llevando como prefijo el signo # . Teclea por ejemplo:

PRINT #9, "HOLA"

La **coma** después del número de cauce es necesaria generalmente como separador. Como puedes ver, la nueva ventana actúa justamente igual que las otras. Si la palabra a exponer no entra dentro de una línea, se genera automáticamente un avance de línea y la escritura continúa en la siguiente. Ahora ensaya este otro ejemplo y observa la ventana cuidadosamente:

PRINT #9,"Provoca corrimiento de imagen"

La ventana actúa en todos los aspectos igual que para las otras ventanas.

Ahora intenta **superponer** escritura en la ventana usando:

AT 5,10:PRINT "Esto solapará #9"

La salida al canal se coloca encima de la de cualquier otro canal.

El canal 9 es una ventana de salida y únicamente funcionarán con él comandos de salida. Así, no es aplicable la sentencia INPUT, y cualquier intento de usar INPUT #9 provocará un error. El teclado está en este momento vinculado a la máquina a través del cauce de entrada #0. Estudia la sección sobre dispositivos en el manual del QL.

Mientras está abierta, se pueden cambiar las características de una **ventana** dada mediante la sentencia WINDOW. La forma general de esta sentencia es:

WINDOW #N,anchura,altura,horiz,vert

Los parámetros están en el mismo orden que para la sentencia OPEN, pero se usan ahora comas como separadores.

También puede cerrarse el cauce hacia una ventana, usando la sentencia de **cierre:**

CLOSE #N

Se puede añadir un **reborde** -un **marco**- a una ventana usando la sentencia BORDER. Esta sentencia acepta tres grupos de parámetros. El primero es el número identificativo del cauce. El segundo es un parámetro de **grosor** que se dará en puntos de imagen, y el tercero es el grupo usual de parámetros de color.

El área de reborde se saca de la propia ventana, por lo que las dimensiones totales permanecen iguales. La forma general de la sentencia es pues:

BORDER #N,grosor,color

Si no se especifica ningún color, el del borde será transparente; pero sigue todavía restringiendo el área de exposición dentro de esa ventana. Por ejemplo, ensaya:

BORDER #9,5

Eso coloca un marco transparente con un grosor de 5 puntos dentro de la ventana 9. Ensaya escribiendo "HOLA" en ese canal para ver el efecto.

Ahora ensaya:

BORDER #9,5,4

Eso colorea el borde pertinente de verde. Ensaya escribiendo otra vez algo. Si usas CLS sobre esa ventana dentro de la cual aparece el borde coloreado, desaparecerá el color del reborde y solamente podrá recuperarse repitiendo la sentencia BORDER. Sin embargo, el borde todavía está ahí y todavía continúa restringiendo el área de exposición.

Es perfectamente admisible definir áreas de ventana unas dentro de otras. Hemos definido la del cauce #9 dentro de la del cauce #1 anteriormente. Como ejercicio, amplía la ventana #9 y define dos o tres más dentro de esa ventana, o incluso solapándose con la parte externa. Luego añade marcos de reborde y practica exponiendo datos en los diversos canales.

Corrimiento de imagen

El contenido de una ventana dada puede también **desplazarse horizontalmente** -como al tomar una PANorámica- o bien **desplazarse hacia arriba o hacia abajo,** ya sea completamente o en parte -como si estuvieramos **desrrollando** (scroll) un papiro. La parte desplazada en uno u otro sentido por el movimiento queda rellena con el color correspondiente al papel.

Para experimentar con SCROLL y PAN, abre una ventana con lo siguiente:

```
CLS
OPEN #7, SCR_100x100a50x50
INK #7,0
PAPER #7,5
```

Luego usa CLS #7 para mostrar la ventana. La primera sentencia anterior **borra** el canal 1, que es el canal normal de salida. La segunda sentencia abre un nuevo canal comenzando en (50,50) y de dimensiones 100 x 100. Luego se fijan los colores de tinta y de papel dentro de ese canal para posteriormente escribir. Si quieres que tenga un reborde, usa el comando BORDER con un grosor de, digamos, 5.

No uses otra vez CLS sobre el canal 1, o el borde desaparecerá.

Ahora haz que se exponga algo en esa ventana usando por ejemplo:

PRINT #7, "Eso es el canal 7"

Todo eso no cabrá en la línea superior del canal. Para hacer que se desplace hacia abajo usaremos el comando SCROLL que tiene tres parámetros posibles:

SCROLL #N, nº de puntos, parte

El primer parámetro es el número de cauce, y está prescrito para ser ante omisiones el canal corriente de salida. El segundo parámetro es el número de puntos que queremos tenga el corrimiento de imagen, y su signo determina la dirección. Puede ser una expresión numérica, y si es positiva el corrimiento es ascendente, y si es negativa, descendente. El último parámetro se refiere a la **parte** de la imagen en la ventana que queremos sea desplazada. Puede ser también una expresión numérica y sólo puede adoptar los valores 0, 1 ó 2. Está prescrito para omisiones el 0, que hace desplazar **toda** la imagen de la ventana. Si se especifica un 1, la parte de la imagen a partir del límite superior hacia abajo, pero excluyendo a la línea de cursor, es la que se mueve. Si se especifica un 2, es la parte inferior de la imagen a partir -y excluyendo- la línea del cursor. Ensaya con lo siguiente:

SCROLL #7,4,1

Eso hará que se desplacen las dos líneas superiores dentro de la ventana en 4 puntos, y se pierda la mitad de cada carácter de la segunda línea. Lo expuesto en la línea del cursor no se ve afectado. Ahora teclea:

SCROLL #7,-4,1

Con lo que conseguirás que lo expuesto vuelva a donde estaba, pero no recuperarás el contenido que cayó fuera de la pantalla durante el último corrimiento.

Para desplazar la parte inferior de la imagen, necesitas subir el cursor hasta la parte superior de manera que haya algo expuesto debajo de él que pueda ser desplazado. La alternativa es usar el comando AT para situar **en** una posición el cursor del #7 y escribir algún dato en esa posición, y luego volver a colocar el cursor por arriba de esos datos. Ensaya tecleando:

AT #7,0,1

Eso ubica el cursor en la posición 0 de la fila 1 de la ventana #7. Ahora el siguiente comando actuará sobre la parte de la imagen situada por debajo del cursor:

SCROLL #7,4,2

y este otro comando:

SCROLL #7,-4,2

lo dejará como estaba.

Ensaya usando el comando de corrimiento (desrrolle) sobre la ventana de comando (#0) para apreciar que posees completo control sobre la pantalla. Recuerda que si estás en el modo de alta o baja resolución, la resolución vertical es la misma; y que puedes por tanto, siempre deslizar justamente un solo punto de imagen, o cualquier número impar de puntos que desees.

El comando de corrimiento en PANorámica, que es lateral, puede actuar también sobre toda la imagen en la ventana, o sobre parte de ella según desees. Acepta el mismo número de parámetros, con los primeros dos siendo igualmente el identificador numérico del canal y el número de puntos que deseas desplazar la imagen hacia la izquierda o hacia la derecha. Los números positivos producen movimiento del contenido de la ventana hacia un lado y los negativos hacia el contrario. El tercer parámetro define la parte de la ventana donde va a ocurrir el corrimiento, y está prescrito para omisiones como 0 indicando que afecta a toda la imagen en la ventana. Ensaya los siguientes comandos:

PAN #7,5 y luego **PAN #7,-5**

El primero mueve el contenido de la ventana 5 puntos hacia un lado, y el segundo hace que vuelva a su posición original.

El tercer parámetro puede añadirse de acuerdo con la siguiente tabla:

Valor	Efecto
0	Corrimiento lateral de toda la ventana.
3	Corrimiento lateral de toda la línea del cursor
4	Parte de la línea del cursor situada a la derecha del mismo, incluyendo la posición del cursor.

Para ver el efecto, teclea:

PRINT #7,"123"

Con lo que mueves el cursor hacia la mitad de una línea en la ventana. Luego teclea:

PAN #7,5,3

Con lo que desplazarás toda la línea del cursor 5 puntos; pero estando en el MODE 8 no tiene ningún sentido usar números impares para las distancias en puntos de imagen, ya que el número real movido es el siguiente número par más inferior. Así, mientras usas una TV, el valor 5 de la última sentencia puede sustituirse por 4 obteniendo el mismo efecto, a no ser que específicamente hayas cambiado a MODE 4.

Ahora teclea:

PAN #7,-4,3

Con eso obtienes el mismo efecto, pero con desplazamiento en sentido contrario.

Para ver moverse parte de la línea del cursor, teclea lo siguiente:

PAN #7,4,4

Con eso mueves la posición del cursor y todo lo situado a su derecha, 4 puntos de imagen. Ensaya por ti mismo cómo volverlo a la posición original.

Si estás produciendo corrimiento de imagen con colores tramados, tendrás que atenerte a los números **pares** en los movimientos para retener el efecto visual estipulado.

Borrado de parte de la pantalla

El comando CLS también puede usarse sobre parte de una ventana específica, si se incluye un parámetro extra que indique la parte que ha de ser borrada. La definición de este parámetro de **parte** se muestra en la siguiente tabla:

Valor	Efecto
0	Limpia toda la pantalla (prescrito)
1	Limpia por arriba pero excluyendo la línea del cursor
2	Limpia por abajo pero excluyendo la línea del cursor
3	Limpia toda la línea de cursor
4	Limpia a la derecha del -e incluyendo al- cursor

Para verlo, usa la ventana #7 anterior, y rellénala de caracteres expuestos usando PRINT con un dato literal muy largo. Luego usa AT para colocar el cursor **en** el centro de la ventana. Para ver dónde está el cursor, haz que se exponga un solo carácter en esa posición, y ten en cuenta que aparecerá en la siguiente posición inmediatamente a la derecha de la que ahora es la posición del cursor.

Ahora teclea lo siguiente en el orden dado, observando la pantalla cada vez que pulses RETURN. Verás cómo se **borra** la parte pertinente de la imagen dentro de la ventana, a medida que lo haces.

```
CLS #7,2
CLS #7,1
CLS #7,4
CLS #7,3
```

Usando el parámetro de parte con valor 0 es lo mismo que usar el comando CLS sin un parámetro de parte (y por eso decimos que es lo prescrito para omisiones).

RECOLoreando

Es posible volver a cambiar el color de cada punto de imagen dentro de una ventana dada usando el comando RECOL. Los 8 parámetros de esta sentencia RECOL especifican un color **sólido** al que se desea cambiar los colores existentes; y en el orden normal para color.

Por tanto, la forma general de la sentencia RECOL será:

RECOL #N, C1,C2,C3,C4,C5,C6,C7,C8

El primer parámetro define el color a 'pintar' sobre todos los puntos en negro; el segundo afecta a todos los azules, y así sucesivamente. Por tanto, para cambiar a rojo todos los puntos en ciano, en la ventana #7, y dejar todos los otros, deberá teclearse la siguiente sentencia:

RECOL #7,0,1,2,3,4,2,6,7

Ensaya para ver el efecto.

Atributo de parpadeo

Se puede dar otro **atributo** a la imagen -puede hacerse que parpadee- mediante el comando FLASH que es sólo efectivo en el modo de baja resolución. Después de haberse ejecutado FLASH 1, todo lo expuesto a través del canal prescrito, #1, a partir de ese momento parpadeará. Y continuará con el parpadeo hasta que se le instruya mediante FLASH 0. También puede incluirse opcionalmente un número identificativo para la ventana, al igual que para otras sentencias de salida. Por ejemplo, teclea:

FLASH #9,1

y todo lo que se exponga a través de esa ventana parpadeará ahora.

Los caracteres parpadeantes no deben usarse para 'sobre-escribir' encima de ellos, ya que la combinación resultante es impredecible. Ensáyalo y verás el efecto.

El resultado pueden ser extraños patrones de caracteres espurios.

Algunas funciones adicionales en la exposición de caracteres

El **tamaño** de los Caracteres expuestos puede alterarse usando la sentencia CSIZE, que acepta dos parámetros en la forma siguiente:

CSIZE anchura,altura

El efecto real de esta instrucción varía con el modo. La anchura puede ser 0, 1, 2 ó 3, y la altura 0 ó 1. Los tamaños prescritos como normales son:

CSIZE 0,0 para MODE 4
CSIZE 1,0 para MODE 8

Ensaya con el siguiente programa y luego cambia el **modo** y ejecútalo de nuevo -verás que cambiando el modo no se vacía el programa de la memoria, incluso aunque se borre la imagen. Este programa te mostrará los tamaños disponibles para cada modo:

```
10 FOR W = 0 TO 3
20 FOR H = 0 TO 1
30 CSIZE W,H
40 PRINT"ESTE ES CSIZE"!W!","!H
50 NEXT H
60 NEXT W
```

La exposición puede también verse reforzada colocando una **banda** de cualquier color como fondo de los caracteres. Para eso se usa la sentencia STRIP, que acepta como parámetros un número de canal y los parámetros normales de color. La forma general es pues:

STRIP #N,color

El canal prescrito para omisiones es la ventana de salida. Teclea lo siguiente:

STRIP #0,2

Eso destacará con un fondo rojo todo lo que se exponga en la ventana de comando. Observa, cuando tecleas, que sólo los caracteres expuestos están sobre una banda roja, con el resto de la ventana todavía en su color normal de papel. Ensaya usando todos los parámetros de color al igual que para BLOCK para ver el efecto que consigues. Una sentencia PAPER anulará el color de STRIP, restituyéndolo al color del papel; y tendrás que volver a reejecutar STRIP si requieres una banda de fondo.

Puedes afectar también la manera en que se exponen los caracteres mediante el comando OVER para superposición. Normalmente, cuando se expone un carácter, la posición que ocupa el carácter queda borrada al situar el otro sobre él. Usando OVER puedes hacer que se expongan caracteres sobre los que ya hubiera en pantalla y sin borrarlos. El efecto es difícil de ver en una TV, y debieras experimentar sólo con OVER 1 y OVER -1 para ver los efectos. Teclea el siguiente programa:

```
10 CLS:CSIZE 3,1
20 FOR I =0,1,-1
30 OVER 0
```

```
40 AT 0,0:PRINT "OVER"!I!"TECLEA CARACTERES"
50 INPUT!C$
60 OVER I
70 AT 5,5:PRINTC$
80 NEXT I
90 GOTO 20
```

Cuando pases el programa, la línea superior te informará del estado corriente del parámetro OVER. Comienza introduciendo unos cuantos 1s cuando se te pide ingreso de datos. Luego la parte de arriba de la pantalla te mostrará que estás en OVER 1. Ahora introduce algunos 2s. Como puedes ver, se **exponen sobre** los 1s sin quitarlos previamente. Lo siguiente que se te pide es para OVER -1, y puedes ver que el color de lo tecleado cambia y la exposición se hace muy confusa. Ensaya de nuevo con el programa, pero simplemente pulsando RETURN en lugar de teclear ningún carácter. Eso permite que el modo OVER cambie a 1. Pulsando RETURN de nuevo pasará a -1, y puedes teclear en este modo con un papel claro. Cuando ensayes los efectos, verás que 0 es la escritura normal, 1 la sobre-escritura y -1 la infraescritura en un color que contraste. Ensaya con el programa con cambios en el color de tinta y de papel, e incluyendo algunos cambios en la banda que destaca. Para ver los efectos de la sobre-exposición en pantalla apropiadamente, es esencial un monitor de video.

El último atributo de exposición disponible es UNDER que **subraya** todo lo expuesto en el canal especificado después de haber ejecutado dicha sentencia. Ensaya:

```
UNDER 1
```

antes de jugar con el programa anterior. Eso pone en marcha el 'conmutador' de subrayar y lo deja hasta que se ejecute de nuevo UNDER 0. Esta sentencia acepta el usual #N antes del valor 1 ó 0, para especificar el canal sobre el que se aplica el subrayado.

Gráficos

Hasta ahora nos hemos encontrado con dos sistemas de coordenadas. El primero es el sistema de coordenadas asociado con la sentencia AT, que tiene su **origen** o punto cero en la esquina superior izquierda de la ventana de salida y en que los puntos de imagen son caracteres medidos a partir de los márgenes izquierdo y superior. El segundo es el sistema de coordenadas de **pixels**, también con su origen en la esquina izquierda, pero esta vez con puntos más pequeños, cada uno igual a 1 pixel.

Las siguientes secciones tratan con el sistema de coordenadas para gráficos. Este sistema puede utilizar **escala** como el papel de dibujar, y tú puedes decidir cuántos puntos se consideran verticalmente, y eso automáticamente ajusta horizontalmente el número para mantener el sistema cuadrado. El origen está en la esquina inferior izquierda, que es el lugar convencional para el papel milimetrado. La escala para los **ejes** puede cambiarse luego a voluntad y todo lo dibujado después de dar una nueva escala será mayor o más pequeño, y estará en diferentes lugares de acuerdo con lo anterior.

El origen para las funciones de dibujar también puede cambiarse a cualquier otro punto, o a la posición corriente del cursor dentro de la ventana. Si las coordenadas se miden a partir del cursor, se denomina **modo relativo.** Todo lo dibujado en una ventana se hace usando el color corriente de INK para dicha ventana, que puede alterarse como ya explicamos anteriormente.

Los colores se tratan exactamente igual que para los puntos de imagen, y puede especificarse cualquier ventana para salida de información. Trataremos primordialmente con la ventana de salida normal. El valor prescrito para omisiones en el sistema de coordenadas para gráficos, es el de 100 puntos verticalmente.

Funciones de gráficos - Pintando puntos

Si has estipulado algún parámetro no prescrito en las últimas secciones, debes restaurar la máquina a sus condiciones iniciales, y comenzar a partir de ahí para experimentar con gráficos.

Se pueden trazar **puntos** usando la sentencia POINT con respecto al origen en la esquina inferior izquierda de la pantalla. Más de un punto se puede especificar en la sentencia usando la palabra clave TO, para indicar el punto **al** que debe saltarse a continuación. La forma general de la sentencia es:

POINT #N, lista de puntos (X,Y) separados por TO

La lista de puntos contiene parejas de coordenadas X e Y para identificar el punto, siendo X la medida horizontalmente, e Y la verticalmente, como es usual. #N está prescrito ante omisiones que sea la ventana de salida normal #1. Ensaya lo siguiente:

```
10 CLS
20 POINT 2,2
30 POINT 4,4 TO 10,10 TO 40,40
```

La primera sentencia deja la pantalla en el color del papel, que es rojo. La segunda sentencia pinta un punto cerca del origen usando el color de INK. La tercera sentencia pinta tres puntos de manera que estén situados en una línea recta, que forma aproximadamente 45 grados. La exactitud de ese ángulo de 45 grados dependerá de la TV o del monitor que estés usando.

La sentencia POINT pinta puntos individuales y a medida que lo hace mueve el **cursor de gráficos** a la posición pertinente. La posición corriente del cursor de gráficos es pues la del último punto pintado por cualquiera de las sentencias de gráficos. Eso es diferente a lo ocurrido con el cursor de textos, que se ve afectado por las sentencias PRINT y AT. Si tecleas PRINT "X" verás que se expone en la esquina superior izquierda que es el punto de origen para el cursor de texto. Repitiendo esta sentencia desplazarás el cursor de textos a lo largo de la línea en la manera normal. El cursor de gráficos no se ve afectado como veremos en breve.

La sentencia POINT también puede usarse de manera **relativa** a la posición corriente del cursor de gráficos, en lugar a la del origen de pantalla.

La forma de la sentencia que lo consigue es:

POINT_R

Esta nueva sentencia se usa exactamente como antes, pero pintará el punto a partir de donde quiera que esté el cursor. Por ejemplo, teclea:

POINT_R 4,4

Eso colocará un punto desplazado 4 unidades a lo largo y 4 hacia arriba a partir del último punto pintado, y el cursor de gráficos quedará situado en estas nuevas coordenadas.

CURSOR

La sentencia CURSOR puede usarse para mover el cursor de textos hasta algún punto relativo al cursor de gráficos. Veremos en breve, cómo eso permite que un modelo gráfico sea asociado con la exposición de caracteres. La sentencia CURSOR acepta los siguientes parámetros:

CURSOR #N, X,Y(gráficos), X,Y(pixel)

El primer par de coordenadas está en el sistema de gráficos; el segundo par es relativo al primer par, y da la posición del pixel superior izquierdo del 'cuadratín' correspondiente a un carácter en el cursor de texto. Por ejemplo, usa CLS y NEW y luego teclea lo siguiente:

10 POINT 40,40 TO 40,60 TO 60,60 TO 60,40

Ejecutándolo se logrará trazar cuatro puntos en los vértices de un cuadrado, y se deja el cursor en 60,40. Observa que se ha usado un número de línea para esta sentencia, aunque pudieramos simplemente haberla tecleado directamente. Si usas un número de línea, y cometes un error de tecleado que no detectas, siempre puedes editar la línea sin tener que volverla a teclear. Eso es una clara ventaja en líneas largas como ésta.

Si quisieras exponer las coordenadas de los puntos pintados anteriormente, escribiendo los valores en sitios adecuadamente cerca de dichos puntos, te sería muy difícil usar la sentencia AT para situar el cursor de texto correctamente. Podemos sin embargo, usar la sentencia CURSOR antes de la PRINT para hacer que la exposición comience a partir de un determinado lugar. Ensaya por ejemplo con:

CURSOR 60,60,2,5

Eso situará la esquina superior izquierda del cuadratín del carácter desplazado (2,5) puntos a partir de la posición de gráficos (60,60). Ahora ensaya con el siguiente comando:

PRINT "(60,60)"

Eso rotulará el punto perfectamente. Ahora ensaya lo mismo para los otros puntos hasta que puedas situar los números fuera del cuadrado en el lugar justo y correcto. Ensaya escribiendo la letra O en la posición central del cuadrado, empleando este método.

Si sólo se especifican dos coordenadas en la sentencia CURSOR, se toman como relativas al sistema de coordenadas de pixels en la pantalla con origen en la esquina superior izquierda. Así, usando CURSOR, puede comenzarse la exposición de caracteres en cualquier posición gráfica de cualquiera de las ventanas, y no estamos restringidos a las filas y columnas normales para exposición de textos vistas hasta ahora.

Trazando líneas

LINE se usa exactamente igual que POINT, pero rellena una raya de puntos entre cada par de coordenadas que se mencione separada por la clave TO. Como es usual, puede aceptar un número de canal. Ensaya lo siguiente:

```
LINE 40,40 TO 60,60
```

El recuadro trazado usando puntos puede ahora completarse usando LINE:

```
10 LINE 40,40 TO 40,60 TO 60,60 TO 60,40 TO 40,40
```

Observa que el conjunto de puntos está ahora cerrado; si dejas fuera el último par de coordenadas anterior, la línea inferior del recuadro no estará presente.

El siguiente programa traza una serie de triángulos de tamaño y color aleatorios, comenzando en (100,75) y desplegándose en abanico hacia el origen.

```
10 CLS
20 REP PATTERN
30 INK RND(8)
40 LINE 100,75 TO RND(50),RND(50) TO
   RND(50),RND(50) TO 100,75
50 END REP PATTERN
```

Algunos efectos impresionantes pueden producirse usando la función RND para cambiar el valor de INK y los parámetros de posición en un programa tan simple como éste.

Círculos y Arcos

Las sentencias CIRCLE y ARC pueden usarse para trazar dibujos circulares en la pantalla con respecto al origen de gráficos. La sentencia fundamental de **círculo** acepta los siguientes parámetros:

CIRCLE #N,coordenadas del centro,radio

Ensaya tecleando:

CIRCLE 50,50,10

Eso dibuja una circunferencia de radio 10 centrada en (50,50) en la ventana de salida. La sentencia CIRCLE también puede trazar diversos círculos separando varias series de parámetros por **punto-y-coma.**

Aparte del número de canal, se pueden añadir dos parámetros más que convierten la circunferencia en una elipse. El primero de ellos define la **excentricidad** de la elipse, y el segundo el ángulo en radianes entre el eje menor y la horizontal. La excentricidad de la elipse es la razón de la longitud del eje menor a la longitud del eje mayor. Si la excentricidad es 1 (que es lo prescrito para omisiones) se dibujará una circunferencia. Si la excentricidad es 0, normalmente se dibujará una línea recta con la orientación definida por el parámetro ángulo. Sin embargo, eso no parece funcionar en las primeras máquinas QL. Teclea lo siguiente:

CIRCLE 50,50,10,0.5,1

Eso da una elipse que encaja dentro del círculo anterior, mostrando que el parámetro radio es la longitud del semieje mayor. Las primeras máquinas trazarán algunos tamaños de elipses con la orientación incorrecta. Debes experimentar por ti mismo para apreciar el efecto de usar diversos ángulos y radios antes de incluir esta sentencia en un programa.

El siguiente programa traza dos conjuntos de elipses concéntricas de diferente color para ilustrar la sentencia.

```
10 CLS
20 INK RND(8)
30 CIRCLE 75,60,56750*RND,0.5,
   2*-1^RND(1 TO 2)
40 GOTO 20
```

La sentencia de círculo dibuja elipses de diversos tamaños y colores, con orientación de, o bien -2 radianes, o bien +2 radianes. Lo que se usa depende de un truco matemático. Si la RND de la línea 30 da 1, entonces -1 se eleva a la potencia 1 (véase el capítulo anterior) y eso es igual a -1, que luego al multiplicarse por 2 da los -2 radianes. Si la RND da 2, entonces -1 se eleva a la potencia 2, dando +1; lo que proporciona los +2 radianes. Este programa también muestra la velocidad con que pueden operar las sentencias de gráficos.

La sentencia CIRCLE también está disponible para ser usada con coordenadas **relativas** al cursor de gráficos usando la versión CIRCLE_R. Sin embargo, eso parece dar error en las primeras máquinas, y deberás experimentar por ti mismo para encontrar si funciona en la tuya.

También es posible trazar solamente un **arco** circular, en lugar de la circunferencia completa, usando la sentencia ARC. Además del habitual número de canal #N, la sentencia ARC acepta tres parámetros. Debes especificar el punto de comienzo, el punto de finalización y el ángulo subtendido al centro del círculo del cual el arco es una parte. La forma general es la siguiente:

ARC #N,punto comienzo TO punto final,ángulo

Para percibir el efecto de esta sentencia, teclea lo siguiente:

```
10 CLS
20 INK 0
30 AT 0,0:PRINT"TECLEA ANGULO"!!
40 INPUT ANGLE
50 ARC 30,50 TO 60,50,ANGLE
60 GOTO 30
```

Cuando ejecutes este programa, te encontrarás que sólo funcionará con una cierta banda de valores para el ángulo. El ángulo está en **radianes** y sólo puedes esperar que su valor varíe entre 0 y 2 pi radianes, porque esa es la posible gama de valores para el ángulo que el arco subtiende en el centro del círculo. Además, en las primeras versiones del QL, algunos ángulos producían efectos extraños y el programa anterior te los mostrará. Ensaya tecleando un 1 para comenzar.

Eso producirá un arco por debajo de los dos puntos, ya que el círculo se traza **en sentido contrario** a las agujas del reloj, a partir del punto de comienzo y hasta el punto final. Si quieres que el arco esté situado por encima de los extremos, debes intercambiar los puntos especificados en la línea número 40. Si quieres usar esta sentencia, deberás experimentar con ella usando este programa.

Dibujos PLENOS de color

Las figuras 'cerradas y convexas' pueden **rellenarse** con el color de INK usando la sentencia FILL. Actúa como una llave conmutadora, que se pone mediante FILL 1 antes de dibujar la figura a rellenar y que se quita mediante FILL 0 después de haber efectuado el dibujo. Por ejemplo, ejecuta lo siguiente:

```
10 CLS
20 INK 7
30 FILL  1
40 CIRCLE 30,70,10
50 FILL 0
60 FILL 1
70 INK 4
```

```
80 CIRCLE 60,70,10
90 FILL 0
100 CIRCLE 90,70,10
```

La línea 30 pone en marcha la función FILL, y el círculo de la línea 40 queda **pleno** de blanco. Luego se quita la función en la línea 50 y se vuelve a poner para rellenar el siguiente círculo de color verde. Vuelve a quitarse de nuevo para dibujar la última 'circunferencia'. Ensaya quitando las líneas 50 y 60 para ver el efecto de permitir que permanezca puesta la llave FILL a lo largo del proceso.

Si estás usando FILL para una figura cualquiera producida, digamos, usando una múltiple sentencia LINE, entonces la figura debe ser **convexa.** Ejemplos de figuras convexas son triángulos, cuadrados y circunferencias. Si la figura es 'reentrante' (cóncava) tal como un cuadrado con una muesca sobre él, la sentencia FILL no funcionará correctamente en general. Estas figuras reentrantes deben trazarse como figuras convexas separadas y rellenarse separadamente.

Cambio de ESCALA

Como explicamos antes, la escala prescrita para omisiones es de 100 puntos verticalmente. Eso significa que una línea vertical de longitud 100 se extenderá desde la parte superior a la inferior de la pantalla. Este valor de **escala** puede cambiarse para trazar diagramas de datos con cualquier banda de valores. La sentencia SCALE también puede usarse para fijar la posición **origen** de coordenadas. La forma general es:

SCALE #N, número de escala,(-)origen coordenadas

Estos son algunos ejemplos de la sentencia:

SCALE 10,-50,-50	fija escala a 10, con origen en 50,50
SCALE 0.8,-0.2,-0.2	fija escala a 0.8, con origen en 0.2,0.2

Las coordenadas del origen tienen que ser negativas y especifican posiciones con respecto a la esquina inferior izquierda de la pantalla. Ensáyala en tu máquina para ver si tu versión requiere coordenadas negativas o positivas para el origen.

Estas sentencias de escala pueden comprobarse dibujando respectivamente los siguientes círculos:

CIRCLE 10,10,10

y

CIRCLE .1,.1,.05

Cada uno de ellos dará una figura razonablemente circular.

Un ejemplo

Como ejemplo de algunas de las funciones anteriores, el siguiente programa abre dos ventanas diferentes en la pantalla. Usa una para exposición normal de caracteres, y la otra en el modo gráfico para dar parejas de círculos aleatorios y corrimientos de un punto de imagen cada vez. El efecto total es uno de los dos procesos ocurriendo simultáneamente. Sin embargo, una inspección concienzuda muestra que son secuenciales, ya que se mueven alternativamente. Tecléalo y ejecútalo como demostración.

```
10 OPEN #5,SCR_150X150A50X40
20 INK #5,1
30 PAPER #5,4
40 CLS #5
50 BORDER #5,4,6
60 OPEN #9,SCR_190X170A250X30
70 PAPER #9,3
80 CLS #9
90 BORDER #9,4,0
100 N=0
110 N=N+1
120 PRINT #5,"NUMEROS:"
130 PRINT #5,N
140 INK #9,RND(7)
150 CIRCLE #9,25,75,RND(25)
160 CIRCLE #9,50,75,RND(20)
170 SCROLL #9,1
180 GOTO 110
```

Debes experimentar con este programa y modificarlo para conseguir efectos diferentes. Intenta **solapar** las dos ventanas, por ejemplo.

Puedes apreciar que el sistema de coordenadas gráficas trabaja igualmente bien en esta pequeña ventana como lo hace en la ventana completa de salida. Recuerda que la escala comienza con 100 puntos verticalmente, de manera que cualquiera que sea el tamaño físico de la ventana, está dividido en 100 filas de gráficos verticales. Sin embargo, en una pequeña ventana no serás capaz de distinguir los 100 puntos de imagen separados verticalmente.

Gráficos con Tortuga

Este sistema de gráficos usa la pantalla en forma muy parecida a la de una **trazadora** en papel (plotter). Debes imaginarte un 'chisme' controlable que lleva una pluma por toda la pantalla y es lo que denominamos **tortuga.** Los comandos disponibles para esta tortuga hacen que se mueva una determinada distancia según una determinada dirección, y apoyando o levantando su pluma del papel para dejar o no dejar una **estela** sobre él, a medida que se desplaza.

La tortuga comienza a partir del origen de gráficos y apuntando hacia el borde derecho de la pantalla, con su pluma levantada. Los comandos para la tortuga son:

PENDOWN	para que **baje** la pluma
PENUP	para que **suba** la pluma
MOVE	para que se **mueva** una distancia
TURN	para que **gire** un ángulo relativo
TURNTO	para que **gire hacia un rumbo** determinado

Cada uno de estos comandos puede tener opcionalmente el número de canal #N. Las primeras dos sentencias controlan la pluma que cuando está bajada deja una estela del color corriente de INK, a medida que se mueve la tortuga. La tercera sentencia necesita un parámetro de **distancia** y hace que se mueva la tortuga a lo largo del **rumbo** que tiene en ese momento, y de acuerdo con la escala de gráficos que esté en operación. La sentencia TURN hace que gire el rumbo llevado por la tortuga un determinado ángulo **en grados.** TURNTO permite que el rumbo sea estipulado a un valor **absoluto** en grados. Los ángulos positivos se miden en sentido contrario a las agujas del reloj y en ángulos medidos a partir de la dirección horizontal que apunta hacia la derecha, en la manera convencional. Para ver cómo funciona, debes experimentar con ángulos dados por ti mismo.

Para comenzar a partir de cero, restaura las condiciones iniciales del ordenador y ensaya lo siguiente:

```
CLS
PENDOWN
MOVE 10
```

CLS es esencial si estás usando una TV, a no ser que fijes INK a algún otro color distinto de blanco. MOVE crea una estela de 10 unidades gráficas a lo largo de la parte inferior de la pantalla. Ahora teclea:

```
TURNTO 45
MOVE 10
```

Eso hará que el rumbo señale hacia el nordeste, y luego avance 10. Usa TURN para girar otros 45 grados más en la dirección positiva, y haz que se mueva de nuevo. ¿Puedes decir qué es lo que hará la tortuga antes de que pulses RETURN?

Este sistema de gráficos es útil para algunos juegos, así como una salida para experimentos sobre la inteligencia de la máquina. Una aplicación típica sería la solución al problema de encontrar el camino a través de un laberinto.

Sonido

El QL tiene una sentencia para generación de sonido muy avanzada, BEEP (que originariamente era simplemente **pite)** que acepta hasta 8 parámetros para conseguir el efecto completo.

Hay además otra función Booleana de clave BEEPING, que entrega el valor TRUE (distinto de cero) si el QL está en ese momento 'pitando', y el valor FALSE (igual a cero) en los demás casos.

El uso más simple de la sentencia BEEP es el de producir un único **tono** durante un tiempo especificado. Eso se consigue usando sólo dos parámetros, tal y como sigue:

BEEP duración,tono

La duración es una variable entera, y tiene como cota superior la de 32767. Esa es la cantidad de períodos de 72 microsegundos, y por tanto este límite representa justamente poco más de 2.3 segundos. El valor 0 para la duración pondrá en marcha la emisión del sonido y durará hasta que se ejecute BEEP sin una lista de parámetros. El tono es un número de 0 a 255, que va de agudo a grave respectivamente. Ensaya lo siguiente:

BEEP 32000,127

Eso da un pitido de unos dos segundos, con una nota que es la media de la gama disponible. Ensaya cambiando esos números para experimentar con las diversas notas. Si usas 0 para la duración, teclea BEEP para detener el sonido cuando lo desees -no serás capaz de pararlo provocando una **ruptura**, mediante BREAK.

El siguiente parámetro que puede añadirse a la lista especifica otra **nota.** El sonido entonces alternará entre esa y la primera nota que se haya especificado. La banda de parámetros para esta nota es la misma que para la primera. Además, pueden especificarse otros dos parámetros adicionales para determinar el **'gradiente'** de estos cambios alternativos. Los dos gradientes tienen las siguientes bandas estipuladas:

Gradiente 1	0 a 15
Gradiente 2	-8 a 7

El siguiente programa te permitirá ensayar los efectos obtenibles:

```
10 INPUT "1ER TONO"!P1
20 INPUT "2DO TONO"!P2
30 INPUT "1ER GRAD."!G1
40 INPUT "2DO GRAD."!G2
50 BEEP 0,P1,P2,G1,G2
60 PAUSE
70 BEEP
80 GOTO 10
```

Cuando pases este programa serás capaz de introducir los parámetros por ti mismo. La línea 50 pone en marcha el generador de sonido, y luego la sentencia PAUSE hace que el programa se detenga en ese punto hasta que se pulse cualquier tecla. BEEP en la línea 70 quita la emisión del sonido y se repite el programa. Es un método conveniente para comprobar la gama de efectos sonoros posibles con la función BEEP. También puedes usar la sentencia FOR para escalonar, digamos por ejemplo, los parámetros de GRAD a través de toda su banda de valores y para una pareja de notas dada. Es un buen ejercicio de programación, y debieras comenzar con las notas de tono 50 y 150 respectivamente.

Los parámetros finales admitidos en la sentencia BEEP son los que hacen que la calidad del sonido sea controlable. Desafortunadamente hay muy poca información disponible sobre la manera en que funcionan, y simplemente es un asunto de ensayo y error por ti mismo, y de llevar cuidadosamente anotado lo que vas haciendo y consiguiendo. De esa manera, aprenderás a usar esa sentencia con toda su potencia completa.

Estos parámetros finales se denominan WRAP (envolvente), FUZZY (reverbero) y RANDOM (ruido aleatorio). Los dos primeros tienen una banda estipulada de 0 a 15, y el siguiente una banda de -32768 a +32767. Debieras añadirlos al programa anterior al final de la sentencia BEEP. Como un ejemplo de la complejidad de esta sentencia BEEP, teclea lo siguiente y escúchalo cuidadosamente durante 15 ó 20 segundos como mínimo. Podrás observar que claramente hay diversos procesos funcionando dentro del sonido.

```
BEEP 0,50,150,2,-1,8,8,
```

Resumen

- La pantalla del QL tiene una resolución máxima en gráficos de 512 pixels a lo ancho por 256 hacia abajo. En MODE 8, se reduce a 256 x 256, pero con 8 colores. En MODE 4, sólo hay 4 colores disponibles, con la imagen completa de 512 x 256 motas.
- El sistema de coordenadas de pixels tiene su origen en la esquina superior izquierda, mientras que el sistema de gráficos lo tiene en la esquina inferior izquierda. La **escala** del sistema de gráficos puede estipularse usando SCALE. La pantalla siempre se ajusta automáticamente para asegurar que el sistema de coordenadas es un cuadrado.
- BLOCK se usa para trazar rectángulos en el sistema de coordenadas de pixels.
- El color puede ser sólido o tramado con dos colores que contrasten, y puede ser usado en PAPER o INK para fijar los colores de fondo y de frente respectivamente.
- Se pueden abrir nuevas ventanas en la pantalla usando OPEN y un número de canal (de cauce) precedido por #. BORDER se usa para darle un 'marco' a una ventana. CLOSE se utiliza para **cerrar** otra vez la ventana.
- El contenido de una ventana puede ser total o parcialmente borrado, corrido verticalmente u horizontalmente (PANorámica), un determinado número de pixels.
- CSIZE puede usarse para cambiar el **tamaño** de los caracteres.
- OVER provoca la **sobre**escritura. STRIP da una **banda** del color del fondo para destacar los caracteres expuestos y UNDER los **subraya**. FLASH hace que los caracteres **parpadeen** regularmente.

- POINT y LINE se usan para trazar **puntos** y **líneas** en el sistema de coordenadas de gráficos. CIRCLE dibuja círculos y elipses; ARC traza partes -arcos- circulares o elípticos.

- CURSOR mueve el cursor de exposición de caracteres hasta una posición dada en el sistema de gráficos para poder escribir anotaciones en los dibujos.

- FILL se usa para **rellenar** figuras convexas con el color corriente de INK.

- Los gráficos de **tortuga** permiten que ésta se mueva por toda la pantalla, dejando o no una estela a medida que avanza.

- BEEP se usa junto con una serie de parámetros para emitir un sonido generado internamente por un procesador independiente. BEEPING es la función que entrega un valor cierto si la máquina está emitiendo un sonido en ese momento.

Capítulo Nueve

Microductoras, Ficheros y Dispositivos

Uno de los mayores avances por lo que el QL es tan bien conocido es la inclusión de dos dispositivos periféricos de almacenamiento similares a discos, que denominamos **microductoras** (microdrives). Estas lecto-grabadoras de cinta magnética en cartuchos miniatura conservan aproximadamente 100 K bytes de información cada una, y permiten que se use un **sistema de archivo** para datos y programas. El uso de las ductoras debe atenderse con especial cuidado, y deberán observarse sin falta las precauciones dadas en el manual del QL.

La programación para depositar y recuperar datos se explica, como se requiere para programas que deben manejar información en cualquiera de sus formas. Se dan algunos ejemplos para mostrar cómo funciona el sistema.

La primera parte de este capítulo es todo lo que se necesita si simplemente quieres almacenar y recuperar programas del cartucho. El resto del capítulo lo puedes leer en una fecha posterior, cuando necesites usar **ficheros de datos** con el QL.

Preparación para el uso de las microductoras

Los cartuchos de la microductora que vienen con el QL están metidos dentro de fundas externas de plástico, y deberán quitarse agarrando el extremo plano del cartucho y tirando de la cubierta externa. Cuando lo hagas, verás la cinta magnética real en que se almacena toda la información. **No toques esa cinta con tus dedos bajo ninguna circunstancia.** El cartucho contiene un lazo continuo de cinta **-sin fin-** y la máquina arrastra la cinta, tardando aproximadamente 7,5 segundos en completar un lazo.

No hay en la cinta marcas distintivas que le digan al ordenador dónde comienza, por ejemplo. Toda esta clase de información está magnéticamente grabada en la cinta, y la primera tarea que has de hacer antes de usarlas, es almacenar estos **datos de formato** en la cinta. Hay una rutina especial llamada FORMAT que consigue eso, y la veremos en breve.

La Fig. 9.1 muestra la disposición física de las microductoras en la máquina y da los identificativos que usamos para designarlas. La microductora de la izquierda es la llamada ductora 1 y será mencionada como MDV1 a partir de ahora; la otra es la MDV2. Cada diodo luminiscente (LED) se ilumina cuando se está accesando la ductora, por lo tanto, dando indicación de lo que está pasando.

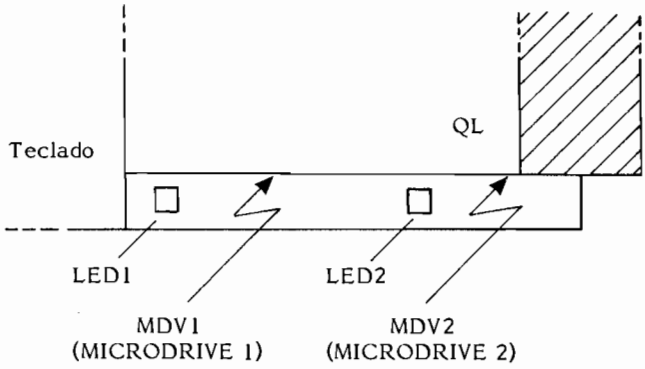


Fig. 9.1. Disposición de microductoras

Todas las funciones y sentencias para las microductoras deben incluir el identificativo MDV1 o el MDV2. Para la mayoría de las actividades, son intercambiables en las funciones que usan las microductoras, pero la MDV1 tiene una función extra. Cuando pulsas F2 o F1 después de restaurar o de poner en marcha, el LED1 se iluminará mostrando que la máquina está intentando accesar la MDV1. Si no hay ningún cartucho presente, la máquina simplemente se va directamente al SuperBASIC, y te permite escribir programas directamente. Sin embargo, si pones en marcha la máquina e insertas dentro de MDV1 uno de los dos cartuchos suministrados con la máquina que contienen el **paquete de programas,** la máquina automáticamente ejecutará ese programa cuando pulses F1 o F2.

Los cartuchos sólo encajan dentro de la ranura de la microductora de una sola manera -no intentes forzar la inserción de un cartucho. Recuerda además que nunca debes poner o quitar la alimentación eléctrica con un cartucho metido en la ranura. Eso puede enviar picos de tensión eléctrica hacia la cinta magnética y destruir la información que haya en ella.

Ensaya pulsando RESET, insertando una cinta en la MDV1 y pulsando F1 o F2 según proceda. El QL cargará en memoria el programa que hayas metido en la cinta de la MDV1, y el LED1 permanecerá encendido hasta que el proceso de carga haya concluido.

La transferencia de programas y datos desde la microductora al ordenador se denomina **carga** (load), y el proceso inverso de transferir información desde el ordenador hasta la microductora es el proceso de **guardar** (save). Justamente habrás auto-cargado un programa a partir de MDV1. Eso significa que tú específicamente no das el comando de carga: la máquina se preocupa de todo el proceso. Este capítulo pretende mostrarte cómo efectuar las operaciones de cargar y guardar programas y datos por ti mismo.

La primera tarea es **formatear** los cartuchos vírgenes que se te suministran -son los que no tienen nada escrito en la parte frontal. Los cartuchos de programas Psion tienen el nombre del programa escrito sobre ellos -déjalos aparte por ahora.

El formateo de un cartucho prepara la cinta magnética para ser utilizable, comprueba la cantidad de espacio disponible para el almacenamiento de información y registra magnéticamente un nombre en la cinta para tus propios propósitos identificativos. El comando FORMAT es el siguiente:

FORMAT MDV1_cinta1

Todos los comandos que guardan relación con los cartuchos de cinta usan la clave identificativa MDV seguida del número 1 ó 2 para distinguir a cuál de las dos microductoras afecta, luego el guión de subrayado y luego un **nombre** o título de alguna clase. El espacio en blanco después de la clave FORMAT es esencial, y no debe haber ningún espacio en la palabra que va detrás de FORMAT y representa el título o nombre. Este comando **formateará** la cinta magnética situada en MDV1, y registrará magnéticamente el título o nombre que la hayas dado: cinta1. Este nombre aparece en la pantalla al utilizar subsiguientemente esa cinta, con lo que te permite identificarla. Prueba a insertar un cartucho en la MDV1, y teclear lo anterior. Cuando pulses ENTER, el piloto luminoso 1 se encenderá y podrás oir cómo dentro de la ductora se produce el arrastre de la cinta. Eso continúa hasta que se haya establecido el formato de la cinta, en lo que sólo tarda unos cuantos segundos. Cuando haya concluido, aparece un mensaje en la pantalla que te indica:

216/218 sectors

Eso te dice que hay 216 **sectores** disponibles para uso. Un sector tiene una capacidad de 512 bytes, por lo que 216 sectores te dan una capacidad total de almacenamiento de 110592 **bytes.** La notación usual para la capacidad de almacenamiento es en Kilo**bytes,** y 1 K es igual a 1024 **bytes.** Por lo tanto, 216 sectores son 108 K de capacidad de almacenamiento (y un **byte** es simplemente un carácter, y aquí es 1 **octeto** = 8 bits).

Coloca otro cartucho en la MDV2, y teclea:

FORMAT MDV2_cinta2

Así usarás la otra ductora. Ahora debieras repetir la operación de formateo para cada cinta aproximadamente seis veces para 'asentar' los cartuchos. Deberás hacerlo con todas las cintas nuevas que tengas.

Los otros comandos que afectan a la microductora tienen una sintaxis similar a FORMAT, como veremos.

Comando para el DIRectorio

En la cinta de los cartuchos se almacena magnéticamente información según **bloques,** que pueden contener programas o datos. Cada bloque es lo que se denomina **un fichero** y está rotulado con un **identificador.** Estos identificadores son los llamados **nombres de ficheros.** La lista completa de los nombres de los ficheros asignados a los almacenados en la cinta se graba en un área especial de cada cinta que se denomina **directorio,** y constituye el "CATálogo" de lo que la cinta contiene.

Para revisar el directorio de una cinta, puedes usar el comando DIR, que tiene la siguiente sintaxis:

DIR MDVN_

siendo N un 1 o un 2 según la microductora afectada. Como pueden añadirse más microductoras a la máquina, tendrán asignados números a partir del 3 en adelante. Prueba a insertar una de las cintas Psion en la MDV1, y teclea lo siguiente:

DIR MDV1_

Eso hará que la microductora comience a arrastrar y leer la cinta, y después de una breve demora aparece un mensaje en la pantalla con el nombre de la cinta, el número de sectores libres y una lista de los ficheros registrados en el directorio.

Si almacenaste el programa de juego dado en un capítulo anterior, serás capaz de usar DIR para comprobar que está almacenado como un fichero en esa cinta.

Dispositivos periféricos

La microductora es un ejemplo de los diversos periféricos de entrada/salida que están controlados por el QL. La palabra dispositivo periférico (peripheral device) en SuperBASIC quiere indicar uno de los siguientes identificativos y equipo:

SCR	- la pantalla
CON	- la consola, que incluye la pantalla y el teclado
MDVN	- microductora número N
SERN	- portal serie número N
NET	- la red (network) de comunicaciones

Hemos visto cómo puede usarse OPEN sobre el equipo SCR para definir una ventana especial en la pantalla. Recuerda que OPEN definía un #N (número de canal o de cauce) que era usado por PRINT y otras sentencias para comunicar información a través de ese canal, y en este caso hacia la ventana definida. La misma filosofía se usa para ponerse en contacto con otros equipos periféricos. Por ejemplo, puedes almacenar información en una de las microductoras usando una sentencia OPEN para asociarle un número de canal, y luego transferir información desde y hasta él mediante las sentencias adecuadas (al igual que PRINT expone -pone fuera- información en el cauce asociado a una ventana).

Algunos de estos dispositivos mencionados tienen valores prescritos para omisiones que son los normales para fijar sus características, y debieras consultar el manual del QL si estás interesado en experimentar con ellos.

El equipo de **portal serie,** identificado por SERN, siendo N 1 ó 2, es un canal que le permite comunicarse con el exterior a través de una forma especial de transmisión de datos que se denomina **serie** (en contraposición a **paralelo).**

Su descripción cae fuera del ámbito de este libro, pero con el cable correcto y con el uso de la sentencia BAUD (para fijar la **rapidez** en Baudios de la transmisión) puede usarse para trabajar con una impresora. También aquí, la sentencia normal PRINT con un número # de un cauce abierto hacia el equipo SER, basta para exponer datos a través de la impresora.

Guardando, cargando y ejecutando programas

Para experimentar con las microductoras, necesitarás tener un programa que puedas **guardar** y posteriormente **cargar** en memoria. Teclea por ejemplo el siguiente:

```
10 REM TEST_PROG
20 CLS
30 PRINT"TEST_PROG"
40 PRINT"LINEA 40"
50 PRINT"LINEA 50"
60 PRINT"LINEA 60"
70 PRINT"LINEA 70"
80 PRINT"LINEA 80"
90 PRINT"LINEA 90"
```

Ahora usa el comando SAVE para llevar el programa desde la memoria hasta la cinta y dejarlo **guardado** en ella, y el comando LOAD para hacer que lo traiga desde la cinta hasta la memoria y lo deje **cargado** en la memoria. Necesitarás colocar una cinta ya formateada en la MDV1 para lo que sigue.

Los comandos SAVE y LOAD se aplican en SuperBASIC únicamente a **programas**, y no a ficheros de datos generales. Estos comandos necesitan algunos parámetros para trabajar correctamente. El parámetro más importante es el identificativo de la microductora con la que se va a hacer la operación de transferencia. La forma general del comando SAVE, para **guardar** un programa en cinta, es la siguiente:

SAVE dispositivo_nombrefichero,lista de líneas

Realmente hay dos grupos de parámetros aquí. El primero identifica al dispositivo y al fichero, y el segundo es una lista de los números de línea que van a ser guardados en cinta. El primero de ellos podría ser de la forma:

MDV2_test_prog

No se permiten espacios en blanco dentro de este identificativo, y tendrás que usar **guiones de subrayado** para separar las palabras en el nombrefichero si te conviene. Por ejemplo, TEST PROG no es un nombrefichero válido, pero TEST_PROG sí lo es. Observa que con todos los identificadores en SuperBASIC, ocurre que se tratan como idénticas las letras en mayúsculas y en minúsculas. MDV2 es el nombre de un dispositivo -la microductora 2- y debe estar empalmado al nombrefichero mediante el signo de subrayar.

La lista de líneas es la otra serie de parámetros requerido con SAVE, y se muestra separada del nombre de dispositivo y del nombrefichero por una **coma.** Esta coma puede que no sea siempre necesaria, pero te ahorrará errores si la empleas. La forma de la **lista de líneas** es idéntica a la de la sentencia LIST. Si se omite, se guardará todo el programa, mientras que una lista de líneas separadas por comas, y de bandas de líneas usando el separador TO, guardará partes del programa. Por ejemplo, el siguiente comando (y no lo teclees):

SAVE MDV2_test_prog, TO 40

guardará en la cinta todas las líneas del programa desde la primera hasta la línea 40, inclusive. Recuerda la sentencia LIST, en un capítulo anterior, para ver todas las otras alternativas.

Para ver el programa ejemplo guardado en el cartucho de la microductora, teclea:

SAVE MDV1_TEST_PROG

Esto registrará todo el programa bajo el nombrefichero de TEST_PROG en la cinta. Para ver que efectivamente ha llegado ahí, basta teclear:

DIR MDV1_

Ahora teclea lo siguiente:

SAVE MDV1_TEST_20_60, 20 TO 60

y luego teclea:

SAVE MDV1_TEST_70_90, 70 TO 90

Esto incluirá dos ficheros adicionales en esa cinta, llamados TEST_20_60 y TEST_70_90 que contienen partes del programa tal y como sugieren los nombrefichero escogidos. Haz que te muestre el DIRectorio para ver la lista completa de los ficheros de la cinta, y observar cuánto almacenamiento te queda libre.

Veamos ahora cómo hacer que traiga un programa desde la cinta y lo deje cargado en la memoria, mediante el comando LOAD. Sólo acepta como parámetro el dispositivo_nombrefichero. Cuando se ejecuta un comando de carga -LOAD- debes primero efectuar uno de borrado -NEW- para quitar el que haya en memoria en ese momento, y luego recibir el que se trae desde la cinta. Prueba a listar el programa en memoria en este momento, antes de continuar. Luego teclea:

LOAD MDV1_TEST_20_60

y ahora lista de nuevo el programa. Verás que el que había ha sido sustituido por el programa parcial con líneas desde la 20 a la 60. Ahora haz lo mismo para el otro programa parcial, y luego para el programa global, y así apreciarás cómo se comporta este comando.

La **comprobación de errores** del QL todavía está en acción durante una carga, y cada línea se comprueba en cuanto a la sintaxis correcta, a medida que se trae de la cinta. Si hay un error, se insertará la palabra MISTAKE (equivocación) al comienzo de la línea, y el error aparecerá cuando se ejecute el programa. Eso no es importante para programas que han sido guardados después de teclearlos, ya que esos han sido comprobados entonces.

Sin embargo, es muy útil si la carga se efectúa procedente desde otro dispositivo, o aprovechando un utensilio de programación para crear y guardar programas.

Podemos ahora guardar y cargar **ficheros de programas** en cualquier dispositivo simplemente especificando el nombre del dispositivo. Un ejemplo de eso sería:

```
SAVE SCR
```

que tomaría el programa corriente en memoria, y lo enviaría hacia el dispositivo de pantalla. Y allí se mostraría acordemente - ¡ensáyalo!

Si tuvieras una impresora conectada al portal serie -usando el identificativo SER_ que le corresponde, lograrías que se transfiriera el programa desde la memoria hasta la impresora. Necesitarás usar la sentencia BAUD correctamente para que funcione -para ello estudia el manual de la impresora y del QL.

Quitando ficheros de la cinta

Cuando deseas **suprimir** un fichero de la cinta, puedes usar el comando DELETE que exige el nombre del dispositivo y el nombre de fichero al igual que para las sentencias ya mencionadas. Prueba con DIR para recordarte los ficheros que hay en la MDV1, y luego teclea lo siguiente:

```
DELETE MDV1_TEST_20_60
```

Con eso eliminarás el fichero mencionado, y puedes hacer que te muestre el directorio para ver que lo has conseguido. Sé muy cuidadoso con este comando o llegarás a perder programas y datos importantes.

Puede que hayas notado que algunas veces, al usar las sentencias anteriores, el cursor reaparece antes de que la cinta haya terminado de avanzar. Puedes continuar tecleando en ese mismo momento, incluso aunque la cinta todavía esté avanzando, ya que el teclado es activo siempre que el cursor indica que está preparado.

Cargando, juntando y ejecutando automáticamente

Un programa puede ejecutarse directamente después de su carga si usamos el comando LRUN, que es la abreviatura de LOAD (cargue) y RUN (ejecute). Simplemente especifica el dispositivo y el nombrefichero. Este comando sólo trabajará para un programa completo, a diferencia de LOAD que permite que un programa parcial sea traído a la memoria.

Por ejemplo, prueba con lo siguiente:

```
LRUN MDV1_TEST_PROG
```

Si tuvieras otro programa en memoria antes de teclear esto, lo habrías perdido completamente, ya que LRUN comienza automáticamente ejecutando un comando NEW. De nuevo diremos, que LRUN comprueba que las líneas entrantes desde la cinta están libres de errores sintácticos, y escribe MISTAKE si encuentra alguno.

Si no deseas perder el programa que tienes en memoria al cargar otro desde un dispositivo de almacenamiento, puedes hacer que **congregue** los dos usando el comando MERGE (mezcle, junte). Eso hará que se traigan las líneas del fichero en cinta, y sobre-escriba cualquiera que hubiera con el mismo número de línea en memoria, pero dejando las otras que no coinciden. Para ver cómo funciona, teclea y guarda el siguiente programa bajo el nombrefichero de SOLAPADOR:

```
60 PRINT"SUSTITUYE LINEA 60 VIEJA"
100 PRINT"ESTAS SON LINEAS EXTRA"
110 PRINT"QUE SERAN AGREGADAS A"
120 PRINT"TEST PROG. "
```

Usa DIR para ver los ficheros que hay en la cinta. Ahora teclea NEW para limpiar la memoria de programas, y confirma listando. Trae de nuevo a la memoria TEST_PROG usando LRUN, para asegurarte de que ha llegado. Ahora teclea el siguiente comando:

MERGE MDV1_SOLAPADOR

Eso **congregará** en memoria las líneas de SOLAPADOR y las de TEST_PROG, sobre-escribiendo las que hay en memoria con las recién traídas de la cinta. Si ahora ejecutas el programa, verás ese efecto. Usa LIST para ver el programa resultante de esta operación de mezclado.

Para que se ejecute inmediatamente la versión combinada, puedes usar MRUN, que es abreviatura de MERGE y RUN. Puedes probar con esto tecleando lo siguiente:

MRUN MDV1_TEST_PROG

Eso hará que se traiga TEST_PROG desde la cinta a la memoria, se mezcle con el que haya en memoria, y se ejecute el resultado de dicha mezcla. La única diferencia estará en la línea 60, como puedes ver.

Tanto MERGE como MRUN también comprueban si hay errores sintácticos en el programa traído de la cinta, y los tratan de la misma manera que con LOAD.

Copia de ficheros

COPY se usa para obtener réplicas de un fichero en otro dispositivo o en el mismo. Puede usarse por ejemplo, para copiar un fichero en cinta desde el cartucho en una microductora hasta otro cartucho, desde una microductora hasta otro dispositivo.

Cuando se almacena información en un cartucho de cinta, un bloque especial de información denominado **cabecera**, se almacena primero en la cinta, y va seguido de la información que realmente constitye el fichero.

El fichero total de la cinta es identificado luego para operaciones posteriores mediante esta información de cabecera.

Si deseas transferir un fichero de información hasta un dispositivo que no sea una microductora, es conveniente ser capaz de enviarlo sin esta cabecera, que en caso contrario interferiría en la operación. Esa es la función de la sentencia COPY_N.

Efectuaremos dos copias, usando COPY, para ejemplo del proceso; y la primera será el copiado de un fichero desde una microductora a otra. Para este experimento necesitas un cartucho adicional ya formateado, colocado en la MDV2.

La sintaxis general de COPY y de COPY_N es como sigue:

COPY dispositivo_nombrefichero TO dispositivo(_nombrefichero)

El segundo dispositivo anterior necesitará que se mencione el nombrefichero si es una microductora.

El primer fichero será entonces transferido hasta la segunda ductora, registrado en ella con el nuevo nombrefichero, que puede ser obviamente el mismo que el primero si lo deseas. Ensaya con lo siguiente:

COPY MDV1_TEST_PROG TO MDV2_TEST2

El comando DIR te mostrará que la cinta en la MDV_2 tiene un nuevo fichero denominado TEST2 y que contiene el mismo programa TEST_PROG. Eso también puede usarse para obtener otra copia de TEST_PROG dentro de la misma cinta, si lo necesitas. Mira a ver si puedes hacer eso por ti mismo, llamando a la segunda copia TEST2. Luego haz LRUN para comprobar que todo ha funcionado.

COPY es útil para sacar copias de **respaldo** (back up) de los ficheros importantes. Siempre debes sacar una copia adicional de los ficheros que deseas mantener. Estas copias de respaldo no deberás usarlas a no ser que te falle la copia principal. Y en ese caso, la de respaldo debieras usarla meramente para producir otra copia de trabajo.

Otro ejemplo de COPY sería el de copiar un fichero directamente de la cinta a la pantalla. Puede hacerse en la forma siguiente:

COPY MDV1_TEST_PROG TO SCR

Prueba para ver el efecto. No necesitas usar COPY_N para esta copia, pero usando COPY_N (que quitaría la cabecera) tendría el mismo efecto.

Algunos comandos adicionales

Hay varios comandos que caen fuera del ámbito de este libro, ya que están orientados hacia la programación en código máquina. Son LBYTES, SBYTES y SEXEC. Para usar estos comandos tendrás que comprender cómo está dispuesta la memoria del QL y la clase de información que se almacena en ella.

Estos comandos se usan para **verter** en cinta y **acopiar** de cinta los datos directamente según notación binaria.

Otro comando muy interesante es el comando EXEC, que permite una clase de ejecución de programas que se denomina **multitarea.** Bajo este régimen, se pueden hacer que sean ejecutados **simultáneamente** varios programas en código máquina, cada uno con su propia área de entrada y salida, quizás como ventanas separadas.

Otro comando importante es el comando NET, que permite conectar una **red** (network) de QLs y Spectrums para intercambiarse rápidamente información y compartiendo la memoria.

Ficheros y tratamiento de ficheros

Hay diversas clases de ficheros que pueden conservarse en la cinta de la microductora. Ya hemos visto los **ficheros de programa.** Esta sección trata justamente con **ficheros de datos** SECUENCIALES que pueden ser usados para depositar información y recuperarla posteriormente, mediante un programa en SuperBASIC.

Uno de los más provechosos atributos de un ordenador es su habilidad para elaborar, depositar y recuperar grandes cantidades de datos. Por ejemplo, puede que desees registrar varios centenares de artículos de inventario en una industria, cada uno con su propia identificación y su precio. Cuando se hace una petición de un artículo, puede que quieras poder teclear su número, examinar su precio y tener la descripción completa impresa como recibo de entrega.

Eso entraña el almacenamiento de información en equipos periféricos tales como las microductoras de cinta en cartucho. Cuando se quita el cartucho con el inventario, y se inserta otro, puede que seas capaz de accesar un fichero con los nombres y las direcciones de tus clientes. Necesitarás preparar otro fichero y recuperar los datos registrados en él de una manera particular. Otra aplicación serían experimentos científicos, donde mantendrías en un fichero los resultados de una serie de pruebas que has efectuado. También aquí, la capacidad de almacenamiento es esencial. Esta sección te muestra cómo depositar y recuperar datos de entrada y elaborados dentro de tus programas.

Debes pensar en un fichero de datos como una **retahíla de fichas** separadas cada una de la siguiente por una marca de **avance de línea.** Normalmente, a cada ficha que representa un grupo de datos relativos a un mismo ente, se denomina **registro.** Cada registro puede ser considerado entonces como una sola línea de constantes numéricas y literales, que representan valores de variables. Por ejemplo, en una guía telefónica, un registro -una ficha- constaría de un nombre, una dirección y un número de teléfono. El siguiente abonado -'miembro del fichero'- tendría sus datos reflejados en una nueva ficha, en un nuevo registro. Hay ventajas en almacenar simplemente cada grupo de datos propios de un miembro del fichero, en una línea separada, y por tanto en un registro separado. Verás que la mínima cantidad de información que puede sacarse de un fichero es precisamente un registro.

Eso significa que si quieres desglosar cada dato en caracteres separados, o los consideras cada uno como un registro separado, o usas las operaciones ya vistas para segregarlos de los datos de acuerdo con las posiciones que ocupan dentro de ellos.

Para definir un fichero, se usan las sentencias OPEN_NEW y OPEN_IN, cuya forma general es la siguiente:

```
OPEN_NEW #N,dispositivo_nombrefichero
OPEN_IN  #N,dispositivo_nombrefichero
```

La primera de éstas abre un **nuevo** fichero llamado nombrefichero y contenido en el dispositivo mencionado, y le asigna un número de canal N para establecer un cauce hacia él. Este fichero está disponible para escritura en él. La segunda sentencia abre un fichero existente para **IN**gresar datos en el programa a partir de él ('lectura'). El número de canal se usa en PRINT y en INPUT igualmente que si dichos ficheros fueran ventanas. Cuando se abre para escritura, el nuevo fichero está vacío, y se usa PRINT para **EX**poner datos en él. El proceso de usar un fichero sólo se completa cuando se encuentra una sentencia CLOSE que afecta a ese número de canal.

Hay muchas operaciones internas que se necesitan para archivar datos, y algunas de ellas sólo concluyen al final del proceso. CLOSE **cierra** todas las que haya pendientes.

El siguiente programa escribe un registro dentro de un nuevo fichero llamado DATA1 en la cinta colocada en la MDV1:

```
10 OPEN_NEW #7,MDV1_DATA1
20 PRINT #7, "ESTO ES UNA FICHA"
30 CLOSE #7
```

La sentencia PRINT usada para dejar registros en un fichero depositará en él los datos exactamente igual a como se exponen normalmente en pantalla. Cada línea expuesta es lo que constituye cada registro del fichero.

Después de ejecutar este programa, la sentencia DIR te confirmará la existencia del fichero, y la sentencia COPY te confirma que contiene la información requerida. Ensaya con lo siguiente:

```
COPY MDV1_DATA1 TO SCR
```

Y siempre puedes comprobar de esta manera tan cómoda el contenido de un fichero. Como puedes ver, los mismos comandos que se usan para almacenar ficheros de programa funcionan también para el almacenamiento de ficheros de datos. Recuerda que si intentabas cargar un fichero que contenía líneas erróneas de programa, aparecía una MISTAKE (equivocación). Prueba con LOAD MDV_DATA1 para ver el efecto.

Observa que el número # elegido para un fichero, no se usa dentro del propio fichero. El número de canal es meramente interno del programa que escribe o lee datos del fichero, y puede ser un número diferente en el programa de creación que en el programa de utilización del fichero. Cualquier número hasta 15, puede usarse siempre y cuando no sea igual a los canales prescritos para omisiones tales como la ventana de salida o la ventana de comando.

Para recuperar la información archivada en DATA1 y usarla en un programa, se usa la sentencia OPEN_IN que define el fichero como de Entrada. Ensaya lo siguiente:

```
10 OPEN_IN #5, MDV1_DATA1
20 INPUT #5, A$
30 CLOSE #5
```

Eso abrirá el fichero DATA1 para ingresar en el programa los datos de él, y traerá el primero (y único) registro que forma el fichero, imponiéndolo como valor de A$, y luego cerrará el fichero. Para ver que ha funcionado, teclea PRINT A$.

Si intentas abrir un nuevo fichero para escritura que tiene el mismo nombre que uno que ya está en la cinta, aparece un mensaje de error en la pantalla para advertirte. Eso te muestra que siempre debes crear un fichero nuevo para escribir datos, incluso aunque pudieras preferir **actualizar** los datos de un fichero ya existente. Para añadir datos a un fichero viejo, su información debe primero transferirse a un nuevo fichero, y luego añadir los nuevos datos. Eso es laborioso, pero esencial al tratar con ficheros de **acceso secuencial.**

Cuando más de un dato está almacenado en una línea, al igual que cuando se va a exponer más de un dato, se usan los separadores normales y los datos aparecerán en el registro de acuerdo con la manera normal de exponer los datos de una línea en pantalla. Por ejemplo, el siguiente programa almacena 10 pares de números en dos columnas, separadas ocho espacios en blanco y usando la coma normal como separador:

```
10 OPEN_NEW #6, MDV1_NUMPARES
20 FOR I=1 TO 10
30 PRINT #6, I, I+1
40 NEXT I
50 CLOSE #6
```

Después de teclearlo y ejecutarlo, teclea COPY MDV1_NUMPARES TO SCR para ver la manera en que están archivados los datos. Eso te confirmará que están exactamente igual a como se exponen mediante PRINT en pantalla. Las siguientes líneas de programa, añadidas a las anteriores, impondrán los datos como valor de la variable D$:

```
60 STOP
70 REM
80 REM Ahora recupero la informacion
90 REM
100 OPEN_IN #7,MDV1_NUMPARES
110 FOR I=1 TO 10
120 INPUT #7,D$
130 PRINT D$
140 NEXT I
150 CLOSE #7
```

STOP en la primera línea es meramente para separar los dos programas, de manera que no pase a ejecutarse directamente el segundo cuando termine el primero.

Cuando se ejecuta el segundo programa usando RUN 100, las líneas de cada ficha del fichero se recogen del mismo, y se imponen como valor de D$. Luego se exponen en pantalla una por una.

Para archivar los números separados por un único espacio en blanco, cambia la línea 30 para que sea:

```
30 PRINT #6,I!I+1
```

Los dos programas sólo pueden ejecutarse después de que el fichero actual NUMPARES haya sido quitado de la cinta usando DELETE.

Como puedes ver, los números anteriores son tratados como literales. Es decir, cuando en la línea 30 se **dejan** en el fichero se hacen como numerales; mientras que en la línea 120 cuando se **cogen** del fichero se hacen como literales.

Con los registros de datos en NUMPARES, sólo el primer número de cada par es aceptado para imponer a una variable numérica, ya que un espacio en blanco -que es lo que lo separa del siguiente- se consideraría como el final de cada número y todo lo demás de esa línea es descartado al hacer la operación INPUT. Puedes haber observado que esa es la utilización normal de la sentencia INPUT procedente del teclado.

Para archivar números en un fichero, cada número debe almacenarse separado del siguiente por un avance de línea -constituyendo por tanto un registro, una ficha por sí misma- lo que significa que la sentencia PRINT debe exponer cada número sucesivo en una nueva línea de pantalla. Prueba cambiando la línea 30 para que sea:

```
30 PRINT #6,I  I+1
```

Eso almacenará cada número en una nueva línea, i.e. lo dejará como un nuevo registro. Ahora cambia las líneas 120 y 130 para que sean:

```
120 INPUT #7,D
130 PRINT D
```

Recuerda que NUMPARES debe eliminarse antes de que se ejecuten los dos programas, o el primer programa provocará un error, ya que estás intentando registrar en el DIRectorio un fichero cuyo nombre ya existe en la cinta.

Cuando pases el primer programa, en la línea 10 se creará un nuevo fichero con cada número dejado en una nueva línea, y por tanto en un nuevo registro. Hay 20 números. Puedes verlo usando COPY. El segundo programa, en la línea 100, abre un fichero para ingreso de datos, y solamente coge de él los primeros 10 de esos números, imponiéndolos como valor de la variable numérica D, y luego escribiéndolos en pantalla.

Para coger más de un registro a la vez, pueden colocarse más variables en la sentencia INPUT exactamente igual a lo que ocurre al imponer datos por teclado. Así podrías cambiar las líneas 120 y 130 para que fueran:

```
120 INPUT #7,D,E
130 PRINT D!!E
```

Eso hace que se INPUT y PRINT los 10 pares de números cogidos del fichero NUMPARES, en los que previamente los habíamos dejado.

Como puedes ver, puedes crear ficheros de cualquier clase y configuración usando correctamente las sentencias de salida PRINT y de entrada INPUT.

En este momento te debemos un par de avisos finales. Los primeros manuales muestran variaciones con lo aquí mencionado, por lo que te aconsejamos que ensayes todos los ejemplos para convencerte por ti mismo de los métodos presentados aquí. El otro aviso concierne a las primeras versiones del programa de control del QL. Algunas veces cuando se menciona una MDV en una sentencia tal como DIR, la microdúctora no se pone en marcha, pero la acción aparenta ser correcta. Eso es el caso en que el programa trabaja a partir de una copia interna del DIRectorio en la MDV sin ir a consultar la propia cinta. Eso no provoca ningún error si no tocas el cartucho que estás usando, pero si colocas un nuevo cartucho en la ductora, obviamente la información conservada en la memoria quedará obsoleta. Asegúrate pues que la ductora hace un ruido cada vez que uses un comando que la afecte, de cualquier clase que sea. Si no lo hace, ensaya un comando DIR con referencia a la otra ductora, incluso aunque esté vacía; eso hace que el sistema vuelva de nuevo a la acción. Si así no funciona, y no quieres restaurar las condiciones, prueba a usar COPY con un intento no destructivo de regresar a la situación normal.

Resumen

- FORMAT se usa para preparar una cinta en cartucho para su uso.
- Las microductoras se identifican por MDV1 y MDV2 (izquierda y derecha respectivamente).
- DIR hace que se muestre el directorio de los ficheros registrados en la cinta.
- Hay cinco grupos de **dispositivos** en el SuperBASIC del QL, siendo las MDVs uno de ellos. SCR es la pantalla; CON es la consola que incluye pantalla y teclado. SER1 y SER2 son los 'portales serie'. NET es la **red** de ordenadores interconectados.
- OPEN se usa para aplicar un número de canal a un dispositivo de manera que las sentencias tales como PRINT e INPUT puedan intercambiar datos con esos dispositivos.
- SAVE, LOAD, LRUN, MERGE, MRUN se usan para el almacenamiento y la recuperación de ficheros de programa en cartuchos de cinta. Cuando se recupera un fichero de programa usando estas sentencias, sus líneas se comprueban en cuanto a sintaxis correcta, y se incluye un mensaje de error si la línea no lo es.
- DELETE se usa para eliminar un fichero de una cinta en cartucho.
- COPY se usa para copiar datos de un dispositivo a otro. Puede usarse, por ejemplo, para producir copias de respaldo en los ficheros importantes.
- Saca como mínimo una copia extra de respaldo (backup) de todos los ficheros importantes para prevenir el caso en que un mal funcionamiento borre la cinta.

- Los ficheros de datos se tratan **secuencialmente** en la cinta, y deben ser abiertos para recoger datos usando OPEN_IN, o para dejar datos en ellos usando OPEN_NEW. Un fichero ya existente no puede alterarse una vez cerrado. CLOSE debe usarse al final del tratamiento de un fichero para asegurar que todas las acciones sobre el mismo han concluido apropiadamente.

- Los datos se cogen y dejan en los ficheros usando PRINT e INPUT, de la misma manera que si fueran expuestos en pantalla o impuestos por teclado, respectivamente. Cada línea separada en pantalla es lo que se denomina un registro del fichero ('una ficha').

Capítulo Diez

Procedimientos, Subrutinas y Funciones

Este capítulo describe las maneras en que la lista de palabras clave del SuperBASIC puede, efectivamente, ampliarse para incluir tus propios comandos y funciones. También muestra cómo puede escribirse parte de un programa como un **módulo** que puede situarse al final del programa principal y al que puede **apelarse** tantas veces como se requiera a lo largo del programa principal. Eso evita tener que escribir una sección del programa varias veces cuando se recurre a él más de una vez.

Las ideas de este capítulo permiten la **programación estructurada** de manera que a medida que se hacen más y más complejos los programas, su estructura puede permanecer comparativamente simple. Los conceptos subyacentes pueden ser difíciles de asimilar, pero aprender a usar estas facilidades puede producir programas fácilmente modificables y netamente estructurados que permiten **desglosar** las más complicadas aplicaciones **en tareas** pequeñas y fácilmente tratables.

Procedimientos

Un **procedimiento** es un bloque separado del programa, que puede situarse al final del programa principal y al que se recurre de manera similar a cualquiera de las sentencias o funciones standard en SuperBASIC. Los valores de las variables pueden ser **transpasados** hasta y desde el procedimiento, como se requiera. Los procedimientos son útiles para reflejar partes de un programa que han de usarse más de una vez.

Considera por ejemplo el problema de comprobar si la respuesta a una pregunta ha sido sí o no. Hemos encontrado este problema hasta ahora en diversas formas, y cada vez esta función se ha realizado dentro de la parte principal del programa. De alguna manera, esta función es un tema lateral con respecto a la función principal de un programa, y sería más correcto si pudiera situarse fuera de la ruta principal y desviarse para ejecutarla cuando sea necesario. El programa principal aparecería así menos abigarrado, y sería más fácil de leer y escribir. Podríamos describir esa tarea mediante un procedimiento, y guardarlo como un módulo separado. Esta filosofía ayuda a estructurar programas de una manera directa e inteligible.

El bloque de un procedimiento se define usando la sentencia DEF PROC. Va seguida del nombre dado al procedimiento, y de un grupo de parámetros **declarados** dentro de paréntesis, que son aquéllos cuyos valores pasan desde y hasta el procedimiento. Un ejemplo podría ser:

```
DEF PROC SUMAR(X,Y)
```

Esto describe un procedimiento llamado SUMAR, que acepta como parámetros de entrada los declarados como X e Y. El 'cuerpo' real del procedimiento se escribe a continuación, terminando de la manera típica en SuperBASIC, con la sentencia END DEF. Este procedimiento sirve meramente para exponer en pantalla la suma de X e Y, y por tanto, vendría descrito por las instrucciones:

```
100 DEF PROC SUMAR(X,Y)
110 PRINT X+Y
120 END DEF
```

Este es un bloque separado de instrucciones de un programa que solamente se ejecuta cuando **se cita** el procedimiento, o se apela a él. Al hacerlo deben darse valores a X e Y, sobre los que procederá a efectuar y exponer su suma. El procedimiento se cita mencionando directamente su nombre, seguido por los valores de los parámetros en el mismo orden en que se declararon en la sentencia DEF PROC. Por ejemplo, para recurrir a este procedimiento con los números 5 y 6, basta usar la siguiente sentencia:

```
SUMAR 5,6
```

Cuando se apela a un procedimiento para que sea ejecutado, debes asegurarte que los valores que van detrás del nombre, están en el mismo orden en que fueron declarados dentro de los paréntesis al hacer la definición del procedimiento.

Como puedes ver, una vez que haya sido definido un procedimiento, el hacer que se ejecute es simplemente escribir su nombre, seguido de los valores de los parámetros que pueda necesitar. En muchas ocasiones no habrá incluso ningún parámetro. Por ejemplo, tomemos el típico mensaje de pantalla que puede necesitarse exponer diversas veces. El procedimiento para efectuar esa acción pudiera ser:

```
100 DEF PROC MENSAJE
110 PRINT FILL$("*",26)
120 READ MESS$
130 PRINT MESS$
140 PRINT FILL$("*",26)
150 DATA "Aqui estaria el primer mensaje"
160 DATA "Aqui estaria el segundo mensaje"
170 DATA "Aqui estaria el tercer mensaje"
180 END DEF
```

Como rutina es bastante simple, pero es lo suficientemente compleja como para incordiar en grado considerable al incluirla en el programa principal, especialmente si se usa varias veces. Aquí simplemente se coloca al final del programa, fuera de la ruta principal, y se cita para ser ejecutada simplemente usando el nombre MENSAJE. Un programa que usara este procedimiento tres veces, podría tener la siguiente forma:

```
10 CLS
20 RESTORE
30 MENSAJE
40 INPUT A$,B$,C$
50 MENSAJE
60 PRINT A$,B$
70 MENSAJE
80 PRINT C$
```

Este programa es extremadamente legible, e iría seguido de la definición del procedimiento con las nuevas líneas que hemos visto. Cada vez que se ejecuta el comando MENSAJE en el programa anterior, el ordenador reconoce que no es una palabra clave normal del SuperBASIC y escruta todo el programa hasta encontrar un procedimiento con ese nombre. Luego ejecuta el procedimiento y **vuelve** a la línea que va detrás del sitio donde se apeló a él, el sitio donde se citó.

Antes de considerar un ejemplo completo de lo anterior, examinaremos cómo traspasar información desde el programa principal hasta el procedimiento, y recoger la que éste produce.

Supongamos que queremos sumar dos números X e Y, pero que su suma no va a ser expuesta en pantalla directamente. Va a ser **entregada** a la parte principal del programa, para que por ejemplo le sume un tercer número obtenido por el teclado. La suma será almacenada en la variable SUM, que también debe incluirse dentro de los paréntesis al hacer la definición del procedimiento, así como al citar dicho procedimiento dentro del programa principal. Así pues, podríamos escribir:

```
10 REM - programa principal:
20 CLS
30 SUMAR 15,38,SUM
40 INPUT"otro numero"!Z
50 PRINT"respuesta="!SM+Z
60 REM
70 REM - ahora viene el procedimiento
80 REM
90 DEF PROC SUMAR(X,Y,SUM)
100 SUM = X+Y
110 END DEF
```

En la línea 30, la cita del procedimiento incluye la pareja de valores que han de ser sumados, y una variable SUM, en la que ha de cargarse el resultado obtenido. Cuando esta instrucción es ejecutada, en realidad es el procedimiento de nombre SUMAR el que se ejecuta, que incluye una asignación de valor a la variable SUM. Por tanto, en la línea 30 se hace que SUM sea evaluada de manera que su valor esté disponible para su uso en las líneas subsiguientes. En particular, la línea 50 la usa para añadir a ella el número impuesto por el teclado. Toda apelación a un procedimiento puede incluir números y/o expresiones numéricas, siempre y cuando estén escritas en el mismo orden tanto en la cita al procedimiento como en la definición del mismo.

Hasta ahora, hemos usado cuidadosamente diferentes identificadores de variables en el programa principal y en los usados por el procedimiento, excepto cuando tienen que compartir los mismos identificadores porque sirven para traspasar valores desde y hasta el procedimiento. De hecho, las variables declaradas en una definición de procedimiento son de la clase denominada LOCAL: no interfieren con los mismos nombres de variables usados en el programa principal, a no ser que estén mencionados en la cita al procedimiento que se haga en el programa principal. Por ejemplo, X e Y están mencionadas en la declaración del procedimiento hecha en la línea 90, y a partir de entonces se consideran como variables de **ámbito** LOCAL. La variable SUM también es local, pero está específicamente sacada fuera del procedimiento al no darle un valor concreto en la cita al procedimiento que se hace en la línea 30. Ya que X e Y son completamente de ámbito LOCAL, puedes usarlas en el programa principal para otros propósitos, dado que no se verán afectadas en sus valores al ejecutar el procedimiento.

El ordenador cuidadosamente conserva aparte los valores que tienen cuando se entra en el procedimiento, y vuelve a restaurar dichos valores de nuevo cuando se vuelve a **re-entrar** en el programa principal. Ensaya con PRINT X y PRINT Y después de ejecutar el programa anterior, y verás que se consideran como variables que no han sido usadas. Sin embargo, si haces PRINT SUM verás que su valor todavía está en memoria.

Las variables usadas dentro de los procedimientos, pero no realmente declaradas en la sentencia DEF PROC no son de ámbito local, y sus valores todavía están retenidos cuando se vuelve a reentrar en el programa principal. Tales variables son las denominadas de ámbito **GLOBAL**. Además de eso, todas las variables usadas en el programa principal son globales, a no ser que específicamente se conviertan en variables locales para ese procedimiento.

Lo siguiente es un ejemplo de una variable (K) no declarada en la DEF PROC y que por tanto es global:

```
500 DEF PROC NUMERO(A,B,C)
510 K = A-B*15
520 C=18^K
530 END DEF
```

Este procedimiento tiene tres variables LOCAL, que son A, B y C, porque están declaradas en la DEF PROC. Sin embargo, K es GLOBAL. Si el programa principal está en el proceso de usar una variable con ese nombre, su valor se verá interferido por la ejecución de este procedimiento. Desde luego, la variable C anterior probablemente será necesaria en el programa principal, y podrá sacarse fuera simplemente incluyendo su nombre en una cita al procedimiento de la siguiente clase:

```
NUMERO 23,D,C
```

Por tanto, los valores 23 y el de la variable D son los traspasados hasta el procedimiento, mientras que el valor de C será el entregado por el procedimiento al concluir su ejecución.

Con el fin de garantizar que no hay ninguna interferencia no autorizada entre las variables dentro de un procedimiento, y las variables del programa principal, puede usarse la sentencia LOC para hacer que una variable sea de ámbito LOCAL. Por ejemplo, podría incluirse al principio del procedimiento anterior, la sentencia:

```
LOC K
```

con lo que K sería una variable de ámbito LOCAL como lo son A y B.

Para escribir programas bien estructurados, merece la pena seguir unas cuantas simples reglas, aunque puedan exigir un poco de esfuerzo adicional. Por ejemplo, no es una mala costumbre usar LOCAL para cualquier variable usada en un procedimiento que no esté realmente declarada en la definición. En un programa simple puedes evitar cualquier problema sin hacer esto. En un programa más complejo que puedas querer ajustar en una fecha posterior, usando LOCAL te evita tener que analizar cada procedimiento para comprobar si estás o no interfiriendo con cualquier nuevo nombre de variable que quieras usar en tu ajuste del programa principal. Si siempre usas local de esta manera, sabrás que todos los procedimientos son seguros.

De la misma manera, siempre debieras colocar tus procedimientos al final del programa, con una sentencia REM para que te recordara que el procedimiento comienza exactamente ahí. También, eso lleva a programas bien estructurados y fácilmente modificables. Trata la escritura de procedimientos como un método de transferir las rutinas más frecuentemente usadas, a un área donde -una vez creadas- no necesitas preocuparte de ellas otra vez. Usar procedimientos hace el programa principal mucho más simple, y no es infrecuente para un programa de tamaño medio tener menos sentencias de programa en la parte principal que en la parte que refleja los procedimientos.

También es una buena idea hacer que los nombres de los procedimientos tengan tanto significado como sea posible. Un procedimiento se usa de manera similar a un comando BASIC, y una de las grandes ventajas del BASIC es lo fácil que es para los angloparlantes. Si sabes suficiente inglés y haces que tus comandos se parezcan a frases del inglés, obtienes esa misma clase de ventajas (y además puedes practicar inglés).

Un ejemplo

El ejemplo en esta sección usa unas cuantas ideas del trabajo previo y tiene un cierto número de procedimientos. Simplemente requiere las tres coordenadas de un triángulo, lo dibuja y luego te pregunta si lo quieres rellenar y con qué color. El programa principal está contenido en un bucle llamado PROGRAM, de manera que en el punto apropiado una sentencia EXIT puede usarse para hacer que **salga** del bucle y finalice el programa. El programa también usa el reloj de tiempo real disponible en el QL, y tendrás que ponerlo en marcha antes de ejecutar el programa.

```
10 REM - Programa Principal
20 REP PROGRAM
30 CLS:INK 7:STRIP 2
40 INICIO
50 PRINT
60 PRINT"POR FAVOR TECLEA TRES PARES"
70 PRINT"DE COORDENADAS."
80 PRINT
90 STRIP 5:INK 0
100 INPUT"X1  Y1"!!X1!!Y1
110 INPUT"X2  Y2"!!X2!!Y2
120 INPUT"X3  Y3"!!X3!!Y3
130 TRACE_TRIANGULO X1,Y1,X2,Y2,X3,Y3
140 AT 0,4:CLS 1
150 STRIP 7
160 INICIO
170 PRINT:PRINT"LO RELLENO?"!!
180 SI_O_NO RESPUESTA
190 IF RESPUESTA THEN
200 INPUT "DIME COLOR O AL 7"!!C
210 INK C
220 FILL 1
```

```
230 TRACE_TRIANGULO X1,Y1,X2,Y2,X3,Y3
240 FILL 0
250 INK 0
260 END IF
270 AT 0,4:CLS 1
280 STRIP 5
290 INICIO
300 PRINT:PRINT"OTRA PASADA?"!!
310 SI_O_NO RESPUESTA
320 IF NOT RESPUESTA THEN
330 PRINT"ADIOS"
340 EXIT PROGRAMA
350 INICIO
360 END IF
370 END REP PROGRAMA
380 REM
390 REM - Aqui comienzan los procedimientos:
400 REM
410 DEF PROC INICIO
420 AT 0,0:
430 PRINT"*****"!DATE$!"*****"
440 END DEF
450 DEF PROC
    TRACE_TRIANGULO(X1,Y1,X2,Y2,X3,Y3)
460 LINE X1,Y1 TO X2,Y2 TO X3,Y3
    TO X1,Y1
470 END DEF
480 DEF PROC SI_O_NO(RESPUESTA)
490 LOC ANS$
500 INPUT ANS$
510 IF ANS$(1)=="S" THEN
    RESPUESTA=1 ELSE RESPUESTA=0
520 END DEF
```

Para seguir el funcionamiento de este programa primero debes repasarlo y encontrar el bucle principal. Comienza en la línea 20 y termina en la línea 370. La salida del bucle se efectúa en la línea 340 bajo ciertas circunstancias. Dentro del bucle hay unas cuantas apelaciones a procedimientos. Estos se citan con nombres que sugieren la función que ejercen, y para cada uno de ellos hay una definición del procedimiento en la última sección del programa, que es la que comienza en la línea 390. Hay tres procedimientos, como sigue:

INICIO — Expone la fecha actual en la pantalla.

TRACE_TRIANGULO — Dibuja el triángulo que tiene por coordenadas X1,Y1; X2,Y2; X3,Y3.

SI_O_NO — Recoge lo impuesto por teclado y comprueba si es "SI" o cualquier otra contestación que empiece con la letra S mayúscula o minúscula, ya que usamos la relación ==. Si es cierto, la variable RESPUESTA se pone a TRUE, al darle un valor distinto de 0. En los demás casos la RESPUESTA se pone a FALSE, al darle el valor 0.

La línea 30 del programa borra la imagen, y luego fija INK y STRIP a los valores normales prescritos. Eso es necesario ya que el programa ajusta esos valores y no los restaura. Eso significa que la última pasada del bucle comenzaría de forma diferente a la primera pasada. La línea 40 apela al procedimiento INICIO.

La siguiente parte del programa pide que se impongan por teclado las coordenadas de los tres puntos que se usan luego en el procedimiento TRACE_TRIANGULO. La línea 140 es un método de borrar la parte superior de la imagen antes de recurrir a INICIO para exponer la hora en ese momento, seguido de la pregunta ¿LO RELLENO? que exige una respuesta SI o NO. Esa cuestión es tratada por el procedimiento SI_O_NO. Si la respuesta es afirmativa, debe imponerse el color y el programa apela de nuevo a TRACE_TRIANGULO para rellenarlo de color. Pregunta luego si el programa va a pasarse otra vez, y también aquí se cita SI_O_NO de nuevo para manejar esta cuestión. El bucle del programa vuelve a cerrarse o se detiene en este momento.

Como puedes ver, los procedimientos se usan todos más de una vez, e intentar conseguir eso usando GOTO o REP, etc., en lugar de procedimientos sería mucho más laborioso.

Debieras teclear este programa y luego guardarlo en la cinta. Puedes entonces pasar algún tiempo modificándolo para cambiar la exposición en pantalla y añadirle más **tareas** en la forma de procedimientos.

Subrutinas

Para mantener la compatibilidad con otras formas del BASIC, el QL admite las sentencias GOSUB y ON...GOSUB con RETURN, que pueden usarse para producir **subrutinas.** Una subrutina es simplemente un bloque de programa, que comienza en algún número de línea, y que termina con una sentencia RETURN para que **vuelva** al punto en que se desvió. El bloque se ejecuta especificando la instrucción GOSUB seguida de un número de línea. Por ejemplo, el siguiente es un programa que expone tres mensajes diferentes, usando la misma subrutina cada vez:

```
10 CLS
20 A$="Primer Mensaje"
30 GOSUB 200
40 A$="Segundo Mensaje"
50 GOSUB 200
60 A$="Tercer Mensaje"
70 GOSUB 200
80 STOP
200 REM - LA SUBRUTINA COMIENZA AQUI
210 PRINT FILL$("*",14)
220 PRINT A$
230 PRINT FILL$("*",14)
240 RET
```

De nuevo, la subrutina contiene un bloque de código al que se recurre siempre que se requiere. Cuando se efectúa un desvío en el curso de la ejecución mediante un GOSUB al número de línea apropiado, el ordenador almacena la posición presente en el programa, y pasa a ejecutar la subrutina. Cuando encuentra la instrucción RETURN, el programa continúa a partir del punto donde se desvió. Las subrutinas pueden usarse para proporcionar el mismo grado de **modularidad** en la programación como los procedimientos, pero son menos estructurados en cuanto al traspaso de variables y a la declaración de ámbito local para ellas.

La sentencia ON...GOSUB trabaja exactamente igual que ON...GOTO, y ambas han sido sustituidas en el SuperBASIC por las sentencias de SELección.

RETurn puede usarse en procedimientos para forzar a que el curso de ejecución **vuelva** al programa principal, quizás cuando en una sentencia IF se comprueba que la condición ha sido satisfecha.

Funciones definidas por el usuario

Al igual que es posible incorporar tus propios comandos usando procedimientos, también puedes incluir tus propias funciones dentro del SuperBASIC.

Las **funciones** son similares a los procedimientos en cuanto que contienen una sección de programación que se requiere a menudo, pero una función difiere de un procedimiento en que su identificador puede ser considerado como un operando cualquiera dentro de una expresión; i.e. como una variable más. En un ejemplo previo de este capítulo, definimos un procedimiento llamado SUMAR. Nunca podríamos escribir una sentencia en la forma:

```
N =3*SUMAR(X,Y)
```

porque SUMAR no es una función. Por otro lado, podemos escribir:

```
N =3*SQRT(X)
```

porque SQRT es una función intrínsecamente definida en SuperBASIC. Podemos añadir nuestras propias funciones a la lista disponible en SuperBASIC, usando la sentencia DEF FN. Al igual que con los procedimientos, tiene que ir seguido de un nombre y de paréntesis dentro de los cuales se declaran los **argumentos** de dicha función. La sentencia DEF FN va pues seguida de una serie de líneas de programa que definen la manera de evaluar esa función, y que han de estar terminadas con la sentencia END DEF. La sentencia RET puede usarse dentro de la evaluación de la función para **devolver** el valor requerido. Eso se muestra a continuación.

El siguiente trozo de programa define una función llamada SUMA que devuelve como resultado la suma de cuatro números de manera que pueden ser usados en expresiones como operandos:

```
100 DEF FN SUMA(X,Y,Z,T)
110 A=X+Y
120 B=Z+T
130 RETurn A+B
140 END DEF
```

Observa la similitud entre una función y un procedimiento. Sin embargo, una función devuelve como resultado justamente un número, que queda asignado al nombre de la función. Este único valor entregado por la función es el que se calcula y se asigna en la línea 130 al usar la sentencia RET (devuelva).

Los nombres de la función deben tener los símbolos de tipo correcto. Aquí SUMA es una función cuyo valor es en coma flotante. Si fuera un literal, tendríamos que haber escrito SUMA$, y si fuera un entero SUMA%.

Al igual que para los procedimientos, una variable usada dentro de la definición de una función puede mantenerse de ámbito local usando la sentencia LOC, y podría merecer la pena añadir aquí la línea:

```
105 LOC A,B
```

para impedir que la evaluación de dicha función interfiera con cualquier posible uso de esas mismas variables dentro del programa principal.

La función SUMA podría usarse dentro de una expresión numérica en la parte principal del programa, añadiendo a la anterior las siguientes líneas:

```
10 CLS
20 INPUT "4 numeros:"!X!!Y!!Z!!T
30 K=123*SUMA(X, Y, Z, T)
40 PRINT K
```

Aquí se impone valor a cuatro variables, y luego se fija K igual a 123 veces el valor de la función SUMA con los argumentos mostrados. Como puedes ver, la función SUMA se usa exactamente igual que las funciones standard incorporadas en el SuperBASIC, y por tanto, puedes añadir tus propias funciones al repertorio del lenguaje, a medida que desees.

Resumen

- Un **procedimiento** es un bloque de sentencias que se ponen en ejecución al citar su nombre dentro del programa principal. El procedimiento comienza con DEF PROC y termina con END DEF. Las variables dentro del procedimiento son de ámbito local si se declaran en la definición, o si se mencionan explícitamente en una sentencia LOC. Se pueden traspasar desde y hasta un procedimiento los valores de variables.

- Una **subrutina** es un bloque de programa, que termina en una sentencia RETurn, y que se cita mediante la sentencia GOSUB seguida de un número de línea.

- DEF FN y END DEF se usan para definir nuevas **funciones** que se pueden usar exactamente de la misma manera que el repertorio normal de las funciones intrínsecas del SuperBASIC. Una función difiere de un procedimiento en que la función **devuelve** exactamente un número que se almacena bajo el propio identificador de la función. Los identificadores de funciones se supone que son de coma flotante a no ser que terminen en $ o en % para literales y numerales enteros, respectivamente.

Apéndice

Lista Alfabética de Palabras Clave, Funciones y Operadores

Este apéndice proporciona una lista completa de las palabras clave en el SuperBASIC del QL, según orden alfabético. Cada palabra clave indica un capítulo, y tiene una breve descripción para recordar al lector la acción o función que ejerce. La lista de los otros operadores y funciones que incluyen, por ejemplo, + y * se puede hallar en la sección de conceptos del propio manual del QL.

Clave	Capítulo	Tipo	Descripción
ABS	7	Función	Valor absoluto
ACOT	7	Función	Arco cotangente - da radianes
ADATE	4	Sentencia	Ajusta reloj en segundos
AND	5	Operador	Yliación Booleana
ARC	8	Sentencia	Traza arco circular
AT	4	Sentencia	Fija posición cursor texto
ATAN	7	Función	Arco tangente - da radianes
AUTO	3	Sentencia	Numeración automática de líneas
BAUD	9	Sentencia	Fija rapidez transmisión portales serie
BEEP	8	Sentencia	Hace que el QL emita un sonido
BEEPING	8	Función	Cierta si se está produciendo sonido
BLOCK	8	Sentencia	Traza rectángulos en pixels
BORDER	8	Sentencia	Traza un marco para una ventana
BREAK	3	Pulsación	Detiene ejecución (CTRL SPACE)
CALL	4	Sentencia	Ejecuta programa código máquina
CHR$	6	Función	Carácter correspondiente a ASCII
CIRCLE	8	Sentencia	Traza círculos y elipses
CLEAR	3	Sentencia	Anula todos los valores de variables
CLOSE	9	Sentencia	Cierra ficheros y canales
CLS	4	Sentencia	Borra ventana total o parcial
CODE	6	Función	Da ASCII del carácter
CONTINUE	5	Sentencia	Siga después interrupción (BREAK)
COPY	9	Sentencia	Copia información entre dispositivos
COS	7	Función	Coseno (radianes)
COT	7	Función	Cotangente (radianes)
CSIZE	8	Sentencia	Cambia tamaño carácter
CURSOR	8	Sentencia	Fija posición PRINT en gráficos
DATA	4	Sentencia	Contiene series de datos
DATE	4	Función	Fecha como número coma flotante
DAY$	4	Función	Día como literal
DEF FN	10	Sentencia	Define nueva función
DEF PROC	10	Sentencia	Define un procedimiento
DEG	7	Función	Cambia radianes en grados
DELETE	9	Sentencia	Quita ficheros de microductoras

Clave	Capítulo	Tipo	Descripción
DIM	6	Sentencia	Fija dimensión de tablas
DIMN	6	Función	Da dimensión corriente tabla
DIR	9	Sentencia	Muestra directorio de cinta en cartucho
DIV	7	Operador	Cociente de división entera
DLINE	3	Sentencia	Suprime bandas de líneas programa
EDIT	3	Sentencia	Revisa una línea programa
ELSE	5	Sentencia	Cláusula ejecutada SI condición falsa
END DEF	10	Sentencia	Fin de procedimientos y funciones
END FOR	5	Sentencia	Finaliza bucle FOR
END IF	5	Sentencia	Finaliza sentencia IF
END REP	5	Sentencia	Finaliza bucle REPEAT
END SEL	5	Sentencia	Finaliza bloque SELECT
EXEC	10	Sentencia	Ejecuta rutinas multitarea
EXIT	5	Sentencia	Salga de un bucle
EXP	7	Función	Eleva e a una potencia
FILL	8	Sentencia	Rellena una figura convexa
FILL$	6	Función	Da repetición de un carácter
FLASH	8	Sentencia	Inicia/Termina parpadeo caracteres
FOR	5	Sentencia	Inicia bucle 'preconfinado'
FORMAT	9	Sentencia	Prepara una cinta en cartucho
GOSUB	10	Sentencia	Desvío a una subrutina
GOTO	6	Sentencia	Salta a número de línea dado
IF	5	Sentencia	Encabeza frase condicional
INK	8	Sentencia	Fija color de salida en pantalla
INKEY$	4	Función	Impone un único carácter tecleado
INPUT	4	Sentencia	Impone valores tecleados
INSTR	6	Operador	Comprueba pertenencia literales
INT	7	Función	Da entero menor que argumento
KEYROW	4	Función	Da columna de tecla pulsada
LBYTES	9	Función	Carga datos en binario
LEN	6	Función	Da longitud de literal
LET	3	Sentencia	Asigna valor a variable
LINE	8	Sentencia	Traza rectas
LIST	3	Sentencia	Lista programa en memoria
LN	7	Función	Logaritmo natural (base **e**)
LOAD	9	Sentencia	Carga programa desde microductora
LOC	10	Sentencia	Hace variable local
LOG10	7	Función	Logaritmo en base 10
LRUN	9	Sentencia	Carga y ejecuta fichero de programa
MERGE	9	Sentencia	Mezcla programa en cinta al de memoria
MOD	7	Operador	Residuo de división entera
MODE	8	Sentencia	Cambia modo resolución pantalla
MOVE	8	Sentencia	Mueve tortuga por pantalla
MRUN	9	Sentencia	Mezcla y ejecuta programa
NET	9	Sentencia	Fija número de estación en red
NEW	3	Sentencia	Quita programa de memoria
NEXT	5	Sentencia	Marca otra ronda de bucle FOR
NOT	5	Operador	Negación o complemento logical
ON GOSUB	10	Sentencia	Desvío a subrutinas según valor
ON GOTO	10	Sentencia	Salto a línea según valor

Clave	Capítulo	Tipo	Descripción
OPEN	8 y 9	Sentencia	Abre canales hacia dispositivos
OPEN_IN	9	Sentencia	Abre fichero para lectura
OPEN_NEW	9	Sentencia	Abre nuevo fichero para escritura
OR	5	Operador	Oliación Booleana
OVER	8	Sentencia	Sobre-escribe caracteres
PAN	8	Sentencia	Desplaza horizontalmente imagen
PAPER	8	Sentencia	Fija color de fondo
PAUSE	3	Sentencia	Pausa en la ejecución
PEEK	4	Función	Da contenido de memoria
PENDOWN	8	Sentencia	Manda a tortuga deje estela
PENUP	8	Sentencia	Termina estela de tortuga
PI	7	Función	Da el valor de pi
POINT	8	Sentencia	Traza un punto en gráficos
POKE	8	Sentencia	Mete valor en memoria
PRINT	4	Sentencia	Expone caracteres en pantalla
RAD	7	Función	Convierte grados a radianes
RANDOMISE	7	Sentencia	Fija gérmen generador números aleatorios
READ	4	Sentencia	Apunta valores tomados de DATA
RECOL	8	Sentencia	Recolorea todos los pixels
REM	3	Sentencia	Permite comentarios en programa
RENUM	3	Sentencia	Renumera líneas de programa
REP	5	Sentencia	Comienzo de bucle REPEAT
RESTORE	4	Sentencia	Redirecciona puntero de serie DATA
RET	10	Sentencia	Vuelta de DEF FN, PROC, GOSUB
RETRY	5	Sentencia	Re-intenta última sentencia ejecutada
RND	7	Función	Da un número aleatorio
RUN	3	Sentencia	Ejecuta programa corriente
SAVE	9	Sentencia	Guarda programa en cinta de cartucho
SBYTES	9	Sentencia	Guarda información binaria en microductora
SCALE	8	Sentencia	Fija la escala para gráficos
SCROLL	8	Sentencia	Desplaza verticalmente imagen ventana
SDATE	4	Sentencia	Fija reloj tiempo real
SEL	5	Sentencia	Selecciona acciones alternativas
SEXEC	9	Sentencia	Guarda para EXEC multitarea
SIN	7	Función	Seno (radianes)
SQRT	7	Función	Raíz cuadrada
STOP	5	Sentencia	Detiene ejecución programa
STRIP	8	Sentencia	Fija banda de fondo en caracteres
TAN	7	Función	Tangente (radianes)
THEN	5	Sentencia	Parte 'entonces' de IF THEN
TURN	8	Sentencia	Gira dirección de tortuga
TURNTO	8	Sentencia	Fija rumbo de tortuga
UNDER	8	Sentencia	Subraya caracteres
WINDOW	8	Sentencia	Fija características de ventanas
XOR	5	Operador	O lógico exclusivo (Oleación Booleana)

Indice

indescomp publicaciones
Avda. del Mediterráneo, 9 — 28007 MADRID