sinclair

QL

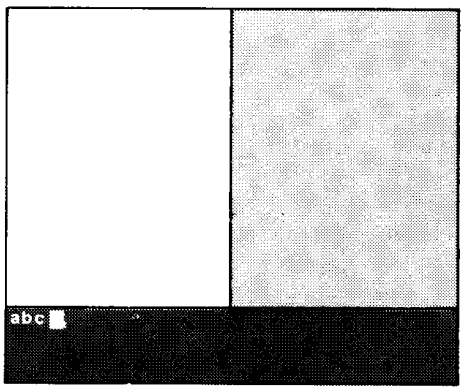
Guía para el principiante

Por Roy Atherton *(Bulmershe College Computer Centre)*

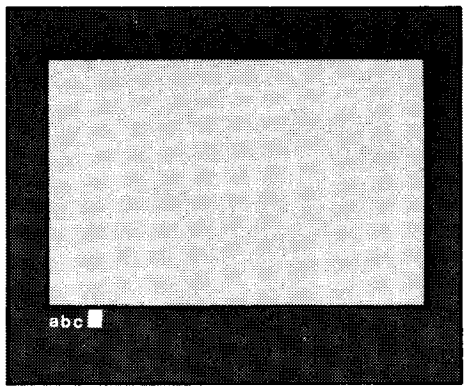
CAPITULO 1
COMIENCE A UTILIZAR SU ORDENADOR

LA PANTALLA

Su QL debe conectarse a una pantalla de TV o a un monitor y a la red. Pulse unas pocas teclas, por ejemplo abc, y la pantalla tendrá el aspecto que se muestra más abajo. La pequeña luz intermitente es el **cursor.**



Monitor



Television

Si su pantalla no tiene este aspecto, lea la sección titulada "Introducción". Con ello probablemente podrá ya resolver cualquier dificultad que se le presente.

EL TECLADO

El QL es un ordenador versátil y potente, por tanto en el teclado dispone de ciertas características que no necesita por el momento. En una primera aproximación vamos a explicar únicamente los elementos que le serán necesarios en este capítulo y los seis siguientes.

INTERRUPCION

Con esta secuencia, se encontrará capacitado para interrumpir situaciones no deseables. Por ejemplo, puede encontrarse con:

una línea que ha decidido abandonar
algo erróneo que usted no comprende
un programa que se está ejecutando y ya no le es interesante
cualquier otro problema

Como la acción de interrupción es tan potente, se ha creado de forma que resulte muy difícil de teclear accidentalmente.

Mantenga pulsado CTRL y luego pulse el espaciador

Si no se añade o elimina nada en un programa interrumpido puede volverse a comenzar tecleando.

```
CONTINUE
```

Si se ha cambiado algo y el programa no puede continuar, el ordenador responderá

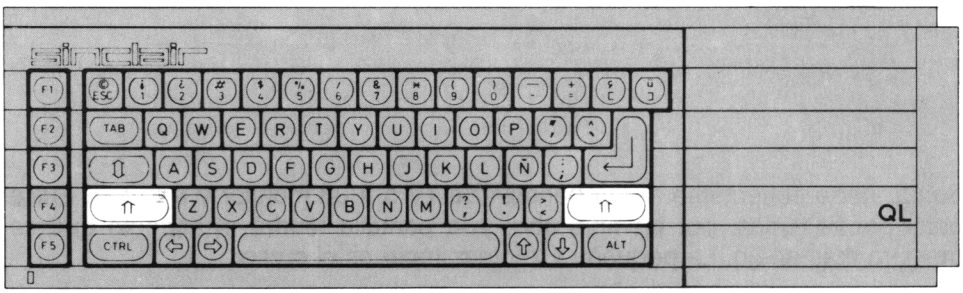
```
Línea incorrecta
```

REINICIALIZACION

En la parte derecha del QL existe un pequeño botón (no es una tecla). Se ha colocado en ese lugar a desmano deliberadamente, ya que sus efectos son más dramáticos que los que se derivan de utilizar las teclas de interrupción. Si

no se obtiene lo que desea con las teclas de interrupción, pulse el botón de reinicialización (RESET). Esta operación es idéntica a desconectar el ordenador y volverlo a conectar. Con ello se obtiene una reinicialización desde cero.

⇑



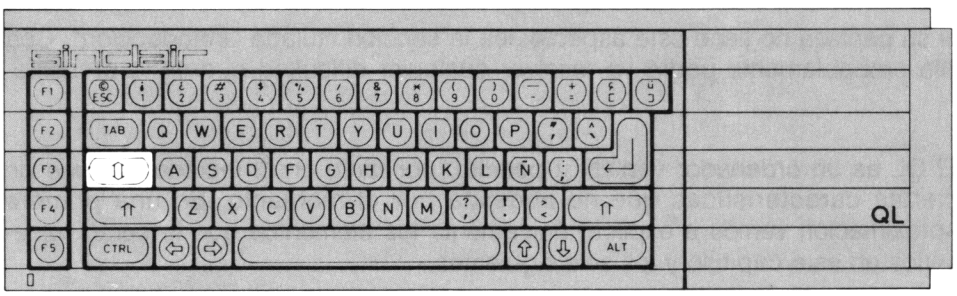
Dispone de dos teclas ⇑ , ya que su uso es muy frecuente y necesitamos utilizarlas con cualquiera de las dos manos.

Mantenga pulsada una de las teclas ⇑ y teclee algunas letras. Obtendrá letras mayúsculas.

Mantenga pulsada una de las teclas ⇑ y teclee otra tecla cualquiera que no sea una letra. Obtendrá el símbolo que se encuentra dibujado en la parte superior de la tecla.

Si no pulsa la tecla ⇑ , escribirá en minúsculas o los símbolos que se encuentran dibujados en la parte inferior de las teclas.

⇳

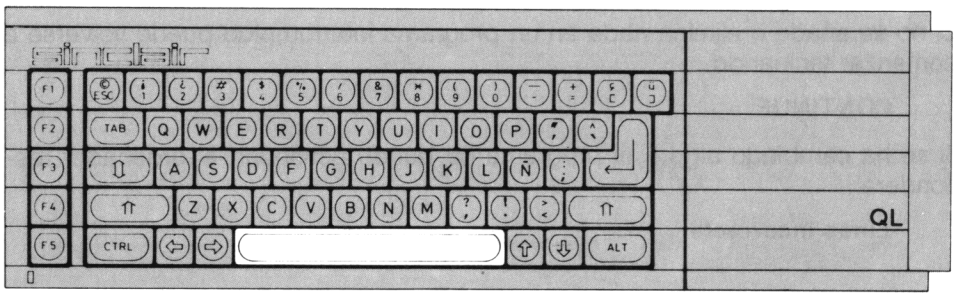


Esta tecla funciona como un conmutador. Púlsela una vez y fijará uno de los dos modos. Estos modos sólo afectan a las letras, que serán mayúsculas o minúsculas.

Pulse algunas teclas de letras
Presione una sola vez ⇳
Pulse algunas otras letras

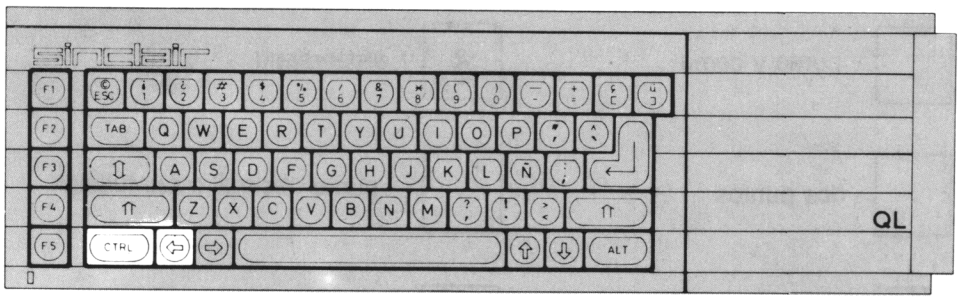
Verá que el modo cambia y permanece hasta que vuelve a pulsar de nuevo ⇳ (seguro para mayúsculas).

BARRA DE ESPACIOS

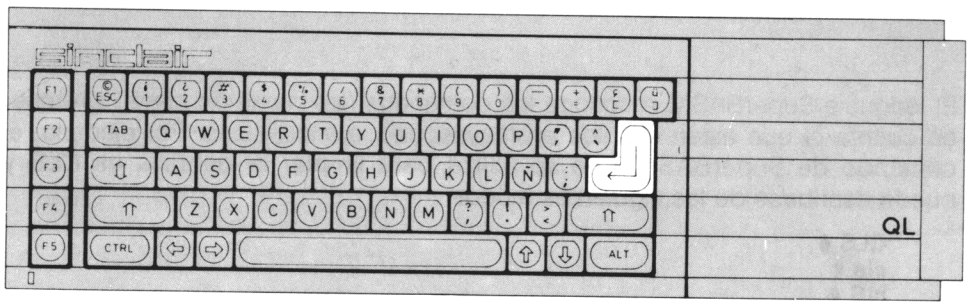


La tecla larga del final del teclado es la que crea los espacios. Como verá en el capítulo dos, en el lenguaje SuperBASIC ésta es una tecla muy importante.

BORRADO



La tecla de flecha izquierda del cursor actuando junto a la tecla **CTRL** actúa como una goma de borrar. Mantenga apretada la tecla de **CTRL** mientras pulsa la tecla del cursor. Cada vez que pulse estas teclas juntas borrará el carácter anterior.



OTROS SIMBOLOS DE UTILIZACION INMEDIATA

El sistema necesita saber cuándo se ha terminado un mensaje o una instrucción. Cuando haya terminado de teclear algo completamente, como por ejemplo **RUN**, pulse la tecla ↵ para introducirlo en el sistema y que éste actúe.

Como esta tecla es de mucho uso hemos utilizado un símbolo especial para ella.

⏎

Utilizaremos este símbolo por conveniencia, mejor presentación y para ahorrar espacio. Pruebe la tecla ⏎ tecleando

```
PRINT "Correcto"⏎
```

Si no comete ningún error, el sistema le responderá

```
Correcto
```

*	multiplicación	+	suma
—	guión	=	se hace igual a
"	comillas	!	exclamación
,	coma	'	apóstrofe


punto y coma


ampersand


dos puntos


punto decimal o puntuación


diagonal


dólar


paréntesis izquierdo


paréntesis derecho

MAYUSCULAS Y MINUSCULAS

El lenguaje SuperBASIC reconoce los comandos (las palabras clave) sin tener en cuenta el que estén escritos en mayúsculas o minúsculas. Por ejemplo, el comando de SuperBASIC que se utiliza para limpiar la pantalla es **CLS** y puede escribirse de los siguientes modos:

```
CLS ↵
cls ↵
clS ↵
```

Estos tres modos son correctos y tienen el mismo efecto. Algunas palabras clave se visualizan escritas con ciertas letras mayúsculas para mostrar las abreviaturas permitidas. Si no puede abreviarse una determinada palabra clave, se visualizará toda ella en mayúsculas.

UTILIZACION DE LAS COMILLAS

Las comillas se utilizan normalmente para definir una palabra o una frase —una cadena de caracteres—. Pruebe a escribir:

```
PRINT "Esto funciona" ↵
```

El ordenador le contestará:

```
Esto funciona
```

Las comillas no se imprimen, pero indican que se debe imprimir un texto, y definen exactamente ese texto —todo lo que se encuentra entre las comillas que abren y cierran el texto—. Si desea utilizar las comillas para que aparezcan en la cadena de caracteres impresa, utilice el símbolo del apóstrofe en lugar de las comillas al dar la instrucción PRINT. Por ejemplo:

```
PRINT 'El símbolo de las comillas es " '
```

que dará el siguiente resultado:

```
El símbolo de las comillas es "
```

ERRORES COMUNES EN EL TECLADO

La tecla cero se encuentra con los demás dígitos numéricos en la parte superior del teclado, y es algo más estrecha.

La letra "O" se encuentra entre las demás letras. Sea cuidadoso y utilice el símbolo adecuado.

Del mismo modo, debe evitarse la confusión entre el uno, que se encuentra entre los dígitos, y la letra "l", que está ubicada entre las letras.

CON LA TECLA SHIFT PULSADA

Cuando utilice la tecla ⇧ , manténgala pulsada mientras teclea la otra tecla, de forma que la tecla ⇧ haga contacto antes que la otra tecla.

Esta regla es también válida para la tecla **CTRL** y la tecla **ALT** alternativa, que son teclas que se usan en conjunción con otras letras, pero por el momento, estas teclas no son necesarias.

Teclee estas instrucciones sencillas:

```
CLS↵
PRINT "Hola"↵
```

Hablando estrictamente, las instrucciones anteriores constituyen un **programa de ordenador;** sin embargo, en informática lo importante son los **programas almacenados.** Las instrucciones anteriores se ejecutan inmediatamente en cuanto usted pulse ↵ (ENTER).

Teclee ahora el programa con los números de línea:

```
10 CLS↵
20 PRINT "HOLA"↵
```

Esta vez, externamente no ocurre nada a excepción de que el programa aparece en la parte superior de la pantalla. Esto significa que el sistema lo ha aceptado como gramaticalmente o sintácticamente correcto. Es conforme con las reglas del lenguaje SuperBASIC pero hasta el momento no ha sido ejecutado, sino simplemente almacenado. Para que funcione, teclee simplemente:

```
RUN↵
```

En el siguiente capítulo se estudiarán las diferencias que existen entre los comandos directos, de acción inmediata, y las secuencias de instrucciones almacenadas. Por el momento, puede experimentar con las ideas anteriores, y dos ideas más:

```
LIST↵
```

Que causa la visualización (listado), en la pantalla o en otro lugar, de un programa almacenado internamente.

```
NEW↵
```

que borra un programa almacenado internamente, de forma que puede teclearse —**NEW**— otro nuevo.

AUTOEXAMEN SOBRE EL CAPITULO 1

En este test puede obtener un máximo de 16 puntos. Compruebe su puntuación con las respuestas de la página siguiente.

1. ¿En qué circunstancias puede utilizarse la secuencia de interrupción?
2. ¿Dónde se encuentra el botón RESET (reinicialización)?
3. ¿Cuál es el efecto que causa el botón RESET?
4. Señale dos diferencias entre las teclas ⇧ y ⇩.
5. Explique cómo puede borrarse un carácter que se acaba de teclear.
6. ¿Para qué se utiliza la tecla ↵?
7. ¿Qué símbolo se utiliza para la tecla ↵?

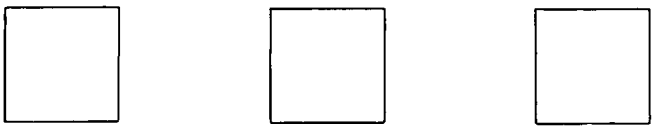
Señale el efecto de los comandos de las preguntas 8 a 11.

8. CLS↵

9. RUN↵
10. LIST↵
11. NEW↵
12. ¿Tienen efecto los comandos que se teclean en minúsculas?
13. ¿Cuál es el significado de la parte de la palabra clave que el QL visualiza en mayúsculas?

CAPITULO 2
LAS INSTRUCCIONES PARA EL ORDENADOR

Los ordenadores necesitan almacenar datos como, por ejemplo, números. Esta forma de almacenamiento puede compararse con el existente en los casilleros.



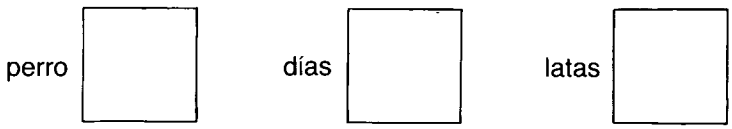
Aunque no pueda verlos, usted necesita dar nombre a los casilleros. Suponga que desea realizar el siguiente cálculo sencillo.

Un criador de perros tiene que alimentar 9 perros durante 28 días, y cada uno de ellos debe recibir una lata de "Perrys" al día. Haga que el ordenador imprima (visualice en pantalla) el número de latas necesario.

Una forma de resolver el problema utilizaría tres casillas de paloma para:

el número de *perros*
el número de *días*
el número total de *latas*

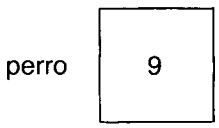
El lenguaje Super-BASIC le permite elegir nombres adecuados para los casilleros, y usted puede elegir los siguientes:



Con una sencilla instrucción puede hacerse que el ordenador cree un casillero, le dé un nombre y almacene en él un número.

```
LET perro = 9
```

Con ello, se establecerá un casillero interno, llamado *perro,* y en él se colocará el número 9, por tanto será:



La palabra **LET** tiene en el lenguaje SuperBASIC un significado muy especial. Se le denomina **Palabra clave.** El lenguaje SuperBASIC tiene muchas otras palabras clave que veremos más adelante. Sea cuidadoso con los espacios que deja después de **LET** y de otras palabras clave. Como el lenguaje SuperBASIC le permite elegir los nombres de los casilleros con gran libertad, *LETperro* puede ser un nombre válido para el casillero.

La palabra clave **LET,** en SuperBASIC es opcional y, por ello, las instrucciones del tipo

```
LETperro = 9
```

son válidas. Estas instrucciones hacen referencia a un casillero llamado **LETperro.**

Como ocurre en nuestro lenguaje, el español, los nombres, números y palabras clave deben ir separados unos de otros por espacios, si no van separados por otros caracteres especiales.

Incluso en los casos en que no son necesarias, las líneas de programas sin el espaciado adecuado no tienen un buen estilo. Las máquinas que no tienen gran cantidad de memoria pueden forzar a los programadores a actuar en esta dirección, pero desde luego, éste no es un problema en el QL.

Compruebe que su casillero existe internamente, tecleando

```
PRINT perro◆
```

En la pantalla aparecerá lo que contiene ese casillero:

```
9
```

De nuevo insistimos en que sea cuidadoso y coloque un espacio después de **PRINT**.

Para resolver el problema podemos escribir un programa que sea una secuencia de instrucciones: Ahora ya puede entender las dos primeras

```
LET perro = 9◆
LET días = 28◆
```

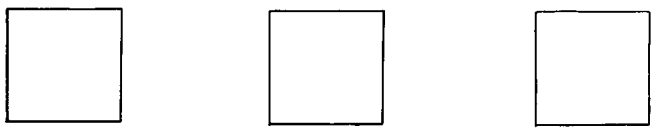
Con ello se habrán creado dos casilleros con sus nombres y se les habrá asignado números o valores.

La siguiente instrucción debe realizar una multiplicación, que utiliza en el ordenador el símbolo * y coloca el resultado en un nuevo casillero llamado en este caso *latas,* esto es:

```
LET latas = perro * días◆
```

1. El ordenador toma los valores 9 y 28 de los dos casilleros de nombre *perro* y *días.*
2. El número 9 se multiplica por 28.
3. Se establece un nuevo casillero con el nombre *latas.*
4. El resultado de la multiplicación es el valor que toma el casillero de nombre *latas.*

Todo ello puede parecer complicado, pero es necesario que usted entienda las ideas, ya que son ideas muy importantes. El efecto puede imaginarse muy simplemente, como se muestra más abajo:



La única tarea que nos resta es que el ordenador imprima el resultado. Esto puede hacerse, tecleando:

```
PRINT latas◆
```

que causará la siguiente salida

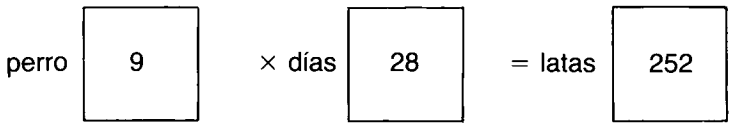
```
252
```

que se visualizará en la pantalla.

En suma, el programa

```
LET perro = 9◆
LET días = 28◆
LET latas = perro * días◆
PRINT latas◆
```

causa efectos internos que se imaginan mejor pensando en tres casilleros que contienen los números:



y la salida en la pantalla será:

```
252
```

Evidentemente, este resultado puede obtenerse más fácilmente con una calculadora o lápiz y papel. Con el QL se puede hacer rápidamente tecleando:

```
PRINT 9 * 28↵
```

que dará la respuesta en la pantalla. Sin embargo las ideas que hemos discutido son puntos esenciales de la programación en SuperBASIC. Son tan importantes, que se dan en muchos lenguajes de programación y tienen nombres especiales.

1. Los nombres como, por ejemplo, *perro, días* y *latas* se llaman **identificadores.**
2. Una simple frase como, por ejemplo:

   ```
   LET perro = 9↵
   ```

   se llama **instrucción.**
3. La asociación del nombre y el casillero asociado se llama **variable.** Al ejecutarse la instrucción anterior, se almacenará el valor 9 en el casillero identificado por su nombre, *perro.*

Una instrucción como, por ejemplo:

```
LET perro = 9↵
```

es una instrucción para un proceso interno muy dinámico, pero el texto que se imprime es estático y usa el signo = tomado del que se utiliza en Matemáticas. Es mejor, por tanto, pensar o decir (pero no escribir)

```
LET perro hacerse 9
```

y pensar en el proceso que tiene una dirección de derecha a izquierda (no teclee esto):

```
perro ◊9
```

La utilización del signo = en una instrucción **LET** es diferente que en Matemáticas. Por ejemplo, si tenemos otro perro, podemos escribir:

```
LET perro = perro + 1↵
```

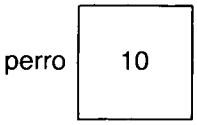
Matemáticamente esto no es correcto, pero en términos de operaciones con ordenadores es muy sencillo. Si el valor de perro antes de la operación era 9, el valor después de la operación debe ser 10. Pruebe todo esto tecleando:

```
LET perro = 9↵
PRINT perro↵
LET perro = perro + 1↵
PRINT perro↵
```

La salida deberá ser:

```
9
10
```

demostrando que el valor final del casillero es, como puede verse:



Un buen sistema para comprender lo que está ocurriendo con los casilleros, o variables, es realizar lo que se llama una pasada en vacío o simulación ("dry run"). Examine simplemente cada instrucción una por una y escriba los valores resultantes de cada instrucción, para ver cómo se crean los casilleros y se les asignan los valores, así como también cómo se retienen los valores mientras se ejecuta el programa.

```
LET perro = 9
LET días = 28
LET latas = perro * días
PRINT latas
```

perro	días	latas
9		
9	28	
9	28	252
9	28	252

La salida será:

```
252
```

Observará que hasta ahora el nombre de la variable se ha usado primero en la parte izquierda de la instrucción **LET.** Cuando ya se ha creado el casillero y se le ha dado un valor, el nombre correspondiente de la variable puede usarse en la parte derecha de la instrucción.

Suponga ahora que quiere convencer a un niño pequeño de que debe ahorrar dinero. Puede darle dos onzas de chocolate por cada billete de 100 pesetas ahorrado. Suponga que intenta hacer el cómputo siguiente:

```
LET onzas = billetes * 2
PRINT onzas
```

Tal y como se encuentra el programa, no puede hacerse una simulación o pasada en vacío, ya que no sabe los billetes de 100 que ha ahorrado el niño.

```
LET onzas = billetes * 2
```

Onzas	Billetes
?	?

Hemos cometido un error deliberado utilizando los *billetes* a la derecha (miembro de la derecha) de una instrucción **LET,** sin haberlo establecido con anterioridad y haberle dado algún valor. Su QL buscará internamente la variable *billetes.* Como no la encuentra, deduce que se ha cometido un error en el programa y envía un mensaje de error. Si hubiéramos intentado imprimir el valor de *billetes,* el QL hubiera impreso un * para indicar que *billetes* no estaba definido. Se dice que la variable *billetes* no ha sido **inicializada** (no se le ha asignado ningún valor inicial). El programa funcionará adecuadamente, si usted hace lo siguiente:

```
LET billetes = 7
LET onzas = billetes * 2
```

Onzas	Billetes
7	
7	14

El programa funcionará adecuadamente y dará como salida:

```
14
```

UN PROGRAMA ALMACENADO

Se puede producir el efecto deseado tecleando las instrucciones sin números de línea, pero existen dos razones por las que este método, si se utiliza como anteriormente, no resulta satisfactorio, si no es a modo de pequeña introducción.

1. El programa sólo se puede ejecutar a la velocidad a la que usted teclea.

No es una velocidad muy buena para una máquina que puede realizar millones de operaciones por segundo.

2. Las instrucciones individuales no se almacenan después de su ejecución y, por tanto, no puede ejecutarse el programa de nuevo, o corregir un error sin volverlo a teclear completo.

En el siglo diecinueve, un pionero de los ordenadores, Charles Babbage, comprendió que un ordenador de éxito necesitaba almacenar las instrucciones, del mismo modo que los datos, en los casilleros internos. Estas instrucciones se ejecutarían rápidamente en orden secuencial, sin otro tipo de intervención humana.

Si utiliza números de línea, las instrucciones de programa se alamacenarán y no se ejecutarán. Pruebe por tanto:

```
10 LET precio = 15↵
20 LET plumas = 7↵
30 LET coste = precio * plumas↵
40 PRINT coste↵
```

Externamente no ocurre nada por el momento, pero el programa se almacena completo internamente. Para ejecutarlo, teclee:

```
RUN↵
```

y la salida:

```
105
```

aparecerá en la pantalla.

La ventaja de esta solución es que los programas pueden editarse o aumentarse con una operación de tecleado mínima.

EDICION DE UN PROGRAMA

Más adelante se verán las características completas de la edición en SuperBASIC, pero incluso en este estado elemental, usted puede realizar tres cosas fácilmente:

sustituir una línea
insertar una nueva línea
borrar una línea

Sustitución de una línea

Suponga que desea alterar el programa anterior debido a que el precio ha cambiado a 20 pesetas cada pluma. Repita sencillamente la línea 10.

```
10 LET precio = 20↵
```

Esta línea sustituirá a la anterior línea 10. Suponiendo que las demás líneas siguen almacenadas, pruebe el programa tecleando:

```
RUN↵
```

y aparecerá la respuesta, 140.

Inserción de una nueva línea

Suponga que desea insertar una línea justo antes de la última, que imprima las palabras "Coste total". Esta situación es muy frecuente, y por ello se eligen números como 10, 20, 30, que dejan espacio para la inserción de líneas adicionales.

Para añadir una línea adicional, teclee:

```
35 PRINT "Costes Totales"↵
```

y esta línea insertará justo antes de la línea 40. El sistema permite una gama de números de línea que va desde 1 a 32768, para permitir una total flexibilidad en la elección de los números. Es muy difícil saber con anterioridad los cambios necesarios.

Teclee ahora:

```
RUN↵
```

y la nueva salida será:

```
Coste Total
140
```

Borrado de una línea

La línea 35 puede borrarse del modo siguiente: Teclee

```
35↵
```

Y el resultado es el mismo que si hubiera sustituido la línea 35 por una línea vacía.

SALIDA-IMPRESION

Observe la gran utilidad de las instrucciones **PRINT**. Para imprimir (**PRINT**) texto, pueden utilizarse comillas o apóstrofes:

```
PRINT "Onzas de chocolate"↵
```

Para imprimir el valor de las variables (lo que contienen los casilleros), teclee instrucciones como la siguiente:

```
PRINT onzas↵
```

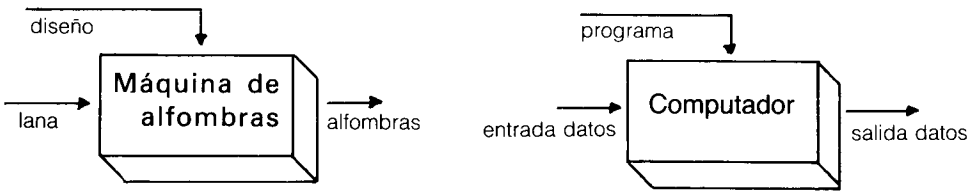
sin utilizar comillas.

Más adelante observará la gran versatilidad de la instrucción **PRINT** en el lenguaje SuperBASIC. Esta instrucción le permitirá colocar texto u otra salida en la pantalla exactamente en el lugar que usted desee. Por el momento, estas dos ventajas son suficientes:

impresión de texto
impresión de los valores de las variables (contenido de los casilleros)

ENTRADA-ENTRADA, LECTURA Y DATOS

Una máquina de fabricar alfombras necesita una introducción de lana. Con ello, fabrica alfombras de acuerdo con el diseño que se le indica en cada momento.



Si se cambia la lana, se puede obtener una alfombra diferente.

En un ordenador, existen los mismos tipos de relaciones.

Sin embargo, si los datos se introducen en el interior de los casilleros utilizando la instrucción **LET**, una vez que supere los programas triviales, tendrá dos tipos de inconvenientes:

la escritura de las instrucciones **LET** es laboriosa
el cambio de entradas es también laborioso

Puede establecerse un modo por el cual los datos se le den al programa mientras se está ejecutando. La instrucción **INPUT** hará que el programa se pare y espere a que usted teclee algo en el teclado.

Teclee en primer lugar:

```
NEW↵
```

de forma que el programa almacenado previamente (si continúa) se borrará y todo quedará preparado para una nuevo programa. Teclee ahora:

```
10 LET precio = 15↵
20 PRINT "Cuantas plumas"↵
30 INPUT plumas↵
40 LET coste = precio * plumas↵
50 PRINT coste↵
RUN↵
```

El programa se parará en la línea 30 y usted debe teclear el número de plumas que desea, por ejemplo:

```
4↵
```

No olvide pulsar tecla ENTER. La salida será:

```
60
```

La instrucción **INPUT** necesita el nombre de una variable para que el sistema conozca dónde debe colocar los datos que vienen desde el teclado (los que usted teclea). El efecto de la línea 30, con lo que usted tecleó es el mismo que el del comando **LET.** En algunos casos, es más conveniente, si se desea una mayor interacción entre el usuario y el ordenador. Sin embargo, las instrucciones **LET** e **INPUT** son útiles sólo cuando no se utilizan demasiados datos. Para mayor número de datos, sin pausas en la ejecución del programa, necesitaremos algo más.

El SuperBASIC, como muchos BASICs, prorporciona otro método de entrada conocido como **READ** (lectura) de las instrucciones de datos. Podemos teclear de nuevo el programa anterior para obtener los mismos efectos que antes, pero sin pausas. Pruebe lo siguiente:

```
NEW↵
10 READ precio, plumas↵
20 LET coste = precio * plumas↵
30 PRINT coste↵
40 DATA 15,4↵
RUN↵
```

La salida será:

```
60
```

como anteriormente.

Cada vez que se ejecuta el programa el SuperBASIC necesita que se le informe dónde debe comenzar a leer los **DATOS.** Esta información puede introducirse al ordenador tecleando **RESTORE** seguido del número de la línea **DATA** o bien tecleando **CLEAR.** Ambos comandos también se pueden insertar al comienzo de los programas.

Cuando ya se ha ejecutado la línea 10, el sistema busca en el programa para encontrar una instrucción **DATA.** A continuación utiliza los valores de la instrucción **DATA** para tomar los valores de las variables de la instrucción **READ,** exactamente en el mismo orden. Normalmente las instrucciones **DATA** se colocan al final de un programa, y se utilizan por él, pero no se ejecutan en el sentido de que todas las otras líneas se ejecutan, una tras otra. Las instrucciones **DATA** puede estar en cualquier parte del programa, pero es mejor colocarlas al final, retiradas. Debemos pensar que estas instrucciones son necesarias, pero realmente no forman parte activa del programa. Las reglas sobre **READ** y **DATA** son las siguientes:

1. Todas las instrucciones **DATA** son consideradas como simples secuencias largas de elementos. Hasta el momento estos elementos han sido números, pero pueden ser palabras.

2. Cada vez que se ejecuta una instrucción **READ,** se copian los elementos

necesarios de la instrucción **DATA** en las variables que aparecen en la instrucción **READ.**

3. El sistema guarda la información de los elementos que se han leído **(READ)** mediante grabación interna. Si un programa intenta leer **READ** más elementos de los que existen en todas las instrucciones **DATA,** se detectará y señalará el correspondiente error.

IDENTIFICADORES (NOMBRES)

Hemos utilizado nombres para los "casilleros" tales como perro, onza, etc. Estos nombres pueden elegirse de forma que cumplan ciertas normas:

El nombre no puede incluir espacios.

El nombre debe comenzar por una letra.

El nombre debe estar formado por **letras, dígitos, $, %, _**(subrayado).

Los símbolos **$, %,** tienen una significación especial que explicaremos más adelante, pero pueden utilizarse para crear nombres, como por ejemplo:

```
perro_comida
mes_salario_total
```

de más fácil lectura.

El lenguaje SuperBASIC no distingue entre mayúsculas y minúsculas y, por ejemplo, *LATAS* y *latas,* son dos nombres iguales.

El número máximo de caracteres para un nombre es 255.

Los nombres que se forman de acuerdo con estas normas se llaman **identificadores.** Los identificadores se utilizan en SuperBASIC para muchas otras tareas, y es necesario que los comprenda. Las reglas permiten gran libertad en la elección de los nombres, y de este modo es más sencillo entender de qué programa se trata. Nombres como por ejemplo, *total, cuenta, plumas,* son de mucha más utilidad que Z, P, Q.

AUTOEXAMEN SOBRE EL CAPITULO 2

En este test puede obtener un máximo de 21 puntos. Compruebe su puntuación comparando sus respuestas con las respuestas de la página siguiente.

1. ¿Cómo piensa usted que es el almacenamiento interno de un número?
2. Explique dos formas de almacenar un valor en un "casillero" interno creado a tal efecto. (Dos formas.)
3. ¿Cómo puede usted averiguar el valor de un "casillero" interno?
4. Diga el nombre técnico que se utiliza para llamar a un "casillero" interno.
5. ¿Cuándo toma el casillero su primer valor?
6. Las variables se llaman así porque su valor puede cambiar mientras se ejecuta el programa. ¿Cuál es el método normal de realizar este cambio?
7. El signo = en una instrucción **LET** no tiene el mismo significado de "igual" que en Matemáticas. ¿Cuál es su significado?
8. ¿Qué ocurre cuando se introduce ↵ una instrucción sin numerar?
9. ¿Qué ocurre cuando se introduce ↵ una instrucción numerada?
10. Con qué fin se utilizan las comillas en una instrucción **PRINT.**

11. ¿Qué ocurre cuando no se utilizan comillas en una instrucción **PRINT**?
12. ¿Qué acción realiza una instrucción **INPUT** que no realice una instrucción **LET**?
13. ¿Qué tipo de instrucción de programa no se ejecuta nunca?
14. ¿Para qué se utiliza la instrucción **DATA**?
15. Especifique otro nombre para designar un casillero (o una variable).
16. Escriba tres identificadores válidos que utilicen letras, letras y dígitos, letras y subrayados (tres ejemplos).
17. ¿Por qué es tan importante el espaciador en SuperBASIC?
18. ¿Por qué son importantes en SuperBASIC los identificadores elegidos libremente?

PROBLEMAS SOBRE EL CAPITULO 2

1. Realice una simulación o pasada en vacío para mostrar los valores de todas las variables, mientras se ejecuta cada línea del siguiente programa.

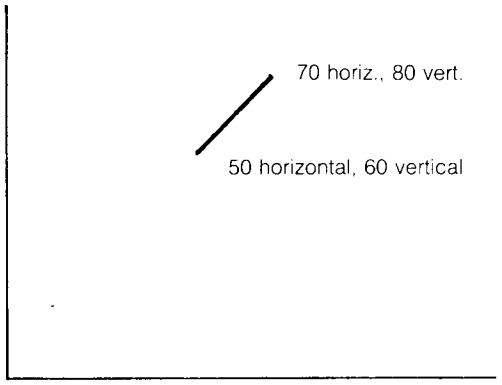
```
10 LET horas = 40
20 LET jornal = 3
30 LET sueldo = horas * jornal
40 PRINT horas, jornal, sueldo
```

2. Escriba y pruebe un programa similar al del problema 1, que obtenga el área de una alfombra de 3 metros de ancho por 4 de largo. Utilice como nombres para las variables *ancho, largo* y *área.*
3. Vuelva a escribir el programa del problema 1 de forma que utilice dos instrucciones **INPUT** en lugar de una instrucción **LET.**
4. Vuelva a escribir el programa del problema 1 de forma que los datos de entrada (40 y 3) aparezcan en una instrucción **DATA** en lugar de una instrucción **LET.**
5. Vuelva a escribir el programa del problema 2 utilizando un método diferente para introducir los datos. Use **READ** y **DATA** si utilizó originalmente **LET**, y viceversa.
6. Juan y Pedro hacen una apuesta. Cada uno de ellos sacará de su billetera todos sus billetes de 100 pesetas y se los dará al otro. Escriba un programa que simule esta acción, sólo con instrucciones **LET** y **PRINT.** Utilice una tercera persona, por ejemplo, Ana, para que sujete el dinero de Juan mientras éste acepta el de Pedro.
7. Vuelva a escribir el programa del problema 6 de forma que los dos números que deben intercambiarse sean introducidos con una instrucción **DATA.**

CAPITULO 3
DIBUJOS SOBRE LA PANTALLA

Con el fin de que se pueda utilizar el ordenador con un aparato de televisión o con un monitor. Disponemos de dos modos de pantalla diferentes para permitir el uso de sistemas de presentación diferentes (televisores domésticos o monitores). El modo MODE 8 permite visualizaciones con ocho colores con gráficos de una resolución de 256 por 256 pixels y un número de líneas máximo de 80, por lo que se necesita un monitor para que la visualización sea correcta. Sin embargo, sería muy desafortunado que un programa que se creó para que dibujara círculos o cuadrados en un modo, produjera elipses o rectángulos en el otro modo (como ocurre en algunos sistemas). Nosotros, sin embargo, disponemos de un sistema de gráficos a escala que evita esos problemas. Usted tiene que elegir sencillamente una escala vertical y trabajar con ella. El otro tipo de gráficos (orientados por pixels) también puede utilizarse y viene descrito en su totalidad en un capítulo posterior.

Suponga, por ejemplo, que elige una escala vertical de 100, y desea trazar una línea desde la posición (50,60) a la posición (70,80).



Gráficos a escala

UNA LINEA COLOREADA

Necesitamos especificar tres elementos:

PAPER-papel (color del fondo)
INK-tinta (color del dibujo)
LINE-línea (comienzo y final)

El siguiente programa dibuja una línea como la que se muestra en la figura anterior, en rojo (código de color 2) sobre fondo blanco (código de color 7).

```
NEW↵
10 PAPER 7 : CLS
20 INK 2↵
30 LINE 50,60 TO 70,80↵
RUN↵
```

En la línea 10 el color del papel se selecciona en primer lugar, pero su efecto tiene lugar con otro comando posterior como el **CLS**, que deja la pantalla limpia con el color de fondo vigente en ese momento.

MODOS Y COLORES

Hasta el momento no es importante el modo que se utilice, pero la gama de colores sí se ve afectada por el modo que se seleccione.

El MODO 8 permite ocho colores básicos
El MODO 4 permite cuatro colores básicos

Los colores tienen unos determinados códigos que se desciben más abajo.

Código	Efecto	
	8 colores	4 colores
0	negro	negro
1	azul	negro
2	rojo	rojo
3	morado	rojo
4	verde	verde
5	ciano	verde
6	amarillo	blanco
7	blanco	blanco

Por ejemplo **INK 3** dará el color magenta (morado) en el **MODO 8** y rojo en el **MODO 4.**

En otro capítulo posterior explicaremos cómo se pueden "mezclar" los colores básicos de varias formas para producir una gama de colores, sombras y texturas realmente increíble.

EFECTOS ALEATORIOS

Se pueden obtener algunos efectos interesantes con números aleatorios que pueden generarse con la función **RND.** Por ejemplo:

```
PRINT RND (1 TO 6)↵
```

y se imprimirá un número entero de la gama que va del 1 al 6, exactamente igual que si hubiéramos tirado un dado de seis lados normal. El siguiente programa ilustra lo anterior:

```
NEW↵
10 LET dado = RND (1 TO 6)↵
20 PRINT dado↵
RUN↵
```

Si se ejecuta el programa varias veces se obtendrán números diferentes.

Pueden obtenerse números enteros dentro de cualquier gama que se desee, por ejemplo:

```
RND (0 TO 100)
```

producirá un número que puede utilizarse en gráficos a escala. El programa puede volverse a escribir de forma que produzca una línea aleatoria en un color aleatorio. En los casos en los que los números aleatorios comiencen en cero, puede omitirse el primer número y escribir:

```
RND (100)
```

```
NEW↵
10 PAPER 7 : CLS↵
20 INK RND(5)↵
30 LINE 50,60 TO RND(100), RND(100)↵
RUN↵
```

Este programa producirá una línea que comienza en algún punto cerca del centro de la pantalla y terminará en algún punto aleatorio. La gama de colores posibles depende del modo seleccionado. Encontrará que en SuperBASIC es frecuente tener una gama de números "algo TO algo" (de algo A algo).

BORDES

La parte de la pantalla sobre la que se dibujan las líneas y se crean otro tipo de salidas se llama **ventana.** Más adelante verá como puede cambiar el tamaño y la posición de una ventana o crear otras ventanas. Por el momento, nos contentaremos con dibujar un borde rodeando la presente ventana. El

área menor de color más tenue que puede dibujarse en la pantalla se llama pixel. En el modo 8, que es el llamado **modo de baja resolución** existen 256 posibles posiciones para los pixels en una de las dos direcciones de la pantalla y otras 256 en la otra dirección. En el modo 4, llamado **modo de alta resolución** disponemos de 512 pixels en la dirección horizontal de la pantalla y 256 en la dirección vertical. Por tanto, el tamaño de un pixel depende del modo.

Para crear un borde que rodee el extremo interior de la ventana, teclee, por ejemplo

```
BORDER 4,2↵
```

Con ello se creará un borde de 4 pixels de anchura de color rojo (código 2). El tamaño efectivo de la ventana se ha reducido con el borde. Esto significa que cualquier impresión de gráficos que se realice después, se creará en el interior del nuevo tamaño de la ventana. La única excepción es el trazado de otro borde posterior que se superpondrá sobre el borde ya existente.

UN BUCLE SENCILLO

Los ordenadores pueden realizar las acciones muy rápidamente, pero no sería posible explotar su gran potencia si cada acción se escribiera en forma de instrucción. Este problema es similar al que puede presentársele a un maestro de obras que manda a un albañil que coloque cien adoquines, y así es, más o menos, como se lo dice. Evidentemente no le da cien instrucciones separadas.

Una forma tradicional de crear bucles o repeticiones en BASIC es usar la instrucción GO TO (o GOTO, indistintamente), como sigue:

```
NEW↵
10 PAPER 6↵
20 BORDER 1,2↵
30 INK RND(5)↵
40 LINE 50,60 TO RND(100), RND(100)↵
50 GOTO 40↵
RUN↵
```

Quizás prefiera usted no teclear este programa ya que el SuperBASIC le permite una forma mejor de realizar las repeticiones. Observe en cada línea los siguientes elementos:

10 20	Parte fija-no se repite
30 40	Parte fija-se repite
50	Controles del programa

Puede volver a escribir los programas anteriores omitiendo la instrucción GOTO y colocando en su lugar REPeat y END REPeat rodeando la parte que se va a repetir.

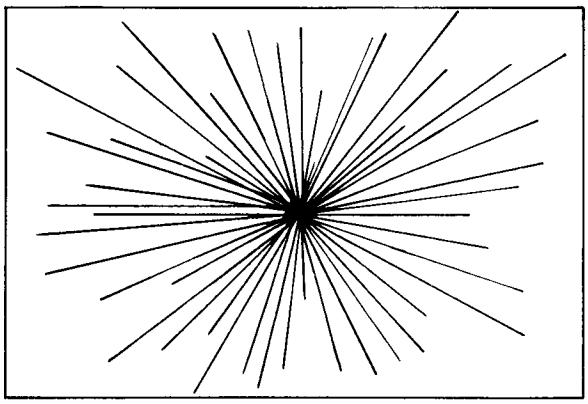
```
NEW↵
10 PAPER 6↵
20 BORDER 1,2↵
30 REPEAT estrella↵
40   INK RND5↵
50   LINE 50,60 TO RND(100), RND(100)↵
60 END REPEAT estrella↵
RUN↵
```

Hemos dado el nombre *estrella* a la *estructura de repetición.* Esta estructura consiste en dos líneas:

```
REPeat estrella
END REPeat estrella
```

todo lo que se encuentre entre estas dos líneas se llama **contenido** de la estructura.

Este programa produce líneas coloreadas indefinidamente, para hacer que la estrella se muestre tal y como en la figura a continuación.



Programa ESTRELLA

Puede pararse pulsando las teclas BREAK de interrupción:

mantenga pulsada la tecla **CTRL** y luego pulse **SPACE**

El lenguaje SuperBASIC le proporciona un método consistente y versátil para parar los procesos repetitivos. Imagínese que se está moviendo varias veces por el interior de las instrucciones de activación del programa. ¿Cómo podrá salir? La respuesta es utilizar una instrucción **EXIT**. Sin embargo debe haber una razón para salir del programa. Podemos, por ejemplo, desear ampliar la elección de los colores e incluir el blanco, tecleando una corrección al programa (no teclee **NEW**).

```
40 INK RND 0,6
```

de este modo si **RND** produce un 6, la tinta será blanca y no se verá. Esta puede ser la razón para finalizar la repetición. Podemos arreglar el programa del siguiente modo:

```
NEW
10 PAPER 6
20 BORDER 1,2
30 REPeat estrella
40   LET color = RND(6)
50   IF color = 6 THEN exit estrella
60   INK color
70   LINE 50,60 TO RND(100), RND(100)
80 END REPeat estrella
```

Lo importante a notar en este ejemplo es que el programa continúa hasta que el color se hace 6. El control sale del bucle inmediatamente después de la línea 90. Como no existe líneas después de la línea 90, el programa se para.

Se ha introducido otro concepto importante, la idea de la decisión:

```
IF color = 6 THEN EXIT estrella
```

Esta estructura es también muy útil, ya que supone una elección para realizar una acción o no realizarla; a este concepto se le llama decisión simple binaria. Su forma general es:

```
IF condición THEN instrucción(es)
```

Más adelante apreciará cómo estos dos conceptos de repetición (o bucle) y la toma-de-decisión (o selección) son dos estructuras fundamentales en el control de programas. Para parar estas estructuras basta pulsar las teclas de interrupción, es decir, mantener pulsado **CRTL** y pulsar luego la barra de espacios.

AUTOEXAMEN SOBRE EL CAPITULO 3

Con el test que se da a continuación, se pueden obtener hasta un máximo de 13 puntos. Obtenga su puntuación comprobando sus respuestas con las de la siguiente página.

1. ¿Qué es un pixel?
2. En el modo de baja resolución, ¿cuántos pixels caben en una línea horizontal de la pantalla?
3. En el modo de baja resolución, ¿cuántos pixels caben en sentido vertical en la pantalla?
4. ¿Qué dos números son los que determinan la dirección o posición de un punto de un gráfico en la pantalla, cuando se utiliza una escala?
5. ¿De cuántos colores se dispone en el modo de baja resolución?
6. Nombre las palabras clave que realizan lo siguiente:

   i. seleccionan la escala para el dibujo
   ii. dibujan una línea
   iii. seleccionan el color del dibujo
   iv. seleccionar el color del fondo
   dibujan un reborde (5 puntos)

7. Establezca las instrucciones que abren y cierran el bucle de **REPetición.**
8. ¿Cuando termina un bucle de **REPetición** que se está ejecutando?
9. ¿Por qué en SuperBASIC los bucles tienen nombres?

PROBLEMAS SOBRE EL CAPITULO 3

1. Escriba un programa que dibuje líneas rectas por toda la pantalla. La dirección y longitud de estas líneas debe ser aleatoria. Cada una de ellas debe comenzar donde termina la anterior, y su color debe elegirse también aleatoriamente.
2. Escriba un programa que dibuje líneas aleatoriamente, con la restricción de que cada línea tenga su comienzo en un punto aleatorio del borde izquierdo de la pantalla.
3. Escriba un programa que dibuje líneas aleatoriamente con la restricción de que las líneas comiencen en el mismo punto situado en el borde inferior de la pantalla.
4. Escriba un programa que produzca líneas de longitud, punto de comienzo y color aleatorios. Las líneas deben ser horizontales.
5. Realice el problema 4, pero con las líneas verticales.
6. Escriba un programa que produzca una "espiral" cuadrada, de forma que la trayectoria de cada línea sea de un color diferente.

   SUGERENCIA: Busque primero las coordenadas de algunos vértices y luego colóquelas en grupos de cuatro. A continuación, descubra el patrón.

CAPITULO 4
CARACTERES Y CADENAS

Algunas veces, los profesores desean asesorarse sobre la habilidad para la lectura que requieren algunos libros o de algunos materiales especiales de clase. Para ello utilizan varios tipos de test, algunos de los cuales computan las longitudes medias de palabras y frases. Vamos a introducir algunas ideas sobre el manejo de las palabras o **cadenas de caracteres,** examinando algunas aproximaciones al estudio de la búsqueda de longitudes medias de palabras.

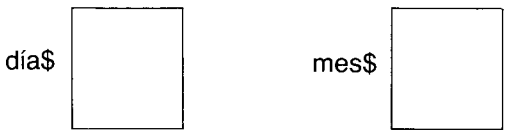
Hablamos de secuencias de letras, dígitos u otros símbolos que pueden ser o no palabras. Esta es la razón por la que se ha inventado el término "cadena de caracteres". Normalmente se abrevia a **cadena.** Las cadenas se manejan de forma muy similar a como se manejan los números, pero, evidentemente, no se realizan con ellas las mismas operaciones que con los números. Las cadenas no pueden multiplicarse o sustraerse. Pueden unirse, separarse, buscarse, y generalmente pueden manipularse como se desee.

NOMBRES Y CASILLEROS PARA LAS CADENAS

Para las cadenas también pueden crearse casilleros. Se pueden colocar cadenas de caracteres dentro de ellos, y utilizar la información del mismo modo que se hacía con los números. Si intentamos almacenar (no todo al mismo tiempo) palabras como:

PRIMERO SEGUNDO TERCERO
y
ENERO FEBRERO MARZO

podemos elegir dos nombres para los casilleros:



Observe el signo de dólar. Los casilleros para cadenas son diferentes internamente de los casilleros para los números, y por tanto el lenguaje SuperBASIC necesita saber de qué tipo de casilleros se trata. Todos los nombres de estos casilleros de cadena deben terminar en **$**. Aparte de esto, las reglas para la elección de nombres son las mismas que las que rigen los casilleros numéricos.

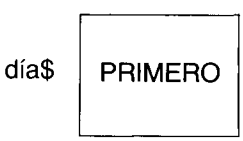
Se puede pronunciar:

días$ como diadólar
mes$ como mesdólar

La instrucción **LET** funciona del mismo modo que para los números. Si teclea:

```
LET día$ = "PRIMERO"
```

se establecerá un casillero interno llamado *día$,* con PRIMERO en su interior, es decir:



Las comillas no se almacenan. Se utilizan en la instrucción **LET** para que resulte absolutamente evidente lo que se debe almacenar en el casillero. Pueden utilizarse apóstrofes en lugar de comillas. Compruébelo escribiendo:

```
PRINT día$
```

y aparecerá en la pantalla el contenido del casillero:

```
PRIMERO
```

En lugar de utilizar comillas puede utilizar apóstrofes.

LONGITUDES DE LAS CADENAS

En el lenguaje SuperBASIC es muy fácil averiguar la longitud o el número de caracteres de una cadena. Sólo tiene que escribir, por ejemplo:

```
PRINT LEN(día$)
```

Si el casillero *día$* contiene PRIMERO, se visualizará en la pantalla el número 7. Veamos el efecto en un programa sencillo:

```
NEW
10 LET día$ = "PRIMERO"
20 PRINT LEN(día$)
RUN
```

En la pantalla aparecerá:

```
7
```

LEN es una palabra clave del lenguaje SuperBASIC.

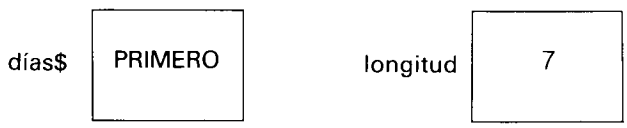
Otro método alternativo para obtener el mismo resultado utiliza ambos casilleros, los de cadena y los numéricos.

```
NEW
10 LET día = "PRIMERO"
20 LET longitud = LEN(día$)
30 PRINT longitud
RUN
```

En pantalla aparecerá:

```
7
```

como antes, y los valores que se muestran estarán en dos casilleros internos:



Volvamos al problema de la longitud media de las palabras.

Escriba un programa que encuentre la longitud media de tres palabras:

```
PRIMERO, DE, AGOSTO
```

DISEÑO DEL PROGRAMA

Cuando los problemas no sean ya tan triviales, es interesante crearse un **diseño de programa**, antes de la creación del propio programa.

1. Almacene las tres palabras en casilleros.
2. Calcule las longitudes y almacénelas.
3. Calcule la media.
4. Imprima el resultado.

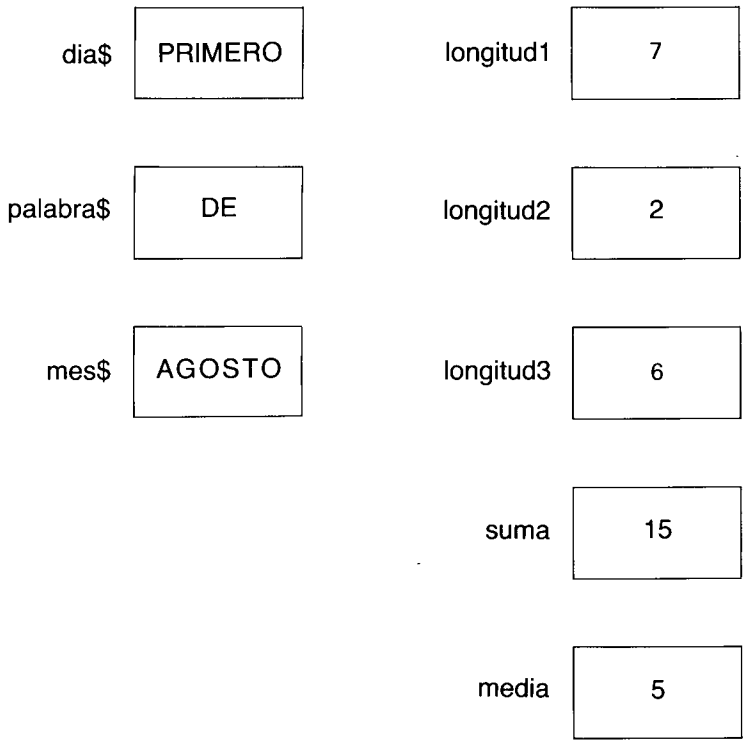
```
NEW
10 LET día$ = "PRIMERO"
20 LET palabra$ = "DE"
```

```
30 LET mes$ = "AGOSTO"
40 LET longitud1 = LEN(día$)
50 LET longitud2 = LEN(palabra$)
60 LET longitud3 = LEN(mes$)
70 LET = longitud1 + longitud2 + longitud3
80 LET media = suma / 3
90 PRINT media
RUN
```

El signo / significa **dividido por.** La salida o el resultado de la ejecución del programa es sencillamente:

```
5
```

y en esta operación habrán participado ocho casilleros internos:



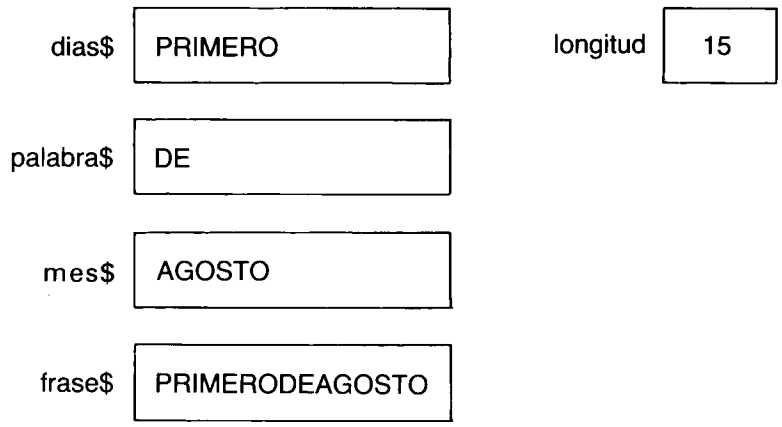
Si piensa que es mucho trabajo para un problema tan sencillo, puede evidentemente acortarlo. La versión más corta sería una línea única, pero resultaría menos fácil su lectura. Una fórmula de compromiso razonable utiliza el signo & que realiza la operación siguiente:

Une dos cadenas

Teclee ahora:

```
NEW
10 LET día$ = "PRIMERO"
20 LET palabra$ = "DE"
30 LET mes$ "AGOSTO"
40 LET frase$ = día$ & palabra$ & mes$
50 LET longitud = LEN(frase$$)
60 PRINT longitud/3
RUN
```

La salida es 5, como antes, pero algunos efectos internos son diferentes:



otra simplificación más razonable consiste en utilizar **READ** y **DATA** en lugar de las tres primeras instrucciones **LET**. Teclee:

```
NEW↵
10 READ día$, = palabra$, mes$↵
20 LET frase$ día$ & palabra$ & mes$↵
30 LET longitud = LEN(frase$)↵
40 PRINT longitud / 3↵
50 DATA "PRIMERO", "DE", "AGOSTO"↵
RUN↵
```

Los efectos internos de esta versión son exactamente iguales a los de la anterior. La instrucción **READ** establece casilleros internos con valores dentro de ellos, de forma similar a la instrucción **LET**.

IDENTIFICADORES Y VARIABLES DE CADENA

Los nombres de casilleros del tipo:

días$
palabra$
mes$
frase$

se llaman identificadores de cadena. El signo dólar implica que los casilleros se han creado para cadenas. Este signo dólar debe ir siempre al final. Es frecuente pronunciar este tipo de palabras como "diadólar", etc.

Los casilleros de este tipo se denominan **variables de cadena**, ya que sólo contienen cadenas de caracteres que pueden variar durante la ejecución del programa.

Los contenidos de estos casilleros se llaman valores. Así pues, las palabras "PRIMERO" y "DE" son valores de las variables de *cadena días$* y *palabra$*.

CARACTERES ALEATORIOS

Para la generación de letras aleatorias, pueden utilizarse códigos de carácter (véase la *Guía de Referencia de Conceptos*). Las letras mayúsculas de la A a la Z tienen los códigos que van desde el 65 al 90. La función **CHR$** convierte esos códigos en letras. El programa siguiente, imprimirá la letra B:

```
NEW↵
10 LET código = 66↵
20 PRINT CHR$(código)↵
RUN↵
```

El programa siguiente genera tríos de letras A, B o C hasta que accidentalmente se cree la palabra inglesa CAB.

```
NEW
10 REPeat taxi
20 LET primero$ = CHR$(RND(65 TO 67))
30 LET segundo$ = CHR$ (RND (65 TO 67))
40 LET tercero$ = CHR$(RND(65 TO 67))
50 LET palabra$ = primero$ & segundo$ & tercero$
60 PRINT ! palabra$
70 IF palabra$ = "CAB" THEN EXIT taxi
80 END REPeat taxi
```

Los caracteres aleatorios (números aleatorios o puntos aleatorios) son útiles, por ejemplo, para aprender a programar. Resulta muy sencillo obtener efectos interesantes en los ejemplos y en los ejercicios.

Observe el efecto de ! en los espacios que aparecen en la salida.

AUTOEXAMEN SOBRE EL CAPITULO 4

En el siguiente test, puede obtener un máximo de 10 puntos. Obtenga su puntuación comparando sus respuestas con las de la página siguiente.

1. Defina que es una cadena de caracteres.
2. ¿Cuál es la abreviación usual del término "cadena de caracteres".
3. ¿Cómo se distingue el nombre de una variable de cadena?
4. ¿Cómo suele pronunciarse una palabra del tipo "palabra$"?
5. ¿Qué palabra clave se utiliza para obtener el número de caracteres de una cadena?
6. ¿Qué símbolo se utiliza para unir dos cadenas?
7. Los espacios pueden ser parte de la cadena. ¿Cuáles son los límites de una cadena definida?
8. En una instrucción del tipo:

   LET carne$ = "solomillo"

   cuando se ejecuta, ¿se almacenan las comillas?
9. ¿Cuál es la función que asigna a cada letra el número de código adecuado?
10. Explique la forma de generar aleatoriamente letras mayúsculas.

PROBLEMAS SOBRE EL CAPITULO 4

1. Almacene la palabra "Buenos" y "días" en dos variables separadas. Utilice una instrucción **LET** para unir los valores de las dos variables en una tercera variable. Imprima el resultado.

2. Almacene las palabras siguientes en cuatro casilleros diferentes.

   deja la caja negra

   Una las palabras para formar una frase añadiendo espacios y el punto. Almacene la frase completa en una variable, *envío$* e imprima la frase y el número total de caracteres que contiene.

3. Escriba un programa que utilice las palabras clave:

   CHR$(RND(65 TO 90))

   para generar cien palabras aleatorias de tres letras. Vea si ha generado accidentalmente alguna palabra española. Compruebe el efecto de:

   a) ; al final de una instrucción **PRINT**.
   b) ! en los dos lados de un elemento de impresión.

CAPITULO 5
PRACTICANDO LO APRENDIDO

Hasta el momento hemos empezado a trabajar con efectividad en programas cortos. Las indicaciones siguientes pueden serle de utilidad:

1. Utilice letras minúsculas para: los nombres de las variables (casilleros) o estructuras de repetición, etc.
2. Cambie el margen de las instrucciones para mostrar el contenido de una estructura de repetición.
3. Elija bien los identificadores, de forma que reflejen para qué se va a utilizar una determinada variable o estructura.
4. Edite un programa:

   sustituyendo una línea
   insertando una línea
   borrando una línea

EJEMPLOS DE PROGRAMAS

Usted ha llegado a un estado en el que es muy útil que estudie los programas para aprender de ellos y tratar de entender lo que realizan. La mecánica de su ejecución estará ya perfectamente comprendida y, por lo tanto, en los capítulos siguientes evitaremos repetirla constantemente.

NEW antes de cada programa
◆ al final de cada programa
RUN Para arrancar el programa

Debe usar estos tres elementos cada vez que desee introducir o ejecutar un programa. Su omisión le imposibilitará observar otros detalles con más claridad mientras imagina lo que hace el programa cuando se está ejecutando.

Si pasamos sobre los detalles anteriores, podemos entender y utilizar más fácilmente los programas sin el embrollo técnico. Por ejemplo, el programa siguiente genera letras mayúsculas aleatoriamente hasta que aparece la Z. No visualiza las palabras **NEW** o **RUN** o el símbolo **ENTER**, pero su utilización sigue siendo necesaria.

```
10 REPeat letras
20   código = RND (65 TO 90)
30   may$ = CHR$ (código)
40   PRINT may$
50   IF may$ = "Z" THEN EXIT letras
60 END REPeat letras
```

En este capítulo y en todos los siguientes, los programas se presentarán sin los símbolos **ENTER**. Los comandos directos también se mostrarán sin estos símbolos, aunque usted debe utilizarlos como siempre. Recuerde también que debe utilizar **NEW** y **RUN** en los casos necesarios.

NUMERACION DE LINEA AUTOMATICA

Resulta muy tedioso introducir los números manualmente. En lugar de hacerlo, puede teclear:

```
AUTO
```

antes de comenzar a programar; y el QL le contestará con un número de línea:

```
100
```

Continúe tecleando líneas hasta que termine su programa, y la pantalla irá mostrando:

```
100 PRINT "Primero"
110 PRINT "Segundo"
120 PRINT "End"
```

Para finalizar la producción automática de líneas utilice la secuencia de interrupción BREAK:

Mantenga pulsado **CTRL** y pulse la barra de espacios. Con ello producirá el siguiente mensaje:

```
130 incompleto
```

y la línea 130 no se incluirá en su programa.

Si comete un error que no causa la interrupción de la numeración automática, puede continuar y editar-**EDIT** la línea posteriormente. Si desea comenzar en un número de línea determinado, por ejemplo en la línea 600, y utilizar un incremento de línea diferente del anterior, por ejemplo 5, en lugar de 10 puede hacerlo del siguiente modo:

```
AUTO 600,5
```

Las líneas se numerarán desde ahora del siguiente modo:

```
600, 605, 610, etc.
```

Para cancelar el comando **AUTO**, pulse **CTRL** y la barra de espacios al mismo tiempo.

EDICION DE UNA LINEA

Para editar una única línea, teclee **EDIT** seguido por el número de línea, por ejemplo:

```
EDIT 110
```

A continuación se visualizará la línea con el cursor al final:

```
110 PRINT "Segundo"
```

Para mover el cursor, utilice:

← un espacio a la izquierda
→ un espacio a la derecha

Para borrar un carácter a la izquierda utilice

CTRL con ←

Para borrar un carácter que se encuentre en la posición del cursor, teclee:

CTRL con →

y el carácter situado a la derecha del cursor se moverá para cerrar el hueco.

UTILIZACION DE LOS CARTUCHOS DE MICRODRIVES

Antes de utilizar un nuevo cartucho de Microdrive, debe formatearlo. Siga las instrucciones que se especifican en la *Introducción* a la Guía del Usuario del QL. La elección del nombre para el cartucho debe seguir las mismas reglas que los identificadores del SuperBASIC, etc., pero limitándolo a 10 caracteres. Es interesante escribir el nombre del cartucho sobre el mismo cartucho, utilizando una de las etiquetas adhesivas que se incluyen.

Debe disponerse siempre de, al menos, una copia de cualquier programa o dato. Siga las instrucciones que se dan en la sección de información al final de la Guía del Usuario.

ATENCION

SI formatea (FORMAT) un cartucho que contenga programas y/o datos, TODOS los programas y los datos se perderán.

ALMACENAMIENTO DE LOS PROGRAMAS

El programa siguiente establece unos bordes de 8 pixels de anchura en color rojo (código 2) en tres ventanas que se designan #0, #1, #2.

```
100 REMark Border
110 FOR k = 0 TO 2 : BORDER #k,8,2
```

Puede guardarlo en un Microdrive insertando el cartucho y tecleando:

```
SAVE mdv1_bord
```

El programa se guardará en un archivo de Microdrive cuyo nombre es **bord.**

COMPROBACION DE UN CARTUCHO

Si desea saber qué programas o archivos de datos se encuentran en un cartucho en particular, colóquelo en el Microdrive 1 y teclee:

```
DIR mdv1_
```

Aparecerá un índice (directorio) en la pantalla. Si el cartucho se encuentra en el Microdrive 2, teclee en su lugar:

```
DIR mdv2_
```

COPIA DE PROGRAMAS Y ARCHIVOS

Una vez almacenado un programa como archivo en un cartucho de Microdrive, podrá copiarlo a otros archivos. Este es un método de realización de copias de seguridad del cartucho del Microdrive. Todos los programas anteriores pueden copiarse en otro cartucho en el Microdrive 2, tecleando:

```
COPY mdv1_bord TO mdv2_bord
```

BORRADO DE UN ARCHIVO DEL CARTUCHO

Un archivo es algo semejante a un programa o a unos datos, almacenados en un cartucho. Para borrar un programa, llamado por ejemplo *prog,* teclee:

```
DELETE mdv1_prog
```

CARGA DE PROGRAMAS

Los programas pueden cargarse desde el cartucho del Microdrive tecleando:

```
LOAD mdv2_bord
```

Si el programa se carga correctamente, significará que ambas copias son buenas. Pruebe el programa utilizando:

LIST para listarlo
RUN para ejecutarlo

También puede, en lugar de utilizar **LOAD** seguido de **RUN,** combinar las dos operaciones en un solo comando:

```
LRUN mdv2_bord
```

y el programa se cargará y se ejecutará inmediatamente.

INTERCALACION DE PROGRAMAS

Suponga que tiene dos programas almacenados en el Microdrive 1 como *prog1* y *prog2:*

```
100 PRINT "Primero"
110 PRINT "Segundo"
```

Si teclea:

```
LOAD mdv1_prog1
```

seguido de:

```
MERGE mdv1_prog2
```

Los dos programas se combinarán en un solo. Para verificarlo, teclee **LIST** y verá en la pantalla:

```
110 PRINT "Primero"
110 PRINT "Segundo"
```

Si intercala **(MERGE)** un programa, asegúrese que sus números de línea sean diferentes de los del programa que ya se encuentra en la memoria principal. Si no lo hace así, sobreescribirá algunas de las líneas del primer programa. Esta característica es muy útil cuando ya se ha alcanzado un cierto grado de eficacia en el manejo de los procedimientos. En ese momento, resultará una operación natural la creación de un programa al que se le van añadiendo procedimientos y funciones.

GENERAL

Sea cuidadoso y metódico con los cartuchos. Disponga siempre de alguna copia, y si piensa que puede haber algún problema con un cartucho o Microdrive, obtenga una segunda copia. Los profesionales informáticos rara vez pierden datos. No desconocen el hecho de que incluso las máquinas o dispositivos mejores tienen fallos ocasionales y, por tanto, lo tienen en cuenta.

Si desea dar a un programa un determinado nombre, por ejemplo, *cuadrado,* puede ser muy útil asignarle nombres como por ejemplo *cdr1, cdr2,...,* en las versiones preliminares. Cuando el programa se encuentre ya en su forma final, realice por lo menos dos copias con el nombre *cuadrado,* y borre las demás copias bien reformateando el cartucho o por otro método más selectivo.

AUTOEXAMEN SOBRE EL CAPITULO 5

En el test siguiente se puede obtener una puntuación máxima de 14 puntos. Compruebe su puntuación comparando sus respuestas con las respuestas de la página siguiente.

1. ¿Cuál es la razón para preferir elegir las letras minúsculas para las palabras del programa?
2. ¿Por qué razón se eligen márgenes diferentes?
3. ¿Qué razonamiento precede su elección de los identificadores para las variables y los bucles?
4. Nombre tres formas diferentes de editar un programa contenido en la memoria principal del ordenador.
5. ¿Qué debe recordar teclear al final de cada comando o línea del programa, cuando lo introduce?
6. ¿Qué teclea normalmente antes de introducir un programa?
7. ¿Qué debe colocarse al comienzo de cada línea que vaya a almacenarse como parte de un programa?
8. ¿Qué debe teclearse para que un programa se ejecute?
9. ¿Cuál es la palabra clave que le permite colocar cierta información en un programa, sin que tenga, sin embargo, efecto alguno en su ejecución?
10. ¿Cuál son los comandos que le permiten almacenar programas y recuperarlos de los cartuchos? (dos comandos).

PROBLEMAS SOBRE EL CAPITULO 5

1. Vuelva a escribir el programa siguiente utilizando letras minúsculas para obtener una presentación mejor. Añada las palabras **NEW** y **RUN.** Utilice numeración para las línas y el símbolo **ENTER** como si deseara introducir y ejecutar un programa. Utilice **REMark** para dar un nombre al programa.

```
LET DOS$ = "DOS"
LET CUATRO$ $ = "CUATRO"
LET SEIS$ = DOS$ & CUATRO$
PRINT LEN(seis$)
```

   Explique cómo es posible que dos y cuatro den 9.

2. Utilice cambios de márgenes, letras minúsculas, **NEW, RUN,** numeración de líneas y el símbolo **ENTER** para mostrar cómo se introduce y ejecuta el programa siguiente:

```
REPEAT Bucle
Código_letra = RND(65 TO 90)
LET letras$ = CHR$ (Código _letra)
PRINT Letras$
IF Letras$ = 'Z' THEN EXIT Bucle
END REPEAT Bucle
```

3. Vuelva a escribir el programa siguiente con un estilo mejor, utilizando nombres de variables inteligibles y una buena presentación. Escriba el programa como si fuera a introducirlo.

```
LET S = 0
REPeat TOTAL
LET N = RND (1 TO 6)
PRINT ! N !
LET S = S + N
IF n = 6 THEN EXIT TOTAL
END REPeat TOTAL
PRINT S
```

Decida lo que realiza el programa y ejecútelo para comprobar su decisión.

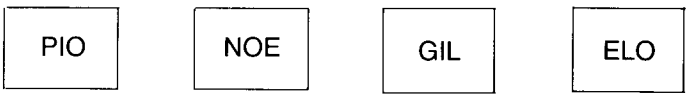
CAPITULO 6
MATRICES Y BUCLES "FOR"

QUE ES UNA MATRIZ

Usted ya sabe que las variables pueden tomar como valores las cadenas de números o de caracteres. Imagínelo como unos números o palabras que se introducen en casilleros internos. Suponga, por ejemplo, que se debe enviar a un pueblo pequeño a cuatro empleados de una compañía, quizá porque se ha descubierto en él petróleo. El pueblo es tan pequeño que sólo llevan nombre las casas, y sólo hay cuatro que se alquilan. Todos los nombres de las casas terminan con el símbolo dólar.

Miralmar$ Bellavista$ Rosaleda$ Robledal$

Los cuatro empleados se llaman:



Puede asignárseles casa por uno de los dos sistemas:

Programa 1

```
100 LET miralmar$ = "PIO"
110 LET bellavista$ = "NOE"
120 LET rosaleda$ = "GIL"
130 LET robledal$ = "ELO"
140 PRINT ! miralmar$ ! bellavista$ ! rosaleda$ ! robledal$ !
```

Programa 2

```
100 READ miralmar$, bellavista$, rosaleda$, robledal$
110 PRINT ! miralmar$ ! bellavista$! rosaleda$ ! robledal$ !
120 DATA "PIO", "NOE", "GIL", "ELO"
```

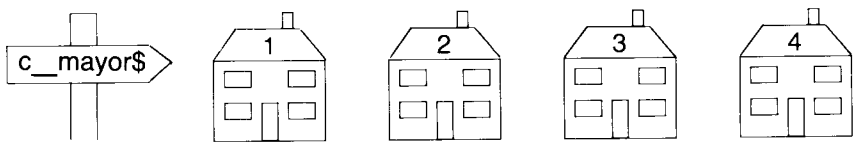
Cuando aumenta el número de los datos se hacen más evidentes las ventajas de las instrucciones **READ** y **DATA** frente a **LET**. Sin embargo, cuando los datos son verdaderamente muy numerosos, el problema de encontrar los nombres para las casas resulta tan dificultoso como encontrar casas vacías en un pueblo pequeño.

La solución a este problema y a muchos otros problemas en el manejo de los datos estriba en un nuevo tipo de casillero o variable en el que pueda compartirse un único nombre. Sin embargo debe poderse distinguir cada variable, de forma que cada una de ellas también tenga un número (como las casas numeradas de una misma calle). Suponga que necesita cuatro casas vacías en la Calle Mayor, numeradas del 1 al 4. En el lenguaje SuperBASIC se denomina una **matriz** de cuatro casas. El nombre de esta matriz es *c_mayor$*, y las cuatro casas se numeran del 1 al 4.

Sin embargo, estas matrices de variables no pueden tratarse como si fueran variables normales (simples). Antes que nada debe declararse la dimensión (tamaño de la matriz). El ordenador busca internamente un espacio para ello y necesita saber cuántas variables de cadena existen en la matriz, así como también la longitud máxima de cada variable de la cadena. Utilice por tanto la instrucción **DIM** del siguiente modo:

```
DIM c_mayor$(4,3)
          |  └──── longitud máxima de la cadena
          └─────── número de variables de cadena
```

Después de haber ejecutado la instrucción **DIM**, las variables ya están dispuestas para su uso. Es una situación semejante a unas casas que ya están terminadas en su construcción pero continúen vacías. Las cuatro "casas" comparten un nombre común, *c_mayor$*, pero cada una de ellas tiene su propio número y puede guardar tres caracteres.



Los cinco programas que se dan más abajo realizan la misma acción: "ocupan" las cuatro "casas", e imprimen **PRINT** el resultado para mostrar que efectivamente se ocuparon. El método final sólo utiliza cuatro líneas, pero los otros cuatro llegan a ello mediante un proceso que conduce desde las ideas ya conocidas a otras nuevas o a nuevos usos de esas ideas ya conocidas a otras nuevas o a nuevos usos de esas ideas ya conocidas. Este proceso va siempre buscando una mayor economía.

Si entiende perfectamente los primeros dos o tres métodos puede que prefiera ir directamente a los métodos 4 y 5. Si tiene dudas, los métodos 1, 2 y 3 le ayudarán a aclarar conceptos.

Programa 1

```
100 DIM c_mayor$(4,3)
110 LET c_mayor$(1) = "PIO"
120 LET c_mayor$(2) = "NOE"
130 LET c_mayor$(3) = "GIL"
140 LET c_mayor$(4) = "ELO"
150 PRINT ! c_mayor$(1) ! c_mayor$(2) !
160 PRINT c_mayor(3) ! c_mayor$(4) !
```

Programa 2

```
100 DIM c_mayor$(4,3)
110 READ c_mayor$(1), c_mayor$(2), c_mayor$(3), c_mayor$(4)
120 PRINT ! c_mayor$(1) ! c_mayor$(2) !
130 PRINT ! c_mayor$(3) ! c_mayor$(4) !
140 DATA "PIO", "NOE", "GIL", "ELO"
```

Este programa muestra cómo economizar en los nombres de las variables, pero la repetición de *c_mayor$* resulta muy tediosa y complica el aspecto general de los programas. De nuevo podemos aplicar una técnica conocida —el bucle de repetición **REPeat**— que mejorará aún más las cosas. Establecemos una cuenta, un número que se va incrementando de uno en uno y el bucle de repetición **REPeat**.

Programa 3

```
100 RESTORE 190
110 DIM c_mayor$(4,3)
120 LET número = 0
130 REPeat casas
140   LET número = número + 1
150   READ c_mayor$(número)
160   IF número = 4 THEN EXIT casas
170 END REPeat casas
180 PRINT c_mayor$(1) ! c_mayor$(2) ! c_mayor$(3) ! c_mayor$(4)
190 DATA "PIO", "NOE", "GIL", "ELO"
```

Este tipo especial de bucle, en el que se debe repetir una acción un cierto número de veces, es muy usual. Para él se ha inventado una estructura especial,

llamada bucle **FOR**. En este tipo de bucle, la cuenta de 1 a 4 se realiza automáticamente. Del mismo modo ocurre con la salida, una vez actuados los cuatro elementos.

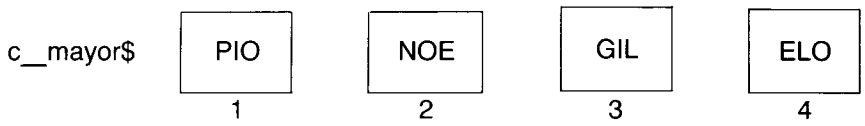
Programa 4

```
100 RESTORE 160
110 DIM c_mayor$(4,3)
120 FOR número = 1 TO 4
130   READ c_mayor$(número)
140   PRINT ! c_mayor$(número) !
150 END FOR número
160 DATA "PIO", "NOE", "GIL", "ELO"
```

La salida de los cuatro programas será la misma:

```
PIO    NOE    GIL    ELO
```

lo que prueba que se han almacenado los datos en el interior en las cuatro variables de la matriz adecuadamente:



El método 4 es claramente el mejor, ya que puede manejar con la misma eficacia 4, 40 ó 400 elementos sin más que cambiar el número 4 o añadir más elementos en **DATA**. Puede utilizar tantas instrucciones **DATA** como sean necesarias.

En su forma más simple el bucle FOR es semejante a la forma más simple del bucle **REPeat**. Vamos a compararlos.

```
100 REPeat salud          100 FOR saludo = 1 TO 40
110   PRINT "hola"        110   PRINT "Hola"
120 END REPeat saludo     120 END FOR saludo
```

Ambos bucles funcionan. El bucle **REPeat** imprimirá "Hola" sin parar (puede pararlo con la secuencia de interrupción), y el segundo bucle imprimirá "Hola" sólo cuarenta veces.

Observe que el nombre del bucle **FOR** es también una variable, *saludo* cuyo valor varía desde 1 a 40 en el curso de la ejecución del programa. Esta variable también se llama **variable del bucle** o **variable de control**, del bucle.

Observe que la estructura de ambos bucles toma la forma:

```
Instrucción de apertura
    Contenido
Instrucción final
```

Sin embargo, ciertas estructuras tienen formas cortas adecuadas para su uso cuando sólo existe una, o unas pocas instrucciones como contenido del bucle. Como están permitidas las formas cortas, podremos escribir el programa de la forma más económica posible.

Programa 5

```
100 RESTORE 140 : CLS
110 DIM c_mayor$(4,3)
120 FOR número = 1 TO 4 : READ c_mayor$(número)
130 FOR número = 1 TO 4 : PRINT ! c_mayor$(número) !
140 DATA "PIO", "NOE", "GIL" y "ELO"
```

Los dos puntos sirven como símbolo final de instrucción, en lugar de **ENTER**, y los símbolos **ENTER** de las líneas 120 y 130 sirven como instrucciones **END FOR**.

Existe otra forma más corta aún de escribir el programa 5. Para imprimir el contenido de la matriz *c_mayor$* podemos sustituir la línea 130 por:

```
130 PRINT ! c_mayor$ !
```

Que utiliza un "delimitador" de matrices que discutiremos en el capítulo 13.

Hemos introducido el concepto de matriz de variables de cadena de forma que los únicos números que aparecen son los índices en el nombre de cada variable. Las matrices pueden ser de cadena o numéricas, y el ejemplo siguiente ilustra una matriz numérica.

Programa 1

Simule que tira unos dados (como las del juego de mesa Monopolio) una centena de veces. Anote el número de apariciones de cada apuntuación desde 2 hasta 12.

```
100 REMark DADO1
110 LET dos = 0:tres = 0:cuatro = 0:cinco = 0:seis = 0
120 LET siete = 0:ocho = 0:nueve = 0:diez = 0:once =0:doce = 0
130 FOR tirada = 1 TO 400
140   LET dado1 = RND(1 TO 6)
150   LET dado2 = RND(1 TO 6)
160   LET puntos = dado1 + dado2
170   IF puntos = 2 THEN LET dos = dos + 1
180   IF puntos = 3 THEN LET tres = tres + 1
190   IF puntos = 4 THEN LET cuatro = cuatro + 1
200   IF puntos = 5 THEN LET cinco = cinco + 1
210   IF puntos = 6 THEN LET seis = seis + 1
220   IF puntos = 7 THEN LET siete = siete + 1
230   IF puntos = 8 THEN LET ocho = ocho + 1
240   IF puntos = 9 THEN LET nueve = nueve + 1
250   IF puntos = 10 THEN LET diez = diez + 1
260   IF puntos = 11 THEN LET once = once + 1
270   IF puntos = 12 THE LET doce = doce + 1
280 END FOR tirada
290 PRINT ! dos ! tres ! cuatro ! cinco ! seis
300 PRINT ! siete ! ocho ! nueve ! diez ! once ! doce
```

En el programa anterior establecimos once variables simples para almacenar el conjunto de puntos. Si traza los lotes diferentes que aparecen impresos al final encontrará que el diagrama de barras es significativamente triangular. Los lotes de puntos más altos son para las puntuaciones seis, siete, ocho y los lotes más bajos (menores) son dos y doce. Como saben muy bien los jugadores de Monopolio, esto refleja muy bien la frecuencia de la gama media de puntuaciones (seis, siete, ocho) y la dificultad (rareza) en la obtención de doses y doces.

Programa 2

```
100 REMark Dado2
110 DIM lote(12)
120 FOR tirada = 1 TO 400
130   LET dado_1 = RND(1 TO 6)
140   LET dado_2 = RND(1 TO 6)
150   LET puntos = dado_1 + dado_2
160   LET lote(puntos) = lote(puntos) + 1
170 END FOR tirada
180 FOR número = 2 TO 12 : PRINT ¡lote(número)!
```

En el primer bucle **FOR**, es tirada, el subíndice de todas las variables es *puntos*. Esto significa que el subíndice correcto de la matriz se elige automáticamente aumentando el lote correspondiente después de cada tirada. Puede imaginar que la matriz lote es como un conjunto de casilleros numerados desde el 2 al 12. Cada vez que aparece una determinada puntuación, el lote que contiene esa punturación se incrementa, añadiendo una unidad en el casillero correspondiente.

En el segundo bucle **FOR** (forma corta), el subíndice es *número*. Como el valor de *número* cambia desde 2 hasta 12, se imprimen todos los valores de los lotes.

Observe que, en las instrucciones **DIM** para matrices numéricas sólo es necesario declarar el número de las variables que se van a utilizar. No importa cuál sea la longitud máxima, como ocurría en el caso de una matriz de cadena, ya que se trata de una matriz numérica.

Si usted ha utilizado otras versiones del lenguaje BASIC, puede preguntarse qué ha ocurrido con la instrucción **NEXT**. Todas las estructuras de SuperBASIC terminan con **END** y algo más. Todo esto es consistente, pero la instrucción **NEXT** tiene, como se verá, su contenido explicado en los próximos capítulos.

AUTOEXAMEN SOBRE EL CAPITULO 6

En este test puede obtenerse un máximo de 16 puntos. Averigue su puntuación comparando sus respuestas con las de la página siguiente.

1. Explique dos dificultades diferentes que aparecen cuando los datos necesarios para un programa se hacen muy numerosos, y usted intenta manejarlos sin utilizar matrices.
2. Si en una matriz, diez variables llevan el mismo nombre ¿Cómo puede usted distinguirlas?
3. ¿Qué acción se realiza normalmente en un programa antes de utilizar una variable matriz?
4. Dé otro nombre al número que distingue una varible determinada de una matriz de las demás variables que comparten el mismo nombre.
5. ¿Puede poner dos ejemplos de la vida real que se correspondan con el concepto de matriz en programación?
6. En un bucle de repetición **REPeat**, el proceso termina cuando alguna condición causa la ejecución de una instrucción **EXIT**. ¿Qué es lo que causa la terminación de un bucle **FOR**? (Dos respuestas.)
7. Un blucle **REPeat** necesita un nombre para poder salir de él adecuadamente **(EXIT)** cuando lleguemos a su final **(END)**. Los bucles **FOR** tambien llevan nombre, pero especifique otra función que lleve el nombre del bucle **FOR**.
8. Con qué dos frases se describe la variable que es además el nombre del bucle **FOR**? (Dos respuestas.)
9. Los valores de una variable de un bucle cambian automáticamente mientras se ejecuta el bucle **FOR**. Expecifique un posible e importante uso de esas variables.
10. ¿Qué tienen en común los siguientes elementos que configuran la forma larga de los bucles **REPeat** y la forma larga de los bucles **FOR**?

    a) Una palabra clave o instrucción de apertura.
    b) Una palabra clave o instrucción de cierre.
    c) Un nombre de un bucle.
    d) Una variable de un bucle o variable de control (cuatro puntos).

PROBLEMAS SOBRE EL CAPITULO 6

1. Utilice un bucle **FOR** para colocar uno de los cuatro números 1, 2, 3, 4, de forma aleatoria en cinco variables de matriz:

   carta(1), carta(2), carta(3), carta(4), carta(5).

   No importa que se repitan algunos de los números. Utilice un segundo bucle **FOR** para dar salida a los valores de las cinco variables "carta".

2. Imagine que los cuatro números representan "Corazones", "Diamantes", "Tréboles" y "Picas". ¿Qué líneas debe insertar al programa para que la salida se obtenga con esas palabras en lugar de números?

3. Utilice un bucle **FOR** para colocar cinco números aleatorios de la gama del 1 al 13 en una matriz de cinco variables.

   carta(1), carta(2), carta(3), carta(4), carta(5).

   Utilice un segundo bucle **FOR** para dar salida a los valores de las cinco variables "carta".

4. Imagine que los números aleatorios generados en el problema 1 representan cartas. Escriba las instrucciones adicionales que causarán las siguientes salidas:

Número	Salida
1	La palabra "As"
Del 2 al 10	El número que aparezca en ese momento
11	La palabra "Jack"
12	La palabra "Reina"
13	La palabra "Rey"

CAPITULO 7
PROCEDIMIENTOS SENCILLOS

Si intentara escribir programas de ordenador para resolver problemas complejos, podría encontrar dificultades en manejar y considerar todos los elementos. Una persona metódica resolvería los problemas dividiendo el trabajo enorme y complejo en secciones, o **tareas,** y más tarde dividiendo esas tareas en otras menores, y así continuando hasta un punto en el que se puedan resolver con facilidad.

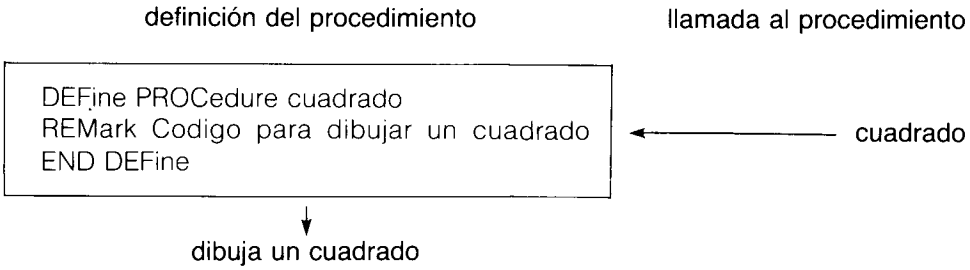
El problema es similar a los problemas complejos humanos. Un gobierno exitoso depende de una delegación responsable. El primer ministro divide el trabajo entre sus ministros, los cuales vuelven a dividirlo entre sus directores generales, y así hasta que las tareas puedan realizarse por personas individuales, sin otra división. Existen además estructuras complicadas, como por ejemplo servicios comunes e interacciones entre niveles diferentes, pero la estructura jerárquica es la que prevalece.

Un buen programador funciona del mismo modo, y un lenguaje de programación moderno, como el SuperBASIC que permite procedimientos adecuadamente nombrados y bien definidos será de mucha más ayuda que otras versiones más antiguas que no tienen estas características.

La idea principal es escribir un determinado bloque de código separado y con un nombre para que realice una tarea determinada. No importa el lugar que ocupa el bloque de código en el programa. Si se encuentra en cualquier lugar, utilizando su nombre, podemos:

activar el código
devolver el control al punto adecuado del programa inmediatamente después de su uso.

Si el procedimiento *cuadrado* dibuja un cuadrado, el esquema puede ser el siguiente:



En la práctica, pueden separarse las tareas de un determinado trabajo y se pueden identificar y nombrar antes de escribir el código de definición. Sólo se necesita el nombre para llamar al procedimiento de forma que se pueden definir los rasgos generales del programa antes de definir las diferentes tareas.

También se pueden escribir primero las diferentes tareas y probarse. Si funcionan, pueden olvidarse los detalles de esa determinada tarea y recordar únicamente la acción que realiza ese determinado procedimiento.

Ejemplo

El ejemplo siguiente puede escribirse muy fácilmente sin procedimientos, pero nos muestra como éstos se pueden utilizar en un contexto razonablemente sencillo. Casi cualquier tarea puede dividirse de forma similar, lo que significa que nunca necesitará preocuparse en un determinado momento por más de cinco a treinta líneas. Si usted es capaz de escribir programas de treinta líneas que funcionen adecuadamente, y que manejan procedimientos, es capaz de escribir programas de trescientas líneas.

Podemos producir frases hechas para políticos u otras personas que quieren dar una impresión de fluidez tecnológica sin saber nada de un determinado tema. Almacene las siguientes palabras en tres matrices y luego produzca diez frase altisonantes aleatoriamente.

adj1$	adj2$	nomb$
total	quinta-generación	sistema
sistemático	basado-en-un-conocimiento	maquinaria
inteligente	compatible	ordenador
controlado	cibernética	alimentado
automatizado	de-fácil-uso	transputer
sincronizado	paralelo	microchip
funcional	didáctico	capacidad
opcional	adaptable	programación
positivo	modular	paquete
equilibrado	estructurado	base de datos
integrado	orientado-lógicamente	hoja-de-trabajo
coordinado	orientado-en-	procesador-de-texto
sofisticado	estandarizado	objetivo

ANALISIS

Escribiremos un programa que produzca diez frases altisonantes. Las diferentes etapas del programa son las siguientes:

1. Almacenamiento de las palabras en tres matrices de cadena.
2. Elección de tres números aleatorios que serán los subíndices de las variables de la matriz..
3. Impresión de la frase.
4. Repetición de 2 y 3 diez veces.

DESIGNACION DE LAS VARIABLES

Identificaremos tres matrices, de las cuales dos contendrán adjetivos describiendo palabras. La tercera matriz contendrá los nombres. Cada sección contiene trece palabras y la palabra más larga contiene 25 caracteres incluyendo los guiones.

Matriz	Objetivo
adj1$(13,12)	primeros adjetivos
adj2$(13, 25)	segundos adjetivos
nom$(13, 19)	nombres

PROCEDIMIENTOS

Utilizaremos tres procedimientos para realizar las diferentes tareas ya identificadas.

almc_datos almacena los tres conjuntos de trece palabras
n_aleat obtiene los tres números aleatorios en una gama que va del 1 al 13
crea_fras imprime una frase

PROGRAMA PRINCIPAL

Resulta muy sencillo ya que el trabajo principal lo hacen los procedimientos:

1. Declaración **(DIM)** de las matrices

2. Almac_datos
3. FOR diez frases
   n_aleat
   crea_fras
   END

Programa

```
100 REMark ****************
110 REMark * Altisonantes *
120 REMark ****************
130 DIM adj1$(13,12), adj2(13, 25), nom$(13, 19)
140 almc_datos
150 FOR frase = 1 TO 10
160   n_aleat
170   crea_fras
180 END FOR frase
190 REMark ***********************************
200 REMark * Definiciones del procedimiento *
210 REMark ***********************************
220 DEFine PROCedure alm_datos
230   REMark *** procedimiento para almacenar los datos altiso-
      nantes ***
240   RESTORE 420
250   FOR elemento = 1 TO 13
260   READ adj1$(elemento), adj2$(elemento), nombre$(elemento)
270   END FOR elemento
280 END DEFine
290 DEFine PROCedure n_aleat
300   REMark *** procedimiento para seleccionar la frase ***
310   LET 1 = RND(1 TO 13)
320   LET 2 = RND(1 TO 13)
330   LET n = RND(1 TO 13)
340 END DEFine
350 DEFine PROCedure crea_fras
360   REMark *** procedimiento para imprimir la frase ***
370   PRINT ! adjetivo_1$ (adj1) ! adjetivo_2$ (adj2) ! nombre$ (n)
380   END DEFine
390 REMark **********
400 REMark *Datos del Programa*
410 REMark **********
420 DATA total, quinta-generación, sistema
430 DATA sistemático, basado-en-44-conocimientos, maquinaria
440 DATA inteligente, compatible, ordenador
450 DATA controlado, cibernético, alimentado
460 DATA automatizado, de-facil-uso, portaordenador
470 DATA sincronizado, paralelo, micro-chips
480 DATA funcional, didáctico, capacidad
490 DATA opcional, adaptable, programación
500 DATA positivo, modular, paquete
510 DATA equilibrado, estructurado, base-de-datos
520 DATA integrado, orientado-lógicamente, hoja-de-trabajo
530 DATA coordinado, archivado, procesador-de-texto
540 DATA sofisticado, standarizado, objetivo
```

SALIDA MUESTRA

```
capacidad automática quinta generación
paquete didáctico funcional
objetivo total paralelo
hoja-de-trabajo positiva práctica
capacidad inteligente archivada
transputer cibernético sincronizado
micro-chips funcionales orientado-lógicamente
realimentación paralela positiva
base de datos equilibrada didáctica
objetivo cibernético controlado
```

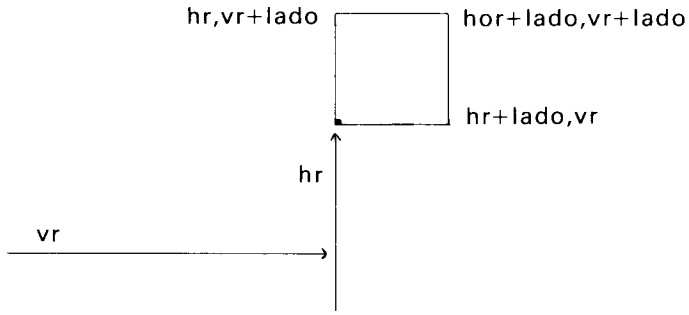
TRANSFERENCIA DE INFORMACION A LOS PROCEDIMIENTOS

Suponga que deseamos dibujar en la pantalla de gráficos a escala cuadrados de varios tamaños, en colores y posiciones también variados.

Si definimos un procedimiento, *cuadrado,* que realice esta acción necesitaremos cuatro elementos de información:

longitud de un lado
color (código del color)
posición (en las dos direcciones)

La posición de los cuadrados viene determinada por dos valores, en horizontal y en vertical. Estos valores establecen el ángulo inferior izquierdo del cuadrado, tal y como puede verse más abajo.



El color del cuadrado se establece con facilidad, pero el cuadrado en sí mismo utiliza los valores *lado* y *hr* y *vr,* tal y como sigue:

```
200 DEFine PROCedure cuadrado(lado,hr,vr)
210   LINE hr,vr TO hr+lado,vr
220   LINE TO hr+lado,vr+lado
230   LINE TO hr,vr+lado TO hr,vr
240 END DEFine
```

Para que este procedimiento funcione, deben darse los valores de lado, *hr* y *vr.* Estos valores se suministran cuando se llama al procedimiento. Por ejemplo, puede obtenerse un cuadrado grande y verde de 20 de lado añadiendo el programa principal que sigue

```
100 PAPER 7 : CLS
110 INK 4
120 cuadrado 20,50,50
```

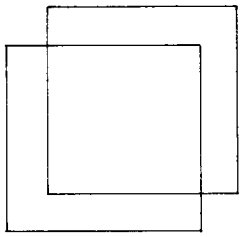
Los números 20, 50, 50 se llaman parámetros y pasan a las variables nombradas en la definición del procedimiento. Por tanto:

```
cuadrado 20,50,50

DEFine PROCedure cuadrado(lado,hr,vr)
```

Los números 20,50,50 se llaman parámetros. En este caso son números, pero pueden ser también variables o expresiones. Las variables *lado, hr, vr,* se llaman parámetros formales. Son variables porque "reciben" valores.

Un programa principal más interesante utiliza el mismo procedimiento para crear un patrón aleatorio de paredes de cuadrados coloreados. Cada par de cuadrados se obtiene desplazando el segundo un quinto de lado tanto en horizontal como en vertical, del siguiente modo:



Suponiendo que el procedimiento *cuadrado* permanezca en la línea 200, el programa siguiente obtendrá el efecto esperado.

```
100 REMark Patrón de Cuadrados
110 PAPER 7 : CLS
120 FOR par = 1 TO 20
130   INK RND(5)
140   LET lado = RND(10 TO 20)
150   LET hr = RND(50) : vr = RND(70)
160   cuadrado lado,hr,vr
170   LET hr=hr+lado/5 : vr=vr+lado/5
180   cuadrado lado,hr,vr
190 END FOR par
```

Enumeraremos pues las ventajas de los procedimientos.

1. Puede utilizarse el mismo código más de una vez en el mismo programa o en otros programas.
2. Una determinada tarea puede partirse en sub-tareas y podemos, pues, escribir procedimientos para cada sub-tarea. Esto resulta muy útil en el análisis y diseño del programa.
3. Los procedimientos se pueden probar por separado, lo que es útil en las pruebas y en el desarrollo de programas.
4. Los programas son comprensibles si se dan nombres con sentido y se definen claramente los comienzos y finales de procedimiento.

Cuando se haya acostumbrado a utilizar procedimientos adecuadamente nombrados, con buenos parámetros, encontrará que sus posibilidades para resolver problemas y como programador habrán aumentado considerablemente.

AUTOEXAMEN SOBRE EL CAPITULO 7

En el test siguiente puede obtenerse un máximo de 14 puntos. Averigue su puntuación comprobando sus respuestas con las de la página siguiente.

1. ¿Cómo se afronta el problema de un tamaño enorme y una gran complejidad en los problemas normales, humanos?
2. ¿Cómo se aplica este principio en programación?
3. ¿Cuáles son las características más evidentes de la definición de un procedimiento? (dos puntos).
4. ¿Cuáles son los dos efectos principales que se producen al utilizar un nombre de procedimiento para "llamarlo"? (dos puntos).
5. ¿Cuál es la ventaja de utilizar nombres de procedimiento en un programa principal antes de escribir las definiciones del procedimiento?
6. ¿Cuál es la ventaja de escribir la definición de un procedimiento antes de utilizar su nombre en el programa principal?
7. ¿Por qué puede considerarse que los procedimientos ayudan a los "programadores de treinta líneas" a escribir programas mucho mayores?
8. Algunos programas usan más memoria en la definición de los procedimientos. Diga en qué circunstancias los procedimientos ahorran espacio de memoria.
9. Nombre dos formas diferentes por las que puede transferirse información desde el programa principal al procedimiento.
10. ¿Qué son los parámetros actuales?
11. ¿Qué son los parámetros formales?

PROBLEMAS SOBRE EL CAPITULO 7

1. Escriba un procedimiento que dé como salida uno de los cuatro palos: "Corazones", "Picas", "Diamantes" o "Tréboles". Llame al procedimiento cinco veces para obtener cinco palos aleatorios.
2. Escriba el programa del problema 1 utilizando un número de la gama del 1 al 4 como parámetro para determinar la palabra que se obtendrá como salida. Si ya lo ha hecho anteriormente, trate de escribir el programa sin parámetros.
3. Escriba un procedimiento que dé como salida el valor de una carta, es decir un número de la gama del 2 al 10, o una de las palabras siguientes: "As", "Jack", "Reina" o "Rey".
4. Escriba un programa que llame a este procedimiento cinco veces, de forma que se obtengan en la salida cinco valores aleatorios.
5. Escriba el programa del problema 3 de nuevo, utilizando un número de la gama del 1 al 13 como parámetro que será transferido al procedimiento. Si éste fue el método que usted utilizó la primera vez, pruebe a escribir el programa sin parámetros.
6. Escriba el programa más elegante que pueda, utilizando procedimientos, de forma que cada uno dé en la salida cuatro manos de cinco cartas cada una. No se preocupe si las cartas aparecen duplicadas. En nuestro caso, la palabra elegancia significa una mezcla de eficacia, brevedad y facilidad de comprensión. Personas diferentes y circunstancias diferentes darán mayor importancia a alguna de estas tres cualidades, que en ciertos casos son contradictorias.

CAPITULO 8
DEL LENGUAJE BASIC AL LENGUAJE SUPERBASIC

Si conoce bien alguna de las versiones anteriores del lenguaje BASIC, puede considerar la posibilidad de omitir el estudio de los primeros siete capítulos y utilizar este capítulo como puente entre los conocimientos que ya tenía y los capítulos siguientes. Si hace esto y encuentra algunas dificultades puede serle útil volver sobre los capítulos anteriores.

Si ya ha estudiado y trabajado sobre los capítulos anteriores, éste le resultará de fácil lectura. Además de presentarle nuevas ideas, podrá ver que le da una visión rápida sobre la forma en que se va desarrollando el lenguaje BASIC. Además de las ventajas en la estructuración de los programas, el lenguaje Super BASIC amplía fronteras en la obtención de mejores presentaciones de pantalla (ediciones), más facilidad de operación y mejores gráficos. En pocas palabras, es una combinación de facilidad de uso y potencia de proceso que no ha existido hasta el momento.

Por tanto, cuando usted realiza la transición del BASIC al SuperBASIC no sólo está pasando a un lenguaje más potente y más útil, sino que además se mueve en un entorno informático notablemente más avanzado.

Discutiremos ahora las características principales del SuperBASIC y las que lo distinguen de otros lenguajes BASIC.

COMPARACIONES ALFABETICAS

Vamos a establecer simples comparaciones aritméticas. Puede escribir por ejemplo:

```
LET ani1$ = "GATO"
LET ani2$ = "PERRO"
IF ani1$ < ani2$ THEN PRINT "Miau"
```

La salida será Miau ya que en este contexto el símbolo < significa:

anterior (en el alfabeto más próxima a la A)

El lenguaje SuperBASIC realiza comparaciones cuidadosas. Por ejemplo, usted espera que:

"gato" esté antes que "PERRO"

y que:

'ERD98L' sea anterior a 'ERD746L'

Una aproximación sencilla, que utilice a ciegas los códigos de caracteres internos dará un resultado equivocado en los dos casos anteriores. Pruebe sin embargo el siguiente programa que busca la anterior de entre las dos cadenas de caracteres.

```
100 INPUT elem1$, elem2$
110 IF elem1$ < elem2$ THEN PRINT elem1$
120 IF elem1$ = elem2$ THEN PRINT "Igual"
130 IF elem1$ > elem2$ THEN PRINT elem2$
```

	ENTRADA	SALIDA
gato	perro	gato
gato	PERRO	gato
ERD98L	ERD746L	ERD98L
ABC	abc	ABC

La sección *Guía de Referencia de Conceptos* le dará detalles completos sobre el sistema de realizar las comparaciones de cadenas en SuperBASIC.

NOMBRES DE VARIABLES E IDENTIFICADORES

Muchos de los lenguajes BASICs tienen variables numéricas y de cadena. Como en los demás lenguajes BASIC, la característica que distingue el nombre de una variable de cadena es el signo de dólar colocado al final, así pues:

numérica:	cuenta	de cadena:	*palabra$*
	suma		*c_mayor$*
	total		*día_de_fiesta$*

Puede que no se haya encontrado todavía estos nombres de variables tan expresivos aunque algunos lenguajes BASIC recientes los aceptan. Las reglas para los identificadores de SuperBASIC vienen descritas en la *Guía de Referencia de Conceptos*. La longitud máxima de un identificador es 255 caracteres. La elección de estos identificadores debe hacerla el usuario. Algunas veces los identificadores largos son más útiles, ya que le explican al lector lo que realiza el programa. Pero en cualquier caso deben teclearse y, como bien dice el refrán, "lo bueno si breve dos veces bueno".

Las palabras cortas son más adecuadas si expresan bien el significado, pero debe procurarse utilizar poco las palabras muy cortas o las letras únicas. Por ejemplo, nombre como *X,Z,P3,Q2* introducen un nivel de abstracción que ayuda muy poco a la mayoría de las personas.

VARIABLES ENTERAS

El lenguaje SuperBASIC permite variables **enteras,** que toman sólo como valores números enteros. Este tipo de variables se distinguen porque llevan un signo de tanto por ciento, esto es:

```
cuenta%
número%
aprox_pesetas%
```

Por tanto, ahora ya tenemos dos tipos de variables numéricas. Al otro tipo, que puede tomar valores enteros o fraccionarios le llamaremos **coma flotante.** Por tanto podemos escribir:

```
LET precio = 9
LET coste = 7.31
LET cuenta% = 13
```

Pero si escribimos:

```
LET cuenta% = 5.43
```

El valor de *cuenta%* será 5. Por otro lado:

```
LET cuenta% = 5.73
```

dará como valor de *cuenta%* 6. Como puede ver, el lenguaje SuperBASIC hace todo lo que puede redondeando el número entero más próximo.

COERCION (CAMBIO FORZADO DE TIPO DE VARIABLE)

El principio que se sigue es intentar ser útil inteligentemente y no por el contrario enviar un mensaje de error o realizar algo que evidentemente no es deseado. Es decir, si una variable de cadena, por ejemplo *marca$* toma el valor:

```
"64"
```

y

```
LET puntos = marca$
```

producirá un valor numérico para la variable puntos de 64. Otro tipo de versiones del BASIC pararían el programa y enviarían el siguiente mensaje:

'Error de tipo'
o 'Sin sentido en BASIC'

Si no se puede convertir la cadena, se enviará el correspondiente mensaje de error.

VARIABLES LOGICAS Y PROCEDIMIENTOS SENCILLOS

Existe otro tipo de variables en SuperBASIC, o mejor dicho, el sistema del SuperBASIC hace que aparezca así. Considere, por ejemplo, la instrucción de SuperBASIC:

```
IF ventoso THEN vuele_cometa
```

En otros BASICs diferentes, debería escribir:

```
IF v=1 THEN GO SUB 300
```

En este caso, v+1 es una condición o expresión lógica que puede ser verdadera o falsa. Si es verdadera, se ejecutará una subrutina que comienza en la línea 300. Esa subrutina puede trabajar con el vuelo de cometa, pero usted no puede decir lo que ocurrirá desde la línea de encima. Un programador cuidadoso escribiría:

```
IF v=1 THEN GOSUB 300 : REM vuele_cometa
```

para hacerlo más razonable. Sin embargo, la instrucción en SuperBASIC puede leerse tal y como está. El identificador *ventoso* se interpreta como falso o verdadero, ya que en ese momento es una variable de coma flotante. Un valor 1 o cualquier otro valor diferente de cero se toma como *verdadero.* El valor cero se toma como falso. De este modo, una única palabra, *ventoso,* tiene el mismo efecto que una expresión lógica.

La otra palabra, *vuele_cometa* es un procedimiento. Realiza la misma acción, pero incluso mucho mejor que GOSUB 300.

El programa siguiente unirá la idea de variables lógicas en tipos de procedimientos con nombre, muy sencillos.

```
100 INPUT ventoso
110 IF ventoso THEN vuela_cometa
120 IF NOT ventoso THEN "Ve a pasear"
130 DEFine PROCedure vuela_cometa
140 PRINT "Mírala en el aire"
150 END DEFine
160 DEFine PROCedure "Ve a pasar".
```

```
170 PRINT "Disfrute de la naturaleza"
180 END DEFine
```

ENTRADA	SALIDA
0	"Disfruta de la naturaleza"
1	"Mirala en el aire"
2	"Mírala en el aire"
-2	"Mírala en el aire"

Puede usted ver que sólo el cero toma el significado de falso. Normalmente los procedimientos no se escriben sólo con una instrucción, pero el programa ilustra la idea en un contexto muy sencillo. Más adelante en este capítulo daremos más información sobre los procedimientos.

LAS INSTRUCCIONES LET

En el lenguaje SuperBASIC, la instrucción **LET** es opcional pero la utilizaremos en este manual para que existan menos posibilidades de confusión en los dos posibles usos del signo =. El significado de = en una instrucción:

```
LET cuenta = 3
```

y en

```
IF cuenta = 3 THEN EXIT
```

es diferente y la instrucción **LET** ayuda a comprenderlo. Sin embargo, podemos hacer una excepción en el caso de que existan dos o instrucciones **LET** (o pocas) que realicen tareas sencillas, como asignar valores iniciales a las variables.

Por ejemplo:

```
100 LET primero = 0
110 LET segundo = 0
120 LET tercero = 0
```

puede escribirse de nuevo como:

```
100 LET primero = 0 : segundo = 0 : tercero = 0
```

sin que se pierda en absoluto claridad o estilo. También se cumple la norma general de permitir formas cortas de otras construcciones en ciertos casos en los que se utiliza de forma muy simple.

Los dos puntos : constituyen un finalizador de la instrucción y se pueden utilizar con otro tipo de instrucciones además de la instrucción **LET**.

LA PANTALLA BASICA

En un capítulo posterior explicaremos cómo pueden manejarse con facilidad otras características gráficas como, por ejemplo, el dibujo de círculos, pero aquí vamos a ocuparnos de las características relacionadas con los pixels. Existen dos modos que pueden activarse mediante cualquiera de los siguientes sistemas:

Baja resolución Modo de 8 colores 256 pixels en horizontal, 256 en vertical	MODO 256 MODO 8

Alta resolución Modo de 4 colores 512 pixels en horizontal, 256 en vertical	MODO 512 MODO 4

En ambos modos, los pixels se posicionan en una gama de números:

0-511 en horizontal
y 0-255 en vertical

Como el modo 8 sólo tiene la mitad de pixels en el sentido horizontal de la pantalla, que el modo 1, los pixels en modo 0 son dobles de ancho que los del modo 1 y, por tanto, los pixels en modo 0 se pueden especificar por dos coordenadas. Por ejemplo:

0 ó 1 2 ó 3 510 ó 511

Esto significa que se utiliza la misma gama de valores para designar los pixels sin tener en cuenta el modo. No lo olvide, 511 pixels en horizontal y 255 en vertical.

Si se utiliza una televisión no todos los pixels serán visibles.

COLORES

Los colores de que se dispone son los siguientes:

MODO 256	Código	MODO 512
negro	0	negro
azul	1	
rojo	2	rojo
morado	3	
verde	4	verde
ciano	5	
amarillo	6	blanco
blanco	7	

CODIGOS DE COLOR

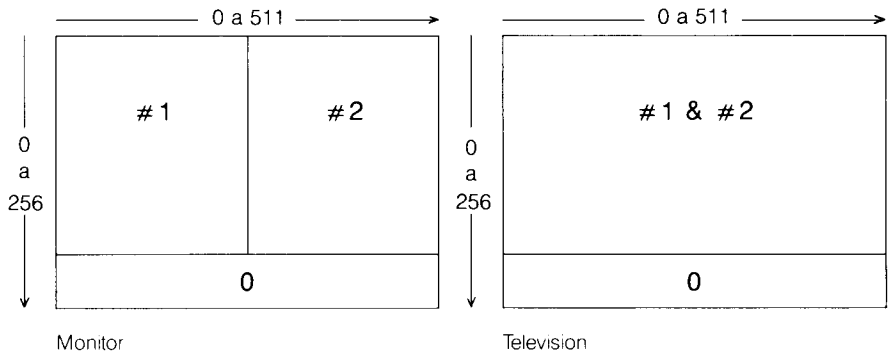
En el modo de *alta resolución,* cada color puede seleccionarse por uno o dos códigos. Más adelante observará la enorme gama de colores y los efectos que se pueden producir si su monitor da buena calidad a los colores.

A continuación se dan algunas de las palabras clave para las presentaciones de pantalla:

INK *color*	color del texto
BORDER *anchura, color*	dibuja un reborde en la pantalla o en la ventana
PAPER *color*	color del fondo
BLOCK, *horz., vert., ancho, alto, color*	colorea un rectángulo cuyo ángulo izquierdo superior esté en la posición dada por las coordenadas horizontal y vertical

ORGANIZACION DE LA PANTALLA DEL QL

Cuando usted conecte su QL, encontrará que la pantalla está dividida en tres zonas llamadas "ventanas", tal y como puede verse en la figura:



Las ventanas se identifican por #0, #1 y #2 de forma que se puedan relacionar ciertos efectos con ventanas especiales, por ejemplo:

```
CLS
```

limpiará la ventana #1 (el sistema elige), por tanto, si desea que se limpie el área de la izquierda, debe teclear:

```
CLS #2
```

Si desea un color de fondo-PAPER diferente, teclee por ejemplo para el verde:

```
PAPER 4 : CLS
```

o

```
PAPER #2,4 : CLS #2
```

si desea limpiar con el color verde la ventana #2.

Los números #0, #1, #2, se llaman *números de canal.* En este caso en particular, le permitirán dirigir ciertos efectos a la ventana de su elección. Más adelante descubrirá que los números de canal pueden utilizarse con otros propósitos, pero por el momento observe que todas las instrucciones que siguen pueden llevar un número de canal. La tercera columna muestra el canal por omisión (que es el que elige el sistema cuando usted no especifica ningún canal).

Observe que las ventanas pueden unirse. Si usted utiliza un aparato de televisión, el sistema une automáticamente las ventanas #1 y #2 para disponer así de más posiciones de carácter por línea para los listados de los programas.

Palabra clave	Efecto	Omisión
AT	Posiciona un carácter	#1
BLOCK	Dibuja un bloque	#1
BORDER	Dibuja un reborde	#1
CLS	Limpia la pantalla	#1
CSIZE	Da el tamaño de un carácter	#1
CURSOR	Posiciona el cursor	#1
FLASH	Causa/cancela intermitencia	#1
INK	Especifica el color del texto	#1
OVER	Tiene efecto en impresión y gráficos	#1
PAN	Mueve la pantalla hacia los lados	#1
PAPER	Especifica el color del fondo	#1
RECOL	Cambia el color	#1
SCROLL	Mueve verticalmente la pantalla	#1
STRIP	Patrón de Fondo para la impresión	#1
UNDER	Subraya	#1
WINDOW	Cambia la ventana existente	#1
LIST	Lista el programa	#2
DIR	Lista el directorio	#1
PRINT	Imprime caracteres	#1
INPUT	Acepta la entrada del teclado	#1

Las instrucciones o comandos directos aparecen en la ventana #0.

Para más detalles sobre la sintaxis o la utilización de las anteriores palabras clave, véanse otras partes del manual.

RECTANGULOS Y LINEAS

El programa de más abajo dibuja un rectángulo verde en el modo 256 sobre un fondo rojo y con un borde amarillo de una anchura de un pixel. El rectángulo tiene su ángulo superior izquierdo en el centro de la pantalla (256,128). Su anchura es de 140 unidades en horizontal (70 pixels) y su altura es de 100 unidades en vertical (100 pixels).

```
100 REMark Rectángulo
110 MODE 256
120 BORDER 1,6
130 PAPER 4 : CLS
140 BLOCK 256, 128, 140, 100,2
```

Debe usted poner el máximo cuidado en el modo 256 porque los valores en el sentido horizontal están en una gama que va desde 0 al 511 aunque sólo existen 256 pixels. Como no podemos decir que el bloque producido por el programa anterior es de 140 pixels de anchura, diremos que tiene una anchura de 140 unidades.

ENTRADA Y SALIDA

El lenguaje SuperBASIC tiene como entradas las instrucciones usuales **LET, INPUT, READ** y **DATA.** La instrucción **PRINT** maneja la mayoría de las salidas de texto de la forma usual, utilizando separadores:

, Tabula la salida

; Sólo separa - no tiene efecto en el formato

\ Fuerza una nueva línea

! Espacio inteligente. Normalmente coloca un espacio pero no al comienzo de la línea. Si algún elemento no cabe al final de una línea crea una nueva línea

TO Permite tabular dirigiéndose a una columna especificada

BUCLES

Se acostumbrará a dos tipos de bucle repetitivo, cuyos ejemplos se dan a continuación:

a) Simula 6 tiradas de un dado normal de seis lados

```
100 FOR tirada = 1 TO 6
110 PRINT RND (1 TO 6)
120 NEXT tirada
```

b) Simula las tiradas de un dado hasta que aparezca el seis.

```
100 dado = RND (1 TO 6)
110 PRINT dado
120 IF dado < > 6 THEN GOTO 10
```

Estos dos programas funcionan en SuperBASIC, pero recomendamos los siguientes. Todos realizan la misma acción, y aunque el programa b) es algo más complejo existen buenas razones para elegirlo.

a)
```
100 FOR tirada = 1 TO 6
110   PRINT RND(1 TO 6)
120 END FOR tirada
```

b)
```
100 REPeat tirada
110   dado = RND(1 TO 6)
120   PRINT dado
130   IF dado = 6 THEN EXIT tirada
140 END REPeat tirada
```

Es lógico que creemos una estructura para el bucle que termine en una condición (**REPeat** bucle), exactamente como ocurría con el que se controlaba por una cuenta.

La estructura fundamental de la instrucción **REPeat** es:

REPeat *identificador*
  instrucciones
END REPeat *identificador*

La instrucción **EXIT** puede colocarse en cualquier lugar de la estructura pero debe ir seguida de un identificador para informar al SuperBASIC de qué bucle debe salir, por ejemplo:

EXIT tirada

transferirá el control a la instrucción siguiente a la

END REPeat tirada

Esto puede parecernos semejante a matar moscas a cañonazos en un problema tan sencillo como el ilustrado. Sin embargo la estructura **REPeat** es muy potente. Puede llevarle muy lejos. Si conoce otros lenguajes podrá ver que realiza la misma acción que las dos estructuras, **REPEAT** y **WHILE** y también maneja otras situaciones más complicadas.

El bucle **REPeat** de SuperBASIC lleva un nombre para poder realizar una salida clara y correcta. El bucle **FOR**, como todas las estructuras de SuperBASIC termina con **END**, y lleva nombre por razones que se explicarán más adelante.

Más adelante también, verá como estas estructuras pueden utilizarse en situaciones sencillas o complejas y se adecúan exactamente a lo que usted necesita. Por el momento, mencionaremos sólo tres características más de los bucles. Le serán familiares si es usted un usuario de BASIC con experiencia.

El incremento de la variable de control de un bucle **FOR** es normalmente 1, pero puede cambiarse a otro valor utilizando la palabra clave **STEP**. Véalo en el siguiente ejemplo:

i.
```
100 FOR par = 2 TO 10 STEP 2
200 PRINT ! par !
300 END FOR par
```

la salida es **2 4 6 8 10**

ii.
```
100 FOR atrás = 9 TO 1 STEP -1
110   PRINT ! atras !
120 END FOR atras
```

la salida es **9 8 7 6 5 4 3 2 1**

La segunda característica es que los bucles se pueden anidar. Puede resultarle familiar el que los bucles **FOR** se aniden. Como ejemplo, el siguiente programa da como salida cuatro filas de diez cruces.

```
100 REMark Cruces
110 FOR fila = 1 TO 4
120   PRINT "número de fila" fila
130   FOR cruz = 1 TO 10
140     PRINT ! 'X' !
150   END FOR cruz
160   PRINT !
170 PRINT \ "fin del número de fila" ! fila
180 END FOR fila
```

la salida es

```
Fila número 1
X X X X X X X X X X
Fin de fila número 1
Fila número 2
X X X X X X X X X X
Fin de fila número 2
Fila número 3
X X X X X X X X X X
Fin de fila número 3
Fila número 4
X X X X X X X X X X
Fin de fila número 4
```

Una enorme ventaja del SuperBASIC es que tiene estructuras para todas las necesidades, y no sólo bucles **FOR**. Estas estructuras pueden anidarse una dentro de la otra, reflejando todas las necesidades de una determinada tarea. Se pueden colocar bucles **REPeat** en el interior de un bucle **FOR**. El programa listado más abajo produce en cada fila en lugar de cruces, la puntuación que dan dos dados, hasta encontrar el siete.

```
100 REMark Tirada de dados
110 FOR fila = 1 TO 4
120   PRINT "Número de fila"! fila
130   REPeat tirada
140     LET dado1 = RND (1 TO 6)
140     LET dado2 = RND (1 TO 6)
160     LET puntos = dado1 + dado 2
170     PRINT ! puntos !
180     IF puntos = 7 THEN EXIT tirada
190   END REPeat tirada
200   PRINT \ "Fin de fila" ! fila
210 END FOR fila
```

Muestra de salidas:

```
Fila número 1
8 11 6 3 7
Fin de fila número 1
Fila número 2
4 6 2 9 4 5 12 7
Fin de fila número 2
Fila número 3
7
Fin de fila número 3
Fila número 4
6 2 4 9 9 7
Fin de fila número 4
```

La tercera característica de los bucles en SuperBASIC es la gran flexibilidad que dan a la gama de valores en los bucles **FOR**. El programa siguiente ilustra esta característica imprimiendo todos los números no primos del 1 al 20.

```
100 REMark Números divisibles
110 FOR núm = 4,6, 8 TO 10, 12, 14 TO 16, 18, 20
120     PRINT ! núm!
130 END FOR núm
```

En un capítulo posterior daremos más información sobre el manejo de las estructuras de repetición, pero con las estructuras que hemos descrito aquí se pueden solucionar casi todas las situaciones excepto alguna poco común o muy compleja.

TOMA DE DECISIONES

Se habrá fijado en un tipo de decisión sencillo:

```
IF dado = 6 THEN EXIT tirada.
```

Esto es válido en muchos lenguajes BASICS, pero el lenguaje SuperBASIC ofrece extensiones a esta estructura, y también otra estructura completamente nueva que maneja situaciones con más de dos posibilidades alternativas.

Sin embargo, las formas largas de la instrucción **IF...THEN**, le pueden resultar útiles. Este tipo de estructuras se explican por sí mismas.

i.
```
100 REMark Forma larga IF ...END IF
110 LET soleado = RND(0 TO 1)
120 IF soleado THEn
130   PRINT "Ponte gafas de sol"
140   PRINT "y sal a pasear"
150 END IF
```

ii.
```
100 REMark Forma larga IF...ELSE...END IF
110 LET soleado = RND(0 TO 1)
120 IF soleado THEN
130   PRINT "Ponte gafas de sol"
140   PRINT "Sal a pasear"
150 ELSE
160   PRINT "Ponte abrigo"
170   PRINT "Ve al cine"
180 END IF
```

El separador **THEN** es opcional en las formas largas o bien puede sustituirse por dos puntos en las formas cortas. Las estructuras de decisión largas deben cumplir las mismas reglas que los bucles. Pueden anidarse o bien se pueden colocar otras estructuras dentro de ellas. Si aparece una variable única en lugar de la condición que usted esperaba, se tomará como falso el valor cero y los otros valores se tomarán como verdaderos.

SUBRUTINAS Y PROCEDIMIENTOS

La mayoría de los lenguajes BASIC incluyen la instrucción **GOSUB** que se utiliza para activar bloques especiales de código, llamados subrutinas. La instrucción **GOSUB** en muchos casos no es satisfactoria y el SuperBASIC ofrece procedimientos con nombre adecuado, algunos de los cuales tienen características muy útiles.

Estudie los programas siguientes, que dibujan (ambos) un "cuadrado" verde con una longitud de 50 pixels por lado en una posición 200 en horizontal y 100 en vertical y sobre un fondo rojo.

a) Utilizando **GOSUB**

```
10 LET color = 4 : fondo = 2
20 LET horiz = 20
30 LET vertic = 100
40 LET lado = 50
```

```
50 GOSUB 80
60 PRINT "FIN"
65 STOP
70 REMark subrutina arrastre cuadrado
80 PAPER fondo : CLS
90 BLOCK lado, lado, horiz, vert, color
100 RETURN
```

b) Utilizando un procedimiento con parámetros

```
10 cuadrado 4, 50, 20, 100, 2
20 PRINT "FIN"
30 DEfine PROCedure cuadrado(color, lado, horiz, vert, fondo)
40   PAPER fondo : CLS
50   BLOCK lado, lado, horiz, vert, color
60 END DEFine
```

En el primer programa los valores de *color, horiz. vert. y lado* se fijan con instrucciones **LET** antes de que la instrucción **GOSUB** active las líneas 80 y 90. El control se devuelve por la instrucción **RETURN.**

En el segundo programa los valores se dan en la primera línea como parámetros en la llamada al procedimiento *cuadrado*, el cual activa el procedimiento y al mismo tiempo le suministra los valores que necesita.

Un procedimiento en su forma más simple no lleva parámetros. Simplemente separa una determinada pieza de código. A pesar de ello, el procedimiento en su utilización más sencilla tiene ventaja sobre **GOSUB** ya que tiene un nombre adecuado y está aislando formando una unidad.

Los efectos de potencia y simplicidad de los procedimientos son más evidentes cuanto más largos son los programas. El trabajo que realizan los procedimientos en los programas largos no es mucho más que prevenir que el programa se alargue demasiado y resulte más dificultoso. El ejemplo anterior sólo ilustra la forma en que funciona un procedimiento en un contexto sencillo.

Ejemplos

Los ejemplos siguientes muestran la gama de vocabulario y sintaxis del SuperBASIC que ya ha sido explicada en este y los demás capítulos anteriores. Forman una base sobre la que se construirá la segunda parte de este manual.

Las letras de un palíndromo se dan en forma de elementos únicos en instrucciones **DATA**. El elemento final es un asterisco y suponemos que usted desconoce el número de letras del palíndromo. Lea las letras **(READ)** en una matriz e imprímalas al revés. Algunos palíndromos como "el ama ámale" o "Dábale arroz a la zorra el abad" solo funcionan si se ignora la puntuación y los espacios. El que damos a continuación funciona adecuadamente.

```
100 REMark Palíndromo
110 DIM text$(30)
120 LET text$ = FILL$ (" ",30)
130 LET cuenta =30
140 REPeat letras
150   READ carácter$
160   IF carácter$ ="*" THEN EXIT letras
170   LET cuenta = cuenta - 1
180 LET text$(cuenta) = carácter$
190 END REPeat letras
200 PRINT text$
200 DATA "Z", "O", "R", "R", "A", "R", "R", "O", "Z",
210 DATA ""Z", "O", "R","R", "A", "Y", "A", "R", "R", "O", "Z", "*"
```

El programa siguiente acepta como entrada números de la gama de 1 a 3999 y los convierte en números romanos equivalentes. No genera la forma más elegante, ya que produce IIII en lugar de IV.

```
100 REMark Números romanos
110 INPUT número
120 RESTORE 210
130 FOR tipo = 1 TO 7
```

```
140 READ letra$, valor
150  REPeat salida
160    IF número < valor : EXIT salida
170    PRINT letra$
180    LET número = número - valor
190  END REPeat salida
200 END FOR tipo
210 DATA "M", 1000, "D", 500, "C", 100, "L", 50, "X", 10, "V", 5, "I", 1
```

Estudie los ejemplos anteriores con cuidado utilizando si es necesario dry runs (pasadas en vacío) hasta que esté absolutamente seguro de que lo entiende todo perfectamente.

CONCLUSION

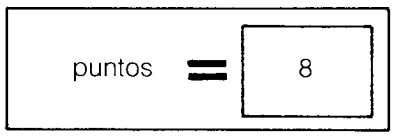
En el lenguaje SuperBASIC existen formas estructurales completas que hacen que los elementos del programa sigan en secuencia o bien se incluyan en otra estructura íntegramente. Todas las estructuras deben identificarse al sistema y deben llevar un nombre. Dispone de muchas características para unificar y simplificar y todo tipo de ventajas adicionales.

Muchas de ellas se explican e ilustran en los capítulos siguientes de este manual, que resulta más sencillo en su lectura que las secciones *Referencias* y *Conceptos* y *Palabras clave*. Así pues, es más fácil de leer porque no da todos los detalles técnicos ni agota todos los temas que trata. Por tanto, habrá ocasiones en las que necesite consultar las secciones de *Referencias*. Por otro lado, algunas de las innovaciones mayores se discuten en los capítulos siguientes; pocos lectores necesitarán utilizarlas todas, y puede ser útil omitir ciertas partes, al menos durante una primera lectura.

CAPITULO 9
TIPOS DE DATOS, VARIABLES E IDENTIFICADORES

Habrá observado que un programa (una secuencia de instrucciones) normalmente toma algunos datos para su funcionamiento de la entrada y produce algún tipo de resultados (salida). Habrá comprendido también que se establecen sistemas internos de almacenamiento de esos datos. Para evitar explicaciones técnicas innecesarias hemos sugerido que se imagine casilleros y que elija algunos nombres explicativos para ellos. Por ejemplo, si necesitamos almacenar algún número que represente la puntuación de dos tiradas de dado, imaginaremos un casillero llamado puntos, que puede contener un número, por ejemplo el 8.

Internamente, los casilleros están numerados y el sistema tiene un diccionario que conecta ciertos nombres con determinados casilleros numerados. Decimos, por tanto, que el nombre *puntos* lleva a un casillero determinado (mediante el diccionario interno).



A esta estructura completa se le llama **variable.**

Lo que usted ve es la palabra *puntos*. Se dice pues que esta palabra, *puntos,* es un identificador, ya que identifica el concepto que necesitamos, en este caso, el resultado de una tirada de dados, es decir 8. Como lo que vemos es el identificador, éste se convierte en el elemento del que hablamos, escribimos o pensamos. Escribimos sobre *puntos* y su valor en cualquier momento.

Existen cuatro tipos de datos simples llamados, respectivamente, de coma flotante, entero, de cadena y lógico. Se explican más abajo. Hablaremos de tipos de datos en lugar de hacerlo de tipos de variables porque los datos existen independientemente, por ejemplo 3, 4 o "sombrero azul" o como valor para una variable. Si usted comprende los diferentes tipos de variables también comprenderá los tipos diferentes de datos.

IDENTIFICADORES Y VARIABLES

1. Un identificador de SuperBASIC debe comenzar con una letra y es una secuencia de:

   letras mayúsculas o minúsculas
   dígitos o subrayados

2. Los identificadores pueden tener una longitud de hasta 255 caracteres y, por tanto, en la práctica no existe límite en su longitud.
3. Un identificador no puede ser igual a una palabra clave del lenguaje SuperBASIC.
4. Los nombres de las variables enteras llevan como identificador un nombre seguido de un signo de tanto por ciento (%).
5. Los nombres de las variables de cadena son identificadores y llevan el signo de dólar $ al final.

6. Los restantes identificadores no utilizarán nunca los símbolos % o $.
7. Los identificadores deben elegirse normalmente de forma que signifiquen algo para el lector, pero en lo que respecta al SuperBASIC, no tienen un significado especial si no es como identificadores de ciertos elementos.

VARIABLES DE COMA FLOTANTE

A continuación daremos algunos ejemplos de variables de coma flotante:

```
10 LET días = 24
20 LET ventanas = 3649.84
30 LET ventas_por_día = ventas/días
40 PRINT ventas_por_día
```

El valor de una variable de coma flotante puede ser cualquiera, dentro de la gama:

-10^{+615} hasta $+10^{+615}$ con ocho cifras significativas.

Suponga que en el ejemplo anterior las ventas son excepcionalmente 50 céntimos. Cambie la línea 20 a:

```
20 LET ventas = 0.50
```

Este sistema lo cambiará a:

```
20 LET ventas = 5E-1
```

Para interpretar esto comience con 5 o con 5.0 y mueva el punto decimal una posición a la izquierda, con ello:

5E-2 es lo mismo que 0.05

Después de ejecutado el programa, obtenemos una media diaria de ventas de:

2.0833333E-2 que es lo mismo que 0.020833333

Los números que llevan E se dice que están en forma exponencial.

(mantisa) E (exponente) = (mantisa) × 10 a la potencia (exponente)

VARIABLES ENTERAS

Las variables enteras sólo pueden tomar como valores números enteros en la gama del −32768 a 32767. A continuación se dan algunos ejemplos de nombres de variables enteras válidos, los cuales deben terminar siempre por %.

```
LET cuenta% = 10
LET seis_lotes% = RND(10)
LET número_3% = 3
```

La única desventaja que presentan las variables enteras cuando se necesitan números enteros es el signo porcentual % que puede ser equívoco. Este símbolo no tiene nada que ver en estas variables con el concepto de porcentaje. Simplemente es un símbolo conveniente que se coloca para mostrar que la variable es entera.

FUNCIONES NUMERICAS

La utilización de una función se parece en cierto modo a cocinar una tortilla. Se coloca un huevo que es procesado siguiendo ciertas reglas (la receta) y la salida es una tortilla. Por ejemplo, la función **INT** toma como entrada cualquier número y da como salida su parte entera. Los elementos que se introducen en una función se llaman parámetros o argumentos. **INT** es una función que da la parte entera de una determinada expresión. Puede escribir por tanto:

```
PRINT INT(5.6)
```

y se obtendrá como salida 5. Al valor 5.6 se le llama parámetro y la función devuelve el valor 5. Una función puede tener más de un parámetro. Por el momento, hemos visto:

```
RND (1 TO 6)
```

que es una función con dos parámetros. Sin embargo, las funciones devuelven siempre un solo valor. Debe ser así porque las funciones pueden colocarse en el interior de expresiones, por ejemplo:

```
PRINT 2 * INT(5.6)
```

Que producirá como salida 10. Es una propiedad importante de las funciones, que puede utilizarse en expresiones. De esto se deduce que devuelven valores únicos que pueden utilizarse en la expresión. **INT** y **RND** son funciones del sistema; están incluidas en él, pero más adelante aprenderá a escribirlas usted mismo.

El ejemplo siguiente muestra una forma común de utilización de la función **INT**.

```
10 REMark Redondeo
20 INPUT decimal
30 PRINT INT (decimal + 0.5)
```

En el ejemplo se introduce una fracción decimal y se devuelve redondeada en la salida. En este caso 4.7 se redondea a 5, pero 4.3 se redondearía a 4.

El mismo resultado puede obtenerse utilizando una variable entera y coerción (cambio forzado de tipos).

Las funciones trigonométricas se estudiarán en una sección posterior pero en la lista que viene a continuación se dan otras funciones numéricas.

Funcion	Efecto	Ejemplo	Valor devuelto
ABS	Absoluto (sin signo)	ABS(7) ABS(−4.3)	7 4.3
INT	Parte entera de un número con coma flotante	INT(2.4) INT(0.4) INT(−2.7)	2 0 −3
SQRT	Raíz cuadrada	SQRT(2) SQRT(16) SQRT(2.6)	1.414214 4 1.612452

Existe un modo de calcular las raíces que es fácil de entender. Para calcular una raíz cuadrada de 8 haga primero una suposición. No importa que esta suposición no sea en absoluto acertada. Suponga por tanto que elige la mitad del número, 8 en este caso, y por tanto su elección es 4.

Como 4 es mayor que la raíz cuadrada de 8 entonces 8/4 debe ser inferior. La inversa es también cierta. Si ha supuesto que 2, es un número inferior a la raíz caudrada, 8/2 será mayor.

A continuación tenemos ya dos números, uno demasiado pequeño y otro demasiado grande. Tomamos la media de esos dos números como primera aproximación para acercarnos a la respuesta correcta.

Este proceso se repite por aproximaciones hasta acercanos tanto al valor correcto que la diferencia sea muy pequeña.

```
10 REMark Raíces Cuadradas
20 LET número = 8
30 LET aprox = número/2
40 REPeat raíz
50   LET nueval = (aprox + número/aprox)/2
60   IF nueval = = aprox THEN EXIT raíz
70   LET aprox = nueval
```

```
80 END REPeat raíz
90 PRINT "La raíz cuadrada de " ! número ! "es "! nueval
```

muestra de la salida

```
La raíz cuadrada de 8 es 2.828427
```

Observe que la salida condicional (**EXIT**) del bucle debe encontrarse en la mitad. Las estructuras tradicionales no superan esta situación pero sí lo hace el SuperBASIC.

El signo = = de la línea 60 significa aproximadamente igual a, es decir, con una aproximación de .0000001.

OPERACIONES NUMERICAS

El lenguaje SuperBASIC permite las operaciones matemáticas normales. Se dará cuenta que son semejantes a funciones, con dos operandos exactamente, cada una de ellas. Se ha convenido también colocar un operando a cada lado del símbolo. Algunas veces la operación se escribe con símbolos familiares como, por ejemplo + o * . Otras, la operación se escribe mediante palabras clave, como **DIV** o **MOD**, pero no existe ninguna diferencia real. Las operaciones numéricas siguen un orden de prioridad. Por ejemplo, el resultado de:

```
PRINT 7 + 3 * 2
```

es 13 porque la multiplicación tiene prioridad.

Sin embargo:

```
PRINT (7 + 3) * 2
```

dará como salida 20, ya que los paréntesis pasan por alto la prioridad usual. Como verá más adelante, se pueden hacer tantas cosas con expresiones en el lenguaje SuperBASIC que una instrucción no puede establecerse completa en esta etapa (véase si es necesario la *Guía de Referencia de Conceptos*), pero las operaciones con las que trabajamos ahora tienen el orden de prioridad siguiente:

el primero es la elevación a potencias

multiplicación y división (incluyendo **DIV, MOD**)

el último la adición y sustracción.

Los símbolos + y − se utilizan también con un solo operando y sólo denotan positivo o negativo. Los símbolos que se utilizan de este modo tienen la prioridad anterior a todas y esta prioridad sólo dejará de considerarse cuando se utilicen paréntesis.

Finalmente, si los dos símbolos tienen la misma prioridad, la operación situada más a la izquierda es la que se realiza primero, por tanto:

```
PRINT 7−2 + 5
```

Realizará primero la sustracción y luego la adición. Este punto es importante si tiene que trabajar con números grandes o muy pequeños.

Operación	Símbolo	Ejemplos	Resultados	Notas
Adición	+	7+6.6	13.6	
Sustracción	−	7−6.6	0.4	
Multiplicación	*	3*2.1	6.3	
		2.1*(−3)	−6.3	
División	/	7/2	3.5	No se
		−17/5	−3.4	divide por 0
Elevación a potencia	^	4^1.5	8	
Elevación a potencia (enteros)	^	3^2	9	Sólo enteros
División de enteros	DIV	−8DIV2	−4	Sólo enteros
		7DIV2	3	No se divide entre cero
Módulos	MOD	13 MOD 5	3	
		21 MOD 7	0	
		−17 MOD 8	7	

La operación Módulos devuelve el resto de una división. Cualquier intento de dividir por cero generará un error y terminará la ejecución del programa.

EXPRESIONES NUMERICAS

Hablando estrictamente, una expresión numérica es una expresión que evalúa respecto a un número, y ofrece más posibilidades de las que necesitamos discutir aquí. El lenguaje SuperBASIC le permite realizar cosas sencillas de forma sencilla. En esta sección nos concentraremos en los usos normales de las expresiones matemáticas.

Básicamente, en SuperBASIC las expresiones son las mismas que en matemáticas, pero estas expresiones deben escribirse completas en forma secuencial.

$$\frac{5 + 3}{6 - 4}$$

se convierte en SuperBASIC (o en otros lenguajes BASIC):

```
(5 + 3) / (6 - 4)
```

En el álgebra que se estudia en la escuela secundaria la solución a la ecuación cuadrática:

$$ax^2 + bx + c = 0$$

tiene una solución en notación matemática que es:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Si comenzamos con la ecuación:

$$2x^2 - 3x + 1 = 0$$

Ejemplo 1

El programa siguiente obtiene la solución:

```
10 READ a, b, c
20 PRINT "La raíz es" ! (-b + SQRT(b^2-4 *a*c)) / (2*a)
30 DATA 2,-3,1
```

Ejemplo 2

En problemas de simulación de los juegos de naipes, se puede hacer corresponder las cartas con los números desde el 1 al 52 como sigue:

del 1 al 13 As, dos... rey de corazones
del 14 al 26 As, dos... rey de pics
del 27 al 39 As, dos... rey de diamantes
del 40 al 52 As, dos... rey de tréboles

Se puede identificar una determinada carta del siguiente modo:

```
10 REM Identificación de cartas
20 LET carta = 23
30 LET palo = (carta-1) DIV 13
40 LET valor = carta MOD 13
50 IF valor = 0 THEN LET value = 13
60 IF valor = 1 THEN PRINT "As de";
70 IF valor > = 2 AND valor < = 10 THEN PRINT valor ! "de";
80 IF valor = 11 THEN PRINT "Jack de ";
90 IF valor = 12 THEN PRINT "Reina de ";
100 IF valor = 13 THEN PRINT "Rey de ";
110 IF palo = 0 THEN PRINT "corazones"
120 IF palo = 1 THEN PRINT "pics"
130 IF palo = 2 THEN PRINT "diamantes"
140 IF palo = 3 THEN PRINT "tréboles"
```

En este programa aparece una nueva idea. Se encuentra en la línea 70. El significado es, evidentemente, que el número se escribe si dos instrucciones lógicas son verdad, estas instrucciones son:

el valor es mayor o igual que 2
AND-(y) el valor es menor o igual que 10

Las cartas que se salen de esta gama son o ases o figuras y deben tratarse de forma diferente.

Observe también la utilización de ! en la instrucción PRINT para obtener espacio y ; para asegurarnos de que la salida continúa en la misma línea.

Existen dos grupos de funciones matemáticas que no hemos discutido todavía. Son las funciones trigonométricas. Puede que las necesite para organizar ciertas presentaciones de pantalla. Este tipo de funciones está además totalmente definido en la sección de referencias.

VARIABLES LOGICAS

Si hablamos con exactitud, el lenguaje SuperBASIC no permite variables lógicas, pero sí le permite utilizar otras variables como variables lógicas. Por ejemplo, ejecute el programa siguiente:

```
10 REMark Variable Lógica
20 LET hambriento = 1
30 IF hambriento THEN PRINT "Toma un bollo"
```

Usted esperaba una expresión lógica en la línea 30 pero en su lugar está la variable numérica hambriento. El sistema interpreta el valor 1 de hambriento como verdadero y por tanto la salida será:

```
Toma un bollo
```

Si en la línea 20 se leyera:

```
LET hambriento = 0
```

no habría salida. El sistema interpreta el cero como una falsedad y todos los demás valores como verdaderos. Todo esto es útil, pero puede disfrazarse la calidad numérica de hambriento escribiendo:

```
10 REMark Variable Lógica
20 LET verdad = 1 : falso = 0
30 LET hambriento = verdad
40 IF hambriento THEN PRINT "Toma un bollo".
```

VARIABLES DE CADENA

Se podría decir mucho sobre el manejo de las cadenas y las variables de cadena y, por tanto, lo dejaremos para otro capítulo.

PROBLEMAS SOBRE EL CAPITULO 9

1. Un rico negociante en petróleo juega con una moneda del siguiente modo: si al tirarla sale cruz obtendrá un 1, y si sale cara vuelve a tirar, pero la posible recompensa se dobla. Esta acción se repite, de forma que los premios son los siguientes:

TIRADAS	1 2 3 4 5 6 7
PREMIOS	1 2 4 8 16 32 64

   Simulando el juego intente decidir cuál sería un pago inicial adecuado para este tipo de juego:

   a) Si el jugador debe tener un límite máximo de siete tiradas por cada juego.

   b) Si no debe haber un número máximo de tiradas.

2. Juan y Pedro se ponen de acuerdo para jugar del siguiente modo. En un determinado momento cada uno de los dos divide sus monedas en dos mitades y le pasa la mitad al otro jugador. A continuación cada uno divide su nuevo total y le pasa la mitad al otro. Averigüe lo que ocurre según avanza el juego si Juan comienza con 16 duros y Pedro con 64.

3. ¿Qué ocurrirá si se cambia el juego y cada uno devuelve al otro una cantidad igual a la mitad de lo que el otro posee?

4. Escriba un programa que forme palabras de tres letras elegidas tomando las letras A, B, C y D y las imprima hasta que aparezca la palabra inglesa BAD.

5. Modifique el programa anterior para que termine cuando aparezca una palabra real cualquiera, de tres letras.

CAPITULO 10
LOGICA

Si ha leído los capítulos anteriores, estará de acuerdo en que la repetición, la toma de decisiones y la ruptura de tareas en otras subtareas menores son los conceptos más importantes en el análisis de los problemas, al diseño de los problemas y en su creación o codificación. Dos de estos conceptos, la repetición y la toma de decisiones necesitan expresiones lógicas, como las que aparecen en las líneas del siguiente programa:

```
IF puntos = 7 THEN EXIT tiradas
IF palo = 3 THEN PRINT "tréboles"
```

La primera posibilita la salida **EXIT** de un bucle de repetición **REPeat.** La segunda es sencillamente una decisión de hacer algo o no hacerlo. Una expresión matemática evalúa hasta millones de cadenas de caracteres. Puede considerar extraño que las expresiones lógicas, a las que se da tanta importancia sólo pueden evaluar uno de dos valores posibles: *verdadero o falso.*

En el caso de

```
puntos = 7
```

es obvio que es correcto. O bien la puntuación toma el valor 7 o no lo toma. La expresión debe ser verdadera o falsa, suponiendo que no sea una expresión sentido. Puede ocurrir que usted no conozca el valor en un momento dado, pero lo sabrá a su debido tiempo.

Debe usted ser muy cuidadoso con las expresiones que lleven palabras como **OR, AND, NOT,** aunque estas palabras son esenciales en una buena programación. Incluso lo serán más, mirando hacia otros tipos de lenguajes basados más en descripciones precisas de lo que necesita el usuario que en lo que debe realizar el ordenador.

AND

La palabra **AND** en SuperBASIC es como la palabra and inglesa, cuyo significado es la y española. Estudie el siguiente programa:

```
10 REMark AND
20 PINT "Introduzca dos valores" \ "1 para VERD o 0 para FALSO"
30 INPUT llueve, faltan_tejas
40 IF llueve AND faltan_tejas TEN PRINT "Te mojarás"
```

Como en la vida real, sólo se mojará usted si llueve y faltan tejas. Si una (o las dos) expresiones de las variables lógicas sencillas

llueve
faltan_tejas

es falsa, la expresión lógica del conjunto:

llueve **AND** *(y) faltan_tejas*

será también falsa. Para que una expresión sea verdadera es necesario que sean verdaderos los dos valores. Esta regla puede verse bien en la tabla que se da a continuación. Sólo cuando la expresión conjunto de las dos es verdadera, usted se moja.

llueve	faltan_ tejas	llueve y faltan_tejas	efecto
FALSO	FALSO	FALSO	SECO
FALSO	VERDADERO	FALSO	SECO
VERDADERO	FALSO	FALSO	SECO
VERDADERO	VERDADERO	VERDADERO	MOJADO

Regla para AND

OR

En la vida real la palabra or (o) se utiliza de dos formas diferentes. Ilustraremos la utilización inclusiva de **OR**, poniendo como ejemplo un capitán de cricket que busca jugadores. Hace la siguiente pregunta: ¿usted bate o tira? Estará encantado si el jugador hace una de las dos cosas bien, pero lo estará también si el jugador puede hacer las dos cosas. En programación, una expresión compuesta que utiliza **OR** es verdadero si cualquiera de las expresiones, o las dos expresiones, son verdad. Pruebe el programa siguiente:

```
10 REMark test OR
20 PRINT "Introduzca dos valores" \ "1 para VERD. o 0 para FALSO"
30 INPUT "Puede pegar", bateador
40 INPUT "Puede tirar", bowler
50 IF bateador OR bowler THEN PRINT "al equipo"
```

Vea los efectos de todas las combinaciones en la tabla que aparece a continuación

bateador	lanzador	bateador OR lanzador	efecto
FALSO	FALSO	FALSO	fuera
FALSO	VERDADERO	VERDADERO	en el equipo
VERDADERO	FALSO	VERDADERO	en el equipo
VERDADERO	FALSO	VERDADERO	en el equipo

Reglas de OR

Cuando se utiliza el **OR inclusivo**, un valor verdadero en cualquiera de las expresiones simples producirá un valor verdadero en la expresión compuesta. Si Ian Botham, jugador inglés, tuviera que responder a la pregunta tanto como lanzador como bateador, las dos expresiones simples serían verdaderas y por tanto la expresión compuesta lo sería también. Y formaría parte del equipo.

Si escribe un 0 para falso y un 1 para verdadero obtendrá todas las posibles combinaciones en sistema binario:

00
01
10
11

NOT

La palabra **NOT** tiene el significado obvio:

NOT *verdadero* es lo mismo que *falso*
NOT *falso* es lo mismo que *verdadero*

Sin embargo es necesario poner mucho cuidado. Suponga que tiene un triángulo rojo, y dice que es:

NOT *rojo* **AND** *cuadrado*
no y

es una expresión ambigua. Si lo que quería decir es:

(**NOT** *rojo*) **AND** *cuadrado*
no y

para un triángulo rojo la expresión anterior es falsa. Si quiere por el contrario decir:

NOT *(rojo* **AND** *cuadrado)*
no y

para un triángulo rojo la expresión completa es correcta. Por tanto, en programación debe existir alguna forma que deje claro el significado. Esta regla es que **NOT** tiene preferencia sobre **AND**, y por tanto la interpretación:

(**NOT** *rojo*) **AND** *cuadrado*
no y

es la correcta, y es la misma que

NOT *rojo* **AND** *cuadrado*

Para obtener la otra interpretación se deben utilizar paréntesis. Si necesita utilizar una expresión lógica compleja es mejor utilizar paréntesis y **NOT**, naturalmente si su uso refleja lo que usted desea. En cualquier caso puede siempre eliminar los paréntesis utilizando las leyes siguientes (atribuidas a Augustus De Morgan)

NOT (a **AND** b)	es igual que	**NOT** a **OR NOT** b
NOT (a **OR** b)	es igual que	**NOT** a **AND NOT** b

Por ejemplo:

NOT (*alta* **AND** *bonita*) es lo mismo que
NOT *alta* **OR NOT** *bonita*
no o no

NOT (*hambriento* **OR** *sediento*) es lo mismo que
NOT *hambriento* **AND NOT** *sediento*
no y no

Pruebe lo anterior introduciendo:

```
10 REMark NOT y paréntesis
20 PRINT "Introduzca dos valores" \ "1 para VERD o 0 para FALSO"
30 INPUT "alta"; alta
40 INPUT "bonita"; bonita
50 IN NOT (alta AND bonita) THEN PRINT "PRIMERA"
60 IF NOT alta OR NOT bonita THEN PRINT "SEGUNDA"
```

Para cualquier combinación de números que se den en la entrada la salida será cualquiera de las dos palabras o ninguna, pero nunca una. Esto significa que las dos expresiones lógicas compuestas son equivalentes.

XOR-Exclusiva OR

Suponga que un profesional del golf desea tener un asistente que pueda llevar una tienda o dar lecciones de golf. Si un aspirante tiene ambas habilidades puede que no le dé el empleo porque el profesional del golf puede tener miedo de que una persona tan competente pueda dejarlo en cualquier momento. El profesional aceptaría un buen jugador de golf que no pueda llevar la tienda o también un jugador de golf mediocre que pueda regentar bien la tienda. Esto es una situación OR exclusiva, una cualquiera de las dos opciones es aceptable, pero no las dos. El programa siguiente probará a los solicitantes.

```
10 REM test XOR
20 PRINT "Teclee un 1 para sí o 0 para no"
30 INPUT "¿Puede regir una tienda?"
40 INPUT "¿Puede dar clases de golf"?
50 IF tienda XOR golf THEN PRINT "Aceptado"
```

Las únicas combinaciones posibles con las respuestas que causarán la salida "Aceptado" son (0 y 1) o (1 y 0). Más abajo se dan las reglas de la instrucción **XOR**.

Apto para tienda	Apto para clases	Tienda y clases	Efecto
FALSO	FALSO	FALSO	no acept.
FALSO	VERDADERO	VERDADERO	aceptado
VERDADERO	FALSO	VERDADERO	aceptado
VERDADERO	VERDADERO	FALSO	no acept.

Reglas de XOR

PRIORIDADES

El orden de prioridades para los operadores lógicos es (de arriba abajo).

NOT
AND
OR, XOR

Por ejemplo, la expresión

rica OR *alta* AND bonita
o y

tiene el mismo significado que

rica OR (*alta* AND *bonita*)
o y

Primero se realiza la operación **AND**. Para probar que las dos expresiones lógicas tienen efectos idénticos ejecute el siguiente programa:

```
100 REMark Prioridades
110 PRINT "Introduzca tres valores"\" Teclee 1 para Sí y 0 para No"!
120 INPUT rica, alta, bonita
130 IF rica OR alta AND bonita THEN PRINT "SI"
140 IF rica OR (alta AND bonita) THEN PRINT "AY"
```

Cualquier combinación con tres ceros o unos que introduzca en la línea 120 dará una salida que puede no tener nada o

SI
AY

Asegúrese de que prueba todas las posibilidades, introduciendo los datos que forman ocho números binarios de tres dígitos, del 000 al 111

000 001 010 011 100 101 110 111

PROBLEMAS SOBRE EL CAPITULO 10

1. Coloque diez números en una instrucción **DATA**. Lea **(READ)** cada número y si es mayor que 20 imprímalo.
2. Pruebe todos los números desde el 1 al 100 e imprima todos los que sean cuadrados perfectos o divisibles por siete.
3. Los juguetes pueden dividirse en varios grupos, Seguros (S) o Peligrosos (P), Caros (C) o Baratos (B), y también para Chicos (O) o para Chicas (A) o para Cualquiera (C). Las cualidades de cada juguete estarán definidas por un trío de letras. Coloque cinco tríos de ese tipo en una instrucción **DATA** y luego busque e imprima sólo los Seguros y adecuados para niñas.
4. Modifique el programa 3 e imprima los que son caros y peligrosos.
5. Modifique el programa 3 e imprima los que son seguros, no son caros y valen para los dos sexos.

CAPITULO 11
MANEJO DE LAS CADENAS DE TEXTO

Usted ya ha utilizado variables de cadena para almacenar cadenas de caracteres y conoce también que las reglas para manipular variables de cadena o constantes de cadena no son las mismas que las válidas para las variables numéricas o constantes numéricas. El lenguaje SuperBASIC ofrece una gama muy amplia de facilidades para manipular con efectividad las cadenas de caracteres. En particular, el concepto de partición de las cadenas extiende y simplifica el manejo de subcadenas o partes de una cadena.

ASIGNACION DE CADENAS

El almacenamiento de las variables de cadena se realiza atendiendo a las necesidades del programa. Por ejemplo, las líneas:

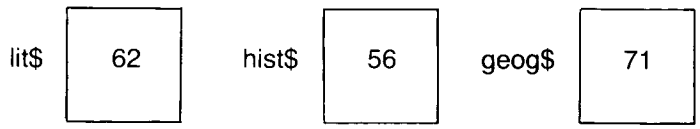
```
100 LET palabras$ = "LARGO"
110 LET palabras$ = "ENORME"
120 PRINT palabras$
```

imprimirán una palabra de cinco letras en total. La primera línea habilita espacio para que esas cinco letras se coloquen, pero esta colocación será rebasada por la segunda línea que necesita espacio para seis.

Sin embargo es posible dimensionar (reservar espacio) para una variable de cadena, y en este caso se define como la longitud máxima y la variable se comporta siempre como una matriz.

UNION DE CADENAS

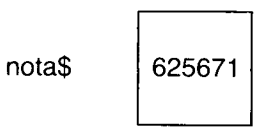
Puede ocurrir que desee construir sus grabaciones de proceso de datos obteniéndolas de diversas fuentes. Suponga por ejemplo que es usted un profesor y desea almacenar un conjunto de tres notas por cada estudiante, de Literatura, Historia y Geografía. Las notas se guardan en variables tal y como se muestra más abajo.



Como parte del historial de cada alumno, puede que usted desee combinar los tres valores de cadena en otra cadena de seis caracteres llamada *notas$*. No tiene más que escribir:

```
LET nota$ = Lit$ & hist$ & geog$
```

Habrá creado otra variable tal y como se muestra:

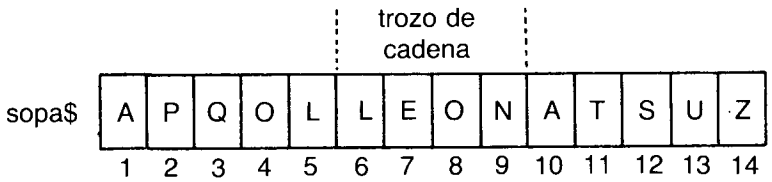


Recuerde, sin embargo, que los números que estamos manejando son en realidad una cadena de caracteres, aunque los caracteres a veces son números.

COPIA DE UN TROZO DE CADENA (SUBCADENA)

Una fracción de cadena (o subcadena) es parte de una cadena. Puede ser cualquier cosa, desde un simple carácter hasta la cadena completa. Para identificar el pedazo de cadena es necesario saber las posiciones de los caracteres que la componen.

Suponga que está construyendo un juego de niños en el que tienen que reconocer una palabra escondida en una sopa de letras. Cada letra tiene un número interno —un índice— que corresponde a su posición en la cadena. Suponga que la cadena completa se almacena en la variable *sopa$* y la clave es Gato grande



Puede verse claramente que la respuesta está definida por los números que van desde el 6 hasta el 9, indicándonos dónde está. Abstrayendo la respuesta, podemos:

```
100 sopa$ = "APQOLLEONATSUZ"
110 LET an$ = sopa$ (6 TO 9)
120 PRINT an$
```

SUSTITUCION DE UNA SUBCADENA

Suponga ahora que desea cambiar el animal escondido y desea ahora colocar un toro. Escriba dos líneas adicionales:

```
130 sopa$ (6 TO 9) = "TORO"
140 PRINT sopa$
```

La salida del programa de cinco líneas completo es:

```
LEON
APQOLTOROATSUZ
```

Todas las variables de cadena están vacías inicialmente, y tienen una longitud cero. Si se intenta copiar una cadena en el interior de una subcadena cuya longitud es insuficiente, esta asignación (de espacio) no será reconocida por el SuperBASIC.

Si desea copiar una cadena dentro de una subcadena es mejor que se asegure que la cadena destino tiene suficiente longitud; por ejemplo, llenándola con espacios.

```
100 LET asignat$ = "INGLES MATEM INFORM"
110 LET estudia$ ="
120 LET estudia$ (8 TO 12) = asignat$ (8 TO 12)
```

Decimos pues que "TORO" es una subcadena "APQOLTOROATSUZ". La frase de definición

```
(6 TO 9)
```

se llama **delimitador.** Se utiliza además para otros objetivos. Observe como

puede utilizarse la misma notación en ambos lados de la instrucción **LET**. Si desea referirse a un solo carácter sería ridículo escribir:

```
sopa$ (6 TO 6)
```

únicamente para sacar la "T" (posiblemente como clave), por tanto puede escribir en su lugar:

```
sopa$ (6)
```

para referirse a un solo carácter.

CAMBIO FORZADO DE TIPOS (COERCION)

Suponga que tiene la variable *nota$* con los resultados de los exámenes, y debe extraer la subcadena con las notas de Historia para realizar un cambio de escala, ya que el profesor de Historia ha sido demasiado estricto con las notas. Las líneas siguientes tendrán el extracto con las notas de Historia:

```
100 LET nota$ = "625671"
110 LET hist$ = nota$(3 TO 4)
```

El problema que surge ahora es que el valor "56" de la variable *hist$* es una cadena de caracteres con datos no numéricos. Si desea modificar las notas multiplicándolas, por un factor de escala de, por ejemplo, 1.125 el valor de *hist$* debe convertirse primero en cadena de datos numéricos. El lenguaje SuperBASIC realiza esta conversión automáticamente cuando tecleamos:

```
120 LET num = 1.125 * hist$
```

La línea 120 convierte la cadena "56" en el número 56 y lo multiplica por 1.125, lo que da un valor de 63.

Ahora debemos reemplazar la nota que teníamos por la nueva nota obtenida, pero esta nueva nota es aún el número 63 y antes de volverlo a insertar en la cadena original, debe convertirse en la cadena "63". Otra vez el SuperBASIC convertirá automáticamente el número cuando tecleemos:

```
130 LET nota$(3 TO 4) = num
140 PRINT nota$
```

La salida del programa completo será:

```
626371
```

que muestra cómo la nota de historia se ha incrementado hasta 63.

Estrictamente, podemos decir que no es correcto mezclar varios tipos de datos en una instrucción **LET** . Sería tonto escribir:

```
LET num = "LEON"
```

y usted obtendría un mensaje de error si lo hiciera, pero si por el contrario escribe:

```
LET num = "65"
```

El sistema concluirá que desea que el número 65 se convierta en el valor de num, y por tanto lo hace. El programa completo es:

```
100 LET nota$ ="625671"
110 LET hist$ = nota$(3 TO 4)
120 LET num = 1.125 * hist$
130 LET nota$(3 TO 4) = num
140 PRINT nota$
```

De nuevo la salida es la misma.

En la línea 120 se convirtió el valor de cadena a forma numérica para que pudiera multiplicarse. En la línea 130 se convirtió un número a forma de ca-

dena. Esta conversión de tipos de datos se conoce con el nombre de **cambio forzado de tipos.**

El programa puede escribirse con más economía si ya ha entendido la partición de cadenas en subcadenas y el cambio forzado de tipos.

```
100 LET nota$ = "625671"
110 LET nota$(3 TO 4)= 1.125 * nota$(3 TO 4)
120 PRINT nota$
```

Si ya ha trabajado con otros lenguajes BASIC apreciará la simplicidad y potencia de la partición de cadenas en subcadenas y el cambio forzado de tipos.

BUSQUEDA DE UNA CADENA

Dada una subcadena, se puede buscar una cadena que concuerde con ella. El programa siguiente presenta una sopa de letras invitando al usuario a encontrar al animal.

```
100 REM Búsqueda de animales
110 LET sopa$ = "MARAGATO"
120 PRINT sopa$
130 INPUT "Cuál es el animal"! an$
140 an$ INSTR sopa$ AND an$(1) = "G"
150   PRINT "Correcto"
160 ELSE
170   PRINT "No es correcto"
180 END IF
```

El operador **INSTR** devuelve un cero si la solución no es correcta. Si la solución es correcta **INSTR** devuelve el número que se encuentra en la posición inicial de la subcadena coincidente, dentro de la cadena, en este caso, 6.

Como la expresión:

```
INSTR sopa$,an$
```

puede tratarse como una expresión lógica, la posición de la cadena es una búsqueda con éxito puede considerarse como verdad mientras que en una búsqueda sin éxito se puede considerar como *"falso"*.

OTRAS FUNCIONES DE CADENA

Se habra encontrado ya con la función **LEN**, que devuelve la longitud (número de caracteres) de una cadena.

Puede usted desear repetir varias veces una cadena en particular, o quizás un carácter. Por ejemplo, si usted desea obtener una fila de asteriscos, en lugar de introducir cuarenta asteriscos en una instrucción **PRINT** o crear un bucle, puede sencillamente escribir:

```
PRINT FILL$ ("*",40)
```

Finalmente, podemos utilizar la función **CHR$** para convertir los códigos ASCII en caracteres de cadena, por ejemplo:

```
PRINT CHR$(65)
```

dará como salida A

COMPARACION DE CADENAS

Una buena parte de la informática tiene como objetivo la organización de datos de forma que se puedan buscar rápidamente. En algunos casos es necesario clasificarlos en orden alfabético. La base de ciertos procesos de clasificación es la facilidad en la comparación de dos cadenas para averiguar cuál

es la anterior. Con letras A, B, C..., tienen códigos internos 65, 66, 67 las instrucciones siguientes son correctas.

A es anterior a B
B es anterior a C

y debido a una comparación interna carácter a carácter, automáticamente se obtiene:

GATO es anterior a Perro
CAN es anterior a GATO.

Puede escribirse, por ejemplo:

```
IF "GATO" < "PERRO" THEN PRINT "MIAU"
```

y la salida será

```
MIAU
```

Del mismo modo:

```
IF "PERRO" > "GATO" THEN PRINT "GUAU"
```

y la salida será

```
GUAU
```

Para la comparación de cadenas se utilizan los símbolos matemáticos de comparación. Las expresiones lógicas que se dan a continuación son ambas cosas, permitidas y *verdaderas*.

```
"ALF" <"BEN"
"KIT" > "BEN"
"KIT" < = "LEN"
"KIT" > ="KIT"
"PAT" > = "LEN"
"LEN" < = "LEN"
"PAT" < > "PET"
```

Por el momento, las comparaciones basadas en códigos internos pueden tener sentido, pero en ocasiones no es posible restringir los datos a letras mayúsculas. Podemos por ejemplo desear:

que Cat sea menor que COT
y K2N sea menos que K27N

Una comparación simple, carácter por carácter basada en códigos internos no dará estos resultados. El SuperBASIC se comporta de una forma más inteligente. El programa siguiente que sugiere la entrada adecuada (e indica la salida que se produce), ilustra las reglas de comparación de las cadenas.

```
100 REMark comparaciones
110 REPeat comp
120   INPUT "introduzca una cadena" ! primero$
130   INPUT "introduzca otra cadena" ! segundo$
140   IF primero$ < segundo$ THEN PRINT "Inferior"
150 IF primero$ > segundo$ THEN PRINT "Mayor"
160   IF primero$ = segundo$ THEN PRINT "Igual"
170 END REPeat comp
```

Entrada		Salida
CAT	COT	Inferior
CAT	CAT	Igual
PET	PETE	Inferior
K6	K7	Inferior
K66	K7	Mayor
K12N	K6N	Mayor

> Mayor que - Comparación dependiente-mayús. números comparados en numérico

< Menor que - Dependiente - mayús. números comparados en orden numérico.

= Igual - dependient - mayús., las cadenas deben ser iguales.

== Equivalente - La cadena debe ser casi la misma, independiente mayúsc., números comparados en orden numérico.

>= Mayor o igual que - Dependiente mayús., números comparados en orden numérico.

<= Menor o igual que - Dependiente mayúsc., números comparados en orden numérico.

PROBLEMAS SOBRE EL CAPITULO 11

1. Coloque 12 letras todas diferentes en una variable de cadena y otras seis letras en una segunda variable. Busque en la primera cadena cada una de las seis letras una tras otra especificando en cada caso si se encuentran o no.
2. Repita el problema utilizando matrices de caracteres sencillos en lugar de cadenas. Coloque veinte letras mayúsculas en una cadena y liste todas las que se repiten.
3. Escriba un programa que lea una muestra de texto todo él en mayúsculas. Cuente la frecuencia con la que aparece cada letra e imprima los resultados.

   "EL GOBIERNO ES UN FIDEICOMISO Y LOS FUNCIONARIOS DEL GOBIERNO SON FIDEICOMISARIOS Y AMBOS, EL FIDEICOMISO Y LOS FIDEICOMISARIOS HAN SIDO CREADOS PARA BENEFICIO DEL PUEBLO".
   HENRY CLAY, 1829.

4. Escriba un programa que cuenta el número de palabras del texto siguiente. Las palabras se reconocen porque comienzan por una letra y continúan por un espacio, un punto u otro carácter.

   "LOS INFORMES SOBRE MI MUERTE SON MUY EXAGERADOS. CABLE DE MARK TWAIN A LA ASOCIACION DE LA PRENSA, LONDRES 1896."

5. Vuelva a escribir el último programa ilustrando la utilización de variables lógicas y procedimientos.

CAPITULO 12
SALIDA DE PANTALLA

El lenguaje SuperBASIC tiene una gama de características para las presentaciones de pantalla tan amplia que las describiremos en dos secciones: *Simple Impresión* y *Pantalla.*

La primera sección describe la salida del texto ordinario, y en ella se explican los métodos mínimamente correctos para visualizar mensajes, textos o salidas numéricas. Incluso en esta mundana sección, hay innovaciones en el concepto de "espacio inteligente", que es un efecto de combinar la facilidad de uso con efectos muy útiles.

La segunda sección es mucho más amplia porque tiene mucho que decir. La gama de tan amplia de características facilita mucho las cosas. Por ejemplo, se puede dibujar un círculo sin más que escribir la palabra **CIRCLE** seguida de algunos detalles que definen elementos como su posición o tamaño. Muchos otros sistemas necesitan que se conozca de antemano algo de geometría o trigonometría para realizar algo que conceptualmente es muy sencillo.

Cada palabra clave ha sido elegida cuidadosamente para que refleje el efecto causado. **WINDOW** define un área de la pantalla; **BORDER** coloca un borde rodeándola, **PAPER** define el color del fondo e **INK** determina el color del texto escrito en la pantalla.

Si estudia este capítulo y practica los conceptos vertidos en él, recordará rápidamente el efecto que produce cada palabra clave. Puede dar gran calidad a sus trabajos muy fácilmente. Con la experiencia comprenderá por qué los gráficos por ordenador se están convirtiendo en una nueva forma de arte.

IMPRESION SIMPLE

La palabra **PRINT** puede ir seguida por una secuencia de elementos de impresión. Un elemento de impresión puede ser cualquiera de los siguientes:

un texto, como por ejemplo "Esto es un texto"
variables, como por ejemplo: *núm, palabra$*
expresiones como 3 * *núm, día$ & semana$*

En cualquier instrucción se pueden mezclar los elementos de impresión, pero deben colocarse uno o más separadores entre cada dos elementos. Los separadores pueden ser cualesquiera entre los siguientes:

; No tiene efecto; sólo separa elementos de impresión.

! Un espacio inteligente inserta normalmente un espacio entre dos elementos de salida. Si un elemento determinado no cabe en la línea se comporta como un símbolo de comienzo de línea. Si el elemento se encuentra al comienzo de una línea no se genera espacio.

, El tabulador causa que la salida esté tabulada en columnas de 8 caracteres.

\ El símbolo de una nueva línea fuerza su comienzo inmediato.

TO Permite tabular

Los números 1, 2, 3 son elementos legítimos de impresión, y nos convienen para ilustrar los efectos de los separadores de impresión.

Instrucción	Efecto
10 PRINT 1,2,3	1 2 3
10 print 1!2!3	1 2 3
10 PRINT 1\2\3	1 2 3
10 PRINT 1;2;3	123
10 PRINT "Esto es texto" 10 LET palabra$="" 10 PRINT palabra$	Esto es texto mueve la posición de impresión
10 LET núm = 13 10 PRINT núm	13
10 LET cont$ ="sí" 10 PRINT "Digo"!cont$	Digo sí
10 PRINT "La suma es"! 4 + 2	La suma es 6

El comando **AT** se utiliza para posicionar la salida impresa en cualquier lugar de la línea que se está utilizando o en cualquier lugar de la pantalla.

Por ejemplo:

```
AT 10,15: PRINT "En la columna 10 y en la fila 15"
```

El comando **CURSOR** se utiliza para colocar la salida impresa en cualquier lugar de la red de pixels de las pantallas. Por ejemplo:

```
CURSOR 100,150 : PRINT "Significa 100 unidades de pixels en sentido
                 horizontal y 15 en vertical"
```

Si lee la *Guía de Referencia de Palabras Clave,* puede parecerle difícil conciliar la sección que trata de la instrucción **PRINT** con la descripción anterior. Dos de esas dificultades desaparecerán si usted entiende que:

El texto entrecomillado, las variables y los números son estrictamente hablando, expresiones, y constituyen las formas más sencillas (degeneradas).

Los separadores de impresión están clasificados extrictamente como elementos de impresión.

LA PANTALLA

Esta sección trata sobre los efectos generales que siguen a su decisión de obtener una salida (imprimir) de texto o gráficos. La instrucción:

```
MODE 8 o MODE 256
```

selecciona el **MODO 8** en el que se encuentran:

256 pixels en la dirección horizontal, numerados de 0 a 511 (dos números por pixel)
256 pixels en la dirección vertical, numerados de 0 a 255
8 colores

Se define un pixel como el área mínima de color que puede visualizarse. Utilizamos el término **colores simples,** porque se comienza con los colores simples ordinarios, que sólo son cuatro. Sin embargo, si utilizamos otros efectos se puede obtener una gran variedad de tonalidades y texturas. Si usted utiliza un QL con una televisión no podrá reproducir ninguno de estos efectos adicionales.

La instrucción:

```
MODE 4 o MODE 512
```

selecciona **MODE 4** en el que existen:

512 pixels en dirección horizontal, numerados desde el 0 al 511
256 pixels en dirección vertical, numerados del 0 al 255
4 colores

COLOR

Para seleccionar un color utilice el código siguiente en combinación con la palabra adecuada, como por ejemplo **PAPER, INK,** etc. Observe que los números en sí mismos no significan nada. Los números se interpretan como colores cuando siguen a una instrucción **PAPER, INK,** etc.

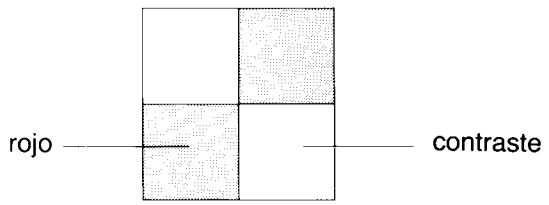
Modo de 8 colores	Código	Modo de 4 colores
negro	0	negro
azul	1	negro
rojo	2	rojo
morado	3	rojo
verde	4	verde
cyano	5	verde
amarillo	6	blanco
blanco	7	blanco

Código de colores

Por ejemplo, en el **MODO 8, INK 3** dará color morado

STIPPLES

Si lo desea, puede especificar dos colores con una única instrucción adecuada. Por ejemplo, 2,4 dará una cuadrícula como se muestra en el dibujo de más abajo. Por cada grupo de cuatro pixels, dos serán rojos (código 2) correspondiendo al color que se seleccionó primero. Los otros dos pixels serán contrastados. Este efecto no es posible obtenerlo en una televisión doméstica.



Si escribimos:

```
INK 2,4
```

se forma una mezcla de los dos colores cuyos códigos son 2 y 4. Les llamaremos a partir de ahora código y contraste.

```
INK color, contraste
```

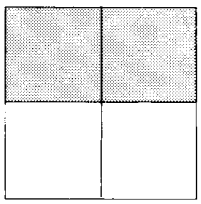
Usted verá los efectos de las mezcla de colores probándolos, pero más abajo daremos más detalles técnicos.

```
10 REMark Color/Contraste
20 FOR color = 0 TO 7 STEP 2
30   PAPER color : CLS
40   FOR contraste = 0 TO 7 STEP 2
50     BLOCK 100,50,40,50, color, contraste
60     PAUSE 50
70   END FOR contraste
80 END FOR color
```

Si desea probar otro tipo de granulación diferente, puede añadir otro tercer número de código a las especificaciones sobre el color. Por ejemplo:

```
INK 2,4,1
```

especificará un efecto de franjas horizontal en rojo y verde. El bloque de cuatro pixels tendrá el siguiente aspecto:



Los efectos posibles se muestran utilizando el rojo ▒ y un contraste □.

Código	Nombre	Efecto
0	Pixel de contraste único	
1	Rayas horizontales	
2	Rayas verticales	
3	Ajedrezado	

Patrones para las mezclas (stipples) de colores)

Como antes describimos, se puede especificar un efecto de granulación para el color utilizando tres números. Por ejemplo:

```
INK color, contraste, granulación
```

que puede usarse con:

un color de la gama de 0 a 7
un contraste de la gama de 0 a 7
un tipo de grano de 0 a 3

PARAMETROS DEL COLOR

El mismo efecto puede conseguirse con un único número, pero no es tan fácil de realizar. Vea la *Guía de Referencia de Conceptos*, bajo el epígrafe, *Color.*

El programa siguiente visualizará todos los posibles efectos de color:

```
10 REMARK efectos de Color
20 FOR num = 0 TO 255
30   BLOCK 100, 50 40,50, num
40   PAUSE 50
50 END FOR num
```

PAPER

La instrucción **PAPER** seguida por uno, dos o tres números especifica el fondo. Por ejemplo:

```
PAPER 2        {rojo}
PAPER 2,4      {rojo/verde ajedrezado}
PAPER 2,4,1    {rojo/verde a rayas horizontales}
```

El color no será visible hasta que se hayan realizado algunas medidas, por ejemplo, hasta que no se haya limpiado la pantalla.

INK

El comando **INK** seguido de uno, dos o tres números, especifica el color para la impresión de los caracteres, las líneas u otro tipo cualquiera de dibujo gráfico. El color y los efectos de granulación son los mismos que los del comando **PAPER**, por ejemplo:

```
INK 2          {tinta roja}
INK 2,4        {tinta en ajedrez rojo/verde}
INK 2,4,1      {tinta en franjas horizontales rojo/verde}
```

La tinta se cambiará para todas las salidas siguientes al comando.

CLS

CLS significa limpiar la ventana con el color del fondo vigente en ese momento. Es un efecto muy parecido al de un profesor limpiando la pizarra, excepto que en este caso es electrónica y multicolor.

INTERMITENCIA

El color del texto, **INK** sólo puede tener intermitencia en el modo 0. Para conectar la intermitencia puede escribir:

```
FLASH 1
```

y para volver a conectar:

```
FLASH 0
```

Si hace que los caracteres con intermitencia se solapen se producirán efectos alarmantes.

ARCHIVOS

Usted habrá utilizado Microdrives para almacenar programas y habrá también utilizado los comandos **LOAD** y **SAVE**. Los cartuchos se pueden utilizar para almacenar tanto datos como programas. La palabra archivo significa normalmente una secuencia de registros de datos, siendo un registro un conjunto de información relacionada, como por ejemplo un nombre, una dirección y un número de teléfono.

Los dos tipos de archivo utilizados más ampliamente son los archivos en serie y los de acceso directo. En un archivo serie, los elementos se leen normalmente en secuencia, comenzando por el primero. Si desea obtener el registro número 50, deberá leer primero los primeros cuarenta y nueve registros, empezando por el primero. Por el contrario, en los registros de acceso directo, el registro número cincuenta se puede encontrar rápidamente, ya que el sistema no necesita trabajar sobre los cuarenta y nueve registros anteriores por llegar a él. La musica Pop de una cassette se asemeja a un archivo en serie, pero si tenemos un disco de larga duración con ocho piezas, tendremos un ejemplo de archivo de acceso directo. Es evidente que se puede colocar el brazo del tocadiscos en cualquiera de las ocho piezas.

El tipo de archivo más sencillo es simplemente una secuencia de números. Para ilustrar esta idea, colocaremos los números del 1 al 100 en un archivo llamado *números*. Sin embargo, el nombre de archivo completo está compuesto de dos partes:

```
nombre del archivo
información añadida
```

Suponga que deseamos crear el archivo números en un cartucho del Microdrive 1. El nombre del dispositivo es

```
mdv1_
```

y la información que se añade es el nombre del archivo

```
números
```

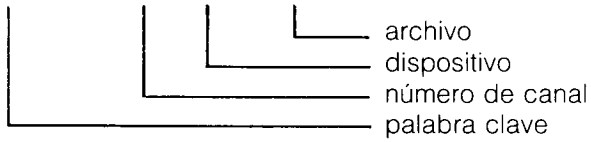
Por tanto, el nombre completo del archivo es:

```
mdv1_números
```

CANALES

Los programas pueden utilizar más de un archivo a un mismo tiempo, pero es más conveniente referirse a un archivo por el número de canal asociado. Este número puede ser cualquier entero de la gama del 0 al 15. Un archivo se asocia a un número de canal utilizando la instrucción **OPEN**, o, en el caso de que sea un arcivo nuevo, **OPEN_NEW**. Por ejemplo, puede elegir el canal 7 para el archivo números y escribir:

```
OPEN_NEW #7,mdv1_números
```



Puede referirse al archivo sin más que teclear #7. El programa completo es:

```
10 REMark Archivo sencillo
20 OPEN_NEW #7, mdv1_números
30 FOR número = 1 TO 100
40 PRINT #7, número
50 END FOR número
60 CLOSE #7
```

La instrucción **PRINT** hace que los números se "inmpriman" sobre el archivo, del cartucho, ya que le hemos asociado el canal número 7. La instrucción **CLOSE** #7 es necesaria porque el sistema tiene que realizar ciertas tareas internas cuando se ha utilizado el archivo. Además, se libera el canal 7 para otros posibles usos. Después de ejecutar el programa, teclee:

```
DIR mdv1_
```

y verá en el directorio que está incluido el archivo *números* en el cartucho del Microdrive, mdv1_.

También es necesario saber si el archivo es correcto, y esto sólo puede hacerse leyendo y comprobando su archivo. La palabra clave que necesitará utilizar es **OPEN_IN**, en caso contrario, el programa para lectura de los datos de un archivo es similar al anterior.

```
10 REMark Lectura de un archivo
20 OPEN_IN #6, mdv1_números
30 FOR elemento = 1 TO 100
40  INPUT #6, número
50  PRINT ! número !
60 END FOR elemento
70 CLOSE #6
```

El programa dará como salidas los números 1 al 100, pero sólo lo hará si el cartucho que contiene el archivo *números* sigue en el Microdrive mdv1_.

DISPOSITIVOS Y CANALES

Hemos visto un ejemplo de un dispositivo, un archivo de datos en un Microdrive. Podemos decir de modo informal que se ha abierto un archivo, pero su significado estricto es que se ha asociado un dispostivo a un determinado canal. Se puede disponer de más información posteriormente, si se considera necesario. Ciertos dispositivos tienen canales asociados a ellos permanentemente por el sistema

canal	uso
#0	SALIDA-Ventana de comandos ENTRADA-Teclado
#1	SALIDA-ventana de impresión
#2	LISTADO-salida para listados

VENTANAS

Se puede crear una ventana de cualquier tamaño en cualquier lugar de la pantalla. El nombre para el dispositivo ventana es

```
scr
```

y la información adicional, puede ser, por ejemplo:

```
scr_360×50a80×40
```

El programa siguiente crea una ventana con el número de canal 5, y la llena con el color rojo (código 2), para terminar cerrándola:

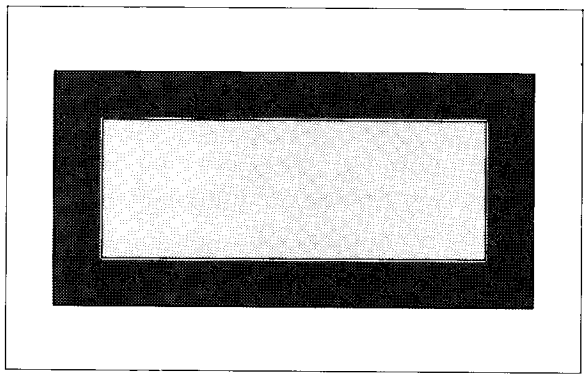
```
10 REMark Creacion de una ventana
20 OPEN #5, scr_400 ×200a20×50
30 PAPER #5,4 : CLS #5
40 CLOSE #5
```

Observe que cada ventana tiene sus propias características, como el fondo (paper), el color del texto (ink), etc. El hecho de abrir una ventana no implica que sea precisamente la ventana de omisión vigente en ese momento.

La forma y la posición de una ventana puede cambiarse sin necesidad de cerrarla y volverla a abrir. Intente añadir dos líneas al programa anterior:

```
34 WINDOW #5,300,150,40,60
36 PAPER #5,4 : CLS #5
```

Vuelva a ejecutar el programa y encontrará una ventana verde interior a la roja original. Esta ventana verde es la que ahora se encuentra asociada con el canal 5. Véase la figura.



BORDER

Se puede colocar un reborde rodeando el final de la pantalla o la ventana. Por ejemplo:

```
BORDER #5,6
```

creará un reborde alrededor de la ventana del canal #5. Tendrá una anchura de 6 unidades y el tamaño de la ventana se verá reducido como corresponde.

El reborde puede ser transparente para proteger cualquier cosa que se encuentre bajo él. Se puede especificar una reborde coloreado por el método usual:

```
BORDER #5,6,2
```

producirá un reborde rojo. Pueden hacerse rebordes de otros colores y texturas por los métodos usuales. Por ejemplo:

```
BORDER 10
```

añadirá un reborde transparente de 10 pixels de anchura a la ventana vigente en ese momento (transparente, porque no se ha especificado ningún color) y,

```
BORDER 2,0,7,0
```

añadirá un borde de 2 pixels de ancho con gránulos blancos y negros.

BLOQUES

Con una única instrucción se puede especificar el tamaño, la posición y color de un bloque. Se coloca en el sistema de coordenadas de pixels relacionado con la ventana o la pantalla vigente en ese momento. Por ejemplo:

```
BLOCK #5,10,20,50,100,2
```

creará un bloque en la ventana #5 en la posición 50 horizontal y 100 vertical. Tendrá una anchura de 10 unidades por una altura de 20. Su color será rojo.

Es importante notar que las instrucciones **WINDOW** y **BLOCK** funcionan sin ningún tipo de alteración en los dos modos, de 4 y 8 colores (aunque los colores pueden variar) ya que los valores en la dirección horizontal son siempre de una escala de 0 a 511 y siempre existen 256 posiciones de pixels en la dirección vertical.

IMPRESIONES ESPECIALES

CSIZE

Podemos alterar el tamaño de los caracteres. Por ejemplo:

```
CSIZE 3,1
```

nos dará los caracteres más largos disponibles

```
CSIZE 0,0
```

nos dará los menores. El primer número debe ser 0, 1, 2 ó 3 y determina la anchura. El segundo debe ser 0 ó 1 y determina la altura. Los tamaños normales son:

```
MODO 4 CSICE 0,0 (25 líneas de 84 caracteres)
MODO 8 CSIZE 1,0 (25 líneas de 42 caracteres)
```

El número de líneas y columnas disponibles para cada tamaño de carácter depende de si la salida se visualiza en un monitor o en una televisión. Los tamaños de fila y columna que se han dado son para un monitor. En el caso de la televisión, serán menores y también variarán de una televisión a otra.

Si está usted utilizando un modo de baja resolución, el QL no le permitirá elegir un tamaño de carácter menor al de omisión.

STRIP

Se puede crear un fondo especial para hacer que los caracteres resalten. Por ejemplo:

```
STRIP 7
```

dará una superficie blanca y

```
STRIP 2,4,2
```

dará una superficie rayada verticalmente en rojo y verde. Pueden utilizarse todas las combinaciones de colores normales disponibles.

OVER

Normalmente la impresión se realiza sobre el fondo que esté vigente en ese momento. Puede, sin embargo, alterarse dicho fondo utilizando strip.

Para realizar otros efectos puede usarse:

OVER 1	El 1 imprime en color ink sobre una franja transparente.
OVER −1	−1 imprime en color ink sobre la presentación que exista en ese momento en pantalla

Para volver a la impresión normal sobre la superficie de pantalla vigente en ese momento, utilice:

```
OVER 0
```

UNDER

Los caracteres se pueden subrayar.

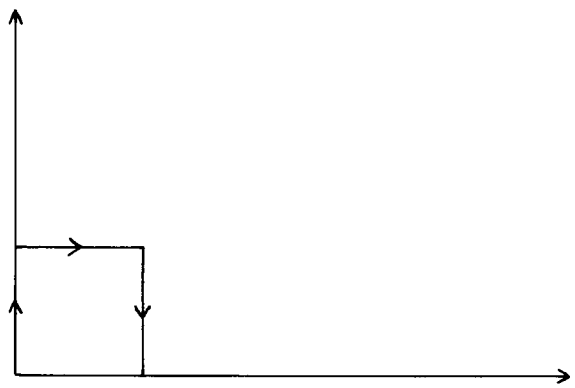
UNDER 1	subraya las salidas subsiguientes en el color ink (de impresión) vigente
UNDER 0	desconecta el subrayado

GRAFICOS A ESCALA

Si desea dibujar figuras geométricas razonablemente reales sobre una pantalla de vídeo o una televisión no es fácil utilizar el sistema basado en los pixels. El sistema realizará el trabajo necesario para asegurarle la posibilidad de dibujar fácil y razonablemente círculos cuadrados y otras formas.

La escala de omisión del sistema gráfico de coordenadas es 100 para la dirección vertical y la que sea necesaria para la dirección horizontal, de modo que las formas que se dibujen con las palabras claves especiales para ello (PLOT, DRAW, CIRCLE) tengan un aspecto real.

El **origen de los gráficos** no es el mismo que el de los pixels que utilizábamos para definir la posición de ventanas y bloques. El origen de los gráficos se encuentra en el ángulo inferior izquierdo de la ventana o pantalla vigente.



PUNTOS Y LINEAS

Si se utilizan los gráficos a escala es muy fácil dibujar puntos y líneas. Si se toma una escala vertical de 100, un punto que se encuentre cerca del centro de la ventana puede trazarse escribiendo:

```
POINT 60,50
```

El punto (60 unidades en horizontal y 50 en vertical) se escribirá en el color de escritura (ink) vigente en ese momento.

Del mismo modo, se puede dibujar una línea con la instrucción:

```
LINE 60,50 TO 80,90
```

Se pueden añadir elementos posteriormente. Por ejemplo, la siguiente instrucción dibuja un cuadrado:

```
LINE 60,50 TO 70,50 TO 70,60 TO 60,60 TO 60,50
```

MODO RELATIVO

Un par de coordenadas, como por ejemplo:

```
horizontal, vertical
```

definen normalmente un punto con respecto al origen, 0,0 que se encuentra en el ángulo inferior izquierdo de una ventana (o en otro lugar, si usted lo desea). Sin embargo, en algunas ocasiones es más conveniente definir puntos con respecto a la posición que tenga en ese momento el cursor. Por ejemplo, el cuadrado anterior puede dibujarse de otra forma utilizando la instrucción LINE_R, que significa:

> «Haz que todos los pares de coordenadas tengan sus valores con respecto a la posición del cursor.»

```
POINT 60,50
LINE_R 0,0 TO 10,0 TO 0,10 TO -10,0 TO 0,-10
```

En primer lugar el punto 60,50 se convierte en el origen. Por tanto, cuando se dibujan las líneas, sus finales se van convirtiendo en orígenes de la línea siguiente.

El programa siguiente dibuja un patrón de cuadrados coloreados, colocados aletoriamente:

```
10 REMark Cuadrados coloreados
20 PAPER 7 : CLS
30 FOR cd = 1 TO 100
40   INK RND (1 TO 6)
50   POINT RND(90), RND(90)
60   LINE_R 0,0 TO 10,0 TO 0,10 TO -10,0 TO 0,-10
70 END FOR cuadrados
```

Se puede obtener el mismo resultado sólo con gráficos absolutos, pero requerirá un esfuerzo superior.

CIRCULOS Y ELIPSES

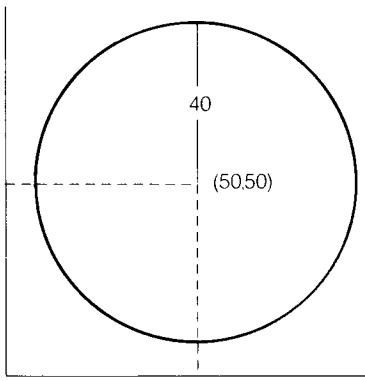
Si desea dibujar un círculo, necesita especificar:

una posición, por ejemplo 50,50
un radio, por ejemplo, 40

La instrucción:

```
CIRCLE 50,60,40
```

dibuja un círculo con su centro en la posición 50,50 y un radio (o altura) de 40 unidades, véase la figura:

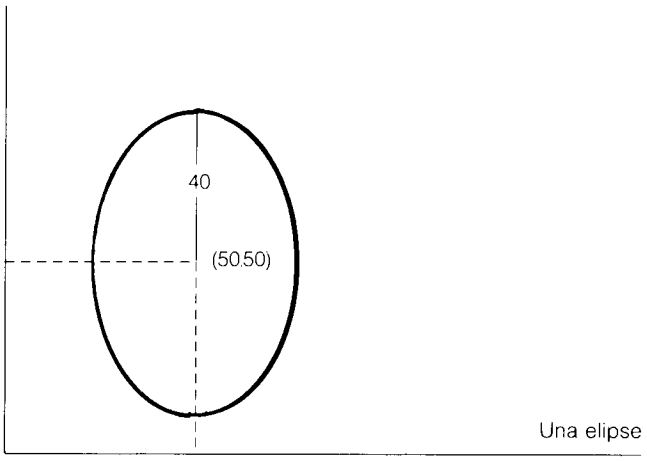


Un círculo

Si añadimos un cuarto parámetro:

```
CIRCLE 50,50,40,.5
```

obtendremos una elipse. Las palabras **CIRCLE** y **ELIPSE** son intercambiables.

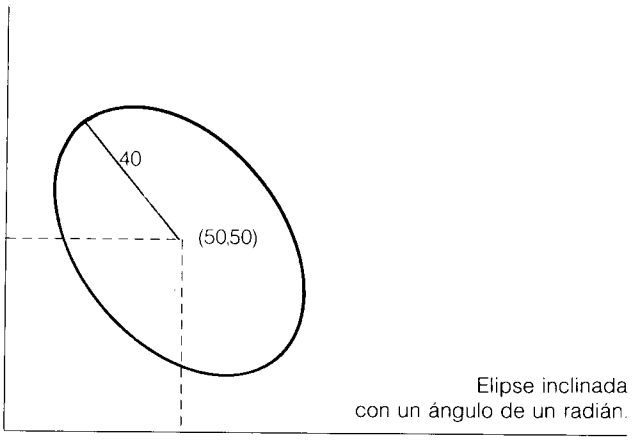


Una elipse

La altura de la elipse es 40, como antes, pero el radio horizontal es ahora 0.5 veces la altura. Este número 0.5 se llama excentricidad. Si la excentricidad es 1, la figura obtenida es un círculo, y si es diferente de 1 será una elipse, y si es 0 se obtendrá una línea. Si deseamos que la elipse esté inclinada, debemos añadir un quinto parámetro, por ejemplo:

```
CIRCLE 50,50,40,.5,1
```

y de este modo la elipse aparecerá inclinada en sentido inverso a las agujas de un reloj un radián (aproximadamente 57 grados) como se puede ver en la figura.



Elipse inclinada
con un ángulo de un radián.

Un ángulo recto tiene 180 grados o PI radianes, por tanto el programa siguiente realizará un patrón de elipses:

```
10 FOR rot = 0 TO 2*PI STEP PI/6
20   CIRCLE 50,50,40,0.5,rot
30 END FOR rot
```

El orden de los parámetros para un círculo o una elipse es el siguiente:

centro __ horizontal, centro__ vertical, altura [excentricidad, ángulo]

Los dos últimos parámetros son opcionales y lo indicamos colocándolos entre paréntesis cuadrados ([]).

Ejemplo Escriba un programa que realice lo siguiente:

1. Abra una ventana (en el 350,180)
2. Con una escala 100 en el modo 8
3. Seleccione fondo negro y la ventana clara.
4. Coloque un reborde rojo de dos unidades de anchura.
5. Dibuje un patrón de seis círculos coloreados.
6. Cierre la ventana.

```
10 REMark patrón
20 MODE 8
30 OPEN #7, scr_100×100 a 100×50
40 SCALE #7,100,0,0
50 PAPER #7,0 : CLS #7
60 BORDER #7,2,4
70 FOR color = 1 TO 6
80   INK #7, color
90   LET rot =2*PI/color
100  CIRCLE #7,50,50,30,0.5,rot
110 END FOR color
120 CLOSE #7
```

Alterando el programa se puede obtener efectos interesantes. Como ejemplo, pruebe las correcciones:

```
70 FOR color = 1 TO 100
80 LET rot = color*PI/50
```

ARCOS

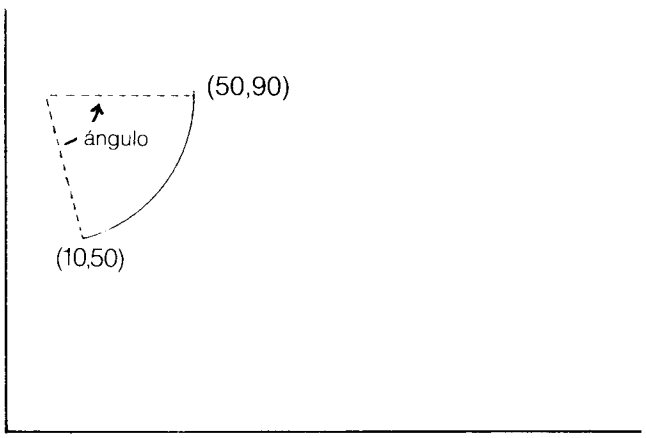
Si desea dibujar un arco necesita decidir:

punto de comienzo
punto final
curvatura

Los dos primeros elementos son fáciles, pero no ocurre lo mismo con la curvatura. Puede hacerse bien dibujándolo con exactitud.o mediante aproximaciones sucesivas, pero se debe decidir el ángulo que sustenta el arco y especificarlo en radianes. Un ángulo de 1.5 radianes dará una curvatura cerrada y un ángulo pequeño dará una curvatura adecuada. Pruebe por ejemplo:

```
ARC 10,50 TO 50,90, 1
```

que da una curvatura moderada con el color **INK** vigente en ese momento.



FILL

Para colorear (llenar) una figura cerrada con el color del texto **(INK)** basta con escribir:

```
FILL 1
```

antes de que se dibuje la figura. El programa listado a continuación produce un círculo verde.

```
INK 4
FILL 1
CIRCLE 50,50,30
```

El comando **FILL** funciona dibujando líneas horizontales entre los puntos adecuados.

La instrucción:

```
FILL 0
```

desconectará el efecto de coloreado.

DESPLAZAMIENTO DE PANTALLA VERTICAL (SCROLLING) Y HORIZONTAL (PANNING)

La presentación de una ventana puede desplazarse vertical u horizontalmente como si fuera un film que está impresionando un cameraman. Este desplazamiento se mide en pixels. Un número positivo de pixels indica un desplazamiento hacia arriba. Así pues,

```
SCROLL 10
```

Mueve la presentación que aparece en la ventana 10 pixels hacia arriba.

```
SCROLL -8
```

Mueve la presentación 8 pixels hacia abajo. Se puede añadir otro segundo parámetro para introducir un desplazamiento de parte de la pantalla.

```
SCROLL -8,1
```

desplazará la parte superior (sin incluir) la línea del cursor y:

```
SCROLL -8,2
```

desplazará la parte inferior (sin incluir) la línea del cursor.

Mientras se realiza el desplazamiento, el espacio que deja libre el movimiento se llena con el color de fondo (paper). Un segundo parámetro, 0 no tiene efecto.

Para desplazar horizontalmente, **PAN,** la ventana vigente hacia la izquierda o la derecha se utiliza la instrucción **PAN.** Esta instrucción funciona de forma similar a la instrucción **SCROLL** pero

PAN 40 mueve la presentación hacia la derecha
PAN -40 mueve la presentación hacia la izquierda

Un segundo parámetro dará un desplazamiento parcial:

0 la pantalla completa
3 la totalidad de la línea ocupada por el cursor
4 la parte derecha de la línea ocupada por el cursor. La zona del cursor también está incluida.

Si se están utilizando mezcla de colores (stipples) o se encuentra en el modo de color 8, las ventanas deben desplazarse vertical u horizontalmente por múltiplos de 2 pixels.

PROBLEMAS SOBRE EL CAPITULO 12

1. Escriba un programa que dibuje una rejilla de "serpientes y escaleras" con diez filas de diez cuadrados.
2. Coloque los números que van del 1 al 100 en los cuadrados comenzando por el ángulo inferior izquierdo y terminando con una F de fin en el último cuadrado.
3. Dibuje una diana sobre la pantalla. Debe consistir en un anillo exterior que pueda llevar números. Debe contener otro anillo de "dobles" y otro de "triples" y un centro rodeado por otro anillo.

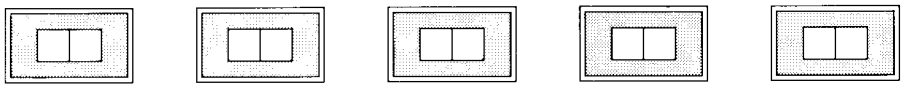
CAPITULO 13
MATRICES
EL PORQUE DE LAS MATRICES

Suponga que es usted el gobernador de una prisión y tiene un bloque nuevo, al que llaman Bloque Oeste. Está preparado para albergar 50 nuevos presos. Usted debe saber qué preso (designándolo por su número) está en cada celda. Puede dar un nombre a cada celda, pero es más sencillo dar los números desde el 1 al 50.

En una simulación por ordenador imaginaremos sólo 5 presos cuyos números se pueden poner en una instrucción **DATA**.

```
DATA 50, 37, 86, 41, 32
```

Establecemos una matriz de variables que comparten el mismo nombre, oeste, y se distinguen por el número que les sigue entre paréntesis.



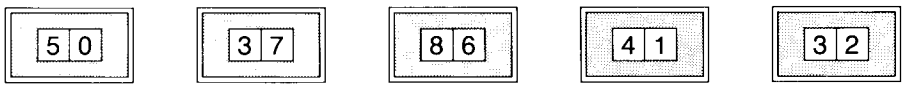
Necesitaremos declarar una matriz y dar sus dimensiones mediante una instrucción **DIM**.

```
DIM oeste(5)
```

Esto permite que el SuperBASIC disponga un espacio que puede ser muy grande. Después de haber ejecutado la instrucción **DIM** pueden utilizarse las cinco variables.

```
FOR celda = 1 TO 5 : READ oeste(celda)
```

Podemos añadir otro bucle **FOR** con una instrucción **PRINT** para probar que los convictos se encuentran en las celdas



El programa completo se muestra más abajo:

```
10 REMark Prisioneros
20 DIM oeste(5)
30 FOR celda =1 TO 5 : READ oeste(celda)
40 FOR celda = 1 TO 5 : PRINT celda! oeste(celda)
50 DATA 50, 37, 86, 41, 32
```

La salida del programa es

```
1  50
2  37
3  86
4  41
5  32
```

Los números del 1 al 5 se llaman *índices* del nombre de la matriz *oeste*. Esta matriz, *oeste* es una matriz numérica que tiene cinco elementos matriciales.

La línea 40 se puede sustituir por:

```
40 PRINT oeste
```

Con ello se obtienen sólo los valores de salida:

```
0
50
37
86
41
32
```

El 0 aparece encabezando la lista porque los subíndices van desde cero hasta el número que se declara. Más adelante explicaremos lo útiles que pueden ser los elementos cero en las matrices.

Observe también que cuando se DIMensiona una matriz numérica, se da un valor cero a todos sus elementos.

MATRICES DE CADENA

Las matrices de cadena son similares a las matrices numéricas, pero llevan además otra dimensión en la instrucción **DIM** que especifica la longitud de cada variable de cadena en la matriz. Suponga que diez de los mejores jugadores del campeonato de Golf Open de España 1984 de El Saler pueden colocar sus nombres en instrucciones **DATA**:

```
DATA "Tom", "Graham", "Seve", "Jack", "Lee"
DATA "Nick", "Bernard", "Ben", "Gregg", "Hal"
```

Se necesitarían diez variables diferentes, pero si tuviera cien o mil jugadores el trabajo sería terriblemente aburrido. Una matriz es un conjunto de variables que se diseñan para resolver este tipo de problemas. Cada nombre de variable tiene dos partes:

un nombre que debe cumplir las normas generales
una parte numérica llamada subíndice

Escriba del siguiente modo las variables

```
piso$(1), piso$(2), piso$(3)... etc.
```

Antes de que se utilicen las variables de la matriz se debe informar al sistema sobre el tipo de matriz y sus dimensiones.

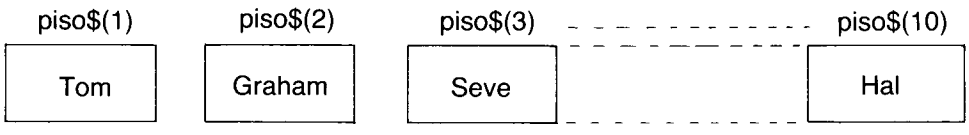
```
DIM piso$(10,8)
```

Con ello habremos reservado diez variables para utilizarlas en el programa. Cada variable de cadena de la matriz puede tener hasta ocho caracteres. Las instrucciones **DIM** deben colocarse normalmente todas juntas cerca del comienzo del programa. Una vez declarada la matriz en la instrucción **DIM** se podrán utilizar todos sus elementos. Una ventaja importante es que la parte numérica (el subíndice) se puede dar como variable numérica.

Puede por tanto escribirse:

```
FOR número = 1 TO 10 : READ piso$(número)
```

Con ello colocaremos a los golfistas en sus "pisos"



Podemos referirnos a la variable de forma usual pero es importante recordar que debe utilizarse el índice adecuado. Suponga que Tom y Seve desean cambiar sus apartamentos. En términos de programación, uno de ellos, por

ejemplo Tom, tendrá que trasladarse a un piso temporalmente, para dar tiempo al traslado de Seve. Podemos pues escribir:

```
LET tem$ = piso$(1); REMark Tom a temporal
LET piso$(1) = piso$(3); REMark Seve a piso$(3)
LET piso$(3) = temp$; REMark Tom a flat$(3)
```

El programa siguiente coloca a los diez golfistas en una matriz llamada piso$ e imprime los nombres de sus ocupantes con sus números de piso (Indices de la matriz) para probar que están alojados. Los ocupantes de los pisos 1 y 3 cambian ahora sus residencias. Debemos pues imprimir otra lista de ocupantes para mostrar que se ha cumplido el cambio de apartamento.

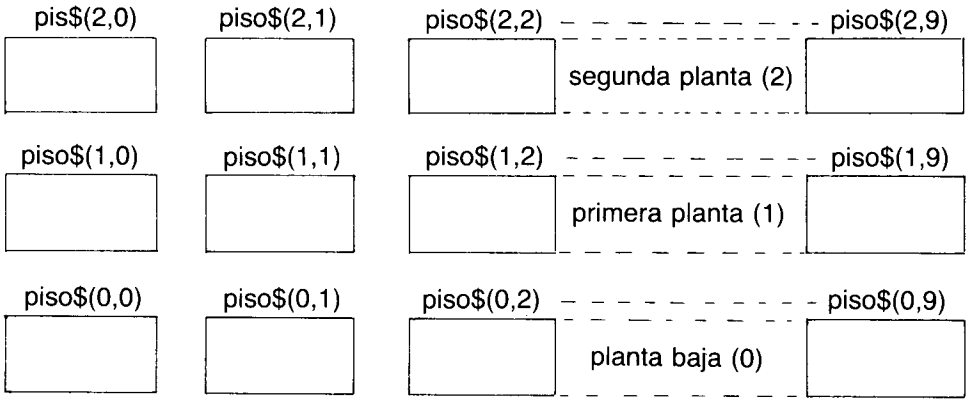
```
 10 REMark Pisos para los Golfistas
 20 DIM piso$(10,8)
 30 FOR número = 1 TO 10 READ piso$(número)
 40 lista
 50 cambio
 60 lista
 70 REMark Fin del programa principal
 80 DEFine PROCedure lista
 90   FOR num = 1 TO 10 : PRINT num, piso$(num)
100 END DEFine
110 DEFine PROCedure cambio
120     LET tem$ = piso$(1)
130     LET piso$(1) = piso$(3)
140     LET piso$(3) = temp$
150 END DEFine
160 DATA "Tom", "Graham", "Seve", "Jack", "Lee"
170 DATA "Nick", "Bernard", "Ben", "Gregg", "Hal"
```

Salida (línea 40)	Salida (línea 60)
1 Tom	1 Seven
2 Graham	2 Graham
3 Seven	3 Tom
4 Jack	4 Jack
5 Lee	5 Lee
6 Nick	6 Nick
7 Bernard	7 Bernard
8 Ben	8 Ben
9 Gregg	9 Gregg
10 Hal	10 Hal

MATRICES BIDIMENSIONALES

Algunas veces la naturaleza del programa aconseja dos dimensiones, como por ejemplo 3 plantas de 10 pisos en lugar de una sola fila de 30.

Suponga que 20 golfistas, o más, necesitan piso y que disponemos de un bloque de 30 pisos divididos en tres plantas de diez pisos cada una. Un modo muy real de representar el bloque es utilizar una matriz de dos dimensiones. Imaginaremos las treinta variables como se muestran más abajo:



Suponiendo una instrucción **DATA** con los 30 nombres, una forma adecuada de colocar cada nombre en su piso será:

```
30 FOR planta = 0 TO 2
40   FOR num = 0 TO 9
50     READ piso$(planta, num)
60   END FOR num
70 END FOR planta
```

También necesitará una instrucción **DIM**

```
20 DIM piso$(2,9,8)
```

que muestre que los primeros índices van desde 0 a 2 (número de planta) y el segundo índice va desde 0 a 9 (número de habitación). El tercer número establece el máximo número de caracteres en cada elemento de la matriz.

Añadimos una rutina de impresión para mostrar que los golfistas se encuentran en sus respectivos pisos. Vamos ahora a usar letras para ahorrar espacio:

```
 10 REMark 30 Golfista
 20 DIM piso$(2,9,8)
 30 FOR planta = 0 TO 2
 40   FOR num = 0 TO 9
 50     READ piso$(planta, num,) : REMark Golfista alojado
 60   END FOR num.
 70 END FOR planta
 80 REMark Fin de la entrada
 90 FOR planta = 0 TO 2
100   PRINT "Número de planta" ! planta
110   FOR num = 0 TO 9
120     PRINT "Piso" ! num ! piso$(planta,num)
130   END FOR num
140 END FOR planta
150 DATA "A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
160 DATA "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T",
170 DATA "U", "V", "W", "X", "Y", "Z", "*", "&", "$", "%"
```

salida E

La salida comienza:

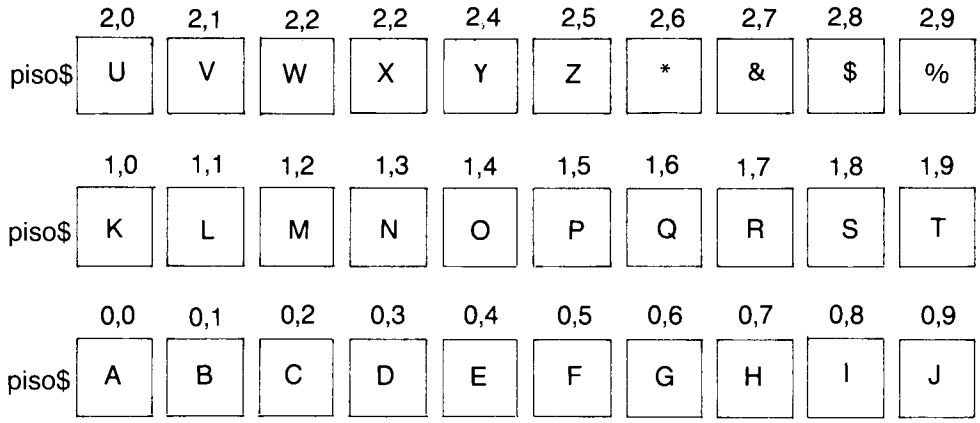
```
Planta número 1
Piso 1 A
Piso 2 B
Piso 3 C
```

y continúa asignando piso a los restantes ocupantes.

CORTES (SLICES) EN LAS MATRICES

Esta sección puede parecerle de difícil lectura, pero esencialmente discute el mismo concepto que el corte de cadenas. Probablemente necesite cortar cadenas cuando avance en el aprendizaje de la programación. La necesidad de cortar matrices es mucho menos frecuente, y puede decidir omitir esta sección, especialmente en una primera lectura.

Simplificaremos ahora el problema de los pisos para golfistas para poder ilustrar el concepto de corte de matrices. Los pisos se numerarán del 0 al 9 para mantenernos con dígitos simples y los nombres serán caracteres únicos, por razones de espacio.



Con los valores anteriores, los cortes en las cadenas será los siguientes:

piso$(1,3)	que significa un elemento único de la matriz con un valor N.
piso#(1,1 TO 6)	que significa seis elementos con valores L M N O P Q.

Elemento de la matriz	Valor
piso$(1,1)	L
piso$(1,2)	M
piso$(1,3)	N
piso$(1,4)	O
piso$(1,5)	P
piso$(1,6)	Q

piso$(1)	significa piso$ (1,0 a 9) diez elementos con valores K L M N O P Q R S T

En estos ejemplos se puede dar el rango de los valores de un subíndice en lugar de dar un valor simple. Si falta completamente el índice se supone el rango completo. En el tercer ejemplo falta el segundo índice, y el sistema supone que es de **0 a 9.**

Las técnicas de corte de cadenas y corte de matrices son similares, aunque estas últimas tienen unas aplicaciones más amplias.

PROBLEMAS SOBRE EL CAPITULO 13

1. CLASIFICACION

   Coloque diez números en una matriz leyéndolos desde una instrucción **DATA.** Busque en la matriz el número más bajo. Haga de este número el primer valor de un nueva matriz. Sustitúyalo en la primera matriz por un número muy grande. Repita este proceso haciendo del segundo valor más pequeño el segundo valor de la nueva matriz y así sucesivamente hasta que haya clasificado la matriz de números que deben imprimirse a continuación.

2. SERPIENTES Y ESCALERAS

   Represente el juego de las escaleras y las serpientes con una matriz numérica de 100 elementos. Cada elemento debe contener:

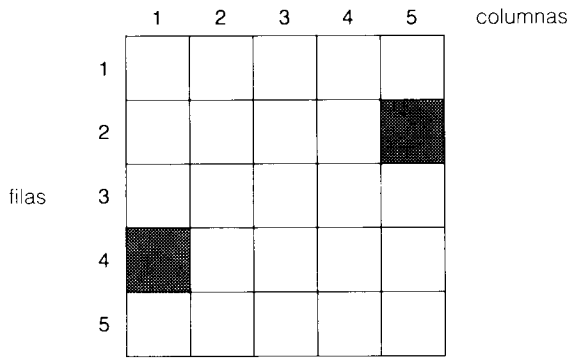
   cero

o un número en una gama del 10 al 90 que significa el número que debe obtener un jugador para "subir una escalera" o "bajar la serpiente"

o los dígitos 1, 2, 3, etc., para indicar la posición de un determinado jugador.

Coloque seis escaleras y seis serpientes introduciendo números en la matriz y simulando una "única" tirada por un único jugador para probar el juego.

3. CRUCIGRAMA CON BLANCOS



Los crucigramas normalmente tienen un número primo de filas y columnas y los cuadrados negros forman un patrón simétrico. Se dice que el patrón tiene simetría rotacional cuando una rotación de 180 grados no lo cambia en absoluto.

Observe que después de la rotación de 180 grados el cuadrado de la fila 4, columna 1, se convierte en el cuadrado de la fila 2, columna 5. Esto es, fila 4, columna 1, se convierten en la fila 6-4 columna 6-1 en una cuadrícula de 5 × 5.

Escriba un programa que genere y visualice un patrón simétrico de este tipo.

4. Modifique el patrón del crucigrama para que no haya secuencias en ninguna de las dos direcciones que sean menores de cuatro cuadrados blancos.

5. BARAJADO DE CARTAS

Las cartas se marcan con los números del 1 al 52 y se guardan en una matriz. Cuando sea necesario pueden convertirse fácilmente en las cartas normales. Para "barajarlas" haremos lo siguiente:

Elija una posición en la gama del 1 al 51, ejemplo 17.

Coloque la carta de esta posición en un lugar de almacenamiento temporal.

Cambie todas las cartas de las posiciones 52 a la 18 a las nuevas posiciones 51 a la 17.

Coloque la carta elegida (y que estaba almacenada temporalmente), en la posición 52.

Repita el mismo proceso para las gamas siguientes, 1-50, 1-49..., hasta la 1-2 de forma que el paquete esté bien barajado.

Obtenga el resultado de este barajado.

6. Establezca seis instrucciones **DATA** cada una de las cuales debe contener un apellido, iniciales y un número de teléfono (código de provincia y número local). Decida la estructura que deben tener las matrices que almacenan esta información y léala (**READ**) dentro de las matrices.

   Imprima (**PRINT**) los datos utilizando un bucle **FOR** separado, y explique cómo no son necesariamente iguales tanto el formato (**DATA**) de entrada, como el interno (matrices) y el formato de salida.

CAPITULO 14
ESTRUCTURA DE LOS PROGRAMAS

En este capítulo vamos a meternos de nuevo en el campo de la estructura de programas: bucles y decisiones, o selecciones. Hemos tratado de presentar las cosas de la forma más sencilla posible, pero el lenguaje SuperBASIC se ha diseñado para trabajar adecuadamente con los problemas simples así como con los complejos y en todos los niveles intermedios. Algunas zonas de este capítulo son difíciles, y si usted es nuevo en la programación puede serle útil omitir ciertas áreas. Los temas cubiertos son los siguientes:

Bucles
Bucles anidados
Decisiones binarias
Decisiones múltiples

Las últimas partes de la primera sección, Bucles, son difíciles porque se estudia la forma en la que el SuperBASIC resuelve problemas que otros lenguajes simplemente ignoran. Salte esas partes si lo considera necesario, pero las otras secciones son más comprensibles.

BUCLES

En esta sección intentamos ilustrar los problemas bien conocidos del manejo de las repeticiones simulando escenas del Salvaje Oeste. El tema puede parecer trivial, pero ofrece una base sencilla de discusión e ilustra las dificultades que surgen en la amplia gama de las aplicaciones de programación.

EJEMPLO 1

Un bandido es herido cerca de la Vieja Escuela. El sheriff tiene seis balas en su pistola. Simule el disparo de los seis tiros.

Programa 1

```
10 REMARK FOR del Oeste
20 FOR balas = 1 TO 6
30   PRINT "Apunta"
40   PRINT "Dispara"
50 END FOR balas
```

Programa 2

```
10 REMark REPeat del Oeste
20 LET balas = 6
30 REPeat bandido
40   PRINT "Apunta"
50   PRINT "dispara"
60   LET balas = balas - 1
70   IF balas = 0 THEN EXIT bandido
80 END REPeat bandido
```

Los dos programas anteriores producen la misma salida:

```
Apunta
Dispara
```

que se imprime seis veces.

En cada programa, el 6 se puede cambiar a cualquier número, bajando hasta 1, y ambos programas siguen funcionando como se esperaba. ¿Qué ocurriría, sin embargo, si la pistola está vacía antes de disparar?

EJEMPLO 2

Suponga que alguien sacó secretamente todas las balas de la pistola del sheriff. ¿Qué ocurriría si sencillamente se cambia el 6 por un 0 en cada programa?

Programa 1

```
10 REMark For del Oeste para Cero
20 FOR balas = 1 to 0
30   REPeat bandido
40   PRINT "Apunta"
50 END FOR balas
```

Funciona correctamente, no hay salida. El "caso cero" funciona correctamente en SuperBASIC.

Programa 2

```
10 REMark REPeat del Oeste con fallo
20 LET balas = 0
30 REPeat bandido
40   PRINT "Apunta"
50   PRINT "Dispara"
60   LET balas = balas - 1
70   IF balas = 0 THEN EXIT bandido
80 END REPeat bandido
```

El programa falla en dos cosas:

1. Apunta
   Dispara

   se imprime aunque no haya balas en la pistola.

2. Cuando se comprueba la variable *balas* en la línea 70, ésta tiene el valor 1, y a continuación no se hace ya cero. El programa sigue en el bucle indefinidamente. Para arreglar esta situación reescriba la línea 60.

   ```
   70 IF balas < 1 THEN EXIT bandido
   ```

Hay una falta inherente en la programación que no permite la posibilidad de tener un caso cero. Esto podemos corregirlo colocando la salida condicional **EXIT** antes de la instrucción **PRINT**.

Programa 3

```
10 REMark REPeat del Oeste, caso Cero
20 LET balas = 0
30 REPeat Bandido
40   IF balas = 0 THEN EXIT Bandido
50   PRINT "Apunta"
60   PRINT "Dispara"
70   LET balas = balas -1
80 END REPeat bandido
```

Este programa funciona ahora adecuadamente para cualquier valor inicial de balas, con tal de que el valor sea positivo, entero o cero. El Método 2 corresponde al bucle **REPEAT... UNTIL** de algunos lenguajes. El Método 3 corresponde al bucle **WHILE... ENDWHILE** de algunos lenguajes. Sin embargo, el bucle **REPeat... END REPeat** con **EXIT** es más flexible que la combinación de ambos.

Si usted ya ha usado otros BASICs, se preguntará qué ha pasado con la instrucción **NEXT**. La reintroduciremos pronto, pero se verá que ambos bucles tienen una estructura similar y ambos tienen nombre.

FOR *nombre* =	(palabra clave de apertura)	**REPeat** *nombre*
(instrucción)	(contenido)	(instrucción)
END FOR nombre	(palabra clave de cierre)	**END REPeat** *nombre*

Además, el bucle **REPeat** debe tener normalmente entre las instrucciones una salida **EXIT**, o nunca terminará.

Observe también que la instrucción **EXIT** hace que el control se dirija inmediatamente después del final (**END**) del bucle.

Las instrucciones **NEXT** pueden colocarse en un bucle, reexpidiendo el control justo después de la palabra clave de apertura, **FOR** o **REPeat**. Debe considerarse en cierto modo como opuesta a una instrucción **EXIT**. Por una curiosa coin-

cidencia, las dos palabras **NEXT** y **EXIT**, contienen **EXT**. Piense en una extensión a los bucles y:

N significa "Nuevo comienzo"
I de la palabra fin

EJEMPLO 3

La situación es la misma que en el ejemplo 1. El sheriff tiene una pistola cargada con seis balas y está dispuesto a disparar al bandido, pero deben cumplirse además dos condiciones:

1. Si hiere al bandido, parará de disparar, y volverá a Dodge City
2. Si se queda sin balas antes de herir al bandido, le dirá a su compañero que vigile al bandido mientras él (el sheriff) vuelve a Dodge City.

Programa 1

```
10 REMark For del Oeste con Epílogo
20 FOR balas = 1 TO 6
30   PRINT "Apunta"
40   PRINT "Dispara"
50   LET herido = RND(9)
60   IF herido = 7 THEN EXIT balas
70 NEXT balas
80   PRINT "Vigila al bandido"
90 END FOR balas
100 PRINT "Vuelta a Dodge City"
```

En este caso, el contenido entre **NEXT** y **END FOR** es un tipo de epílogo que sólo se ejecuta si el bucle **FOR** se ejecuta completamente. Si tiene lugar una salida **EXIT** prematura, el epílogo no se ejecuta.

Se puede conseguir el mismo efecto con un bucle **REPeat** aunque no es necesariamente la mejor manera de hacerlo. Sin embargo, puede ser útil estudiarlo (quizás en una segunda lectura) si desea entender las estructuras que son tan sencillas que pueden utilizarse con facilidad y sin embargo son suficientemente potentes como para solucionar situaciones difíciles, cuando éstas surgen.

Programa 2

```
10 REMark REPeat del Oeste con Epílogo
20 LET balas = 6
30 REPeat Bandido
40 PRINT "Apunta"
50   PRINT "Dispara"
60   LET herido = RND(9)
70   IF herido = 7 THEN EXIT Bandido
80   LET balas = -1
90   IF balas <>0 THEN NEXT Bandido
100   PRINT "Vigila al bandido"
110 END REPeat Bandido
120 PRINT "Vuelve a Dodge City"
```

El programa funciona adecuadamente mientras el sheriff tenga al menos una bala al comienzo. No funciona si la línea 20 dice:

```
20 LET balas =0
```

Usted puede pensar que el sheriff es un loco si empieza una empresa de ese tipo sin una sola bala, y tendría razón. Nosostros estamos discutiendo ahora como preservar la buena estructura del programa en las situaciones más complejas. Al menos hemos mantenido la sencillez del contexto y sabemos lo que intentamos obtener. Los problemas estructurales complejos aparecen en situaciones más difíciles que las simulaciones del Salvaje Oeste. Si realmente usted desea una solución al problema que aparece si se considera una posible herida, falta de balas o un epílogo, además del caso cero añada la línea siguiente al programa anterior:

```
35 IF balas = 0 THEN PRINT "Vigila al bandido": EXIT bandido
```

No podemos concebir un problema más complejo que el anterior, con un bucle simple. El SuperBASIC puede solucionarlo, si usted realmente lo desea.

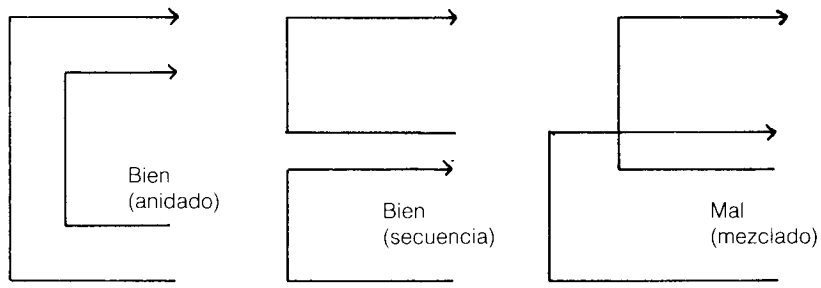
BUCLES ANIDADOS

Considere el siguiente bucle **FOR** que dibuja, **PLOT**, una fila de puntos de varios colores elegidos aleatoriamente (no negros).

```
10 REMark Fila de pixels
20 PAPER 0 : CLS
30 LET vert = 50
40 FOR horz = 20 TO 60
50   INK RND (2 TO 7)
60   POINT horz, vert
70 END FOR
```

Este programa dibuja una fila de puntos como el dibujo siguiente:

Si usted desea, por ejemplo 51 filas de puntos, debe trazar **(PLOT)** una fila por cada valor de vert desde 30 a 80. Sin embargo, siempre debe cumplir la regla de que una estructura debe estar enteramente en el interior de otra, o debe rodearla. También puede seguirla en secuencia, pero no puede "mezclarse" con otra estructura. Muchos libros de programación suelen mostrar con frecuencia como se relacionan los bucles **FOR** utilizando diagramas como los que siguen a continuación.



En el SuperBASIC la regla se aplica a todas las estructuras. Usted podrá resolver todos los problemas utilizándolas adecuadamente. Vamos, por tanto, a tratar el bucle **FOR** como una entidad diseñando un nuevo programa:

```
FOR vert = 30 TO 80

FOR horz = 20 TO 60
INK RND (2 TO 7)
    POINT horz, vert
END FOR horz

END FOR vert
```

Cuando llevemos esto a un programa, estaremos seguros no sólo de que funciona, sino de lo que realmente hace: dibuja un rectángulo realizado con filas de pixels.

```
10 REMark Filas de pixels
20 PAPER 0 : CLS
30 FOR vert = 30 TO 80
50   FOR horz = 20 TO 60
60     INK RND(2 TO 7)
70     POINT horz, vert
80   END FOR horz
90 END FOR vert
```

Se pueden anidar estructuras diferentes. Suponga que sustituimos el bucle interior **FOR** del programa anterior por un bucle **REPeat** . Terminaremos el bu-

cle **REPeat** cuando el código de color cero aparezca en alguna selección dentro de la gama del 0 al 7.

```
10 REMark REPeat dentro de FOR
20 PAPER 0 : CLS
30 FOR vert = 30 TO 80
40   LET hrz = 19
50   REPeat puntos
60     LET color = RND(7)
70     INK color
80     LET hrz =hrz + 1
90     POINT hrz, vert
100 IF color = 0 then EXIT puntos
110   END REPeat puntos
120 END FOR vert
```

En dos únicas reglas se puede expresar mucha sabiduría sobre el control de programas y estructuras:

1. Construya su programa utilizando sólo las estructuras de bucles y toma de decisiones válidas.
2. Debe relacionar adecuadamente cada estructura bien en secuencia o bien enteramente en el interior de otra.

DECISIONES BINARIAS

Los tres tipos de decisiones binarias se pueden ilustrar fácilmente considerando por ejemplo lo que debe hacerse cuando llueve.

i.
```
10 REM Forma corta de IF
20 LET lluvia = RND(0 TO 1)
30 IF lluvia THEN PRINT "Abre el paraguas"
```

ii.
```
10 REMark Forma larga de IF ...END IF
20 LET lluvia = RND(0 TO 1)
30 IF lluvia THEN
40   PRINT "ponte abrigado"
50   PRINT "abre el paraguas"
60   PRINT "Anda deprisa"
70 END IF
```

iii.
```
10 REMark Forma larga de IF...ELSE...END IF
20 LET lluvia = RND (0 TO 1)
30 IF lluvia then
40   PRINT "Toma el autobús"
50 ELSE
60   PRINT "Pasea"
70 END IF
```

Todas ellas son decisiones binarias. Los primeros dos ejemplos son sencillos: o bien hay algo que ocurre, o no ocurre. El tercero es una decisión binaria general con dos posibles acciones, que deben estar definidas.

Si se desea puede omitirse **THEN** en las formas largas. En las formas cortas pueden sustituirse los dos puntos : por **THEN**

EJEMPLO

Considere un ejemplo más complejo en el que parezca natural anidar decisiones binarias. Este tipo de nido puede resultar confuso y sólo se debe hacer si es lo mejor que puede hacerse. Es muy importante cuidar la presentación, especialmente los diferentes márgenes del programa.

Analice una parte de un determinado texto para contar el número de vocales, consonantes y otros caracteres. Ignore los espacios. Para mayor simplicidad, el texto está todo él en mayúsculas.

Data

```
"EN 1984 SE REALIZO LA HISTORIA DE LOS ORDENADORES"
```

Designación

```
Lea los datos
  FOR cada carácter
    IF letra THEN
      IF vocal
        incrementa la cuenta de vocales
      ELSE
        incrementa la cuenta de consonantes
      END IF
    ELSE
      IF no hay espacios, incrementa otra cuenta
    END IF
END FOR
PRINT resultados
```

Programa

```
100 REMark Cuenta de caracteres
110 RESTORE 290
120 READ texto$
130 LET vocales = 0 : cons = 0 : otros = 0
140 FOR num = 1 TO LEN(texto$)
150   LET cr$ = texto$(num)
160   IF cr$ >= "A" AND cr$< = "Z"
170     IF cr$ INSTR "AEIOU"
180       LEt vocales = vocales + 1
190     ELSE
200       LET cons = cons + 1
210     END IF
220   ELSE
230     IF cr$ < > "" THEN otros = otros + 1
240   END IF
250 END FOR num
260 PRINT "La cuenta de vocales da" ! vocales
270 PRINT "La cuenta de consonantes da" ! cons
280 PRINT "La otra cuenta da" ! otros
290 DATA "LA HISTORIA DE LOS ORDENADORES SE HIZO EN 1984"
```

Salida

```
La cuenta de vocales da 16
La cuenta de consonantes da 18
La otra cuenta da 4
```

DECISIONES MULTIPLES SELect

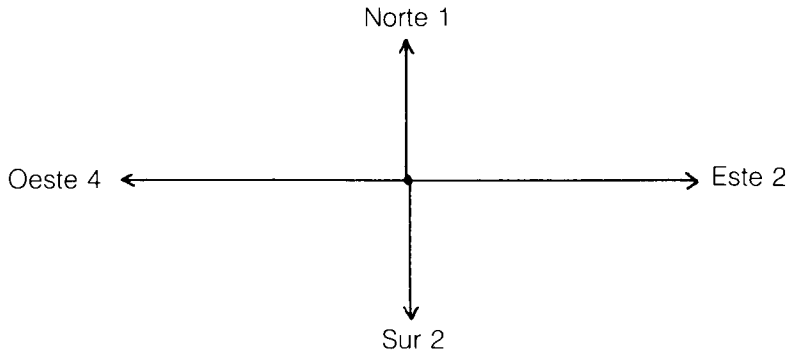
Si se pueden realizar tres o más posibles acciones, y ninguna de ellas depende de la elección anterior, la estructura natural que debe utilizarse es **SELect**, queda la opción de elegir entre un número cualquiera de posibilidades.

EJEMPLO

Una serpiente mágica puede crecer sin límite añadiendo sectores a su parte delantera. Cada sector puede tener una longitud de hasta veinte unidades, y puede ser del mismo color o diferente color que el resto de la serpiente. Cada nuevo sector puede crecer en alguna de las direcciones, norte, sur, este y oeste. La serpiente comienza en el centro de la ventana.

Método

En cualquier momento mientras la serpiente continúa en la pantalla es fácil elegir la longitud y el color del dibujo (ink). La dirección puede elegirse mediante un número 1, 2, 3 ó 4 como se muestra:



Diseño

Select **PAPER**
Establece la serpiente en el centro de la ventana
REPeat
  Selecciona la dirección color, longitud de crecimiento
  FOR unid = 1 TO longitud
    Hace crecer la serpiente norte sur, este u oeste
    Si la serpiente se sale de la ventana **THEN EXIT**
  END REPeat
PRINT final del mensaje

Programa

```
 10 REMark Serpiente Mágica
 20 PAPER 0 : CLS
 30 LET horz = 50 : vert = 50
 40 REPeat serpiente
 50   LET direc = RND (1 TO 4) : color = RND (2 TO 7)
 60   LET crec = RND (2 TO 20)
 70   INK color
 80   FOR unid = 1 TO crec
 90     SELect ON DIREC
100       ON direc = 1
110         LET vert = vert + 1
120       ON direc = 2
130         LET horz = horz + 1
140       ON direc = 3
150         LET vert = vert - 1
160       ON direct = 4
170         LET horz = horz - 1
180     END SELect
190     IF horz<1 OR horz >99 OR vert<1 OR vert>99 THEN EXIT
serpiente
200     POINT horz, vert
210   END FOR unid
220 END REPeat serpiente
230 PRINT "La serpiente se desborda"
```

La sintaxis de la estructura **SELect ON** permite siempre seleccionar de entre una lista de valores, como por ejemplo:

```
5,6,8,10 TO 13
```

También se puede ejecutar una acción si no se encuentra ninguno de los valores dados. La estructura completa se muestra en la forma que se explica más abajo.

FORMA LARGA

SELect ON num
ON num = lista de valores
  instrucciones
ON num = lista de valores
  instrucciones
  —
  —

```
ON num = REMAINDER
  instrucciones
END SELect
```

donde num es cualquier variable numérica, y la cláusula **REMAINDER** es opcional.

FORMA CORTA

Tenemos una forma corta de la estructura **SELect**. Por ejemplo:

```
10 INPUT num
20 SELect ON num = 0 TO 9 : PRINT "dígito"
```

que funcionará como usted esperaba.

PROBLEMAS SOBRE EL CAPITULO 14

1. Almacene diez números en una matriz y realice una clasificación "burbuja". Esta clasificación se realiza comparando el primer par e intercambiándolo si es necesario, y continuando así con el segundo par (segundo y tercer números) hasta el noveno par (noveno y décimo números). La primera pasada de comparaciones y posibles cambios garantiza que el número más alto alcance su posición correcta. Otras ocho pasadas garantizarán otras ocho posiciones correctas, dejando sólo el número más bajo que debe, pues estar en la única posición (correcta) a la izquierda. La forma más simple de clasificación "burbuja" de diez números necesita nueve pasadas de nueve comparaciones.

2. Piense en algún sistema para acelerar la clasificación tipo "burbuja", pero no espere nunca demasiado eficacia.

3. Un subastador desea vender un viejo reloj y tiene instrucciones de comenzar su subasta con un precio de 50£. Si nadie puja puede bajar a 40£, 30£ ó 20£, pero no puede bajar de ahí, para comenzar la subasta. Si nadie puja, se retirará el reloj de la venta. Cuando comience la subasta, podrá incrementar el precio de cinco en cinco libras hasta alcanzar el precio final (precio de reserva) que debe ser de 25£, para que el reloj se venda. Si no se alcanza ese precio, el reloj no se vende.

   Simule la subasta utilizando el equivalente de un dado de seis lados para comenzar la puja. Cuando se obtenga un seis, mientras se está en cualquiera de los precios de comienzo arrancará la puja.

   Cuando haya comenzado la puja, el programa simulará que hay tres de cuatro posibilidades de aumentar la puja en cada pregunta del subastador al público.

4. En una pelea del salvaje oeste, el sheriff se ha quedado sin munición y quiere arrestar a un pistolero que se ha refugiado en un bosque. Cabalga, por tanto, entre los árboles intentando que el pistolero le dispare. Su idea es que cuando el pistolero haya disparado seis balas, pueda rápidamente vencer al pistolero mientras éste intenta cargar la pistola. Simule el encuentro dando al pistolero una posibilidad entre veinte de herir al sheriff con cada uno de los disparos. Si no resulta herido después de los seis disparos, supondremos que arresta al pistolero.

5. Las instrucciones del sheriff a su ayudante son las siguientes:

   > "Si la pistola está descargada, recárgala y si no, sigue disparando hasta herir al bandido o hasta que se rinda. Si Pit Railnish se vuelve, sal corriendo.»

Escriba un programa que simule adecuadamente todas esas situaciones.

Pase lo que pase, vuélvete a Dodge City.
Si Pit Railnish se vuelve, retorna inmediatamente.
Si la pistola está vacía cárgala de nuevo.
Si la pistola no está vacía pide al bandido que se entregue.
Si el bandido se rinde, arréstale.
Si no se rinde, dispara.
Si resulta herido, arréstale y atiéndele la herida.

Suponga que la munición es ilimitada. Utilice un dado de "veinte caras", con el siete que signifique que el bandido "se rinde" y el trece que signifique que "está herido".

CAPITULO 15
PROCEDIMIENTOS Y FUNCIONES

En la primera parte de este capítulo explicamos las características más elementales de los procedimientos y funciones del SuperBASIC. Lo hacemos con ejemplos muy sencillos de forma que usted pueda comprender el funcionamiento de cada característica mientras se describe. Aunque los ejemplos son muy sencillos, usted apreciará que cuando lo haya comprendido podrá aplicar inmediatamente esas ideas a situaciones más complejas, donde son de verdad útiles.

Después de la primera parte, se discute el "porqué de los procedimientos". Si usted comprende más o menos todo esto, está en el buen camino y será capaz de utilizar los procedimientos y funciones con una efectividad que se irá incrementando.

El SuperBASIC le permite en primer lugar realizar las cosas más sencillas de forma sencilla, y luego le ofrece una mayor complejidad, si lo desea. Ciertas características adicionales y algunas otras materias técnicas se explican en la segunda parte de este capítulo, pero pueden omitirse (desde luego en una primera lectura) y a pesar de ello estar en una posición más fuerte que otros usuarios de antiguos tipos de BASIC

PARAMETROS DE VALORES

Hemos visto en los capítulos anteriores cómo puede introducirse un valor en un procedimiento. Veamos ahora otro ejemplo:

EJEMPLO

En la tienda de comidas preparadas "Platos Chinos Chan" tienen un menú con seis platos:

Platos de arroz	Dulces
1. gambas	4. helado
2. pollo	5. fritos
3. especial	6. lychees

Chan tiene un método sencillo para los precios. Los precios son los siguientes (en pesetas)

Los platos de arroz 300 + 10 veces el número del menú.
Los dulces, 12 veces el número del menú.

De este modo, un cliente que pida arroz especial y un helado, pagaría:

300 + 10 * 3 + 12 * 4 = 378 pesetas.

Programa

```
10 REMark Coste del Plato
20 plato 3
30 plato 4
40 DEFine PROCedure plato (num)
50   IF num < = 3 THEN LET precio = 300 + 10 * num
60   IF num > = 4 THEN LET precio = 12 * num
70   PRINT ! precio !
80   END DEFine
```

Salida

```
330 48
```

En el programa principal, se utilizan los parámetros actuales 3 y 4 (valores actuales de los parámetros formales). La definición del procedimiento tiene un

parámetro formal, num, que toma el valor que le suministra el programa principal. Observe que los parámetros formales deben ir entre paréntesis pero en el caso de los parámetros actuales no es necesario.

EJEMPLO Imagine ahora la variable de trabajo *precio*, que también se utiliza en el programa principal, con otro significado, por ejemplo, el precio de un vaso de cerveza, es decir 70 ptas. El programa siguiente falla y no da el resultado apetecido.

Programa

```
10 REMark Precio Global
20 LEt precio = 70
30 elem 3
40 elem 4
50 PRINT ! precio !
60 DEFine PROcedure elem(num)
70   IF num < = 3 THEN precio = 300 + 10 * num
80   IF num > = 4 THEN precio = 12 * num
90   PRINT ! precio !
100 END DEFine
```

Salida

```
330 48 48
```

El procedimiento ha alterado el precio de la cerveza. Decimos que la variable *precio* es **global** porque se puede utilizar en cualquier parte, dentro del programa.

EJEMPLO Haga que la variable *precio,* sea **LOCAL**, dentro del procedimiento. Esto significa que el SuperBASIC la tratará como una variable especial, accesible sólo desde el procedimiento. La variable *precio* del programa principal será diferente, aunque lleve el mismo nombre.

Programa

```
10 REMark LOCAL precio
20 LET precio = 70
30 elem 3
40 elem 4
50 PRINT ! precio !
60 DEFine PROCedure elem(num)
70     LOCAL precio
80     If num < = 3 THEN LET precio = 300 +10*num
90     IF num > = 4 THEN LET precio = 12*num
100   PRINT ! precio !
110 END DEFine
```

Salida

```
330 48 70
```

Esta vez todo ha funcionado perfectamente. La línea 70 hace que la variable *precio* del procedimiento se marque como "perteneciente" sólo al procedimiento. La otra variable *precio* no se ve afectada. Como puede verse, las variables locales son muy útiles.

EJEMPLO Las variables locales son tan útiles que automáticamente hacemos locales a los parámetros del procedimiento. Aunque no lo hemos mencionado antes, los parámetros como por ejemplo *num,* de los programas anteriores no pueden interferir con las variables del programa principal. Para probarlo tomamos la instrucción **LOCAL** del programa anterior y utilizamos la variable *num* para el precio de la cerveza. Como *num* es local en el procedimiento, todo sigue funcionando perfectamente.

Programa

```
10 REMark Parámetro LOCAL
20 LET núm = 70
30 elem 3
40 elem 4
```

```
50 PRINT ! num !
60 DEFine PROCedure elem(num)
70   IF num < = 3 THEN LET precio = 300 + 10 * num
80   IF núm > = 4 THEN LET precio = 12*num
90   PRINT ! precio !
100 END DEfine
```

```
330 48 70
```

Salida

PARAMETROS VARIABLES

Hasta el momento sólo hemos utilizado los parámetros de procedimiento para introducir valores dentro del procedimiento. Suponga, sin embargo, que el programa principal necesita que el coste de un elemento vuelva atrás de forma que se pueda calcular el coste total. Esto puede realizarse fácilmente colocando otro procedimiento en la llamada del procedimiento. Debe ser una variable porque debe recibir un valor del procedimiento. Sin embargo, nosotros lo llamaremos parámetro variable y debe corresponderse con el parámetro variable de la definición del procedimiento.

EJEMPLO

Utilice los parámetros variables actuales, *coste __ 1 y coste __ 2* para recibir los valores de las variables precios del procedimiento. Haga que el programa principal calcule e imprima la factura total.

Programa

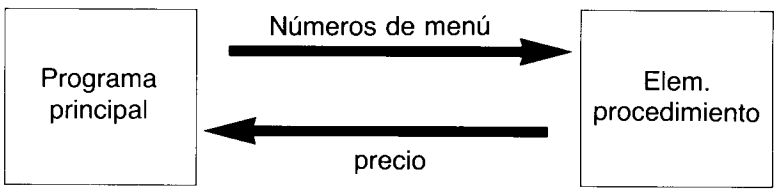
```
10 REMark Parámetro variable
20 LET num = 70
30 elem 3,cost__1
40 elem 4,cost__2
50 LET fact = num + cost__1 + cost__2
60 PRINT fact
70 DEFine PROCedure elem(num, precio)
80   IF núm < = 3 THEN LET precio = 300 + 10*num
90   IF num > = 4 THEN LET precio = 12*num
100 END DEFine
```

```
448
```

Salida

Los parámetros num y precio son los dos locales automáticamente, y por tanto no pueden causar problemas.

Los diagramas muestran cómo la información pasa desde el programa principal al procedimiento y de nuevo al programa principal.



Por el momento, sabemos suficiente sobre procedimientos y parámetros.

FUNCIONES

Usted ya conoce cómo funciona una función del sistema. Por ejemplo:

```
SQRT (9)
```

computa el valor 3, que es la raíz cuadrada de 9. Decimos pues que la función devuelve el valor 3. Una función, al igual que un procedimiento, puede tener uno o más parámetros, pero lo que distingue intrínsecamente una fun-

ción es que devuelve exactamente un valor. Esto significa que se puede utilizar en expresiones ya existentes. Teclee por ejemplo:

```
PRINT 2*SQRT(9)
```

y obtendrá una salida de 6. Así pues, una función, se comporta como un procedimiento con uno o más parámetros de valores y sólo un parámetro variable que contiene el valor que la función devuelve; ese parámetro variable tiene el nombre de la función.

Los parámetros no tienen por qué ser exclusivamente numéricos.

```
PRINT LEN("cadena")
```

tiene un argumento de cadena, pero devuelve el valor numérico 6.

EJEMPLO

Escriba de nuevo el programa de la última sección, que utiliza *precio* como parámetro variable. Dé a la función el nombre *precio.*

El valor devuelto está definido en la instrucción **RETurn.** Veámoslo más abajo:

Programa

```
10 REMark FUNción
20 LET num = 70
30 LET fact = num + precio(3) + precio(4)
40 PRINT fact
50 DEFine FUNction precio(num)
60   IF num < = 3 THEN return 300+10*num.
70   IF num > = 4 THEN return 12*num.
80 END DEFine
```

Salida

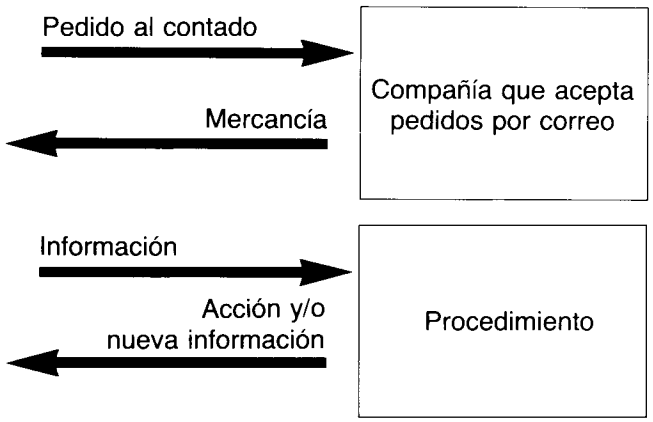
```
448
```

Observe la sencillez de la llamada a las funciones comparada con la llamada a los procedimientos.

¿POR QUE LOS PROCEDIMIENTOS?

El último concepto de un procedimiento es que debe ser una "caja negra" que reciba información específica del "exterior" y realice ciertas operaciones que pueden incluir la reexpedición de cierta información al "exterior". El "exterior" puede ser el programa principal u otro procedimiento.

El término "caja negra" implica que el trabajo interno no es importante, lo que realmente importa es lo que entra y lo que sale. Si, por ejemplo, un procedimiento utiliza la variable *cuenta,* y cambia su valor, puede afectar a una variable que lleve el mismo nombre en el programa principal. Piense en una Compañía que recibe encargos por correo. Usted le envía el pedido y el dinero, y ellos le envían la mercancía. La información se envía a un procedimiento y éste reexpide alguna acción y/o nueva información.



Puede que no deseemos que la Compañía antes mencionada utilice nuestro nombre y dirección u otra información con otros fines. Sería un efecto colateral no deseable. Del mismo modo, tampoco deseamos que el procedimiento cree cambios no planificados en los valores de las variables que se utilizan en el programa principal.

Evidentemente se puede hacer de forma que no se utilicen dos veces los nombres de las variables del programa. Esta es una posible solución al problema, pero ya hemos visto en este capítulo cómo evitar problemas si usted olvida las variables que ha utilizado en un determinado procedimiento.

Otro objetivo para la utilización de procedimientos es hacer que el programa sea modular. En lugar de tener un programa largo, éste se puede partir en determinadas tareas, que Seymour Papert, inventor del **LOGO**, llamó "elementos abarcables por la mente" (mindsized), que no son sino procedimientos, cada uno de ellos de un tamaño tal que se pueda comprender y controlar con facilidad. Estos procedimientos se unen por llamadas realizadas en secuencia o de un modo jerárquico.

Un tercer objetivo es evitar la escritura del mismo código dos veces. Es mejor escribirlo una sola vez y llamarlo dos veces, si es necesario.

Más abajo damos otro ejemplo que muestra cómo los procedimientos configuran un programa de forma modular.

EJEMPLO

Se hace un pedido de seis platos en la tienda de platos preparados Chan, cuyo menú es el siguiente:

Elem. Núm. Precio	Plato
1 350	Gambas
2 280	Pollo
3 330	Especial

Escriba un procedimiento que realice las siguientes tareas:

1. Establezca dos matrices de tres elementos que muestren el menú, platos y precios. Utilice una instrucción **DATA.**
2. Simule un pedido de seis platos aleatoriamente, utilizando un procedimiento. Elija platos y haga una lista detallada del número de veces que es elegido un determinado plato.
3. Pase los tres números a un procedimiento llamado *camarero* que devolverá el coste del pedido al programa principal utilizando un parámetro llamado *coste.* El procedimiento *camarero* llama a otros dos procedimientos, *cómputo* y *cocina,* que computan el coste y simulan la "cocina".
4. El procedimiento *cocina* imprime sencillamente el número pedido y el nombre de cada plato.

El programa principal debe llamar a los procedimientos las veces que sea necesario, obtener el coste total del procedimiento *camarero* añadir el 10% de propina e imprimir el total de la factura.

Diseño

Este programa ilustra el paso de parámetros de una forma muy compleja, y por tanto explicaremos el programa paso a paso antes de unirlo.

```
100 REMark Procedimientos
110 RESTORE 490
120 DIM elem$(3,8), precio(3), plato(3)
130 REMark ***PROGRAMA***
140 LET prop = 0.9
150 arranque
    —
    —

210 DEFine PROCedure arranque
220   FOR k = 1 TO 3
230     READ elem$(k)
240     READ precio(k)
250   END FOR k
260 END DEFine
    —
    —
    —
490 DATA "Gambas", 350, "Pollo", 280, "Especial", 330
```

Los nombres de los elementos del menú y sus precios se colocan en las matrices *elem$* y *precio.*

El paso siguiente es elegir un número de menú para cada uno de los seis clientes. Necesitaremos una lista del número de cada plato, pedido que se guardará en la matriz platos.

```
160 elija plato
    —
    —
    —
270 DEFine PROCedure elija(plato)
280   FOR tome = TO 6
290     LET núm = RND(1 TO 3)
300     LET plato(núm) = plato(núm) + 1
310   END FOR tome
320 END DEFine
```

Observe que el parámetro formal *plato* es tanto:

local dentro del procedimiento que elija
como una matriz en el programa principal

Los tres valores son enviados de nuevo a la matriz global llamada también *plato.* Estos valores pasan después al procedimiento *camarero.*

```
170 camar plato, fact
    —
    —
    —
    —
330 DEFine PROCedure camar(plato, coste)
340   calcula plato, coste
350   coci plato
360 END DEFine
```

camar pasa la información sobre el número de cada plato pedido al procedimiento calcula, el cual calcula el coste y lo devuelve.

```
370 DEFine calcula(plato, total)
380   LET total = 0
390   FOR k = 1 to 3
400     LET total = total + plato(k)*precio(k)
410   END FOR k
420 END DEFine
```

El camarero pasa la información a la cocina que sólo imprime el número pedido para cada elemento del menú.

```
430 DEFine PRocedure coci(plato)
440   FOR c= 1 TO 3
450     PRINT ! plato(c) ! elem$$(c)!
460   END FOR c
470 END DEFine
```

De nuevo en la matriz *platos* el procedimiento *coci* es local y recibe la información que utiliza el de una instrucción **PRINT**.

El programa completo se lista a continuación.

Programa

```
100 REMark Procedimientos
110 RESTORE 490
120 DIM elem$(3,8), precio(3), plato(3)
130 REMark ***PROGRAMA***
140 LET prop = 0.1
150 arranque
160 elija plato
170 camar plato, fact
180 LET fact = fact + prop * fact
190 PRINT "El coste total es pts." ! fact
200 REMark ***DEFINICIONES DE PROCEDIMIENTOS***
210 DEFine PROCedure arranque
220   FOR k = 1 TO 3
230     READ elem$(k)
240     READ precio(k)
250   END FOR k
260 END DEFine
270 DEFine PROcedure elija(plato)
280   FOR tome = 1 TO 6
290     LET núm = RND (1 TO 6)
300     LET plato(núm) = plato(núm) + 1
310   END FOR tome
320 END DEFine
330 DEFine PROCedure camar(plato, coste)
340   calcula plato, coste
350   coci plato
360 END DEFine
370 DEFine PROCedure calcula(plato, total)
380   LET total = 0
390   FOR k = 1 TO 3
400     LET total = total + plato(k)*precio(k)
410   END FOR k
420 END DEFine
430 DEFine PROCedure coci(plato)
440   FOR c = 1 TO 3
450 PRINT ! plato(c) ! elem$(c)
460   END FOR c
470 END DEFine
480 REMark ***DATOS DEL PROGRAMA***
490   DATA "Gambas", 350, "Pollo", 280, "Especial", 330
```

Salida

La salida depende de la selección aleatoria de los platos, pero la elección que sigue a estas líneas ilustra el tema y da una muestra de salida.

```
3 gambas
1 pollo
2 especial

El coste total es pts. 2040
```

COMENTARIOS

Obviamente, en un programa tan sencillo no es necesaria la utilización de los procedimientos y parámetros, pero es evidente que cada subtarea puede ser mucho más compleja. En este tipo de situaciones, la utilización de procedimientos permite la creación de programas modulares que pueden comprobarse por etapas. El ejemplo anterior ilustra muy por encima las notaciones principales y las relaciones entre los diferentes procedimientos.

Del mismo modo, el ejemplo siguiente ilustra la utilización de funciones.

Observese que en el ejemplo anterior, los procedimientos *camarero* y *calcula* devuelven exactamente un valor. Vuelva a escribir los procedimientos como funciones y muestre los cambios que le parezcan necesarios como cosecuencia de este cambio.

```
DEFine FuNction camar(plato)
  coci plato
  RETurn calcula plato
END DEFine

DEFine FuNction calcula(plato)
  LET total = 0
  FOR k = 1 TO 3
    LET total = total + plato(k)*precio(k)
  END FOR k
RETurn total
END DEFine
```

La función de llamada a *camarero* toma también una forma diferente:

```
LET fact = camar(plato)
```

Este programa funciona como anteriormente. Observe que hay menos parámetros aunque la estructura del programa es similar. Esto se debe a que los nombres de la función también sirven como parámetros devolviendo la información a la fuente de llamada de la función.

EJEMPLO

Todas las variables que se utilizan como parámetros formales en los procedimientos o funciones están "seguras" porque son locales automáticamente. Especifique las variables utilizadas en los procedimientos o funciones que no son locales, así como las instrucciones adicionales necesarias para convertirlas en locales.

Cambios en el programa

Las variables *k, tome* y *núm.* no son locales. Los cambios necesarios para convertirlas en locales son los siguientes:

```
LOCAL K
LOCAL toma, num
```

PARAMETROS SIN TIPO

Los parámetros formales no tienen tipo. Hasta el momento no hemos mencionado su aspecto porque usted puede trabajar perfectamente sin su conocimiento. Su apariencia es como si lo tuvieran y usted puede preferir que una variable que maneja números tenga apariencia de variable numérica y una variable que maneje cadenas tenga el aspecto de lo que es. Sin embargo, usted escribe parámetros sin tipo. Como prueba escriba el siguiente programa:

Programa

```
10 REMark Número de palabra
20 camar 2
30 camar "Pollo"
40 DEFine PROCedure camar(elem)
50   PRINT elem
60 END DEFine
```

Salida

```
2 Pollo
```

El tipo del parámetro se establece cuando se llama al procedimiento y "llega" un parámetro actual.

GAMA DE VARIABLES

Considere el programa siguiente y trate de averiguar qué numeros saldrán.

```
 10 REMark gama
 20 LET número = 1
 30 test
 40 DEFine PROCedure test
 50   LOCAL número
 60   LET número = 2
 70   PRINT número
 80   prueba
 90 END DEFine
100 DEFine PROCedure prueba
110   PRINT número
120 END DEFine
```

Obviamente el primer número que se imprimirá será 2, pero ¿será global la variable *número* de la línea 110?

La respuesta es que el valor del *número* en la línea 70 se llevará al procedimiento *prueba*. Una variable local dentro de un procedimiento, será la misma variable en un segundo procedimiento llamado por el primero.

Del mismo modo, si el procedimiento *prueba* se llama desde el programa principal, la variable *número* será el mismo número en ambos, el programa principal y el procedimiento *prueba*. Estas implicaciones pueden parecer extrañas al principio, pero son lógicas.

1. La variable *número* en la línea 20 es global.
2. La variable *número* en el procedimiento *prueba* es definitivamente local al procedimiento.
3. La variable *número* en el procedimiento *prueba* "pertenece" a la parte de programa que fue la última que llamó a ese procedimiento.

En este capítulo hemos descubierto muchos conceptos porque las funciones y procedimientos del lenguaje SuperBASIC son muy potentes. Sin embargo, no piense el lector que va a utilizar todas estas características inmediatamente. Utilice procedimientos y funciones de forma sencilla al principio. Pueden ser muy efectivas y potentes si es necesario.

PROBLEMAS SOBRE EL CAPITULO 15

1. Seis empleados de una compañía se identifican sólo por sus apellidos. Cada empleado cotiza una pensión que viene expresada en forma de porcentaje. Los datos siguientes representan los salarios totales y porcentajes de las pensiones de los seis empleados.

Fernández	13.800	6,25
González	8.700	6,00
Martínez	10.300	6,25
Suárez	15.000	7,00
Duralde	6.200	6,00
García	5.100	5,75

   Escriba los procedimientos necesarios para:

   Introducir los datos en las matrices.
   Computar las pensiones actuales y las contribuciones.
   Obtener en la salida una lista de los nombres y contribuciones contabilizadas.

   Una los procedimientos con un programa principal que los llame secuencialmente.

2. Escriba una función que seleccione (*select*) dos argumentos *gana y falta*. La función debe devolver un número entero aleatoriamente dentro de la *gama* dada, pero debe ser diferente del valor de *falta*.

Utilice la función en un programa que elija un color de fondo (**PAPER**) aleatorio y dibuje círculos también aleatorios en colores de texto (**INK**), de forma que ninguno sea del mismo color que el fondo.

3. Vuelva a escribir la solución al ejercicio 1 de forma que la función *pensión* tome como argumentos el salario y la contribución al fondo y devuelva la pensión contabilizada. Utilice dos procedimientos, uno como entrada de los datos y otro para dar salida a la información necesaria utilizando la función *pensión*.

4. Escriba lo siguiente:

   Un procedimiento que establezca una baraja de cartas.
   Un procedimiento que baraje las cartas.
   Una función que tome un número como argumento y devuelva un valor de cadena describiendo la carta.
   Un procedimiento que saque cuatro manos de poker de cinco cartas cada una.
   Un programa principal que llame a los procedimientos anteriores.
   (Véase el capítulo 16 donde se discute un problema similar.)

CAPITULO 16
ALGUNAS TECNICAS

En este capítulo final presentamos algunas aplicaciones de conceptos y características que ya se han discutido antes, y mostramos cómo deben aplicarse unas ideas posteriores.

SIMULACION DE UNA PARTIDA DE CARTAS

Es muy fácil almacenar y manipular cartas de juego, representándolas con los números desde el 1 al 52. De este modo se puede convertir un número en la carta equivalente. Suponga, por ejemplo, que aparece el número 29. Podemos decidir, por ejemplo, que:

Las cartas del 1 al 13 son corazones
Las cartas del 14 al 26 son pics
Las cartas del 27 al 39 son diamantes
Las cartas del 40 al 52 son tréboles

entonces usted sabrá que 29 significa que usted tiene un "diamante". Su QL puede programarse para realizar esto:

```
LET palo = (carta-1) DIV 13
```

que producirá un valor de la gama de 0 a 3 que puede utilizarse para imprimir el palo determinado. Este valor se puede reducir a una gama del 1 al 13 escribiendo:

```
LET valor = carta MOD 13
IF valor = 0 THEN LET valor = 13
```

Programa

Puede hacerse que los números del 1 al 13 impriman: As, 2, 3... Jack, Reina, Rey, o si lo prefiere se pueden imprimir frases como "dos de corazones". El programa siguiente imprime el nombre de la carta que corresponde al número de la entrada.

```
 10 REMark Cartas
 20 DIM nompalo$(4,8),carval$(13,5)
 30 LET d$ = "de"
 40 set_up
 50 REPeat cartas
 60   INPUT "Introduzca un número de carta 1-52:" ! carta
 70   IF carta <1 OR carta> 52 THEN EXIT cartas
 80   LET palo = (carta-1) DIV 13
 90 LET valor = carta MOD 13
100 IF valor = 0 THEN LET valor = 13
120   PRINT carval$(valor) ! d$ ! nompalo$(palo)
130 END REPeat cartas
140 DEFine PROCedure set-up
150   FOR s = 1 TO 4: READ nompalo$(s)
160   FOR v = 1 TO 13 : READ carval$(v)
170 END DEFine
180 DATA "corazones", "clubs", "diamantes", "tréboles"
190 DATA "As", "Dos", "Tres", "Cuatro", "Cinco", "Seis", "Siete"
200 DATA "Ocho", "Nueve", "Diez", "Jack", "Reina", "Rey".
```

Entradas y salidas

```
13
Rey de corazones
49
Diez de tréboles
27
As de diamantes

0
```

COMENTARIO Observe que al usar las instrucciones **DATA** necesitaremos un archivo de datos permanente, que el programa utilizará siempre. Los otros datos que cambian cada vez que el programa se ejecuta, se introducen a través de instrucciones **INPUT**. Si se conocen los datos de entrada antes de ejecutar el programa, sería también correcta la utilización de otra instrucción **READ** y más instrucciones **DATA**. Con ello se obtendría un control mejor.

ARCHIVOS DE DATOS SECUENCIALES

El programa siguiente establece un archivo con una centena de números.

```
10 REMark Archivo Número
20 OPEN_NEW #6, mdv1_números
30 FOR num = 1 TO 100
40   PRINT #6,num
50 END FOR num
60 CLOSE #6
```

Después de ejecutar el programa, compruebe que el nombre de archivo "números" se encuentra en el directorio tecleando:

```
DIR mdv1_
```

Se puede obtener una visión del archivo sin ningún formateo especial, sin más que copiarlo del Microdrive a la pantalla:

```
COPI mdv1_números to scr
```

También se puede usar el programa siguiente para leer el archivo y visualizar sus registros en la pantalla.

```
10 REMark Lectura de archivo
20 OPEN_IN #6, mdv1_números
30 FOR num = 1 TO 100
40   INPUT #6,elem
50   PRINT ! elem !
60 END FOR num
70 CLOSE #6
```

Si lo desea, puede alterar el programa para obtener la salida en forma diferente.

De forma similar, los programas siguientes establecen un archivo de cien letras elegidas aleatoriamente, y las leen.

Archivos de carácter

```
10 REMark Archivo letras
20 OPEN_NEW #5,mdv1_archcar
30 FOR num = 1 TO 100
40   LET ch$ = CHR$(RND(65 TO 90))
50   PRINT #6, ch$
60 END FOR num
70 CLOSE #6
```

```
10 REMark Obtención de Letras
20 OPEN_IN #6,mdv1_archcar
30 FOR num = 1 TO 100
40   INPUT #6, elem$
50   PRINT ! elem$ !
60 END FOR num
70 CLOSE #6
```

ESTABLECIMIENTO DE UN ARCHIVO DE DATOS

Suponga que desea establecer un archivo sencillo de nombres y números de teléfono.

JUAN	678462
PEDRO	896487
EVA	455737
MARTA	255145
LOLA	345212
PEPE	121234
ARTY	303003

Puede hacerlo con el siguiente programa.

```
10 REMark Números de teléfono
20 OPEN_NEW #6, mdv1_teléf
30 FOR registro = 1 TO 7
40   INPUT nombres$,núm$
50   PRINT #6; nombre$; núm$
60 END FOR registro
70 CLOSE#6
```

Pulse **RUN** e introduzca un nombre seguido de la tecla **ENTER** y un número seguido por la tecla **ENTER**. Repita la operación siete veces.

Observe que los datos se "guardan". Se almacenan internamente hasta que el sistema esté preparado para transferirlo al Microdrive. El acceso al Microdrive se realiza sólo en un momento dado, como puede usted apreciar si oye y atiende.

COPIA DE UN ARCHIVO

Una vez establecido el archivo, debe hacerse inmediatamente una "copia de seguridad". Para realizarlo, teclee:

```
COPY mdv1_teléf TO mdv2_
```

LECTURA DE UN ARCHIVO

Es necesario asegurarse de que el archivo existe con el formato correcto, para que se pueda leer de nuevo desde el Microdrive y visualizar en la pantalla. Esto puede hacerse fácilmente utilizando lo siguiente:

```
COPY mdv2_teléf TO scr
```

La salida en pantalla no suministra automáticamente los espacios entre el nombre y el número, pero sí da una "nueva línea" al final de cada registro. La salida será por tanto:

```
JUAN6784632
PEDRO896487
EVA455737
MARTA255145
LOLA345212
PEPE121234
ARTY303003
```

Con el programa siguiente, podemos obtener una presentación más controlada de los datos:

```
10 REMark Lectura de Números de Teléfono
20 OPEN_IN #5,mdv1_teléf
30 FOR registro = 1 TO 7
40   INPUT 5,reg$
50   PRINT,reg$
60 END FOR registro
70 CLOSE #5
```

Los datos se imprimen como antes, pero en este caso, cada par de campos se guardan en la variable *reg*$ antes de imprimirse en la pantalla, y usted tiene la oportunidad de manipularla como desee.

Observe que se puede utilizar más de una variable de cadena en la etapa de creación de un archivo con **INPUT** y **PRINT**, pero el registro completo creado de este modo puede obtenerse desde el archivo de Microdrive con una única variable de cadena (reg$ en el ejemplo anterior)

CLASIFICACION POR INTERCALACION

Los colores siguientes están disponibles en el modo de baja resolución (su número de código va desde el 0 al 7)

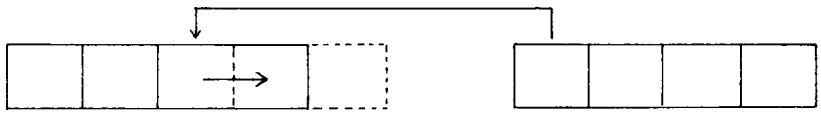
negro azul rojo morado verde ciano amarillo blanco

EJEMPLO

Escriba un programa que clasifique los colores en orden alfabético, utilizando una clasificación por inserción.

Método

Colocamos los ocho colores en una matriz, *color$,* que dividimos en dos partes:



Se toma el elemento situado más a la izquierda de la parte sin clasificar y se compara con cada uno de los elementos de derecha a izquierda en la parte clasificada hasta encontrar su lugar correcto. Mientras se van realizando las comparaciones, se desplazan los elementos clasificados a la derecha, de forma que cuando encontremos el lugar adecuado se pueda insertar el elemento inmediatamente sin tener que realizar otro desplazamiento posterior.

Suponga que hemos llegado a un punto en el que se han clasificado cinco elementos, y nos fijamos ahora en el ciano que es el elemento situado más a la izquierda en la zona sin clasificar.

1	2	3	4	5		6	7	8
azul	morado	negro	rojo	verde		ciano	amarillo	blanco
		parte clasificada					parte sin clasificar	

1. Colocamos el ciano en la variable *comp$* y asignamos a una variable *p* el valor 6.
2. La variable *p* indicará por el momento el lugar en el que pensamos que debe colocarse el ciano. Si se debe mover a la izquierda decrementaremos el valor de *p*.
3. Comparamos el ciano con el verde. Si el ciano es mayor (más cercano a la Z) o igual que el verde, salimos (exit) y el ciano permanece en su lugar.

   En caso contrario, copiamos el verde en la posición 6 y decrecemos el valor de *p;* por tanto queda:

1	2	3	4	5	6		7	8
azul	morado	negro	rojo	ciano	verde		amarillo	blanco

4. Repetimos de nuevo el proceso, pero en este caso el ciano se compara con el rojo, y se obtiene:

1	2	3	4	5	6		7	8
azul	morado	negro	ciano	rojo	verde		amarillo	blanco

5. Realizando dos veces más este proceso, obtenemos sucesivamente:

1	2	3	4	5	6		7	8
azul	morado	ciano	negro	rojo	verde		amarillo	blanco

y después:

1	2	3	4	5	6		7	8
azul	ciano	morado	negro	rojo	verde		amarillo	blanco

6. Finalmente nos movemos de nuevo a la izquierda comparando el ciano con el azul. Esta vez no es necesario mover o cambiar nada. Salimos del bucle y colocamos el ciano en la posición dos. Ya estamos preparados para fijarnos en el séptimos elemento, el amarillo.

1	2	3	4	5	6		7	8
azul	ciano	morado	negro	rojo	verde		amarillo	blanco

ANALISIS DEL PROBLEMA

1. En primer lugar, almacenamos los colores en una matriz, *color$(8)* y utilizamos

   comp$ para el color que se está comparando
   p pensamos que puede estar situado el color de **comp$.**

2. Un bucle **FOR** cuestionará las posiciones 2 a la 8 una tras otra (los elementos únicos están clasificados).

3. Un bucle **REPeat** permitirá las comparaciones hasta que encontremos dónde debe colocarse el valor *comp$*.

   REPeat comparación
   IF *comp$* no necesita moverse izda. **(EXIT)**
   copia un color en la posición a su
   derecha y decrementa *p*.
   END REPeat comparación

4. Después de salir del bucle **REPeat** el color que se encuentra en *comp$* se coloca en la posición *p* y el bucle **FOR** continúa.

Diseño del programa

1. Declaración de la matriz color$
2. Lectura de los colores dentro de la matriz
3. **FOR** elemento = 2 **TO** 8
   LET p = elemento
   LET comp$ = color$(p)
   REPEAT comparacion
   IF comp$ >= color$(p-1) : **EXIT** comparación
   LET color$(p) = color$(p-1)
   LET p = p-1
   END REPeat comparación

```
      LET color$(p) = comp$
    END FOR elemento
4.  PRINT matriz clasificada color$
5.  DATA
```

Realmente el programa tiene un fallo consistente en suponer que el primer elemento de la tabla original está bien clasificado en la primera posición (como sucedería si tomáramos los nombres ingleses de los colores clasificados por su orden numérico −1 al 8−: **blac blue red... white.**

En efecto, cuando comparamos el azul con el negro, debemos disminuir *p* al valor 1 y en el paso siguiente compararemos de nuevo el azul con una variable inexistente —pues *color$(p-1)* será ahora color$(0).

Este problema es bien conocido en programación y la solución se hace "a posteriori" al final de la matriz. Necesitaremos colocar justo antes del bucle **REPeat** lo siguiente:

```
LET color$(0) = comp$
```

Afortunadamente, el lenguaje SuperBASIC permite índices cero, de otro modo el problema habría tenido que solucionarse a expensas de la facilidad de lectura del programa.

PROGRAMA MODIFICADO

```
10 REM clasificación por Inserción
20 DIM color$(8,7)
30 FOR elem = 1 TO 8 : READ color$(elem)
40 FOR elem = 2 TO 8
50   LET p = elem
60   LET comp$ = color$(p)
70   LET color$(0) = comp$
80   REPeat compara
90     IF comp$ >= color$(p-1) : EXIT compara
100    LET color$(p) = color$(p-1)
120    LET p = p-1
130  END REPeat compara
140  LET color$(p) = comp$
150 END FOR elem
160 PRINT "Clasificación ; " \ color$(1 to 8)
170 DATA "Negro", "Azul", "Morado", "Rojo"
180 DATA "Verde", "Ciano", "Amarillo", "Blanco"
```

COMENTARIOS

1. El programa funciona bien. Se ha probado con datos de ensayo:

   A A A A A A A
   B A B A B A B
   A B A B A B A
   B C D E F G H
   G F E D C B A

2. La clasificación por inserción no es muy rápida, pero puede ser útil para añadir ciertos elementos a alguna lista que ya ha sido clasificada. En algunos casos es conveniente dar pequeños tiempos con cierta frecuencia para mantener los elementos en orden que gastar un tiempo considerable con menos frecuencia para realizar una reclasificación total.

Ahora ya tiene usted conocimientos suficientes para seguir el desarrollo del manejo del archivo de siete nombres con sus números de teléfono.

CLASIFICACION DE UN ARCHIVO DE MICRODRIVE

Para ordenar el archivo "Telef" en orden alfabético de nombres debemos leerlo en el interior de una matriz interna, clasificarla y luego crear un nuevo archivo que ya tenga los nombres en orden alfabético.

No es una buena norma el borrar un archivo antes de sustituirlo, así lo hemos establecido claramente y se ha probado que es el mejor sistema. Por tanto, como norma de seguridad, se debe copiar en primer lugar el archivo utilizando un nombre diferente. El proceso necesario es el que sigue:

1. Copiar el archivo "telef" en "telef-temp".
2. Leer el archivo "telef" en una matriz.
3. Clasificar la matriz.
4. Parar para comprobar que todo va bien.
5. Borrar el archivo "telef".
6. Crear el nuevo archivo "telef".

Todo esto es lo que el programa debe hacer, pero además debemos probarlo inmediatamente utilizando lo siguiente:

```
COPY mdv1_telef TO scr
```

y se realizarán las comprobaciones necesarias, por tanto:

```
DELETE mdv2_telef
COPY mdv1_telef to mdv2_telef
COPY mdv1_telef to scr
DELETE mdv1_telef_temp
```

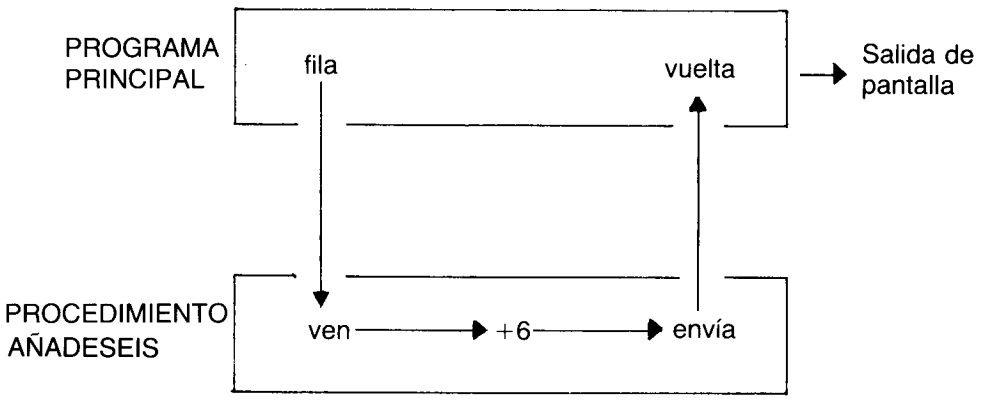
Las operaciones anteriores completan el proceso de sustitución de un archivo clasificado por el archivo original sin clasificar, tanto en originales como en sus copias.

PARAMETROS QUE SON MATRICES

En el programa siguiente ilustramos el paso de matrices completas entre el programa principal y el procedimiento. Los datos pasan en las dos direcciones.

En la línea 40, la matriz llamada *fila* que guarda los números 1,2,3 pasa al procedimiento *añadeseis* . El parámetro *ven* recibe los datos que entran y el procedimiento añade seis unidades a cada elemento. El parámetro *envía* tiene, en este punto, los números 7,8,9.

Estos números pasan de nuevo al programa principal para convertirse en valores de la matriz *vuelta*. Los valores se imprimen para probar que los datos se han movido como deseábamos.



Programa

```
10 REMark Paso de matrices
20 DIM fila(3), vuelta(3)
30 FOR k = 1 TO 3 : LET fila(k) = k
40 añadeseis fila, vuelta
50 FOR k = 1 TO 3 : PRINT vuelta(k) !
60 DEFine PROCedure añadeseis(ven,envía)
70    FOR k = 1 TO 3 : LET envia(k) = ven(k) + 6
80 END DEFine
```

Salida 7 8 9

El procedimiento siguiente recibe una matriz con los datos que deben clasificarse. El elemento cero contendrá el número de elementos. Observe que no importa que la matriz sea numérica o de cadena. Debido al cambio forzado de tipos se cambiarán los datos de cadena a numéricos, si fuera necesario.

Otro segundo punto de interés es observar que el elemento *ven(0)* se utiliza con dos propósitos:

lleva el número de elementos que se van a clasificar
se utiliza para contener el elemento que en ese momento está bien colocado.

```
800 DEFine PROCedure clasif(ven,envía)
810   LET num = ven(0)
820   FOR elem = 2 TO num
830     LET p = elem
840     LET ven(0) = ven(p)
850     REPeat compara
860       IF ven(0)>=ven(p-1) : EXIT compara
870       LET ven(p) = ven(p-1)
880       LET p = p-1
890     END REPeat compara
900     LET ven(p) = ven(0)
910   END FOR elem
920   FOR k = 1 TO 7 : envía(k) = ven (k)
930 END DEFine
```

Las líneas adicionales siguientes comprobarán el procedimiento de clasificación.

```
10 REMark Prueba de Clasificación
20 DIM fila$(7,3),vuelta$(7,3)
25 LET fila$(0) = 7
30 FOR k = 1 TO 7 : READ fila$(k)
40 clasif fila$,vuelta$
50 PRINT ! vuelta$ !
60 DATA "EEL", "DOG", "ANT", "GNU", "CAT", "BUG", "FOX"
```

Salida ANT BUG CAT DOG EEL FOX GNU

COMENTARIOS

Este programa ilustra la facilidad con la que pueden manejar las matrices en SuperBASIC. Todo lo que debe hacer es utilizar el nombre de la matriz para pasarla como un parámetro de un procedimiento o bien para imprimirla completa. Una vez almacenado (SAVE) el procedimiento se puede usar **MERGE mdv1_clasif** para añadirlo al programa que esté en la memoria principal.

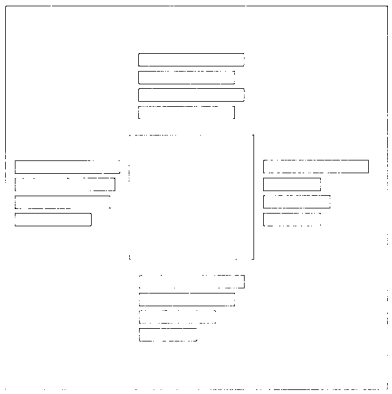
En ese momento, usted tendrá suficiente conocimiento de las técnicas y de la sintaxis para manejar presentaciones de pantalla más complejas. Suponga que desea representar las manos de cuatro jugadores de cartas. Un mano se puede representar por:

```
H: A  3 7 Q
C: 5  9 J
D: 6 10 K
S: 2  4 Q
```

Para ayudar en la presentación, imprimiremos los Corazones y Diamantes en rojo y los Pics y Tréboles en negro. Un color adecuado para las franjas **STRIP** puede ser el blanco. El fondo general puede ser verde y la mesa puede obtenerse mezclando dos colores.

METODO

Como vamos a tratar con caracteres de impresión es mejor comenzar planeando el esquema de pantalla en términos de pixels. Podrá ver que se necesitan doce líneas de caracteres con algunos espacios entre líneas y una altura total de pantalla de 256 pixels.



Es útil recalcar que las alturas posibles para los caracteres son de 10 pixels o 20 pixels. Es obvio que nos irá mejor la altura de 10 pixels para permitir el espacio para una presentación esmerada.

Estimemos el número de posiciones de caracteres en el sentido horizontal de la pantalla. Si adoptamos el signo convenido "D" para el "10" todos los valores de las cartas se pueden representar como caracteres simples. Suponga que en una primera aproximación permitimos un máximo de ocho cartas del mismo palo. Si es necesario podemos reconsiderar el problema. Necesitaremos para ello un total de 10 caracteres por cada mano. Las imposiciones en sentido horizontal son las siguientes:

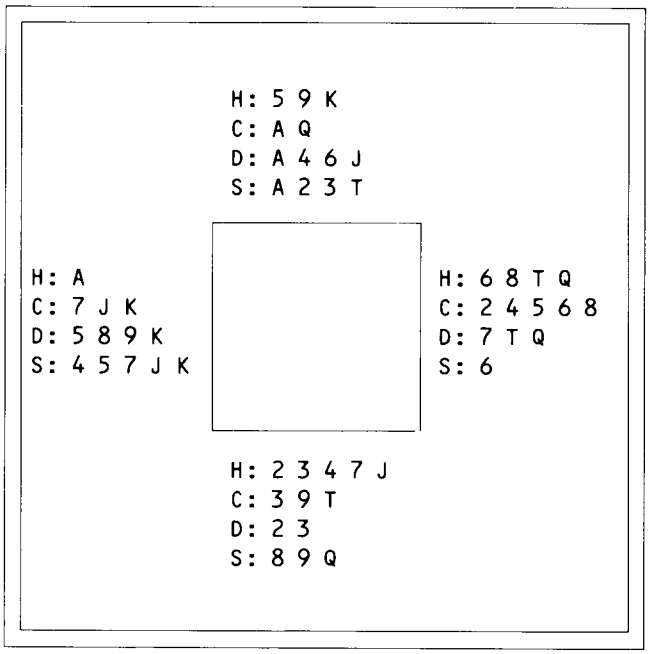
mano oeste + anchura de la mesa + mano este

y si permitimos espacios entre los caracteres, será:

20 + anchura de la mesa + 20

La decisión depende ahora del modelo de pantalla elegido. El modo de 256 solucionará el problema, como se verá más adelante, pero en primer lugar trabajaremos en el modo de 512 pixels. La anchura menor de un carácter es seis pixels, que nos dan un total de 240 pixels + anchura de la mesa. El diagrama nos dará un primer balance si consideramos una anchura para la mesa de la mitad de 240.

Experimentemos con un ancho para la mesa de unos 120 pixels, que puede ser adecuado. Después de una pequeña prueba, produciremos la presentación de pantalla que aparece más abajo.



VENTANA 440 × 220 en 35,15
Verde con un borde blanco de 10 unidades

MESA 100 × 60 en 150,60
Superficie con patrón Stiplle ajedrezado en rojo y verde

MANOS Espacio para al menos ocho símbolos de cartas. Las posiciones iniciales del cursor son:

Norte	150,10
Este	260,60
Sur	150,130
Oeste	30,60

TAMAÑO DE LOS CARACTERES: estándard para el MODO 512.
NUMERO DE PIXELS: 12 entre líneas.

COLOR DE LOS CARACTERES: Blanco.
BANDA DE FONDO DE LOS CARACTERES: Rojo para los Corazones y Diamantes. Negro para los Pics y Tréboles.

VARIABLES

cart(52)	almacena los números de las cartas
clas(13)	clasifica cada mano
pal$(4,2)	almacena palos C:, P:, T:, D:
kmcmh	variables de un bucle
ran	posición aleatoria para cambio de cartas
temp	se usa en los cambios de cartas
elem	carta que se inserta clasificada
dard	puntero para encontrar la posición ya clasificada
comp	mantiene el número de la carta en la clasificación
inc	incremento en pixels en las pilas de cartas
sit	posición actual del juego
ac,dn	posición del cursor para los caracteres
fila	fila en curso de los caracteres
lin$	crea una fila de caracteres
maz	número más alto para una carta
p	apunta la posición de una carta
n	número que toma en ese momento una carta

PROCEDIMIENTOS

barajado	baraja 52 cartas
reparto	reparte cartas en cuatro manos y llama a clasif para que clasifique cada mano
clasif	clasifica 13 cartas en orden ascendente
presen	crea un color de fondo, borde y mesa
imprim	imprime cada línea de símbolos de cartas
nuelin	toma una fila de cartas y convierte los números en símbolos A,2,3,4,5,6,7,8,9,D,J,Q,K

SOMERO DISEÑO DEL PROGRAMA

1. Declaración de matrices, elección de "palo" y colocación de 52 números en la matriz cart.
2. Barajado de cartas.
3. Reparto de 4 manos y clasificación de cada una de ellas.
4. Apertura (OPEN) de una ventana en la pantalla.
5. Establecimiento de un esquema de pantalla.
6. Impresión de las cuatro manos.
7. Cerrado (CLOSE) de la ventana de la pantalla.

```
10 DIM cart(52),clas(13),pal$(4,2)
20 FOR k = 1 TO 4 : READ pal$(k)
30 FOR k = 1 TO 52 : LET cart(k) = k
40 barajado
50 reparto
```

```
  60 OPEN #6,scr_440x220a35x15
  70 Presen
  80 imprim
  90 CLOSE #6
 200 DEFine PROCedure barajado
 210   FOR c = 52 TO 3 STEP -1
 220     LET ran = RND(1 TO c-1)
 230     LET temp = cart(c)
 240     LET cart(c) = cart(ran)
 250     LET cart(ran) = temp
 260   END FOR c
 270 END DEFine
 300 DEFine PROCedure reparto
 310   FOR h = 1 TO 4
 320     FOR c = 1 TO 13
 330       LET clas(c) = cart((h-I)*13+c)
 340     END FOR c
 350     clasif
 360     FOR c = 1 TO 13
 370       LET cart((h-1)*13+c) = clas(c)
 380     END FOR c
 390   END FOR h
 400 END DEFine
 500 DEFine PROCedure clasif
 510   FOR elem = 2 TO 13
 520     LET dard = elem
 530     LET comp = clas(dard)
 540     LET clas(∅) = comp
 550     REPeat compara
 560       IF comp >=clas (dard-1) : EXIT compara
 570       LET clas(dard) = clas(dard-1)
 580       LET dard = dar-1
 590     END REPeat compara
 600     LET clas(dard) = comp
 610   END FOR elem
 620 END DEFine
 630 DEFine PROCedure presen
 640   PAPER #6,4 : CLS #6
 650   BORDER #6,10,0
 660   BLOCK #6,100,60,150,60,2,4
 670 END DEFine
 800 DEFine PROCedure imprim
 810   LET inc = 12 : INK #6,7
 820   LET p = 0
 830   FOR sit = 1 TO 4
 840     READ ac,dn
 850     FOR fila = 1 TO 4
 860       nuelin
 870       CURSOR #6,lin$
 880       PRINT #6,lin$
 890       LET dn = dn + inc
 900     END FOR fila
 910   END FOR sit
 920 END DEFINE
1000 DEFine PROCedure nuelin
1010   IF fila MOD 2 = 0 THEN STRIP #6,0
1020   IF fila MOD 2 = 1 THEN STRIP #6,20
1030   LET lin$ = pal$(fila)
1040   LET maz = fila*13
1050   REPeat un_palo
1060     LET p = p+1
1070     LET n = cart(p)
1080     IF n > maz THEN p-1 : EXIT un_palo
1090     LET n = n MOD 13
1100     IF n = 0 THEN n = 13
1110     IF n = 1 : LET ch$"A"
1120     IF n >= 2 and n <= 9 : LET ch$ = n
1130     IF n = 10 : LET ch$ = "D"
1140     IF n = 11 : LET ch$ = "J"
1150     IF n = 12 : LET ch$ = "Q"
1160     IF n = 13 : LET ch$ = "K"
1170     LET lin$ = lin$ & " " & ch$
1180     IF p = 52 : EXIT un_palo
```

```
1190     IF cart(p)>cart(p+1) : EXIT un_palo
1200   END REPeat un_palo
1210 END DEFine
1220 DATA "C:", "P:", "T:", "D:"
1230 DATA 150,10,260,60,150,130,30,60
```

Debe cambiarse el 10 = "D" por "X" para no confundirlo con los diamantes "D".

COMENTARIOS

El programa funciona en el modo 256 pero las líneas con los símbolos de las cartas se pueden sobreponer sobre la "mesa" o salirse del borde de la ventana. Para corregirlo, podemos realizar los siguiente cambios:

```
1170 LET lin$ = lin$ & " " & ch$
```

por

```
1170 LET lin$ = lin$ & ch$
```

y lo habremos corregido. Los espacios entre los caracteres desapareceran, pero como su tamaño es mayor las filas son perfectamente legibles. Así pues, el programa funciona bien en cualquiera de los dos modos.

CONCLUSION

Hemos tratado de mostrar cómo se puede utilizar el lenguaje SuperBASIC para resolver problemas. También hemos mostrado lo sencillas que pueden resultar las tareas si se realizan de forma simple. Si la tarea es compleja inherentemente, como la manipulación de matrices o el diseño de gráficos en pantalla, el SuperBASIC le permite manejar el problema con eficacia y un máximo de claridad.

Si es usted principiante y ha trabajado sobre gran parte de esta guía, habrá empezado bien el camino a la buena programación. Si ya tiene experiencia, esperemos que haya apreciado y aprovechado las características extra que ofrece el SuperBASIC.

La gama de tareas que puede realizar el SuperBASIC es enorme, y en esta guía sólo hemos tratado una pequeña fracción. No podemos saber al nivel de posibilidades que puede usted llegar, pero esperamos que encuentre todo fructífero, estimulante y divertido.

CONTESTACIONES AL AUTOEXAMEN SOBRE EL CAPITULO 1

1. Utilice la secuencia BREAK de interrupción para abandonar un programa que se está ejecutando cuando suceda una de las siguientes cosas:
   a) Algo está equivocado y usted no lo comprende.
   b) Ya ha dejado de tener interés.
   c) Cualquier otro problema (tres puntos)
2. El botón de reinicialización (RESET) se encuentra en el lado derecho del ordenador.
3. El efecto del botón RESET es muy semejante a desconectar y conectar de nuevo el ordenador.
4. La tecla **SHIFT**.
   a) Sólo es efectiva mientras permanece presionada; por el contrario, la tecla **CAPS LOCK** (seguro de las mayúsculas) es efectiva nada más pulsarse. (un punto)
   b) La tecla **SHIFT** afecta a todas las letras, dígitos y símbolos, pero la tecla **CAPS LOCK** (seguro de las mayúsculas) sólo afecta a las letras (otro punto).
5. Las teclas de **CTRL** con la flecha izda. del cursor borran el carácter situado inmediatamente a la izquierda del cursor.
6. La tecla ↵ **(ENTER)** causa la introducción de un mensaje o instrucción en el ordenador para que este actúe adecuadamente.
7. Para representar la tecla **ENTER**, se utiliza el signo ↵.
8. El comando **CLS**↵ hace que parte de la pantalla quede limpia (vacía).
9. El comando **RUN**↵ hace que se ejecute un programa almacenado.
10. El comando **LIST**↵ hace que se visualice en la pantalla (listado), el programa almacenado que va a ejecutarse.
11. El comando **NEW**↵ limpia la memoria principal dejándola preparada para un nuevo programa.
12. Las palabras clave del lenguaje SuperBASIC son reconocidas por el ordenador tanto en mayúsculas como en minúsculas.
13. La parte de la palabra clave que aparece en mayúsculas indica al usuario la abreviación permitida.

COMPRUEBE SU PUNTUACION

De 14 a 16 es muy buena. Continúe la lectura.

12 ó 13 es una puntuación buena, pero vuelva a leer algunas partes del capítulo uno.

10 u 11 es aceptable. Vuelva a leer algunas partes del capítulo uno y repita el test.

Si obtiene menos de 10 debe trabajar cuidadosamente con el capítulo uno y repetir de nuevo el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 2

1. El almacenamiento interno de un número se asemeja a un casillero, al que se pueden poner nombres en las casillas y colocar números en su interior.

2. La instrucción **LET** que utiliza por primera vez un determinado nombre, causará la creación de un casillero y le asignará un nombre, por ejemplo:

   LET cuenta = 1↵ (1 punto)

   La instrucción **READ** que utiliza por primera vez un nombre tendrá el mismo efecto, por ejemplo:

   READ cuenta↵ (1 punto)

3. Para averiguar el valor de un determinado casillero puede utilizarse la instrucción **PRINT**.

4. El nombre técnico de un casillero de palomar es "variable", ya que sus valores pueden variar mientras se va ejecutando el programa.

5. Una variable toma su primer valor la primera vez que se utiliza en una instrucción **LET, INPUT** o **READ.**

6. Los cambios en los valores de una variable son causados generalmente por la ejecución de una instrucción **LET.**

7. El signo = en una instrucción **LET** representa una operación:

   Valore lo que se encuentra en el miembro de la derecha y colóquelo en el casillero cuyo nombre se indica en el miembro de la izquierda, esto es, el miembro de la izquierda se hace igual al miembro de la derecha.

8. Las instrucciones sin numerar se ejecutan inmediatamente.

9. Las instrucciones numeradas no se ejecutan inmediatamente, se almacenan.

10. Las comillas en una instrucción **PRINT** incluyen el texto que se desea imprimir.

11. Cuando no se utilizan las comillas, se imprime el valor de una variable.

12. Una instrucción **INPUT** hace que el programa se pare para que usted pueda teclear los datos en el teclado.

13. Las instrucciones **DATA** nunca se ejecutan.

14. Se utilizan para suministrar los valores para las variables en las instrucciones.

15. La palabra técnica utilizada para designar el nombre de un casillero es "identificador".

16. Ejemplos:

    i. día
    ii. día_23
    iii. día_de_fiesta (3 puntos)

17. La barra de espacios es especialmente importante para colocar espacios antes o detrás de las palabras clave, de forma que no puedan tomarse como identificadores (nombres) elegidos por el usuario.

18. Los identificadores elegidos libremente son importantes porque le ayudan a que sus programas sean más comprensibles. Estos programas tienen menos tendencia a incluir errores y son más adaptables.

COMPRUEBE SU PUNTUACION

De 18 a 21 puntos, muy bien. Continúe la lectura.

16 ó 17 puntos, bien, pero vuelva a leer ciertas zonas del capítulo dos.

14 ó 15, aceptable, pero vuelva a leer ciertas zonas del capítulo dos, y vuelva a hacer de nuevo el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 3

1. Un pixel es el área menor de luz que puede visualizarse en la pantalla.
2. Se dispone de 256 posiciones de pixels en la dirección horizontal de la pantalla de omisión de baja resolución.
3. En la pantalla de baja resolución, se dispone de 256 posiciones de pixels en dirección vertical.
4. Una dirección viene determinada por:

   el valor superior, de 0 a 100
   el valor inferior, de 0 a un número evaluado por el sistema.
5. En la pantalla de omisión, se dispone de ocho colores; en ellos está incluido el blanco y el negro.
6. i. **LINE**, para dibujar una línea, ej. **a,b TO x,y.**
   ii. **INK**, para seleccionar un color de dibujo, ejemplo **INK 5.**
   iii. **PAPER**, para seleccionar el color del fondo, ej., **PAPER 7.**
   iv. **BORDER**, para dibujar un reborde, ej. **BORDER 1,5.**
7. **REPeat** *nombre...* **END REPeat** *nombre.*
8. Un bucle de repetición **REPeat** termina cuando se ejecuta la instrucción **EXIT.**
9. En el lenguaje SuperBSIC, los bucles llevan nombres para posibilitar la salida de ellos de forma sencilla.

COMPRUEBE SU PUNTUACION

De 11 a 13, muy bien. Continúe leyendo.

De 8 a 10, bien, pero vuelva a leer algunas zonas del capítulo tres.

6 ó 7 es suficiente, pero relea algunas partes del capítulo tres, y vuelva a hacer el test.

Por debajo de 6. Debe trabajar más sobre el capítulo tres, y volver a realizar el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 4

1. Una cadena de caracteres es una secuencia de caracteres, como por ejemplo letras, dígitos u otros símbolos.
2. El término "cadena de caracteres" se abrevia frecuentemente por "cadena".
3. Una variable de cadena termina siempre por **$**.
4. Los nombres, como por ejemplo *palabra$*, suelen pronunciarse algunas veces como palabradólar.
5. La palabra clave **LEN** da la longitud o número de caracteres de una cadena. Por ejemplo, si la variable *carne$* tiene el valor "solomillo", la instrucción:

   ```
   PRINT LEN(carne$)
   ```

   dará como salida 9
6. El símbolo que se utiliza para unir dos cadenas es **&**.
7. Los límites de una cadena pueden definirse con unas comillas o apóstrofes.
8. Las comillas no forman parte de la cadena y no se almacenan.
9. La función es **CHR$**, que debe utilizarse con paréntesis, como por ejemplo en **CHR$(66)** o en **CHR$(RND (65 TO 67))**.
10. Para generar letras aleatorias, utilice instrucciones del tipo de la siguiente:

    ```
    código = RND(65 TO 90)
    PRINT CHR$(código)
    ```

COMPRUEBE SU PUNTUACION

9 ó 10 es muy buena puntuación. Continúe leyendo.

7 u 8 es una puntuación buena, pero vuelva a leer ciertas zonas del capítulo cuatro.

5 ó 6 es suficiente, pero relea algunas partes del capítulo cuatro y vuelva a hacer de nuevo el test.

Por debajo de 5 debe estudiar cuidadosamente el capítulo cuatro y repetir el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 5

1. Las letras minúsculas que se utilizan en los nombres de las variables o bucles, contrastan con las palabras clave, las cuales se visualizan al menos con parte de su nombre en mayúsculas.
2. Con el cambio de margen se evidencia claramente el contenido de los bucles (y otras estructuras).
3. Los identificadores (nombres) deben elegirse normalmente para que tengan un cierto significado, por ejemplo, *cuenta palabra$*, son mejores que *C* o *W$*.
4. Puede editar un programa almacenado:

   sustituyendo una línea
   insertando una línea
   borrando una línea (tres modos)
5. La tecla **ENTER** debe utilizarse para introducir un comando o una línea de un programa.
6. La palabra **NEW** borra cualquier programa previo escrito en SuperBASIC de su QL, asegurándole que el nuevo programa que introduzca no se mezcla con ningún otro ya existente.
7. Si desea que una línea se almacene como parte de un programa deberá utilizar un número de línea.
8. La palabra **RUN** seguida de ◆ hace que el programa se ejecute.
9. La palabra **REMark** es la que le permite colocar información en un programa y que ésta se ignore en el momento de la ejecución.
10. Las palabras clave **SAVE** y **LOAD** permiten que los programas puedan almacenarse y recuperarse de los cartuchos.

COMPRUEBE SU PUNTUACION

De 12 a 14 es muy buena. Continúe leyendo.

10 ó 11 es buena, pero vuelva a releer ciertas zonas del capítulo cinco.

8 ó 9 es suficiente, pero vuelva a leer el capítulo cinco y haga de nuevo el test.

Por debajo de 8, debe leer de nuevo el capítulo cinco cuidadosamente y volver a realizar el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 6

1. No resulta fácil pensar muchos nombres diferentes para almacenar los datos. Aunque pueda pensar todos estos nombres, debe escribirlos todos mediante las instrucciones **LET** o **READ**, si no utiliza matrices.
2. Mediante un número llamado subíndice, que forma parte del nombre variable de una matriz. Todas las variables de una matriz llevan el nombre, pero cada una de ellas lleva un subíndice diferente.
3. Para declarar una matriz debe decirse su tamaño (dimensión) en una instrucción **DIM** que normalmente se coloca cerca del comienzo del programa.
4. El número que distingue un elemento de una matriz se llama subíndice.
5. Por ejemplo las casas de una misma calle, comparten el nombre de la calle, pero cada una de ellas tiene su propio número.

   Las camas de una determinada sala en un hospital pueden compartir el número de sala, pero cada una de ellas está numerada.

   Las celdas de una determinada galería de una cárcel, tienen de común el nombre de la galería pero cada una de ellas tiene su propio número.

   Los agujeros de una partida de golf. Ejemplo, el quinto agujero del Open de España en El Saler.
6. Los bucles **FOR** terminan cuando se ha completado el último proceso correspondiente al último valor de la última variable del bucle.
7. El nombre de un bucle **FOR** es también el de la variable que controla el bucle.
8. Las dos frases que definen esta variable son "variable de bucle"o "variable de control".
9. Los valores de una variable de bucle pueden utilizarse como subíndices para los nombres variables de las matrices. De este modo, durante la ejecución del bucle, cada elemento de la matriz se "visita" una sola vez.
10. Tanto los bucles **FOR** como los bucles **REPeat:**
    a) Tienen una palabra de apertura:

       REPeat, FOR

    b) Tienen una instrucción para su cierre:

       END REPeat *nombre,* **END FOR** *nombre*

       Tienen un nombre de bucle.

d) Sólo los bucles **FOR** tienen variables de bucle o variables de control.

COMPRUEBE SU PUNTUACION

Este test es más analítico que los anteriores.

15 ó 16 puntos es excelente. Continúe leyendo.

13 ó 14 es una puntuación muy buena, pero debe meditar algo más algunas ideas. Vea los programas y estudie como funcionan.

11 ó 12 es una puntuación buena, pero relea algunas zonas del capítulo seis.

De 8 a 10 es suficiente, debe, sin embargo, leer algunas zonas del capítulo seis y volver a hacer el test.

Por debajo de 8 debe releer el capítulo seis cuidadosamente y rehacer el test.

RESPUESTAS AL AUTOEXAMEN SOBRE EL CAPITULO 7

1. Normalmente los trabajos largos y complejos se dividen en tareas menores, de un tamaño tal que se puedan realizar totalmente.
2. Este principio puede aplicarse en programación, partiendo el trabajo total y escribiendo procedimientos que realicen cada tarea.
3. Un procedimiento sencillo es

   Un bloque de código separado,
   Con un nombre apropiado

4. La llamada a un procedimiento realiza los siguiente:

   Activa el procedimiento.
   Devuelve el control inmediatamente después del punto de llamada

5. Los nombres de los procedimientos pueden usarse en un programa principal antes de escribir los propios procedimientos. Con ello el programador puede pensar en la totalidad del trabajo y tener una visión de conjunto sin preocuparse por los detalles.
6. Si se escribe un procedimiento antes de utilizar su nombre, puede probarse, y si funciona adecuadamente podremos olvidar los detalles. Sólo necesitaremos recordar su nombre y muy superficialmente lo que realiza.
7. Un programador que puede escribir programas de treinta líneas puede partir un trabajo complicado en procedimientos, de forma que ninguno de ellos tenga más de una treintena de líneas, y la mayoría incluso serán menos. De este modo, sólo necesitará preocuparse de una porción del trabajo en cada momento.
8. La utilización de un procedimiento ahorrará espacio de memoria si es necesario llamarlo más de una vez desde partes diferentes del programa. Los procedimientos sólo se definen una vez, pero pueden llamarse tantas veces como sean necesarios.
9. El programa principal puede colocar la información en "casilleros", que son accesibles por el procedimiento. De esta forma el procedimiento utiliza la información establecida originalmente por el programa principal.

   Otro segundo método es utilizar parámetros en la llamada al procedimiento. Estos valores se transfieren a las variables en la definición del procedimiento, que los utiliza cuando son necesarios.
10. Un parámetro actual es el valor actual que es transferido desde la llamada a un procedimiento, en el programa principal, al propio procedimiento.
11. Un parámetro formal es una variable de la definición de un procedimiento que recibe el valor que envía el programa principal al procediiento.

COMPRUEBE SU PUNTUACION

Este test es más analítico que los anteriores. Puede que necesite una experiencia mayor en el uso de los procedimientos antes de apreciar completamente todas las ideas. Sin embargo, son ideas que una vez comprendidas resultan muy útiles y le conferirán gran potencia. Todo el esfuerzo necesario en su comprensión tendrá su recompensa.

De 12 a 14, excelente. Siga leyendo con confianza.

10 u 11 muy bien. Compruebe sólo ciertos puntos.

8 ó 9 bien, pero relea ciertas zonas del capítulo siete. Trabaje con cuidado sobre los programas.

6 ó 7 suficiente, pero vuelva a leer ciertas partes del capítulo siete. Trabaje cuidadosamente sobre los programas, escribiendo todos los cambios de valores de las variables. A continuación vuelve a hacer el test.

Por debajo de 6, lea de nuevo el capítulo siete, y tómese el tiempo necesario en cada programa. Las ideas que contienen no son fáciles, pero merecen un esfuerzo. Cuando se considere preparado vuelva a hacer el test.