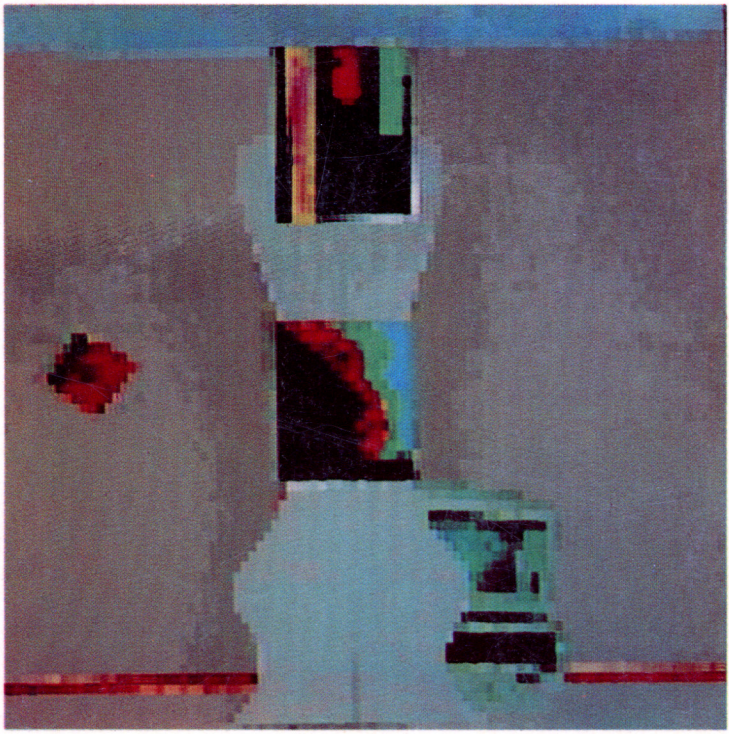
GRAN BIBLIOTECA

AMSTRAD



LENGUAJE BASIC II

UN INTÉRPRETE EFICAZ

EDICIONES INGELEK

GRAN BIBLIOTECA AMSTRAD

15

LENGUAJE BASIC II

Director editor:
Antonio M.ª Ferrer Abelló

Director de producción:
Vicente Robles

Director de la obra:
Fernando López Martínez

Redactor técnico:
Antonio García Verdugo

Colaboradores:
L-H Servicios Informáticos
Pilar Manzanera Amaro

Diseño:
Bravo/Lofish

Maquetación:
Carlos González Amezúa

Dibujos:
José Ochoa

Fotografía:
Grupo Gálata

ISBN del tomo: 84-7708-064-X.
ISBN de la obra: 84-7708-004-6.
Fotocomposición: Andueza, S. A.
Imprime: Eurosur, S. A.
Depósito Legal: M. 16.163-1987.
Precio en Canarias, Ceuta y Melilla: 435 ptas.

Agosto 1987

LENGUAJE BASIC II

INTRODUCCIÓN

En el octavo volumen de esta GRAN BIBLIOTECA AMSTRAD iniciamos nuestra andadura hacia el conocimiento del BASIC, el lenguaje de mayor importancia en la actualidad, y prácticamente imprescindible para los usuarios AMSTRAD, dado que en todos sus modelos se acompaña gratuitamente un intérprete del mismo; ya sea en ROM o disco.

Ha llegado el momento de perfilar nuestros conocimientos sobre este lenguaje, estudiando en más detalle aquellas instrucciones propias de cada aparato, motivo por el cual este libro se halla dividido en dos partes bien diferenciadas: una de ellas dedicada por entero a los PCW y otra a los CPC.

En el primer caso, y dada la completa compatibilidad del Mallard BASIC para PCW 8256 y 8512, su tratamiento se hace más simple. No obstante, el hecho de que el intérprete para los CPC 664/6128 presente algunas mejoras respecto al de los 464/472, puede llevar en ocasiones a cierta confusión. Por este motivo, conviene que los poseedores de un CPC 464 ó 472, se dirijan previamente al apéndice V del

final del libro, en el cual encontrarán una lista de las palabras reservadas que su aparato no implementa.

Asimismo, determinadas instrucciones de campos muy concretos, como pueden ser la música o los gráficos, han sido dejados al margen, intencionadamente, pues recibirán un tratamiento más amplio en volúmenes de esta obra, dedicados a su estudio concreto.

INTRODUCCIÓN AL BASIC PCW

El PCW es una máquina vacía. Esto significa que no tiene memorias ROM, y por lo tanto, al encenderlo necesita cargar el sistema operativo CP/M. Mallard BASIC es un programa más de los accesibles por CP/M, por lo que para trasladarlo a la memoria, debemos teclear:

A>basic<RETURN>

Aparecerá entonces el mensaje de *copyright,* la versión y la memoria disponible para ejecución de programas y almacenamiento de variables. El aspecto de la pantalla queda entonces así:

```
Mallard-80 BASIC with Jetsam Version X.XX
© Copyright 1984 Locomotive Software Ltd
All rights reserved
31597 free bytes
Ok
_
```

El «80» se refiere a los microprocesadores a los cuales va destinado el programa. Son el 8080 Intel y el Z80 Zilog, ambos de ocho bits. Este último es el que tiene nuestro PCW.

Jetsam es un sistema especial de gestión de datos a través de ficheros indexados. Lo estudiaremos detenidamente más adelante, pero anticipamos ya que se trata de un sistema extremadamente interesante, que hace más útil y eficaz esta adaptación de BASIC.

La versión que poseamos no ha de preocuparnos demasiado. Normalmente, los cambios de una a otra son internos (ahorro de memoria, rapidez de ejecución...), cualquier alteración notable sería indicada en el manual.

La memoria libre no varía si utilizamos un PCW8512 en lugar de un PCW8256, ya que ésta depende del sistema operativo CP/M. En efecto, CP/M alberga a BASIC en un bloque de 64K (se trata de la máxima cantidad de memoria controlable por el Z80 a la vez) y lo que queda libre de ese bloque es lo que indica BASIC al comenzar su ejecución. Tendremos que conformarnos con 30.8K para nuestros programas o variables, o quizá un poco más.

CARACTERÍSTICAS

Mallard BASIC es un intérprete muy rápido. La capacidad de cálculo resulta de gran utilidad en programa con aplicación científica; la precisión será suficiente en la mayoría de los casos, aunque a veces se producen ciertos errores.

Aprovechando el especial trato que se da a parte de la memoria del PCW (lo que se llama disco virtual), podemos sacar aún más provecho a la hora de procesar y ordenar ficheros, puesto que el acceso a la memoria RAM es incomparablemente más rápido.

Si añadimos a eso la capacidad para manejar gran número de ficheros a un mismo tiempo, y la posibilidad de gestión por claves, podemos asegurar que el PCW, a través del BASIC, se presta a gestionar datos con la misma eficacia que un ordenador de la serie PC.

El punto negro (en cierto modo) de este intérprete es la ausencia de comandos gráficos, debido a su compatibilidad con otras máquinas muy distintas. Pero debemos olvidar estas posibilidades, más propias de ordenadores «videojuego» y aprovechar caracteres gráficos, como los que Locoscript utiliza, para nuestros propios programas, obteniendo una presentación tan estética y profesional como la de este procesador de textos.

TECLADO EN BASIC

La configuración del teclado ejecutando BASIC no varía mucho de la de CP/M. Las pulsaciones <RETURN>, <INTRO> y <ALT>+<M> producen el mismo efecto. <ALT>+<S> detiene la ejecución de BASIC temporalmente, y <ALT>+<C> o <STOP> detienen los programas o listados.

Tal como ocurre en Locoscript, las pulsaciones <ALT>+<INTRO> y <ALT>+<JUST> provocan el bloqueo de mayúsculas y teclado numérico, respectivamente.

En el apéndice IV se explican con detalle todas las órdenes del teclado en BASIC, sin reconfirgurarlo previamente.

OPERACIONES NUMÉRICAS. REPRESENTACIONES

En el anterior volumen sobre BASIC de esta misma colección (número 8) ya se dio un repaso a todas las funciones numéricas, válidas para la versión PCW. Sin embargo, dadas las posibles aplicaciones de este ordenador, es conveniente reincidir en las funciones trigonométricas TAN, SIN y COS (tangente, seno y coseno), cuyo argumento sólo puede expresarse en radianes, y sin superar un valor absoluto de aproximadamente 200.000, ya que de lo contrario se provoca el error *Improper Argument.*

La función TAN no admitirá, como es de suponer, los valores para los cuales la función no existe: pi/2, 3pi/2 y múltiplos de estas expresiones. Se emitirá el mensaje «Overflow» y el resultado será el máximo valor admisible para BASIC: 1.701412e38.

La única función inversa disponible es el arco tangente (ATN), que devuelve el ángulo en radianes. Para obtener el valor de pi, no incluido en este BASIC, se puede utilizar la expresión:

```
pi=ATN(1)*4
```

o bien,

```
pi=ATN(1E38)*2
```

Dado que el PCW puede tener importantes aplicaciones en el cálculo, incluimos una tabla para obtener cualquier función trigonométrica a partir de las implementadas.

OBTENCION DE FUNCIONES TRIGONOMETRICAS

SECANTE	1/COS(X)
COSECANTE	1/SIN(X)
COTANGENTE	1/TAN(X)
ARCO SENO	ATN(X/SQR(−X*X+1))
ARCO COSENO	ATN(X/SQR(−X*X+1))+1.5708
ARCO SECANTE	ATN(X/SQR(−X*X+1))+SGN(SGN(X)−1)*1.5708
ARCO COSECANTE	ATN(X/SQR(−X*X+1))+(SGN(X)−1)*1.5708
ARCO COTANGENTE	ATN(X)+1.5708

Representación De Los Números

Si queremos dar valores al ordenador en distintas bases, podemos hacerlo. En concreto, pueden expresarse valores en base 8 y en base 16. Está claro que en principio BASIC no los distinguiría de los entregados en base diez, y por lo tanto son necesarios unos prefijos.

El signo «&», común a las dos notaciones, previene a BASIC de que lo que a continuación se indica es un número en notación no decimal. Para especificar un valor octal, escribiremos una «O», y en el caso de un valor hexadecimal, una «H». Por ejemplo, para expresar los valores FFFE (hexadecimal) y 1322 (octal), escribiremos:

```
print &hfffe
print &o1322
```

Estas notaciones están acotadas. Se opera internamente con dos bytes, y por lo tanto los valores mínimo y máximo son − 32767 y 32768 respectivamente. Si se quiere operar sólo con valores positivos, habrá que sumar 65536 al resultado negativo.

Según esta última observación, el valor máximo admisible en hexadecimal es &hFFFF, y en octal &o177777. Ambos dan el valor en base diez − 1, o bien 65535 si se pasa a positivo.

También podemos convertir un valor en decimal a octal o hexadecimal, pero dado que tales expresiones no tienen un valor directamente asimilable por el intérprete, el resultado de las funciones es una cadena en lugar de un número. Las funciones que convierten son OCT$(X) y HEX$(X).

Por ejemplo, con:

print oct$(256)

obtendremos como resultado la cadena «400».

print hex$(256)

dará «100».

```
10 INPUT "Valor decimal ";n
20 IF n>65535 THEN PRINT "Valor no adecuado.":STOP
30 res$=""
40 FOR a=15 TO 0 STEP -1
50 IF n>=2^a THEN res$=res$+"1":n=n-2^a ELSE res$=res$+"0"
60 NEXT
70 WHILE LEFT$(res$,1)="0"
80 res$=RIGHT$(res$,len(res$)-1)
90 WEND
100 PRINT res$
110 STOP
```

```
10 INPUT "Valor binario ";a$
20 FOR a=LEN(a$) TO 1 STEP -1
30 IF INSTR("10",MID$(a$,a,1)=0 THEN PRINT "Valor no adecuado"
   STOP
40 IF VAL(MID$(LEN(a$)-a+1,1))=1 THEN n=n+2^(a-1)
50 NEXT
60 PRINT n
```

Más de una vez nos hará falta una función para manejar valores en base 2. No existen tales funciones, pero podemos sustituirlas por unas subrutinas, como las que aquí mostramos.

Estos programas contienen muchas precauciones. El paso de binario a decimal puede hacerse de manera mucho más sencilla, aunque, eso sí, con estrictas normas de entrada. Por ejemplo, la función

```
10 DEF FN a(a$)=val(RIGHT$(a$,1)+2*VAL(MID$(a$,3,1)
+4*VAL(MID$(a$,2,1))+8*VAL(LEFT$(a$,1))
```

admite cadenas de 4 caracteres (representando 4 bits) para pasar a decimal. Si la cadena no contiene 4 caracteres el cálculo será erróneo.

CONTROL Y REPRESENTACIÓN DE VARIABLES

Las operaciones numéricas con Mallard BASIC son un tanto especiales. Se distinguen tres tipos de valores internamente: enteros, de precisión sencilla, y de doble precisión. BASIC trabaja implícitamente con valores en precisión sencilla.

Para que BASIC distinga las características de un número, es posible añadirle un sufijo que designa la precisión con la cual ha de ser tratado. Estos sufijos son «%», «!» y «#», para valores enteros, sencillos y de doble precisión, respectivamente.

Para los primeros, utiliza como almacenamiento dos bytes; los valores extremos son, por tanto, − 32767 y 32768. Los valores con precisión sencilla ocupan cuatro bytes, lo cual permite que tengan exponente y decimales; en total, siete cifras significativas. La doble precisión se consigue ocupando ocho bytes, permitiendo 16 cifras significativas.

Seguro que los sufijos «!» y «#» nos han aparecido en alguna ocasión en un listado sin haberlos tecleado, y no hemos sabido muy bien

por qué. La razón es que BASIC indica al programador si el número ocupa cuatro u ocho bytes con el correspondiente signo. Esto se hace sólo si el número está tan cerca del límite de los cuatro bytes, que no sabemos muy bien si es así o no. En tal caso BASIC añade al número un «!».

Si el número sobrepasa el límite de los cuatro bytes, llevará el signo «#». Un ejemplo: si tenemos un listado con la línea

200 print 356

al listarla, BASIC no añadirá nada. Sabemos perfectamente que se trabaja implícitamente con precisión sencilla, y esta cifra está en ese margen, lo comprobamos a simple vista. Si, por el contrario, tenemos

200 print 3987459

al listarlo, aparecerá una admiración al final del número. Realmente es difícil saber si esa cantidad supera los cuatro bytes, y es por eso que BASIC nos saca de dudas colocando ese signo. Con

200 print 1234567890

BASIC añadirá el signo «#» al final. Esto se debe a que el número sobrepasa los límites (tiene 10 cifras) y BASIC sólo puede almacenarlo tal cual en ocho bytes. Si quisiéramos forzar a BASIC a guardarlo en cuatro, deberíamos añadir nosotros mismos al teclear la línea el signo de precisión sencilla. Tecleemos:

```
200 print 123456789!
list
200 print 1.234568E+09
Ok
```

esto significa que BASIC nos ha obedecido pero, claro está, sacrificando el número de cifras significativas debido a las características de «!».

Vamos con unos ejemplos más generales. Para operar con enteros, se utiliza el «%». Hagamos la prueba:

print 34.234%

el resultado es 34. BASIC tiene la «delicadeza» de redondear en lugar de cortar, antes de almacenar en los dos bytes correspondientes a estos valores. Si probamos con un valor fuera de rango, como:

print 43000%

obtendremos el doble mensaje *overflow-syntax error* debido el primero al desbordamiento y el segundo a un posterior intento de interpretación, erróneo.

Estos mismos sufijos pueden utilizarse en el manejo de variables, es decir, la variable a% almacenará valores enteros, *gana!* valores en precisión sencilla, y *double#* valores en doble precisión. Tal y como ocurre con variables numéricas y alfanuméricas, BASIC distingue perfectamente entre tipos, por lo tanto, *a%*, *a!* y *a#* son tres variables distintas, y pueden coexistir en un solo programa (dificultando la comprensión del mismo, eso sí).

En programas muy largos, comprobaremos que escribir sufijos se convierte en una tarea excesivamente pesada. Para evitarlo, BASIC dispone de unas órdenes de definición de tipos de variables. Estas son DEFSTR, DEFINT, DEFSNG, DEFDBL. Tras éstas se indica una lista de letras, separadas por comas, o por guiones, para indicar un margen. Las variables afectadas entonces para estas órdenes son todas aquellas cuya letra inicial coincida con las especificadas.

Por ejemplo, para indicar que la a, d y f son iniciales afectadas, se escribirá *a,d,f*. Si queremos un margen de letras de la b a la f, escribiremos *b-f*. Se pueden indicar distintos márgenes, o letras sueltas, siempre separados por comas. En el primer ejemplo, las variables afectadas son las que empiezan por a, d o f. En el segundo caso, por b, c, d, e y f.

DEFSTR (DEFine STRing, definir como cadena) hace que la variable o variables especificadas sean alfanuméricas.

```
defstr d,e
```

hace que las variables cuyo nombre empiece por d o e sean alfanuméricas sin tener que indicar el prefijo, y por tanto las asignaciones

```
dia="ACB"
espacio=chr$(32)
```

que en otro caso provocarían el error «Type Mismatch», serán aceptadas.

Debemos tener en cuenta la confusión que esta operación provoca; en un listado, veremos una asignación que en principio parece incorrecta, pero no lo es, gracias al correspondiente DEF. Es una buena práctica el utilizar siempre las mismas letras para los tipos, a fin de acostumbrarnos y evitar errores.

DEFDBL (DEFine DouBLe, definir como doble —precisión—) tiene la misma forma general y el mismo resultado que DEFSTR en cuanto a nombre de variables afectadas, que en este caso ocuparán ocho bytes para obtener la doble precisión numérica. Igualmente, DEFSNG (DEFine SiNGle, definir como sencilla) y DEFINT (DEFine INTeger, definir como enteras) hacen que las variables sean implí-

citamente tratadas como de precisión sencilla y enteras, respectivamente.

Así, al asignar un valor con decimales a una variable definida como entera, el número se redondeará antes de almacenarlo. Para comprobarlo basta con teclear esta secuencia:

```
defint a
Ok
a=34.23879
Ok
print a
 34
```

La definición de tipos de variables puede resultar muy útil en algunas ocasiones. Por ejemplo, en un programa que no efectúe ninguna clase de cálculo y se dedique exclusivamente a numerar registros, etc... será un acierto el definir todas las variables numéricas como enteras. Se consigue un considerable ahorro de memoria y rapidez en la ejecución.

INTERCAMBIO DE VARIABLES

En muchas ocasiones necesitamos intercambiar el contenido de dos variables (del mismo tipo, por supuesto). La operación se realiza normalmente utilizando una variable intermedia (que, por lo tanto, debe ser del mismo tipo) en la cual guardar un contenido.

Por ejemplo, para cambiar el valor que contiene **a%** por el de **b%**, tomaremos una variable, pongamos **x%**, para el traslado:

```
x%=a% : a%=b% : b%=x%
```

BASIC nos ofrece una manera mucho más sencilla de realizar esta operación, ahorrando además la memoria que ocupa **x%**. Se trata de **SWAP v1,v2,** donde **v1** y **v2** son variables del mismo tipo. Con esta orden, el intercambio anterior se habría llevado a cabo de la siguiente manera:

```
swap a%,b%
```

Como podemos suponer, la utilización de **SWAP** puede dar lugar a algunas confusiones. Como ya dijimos, los tipos de las variables han de ser los mismos, de manera que una expresión como

```
swap a%,b!
```

provocará un error (**Type mismatch**). Este problema se complica si

además hemos definido los tipos de alguna variable. Por ejemplo, tras la secuencia

```
defint d
d=34
e=35
```

una expresión como **swap d,e** provocará el error entre tipos, debido a que **e** es una variable en precisión sencilla (ya que no hemos indicado prefijo) y **d** es entera. La confusión se debe a que ambas tienen asignado un tipo implícitamente y eso no se refleja claramente en el listado.

También podemos intercambiar el contenido de variables alfanuméricas (**swap a$,b$**) y podemos incluir también variables con subíndice, siempre que cumplan las condiciones necesarias. Por ejemplo, si no se han definido tipos, serán válidas las siguientes operaciones:

```
swap a%,d%(3,2,1)
swap a$,d$(10)
swap j(i),j(i+1)
swap uno#,dos#(2,3)
```

ELIMINAR EL SUBÍNDICE 0

Ya comentábamos en el anterior número sobre BASIC que en algunas ocasiones es más cómodo para la codificación del programa no utilizar el subíndice 0. Pero esto significa que desaprovechamos un espacio de memoria, que será más grande cuanto mayor sea el número de dimensiones de la variable.

El BASIC del PCW dispone de una orden nada usual entre otros ordenadores, y que permite elegir entre el 0 y el 1 como primer índice de las matrices. La forma general es

```
option base n
```

n es 0 ó 1. Si indicamos un número mayor que 1, se provoca el error **Syntax error.** Si, por el contrario, se da un argumento negativo, aparece **Improper argument.**

OPTION BASE debe ser ejecutado ANTES que cualquier **DIM,** ya que es el primero el que determina las características del segundo. Así, si ya hemos dimensionado una variable y a continuación tecleamos **option base 1,** el intérprete emite el error **array already dimensioned** (variable ya dimensionada). Tampoco está permitido ejecutar dos **OPTION BASE** seguidos, aunque indiquen el mismo número:

option base 1
Ok
option base 1
Array already dimensioned
Ok

Para evitar esta clase de errores al intentar reestructurar nuestras variables con subíndice, habrá que escribir la orden **CLEAR,** pero teniendo en cuenta que ésta borra todas las variables.

Aunque borremos las matrices con **ERASE,** el efecto de **OPTION BASE** sigue activo, y por lo tanto esta operación no evitará el error.

ASIGNACIONES «JUSTIFICADAS»

Dentro del control y modificación de variables, debemos incluir este tipo de asignaciones y de las conversiones cadena-número. Todavía no son de gran utilidad, pero sí imprescindibles para el manejo de ficheros, que trataremos más adelante.

LSET y **RSET** (que significan colocar a la izquierda y a la derecha, respectivamente) son órdenes que transfieren el contenido de una cadena a otra, pero justificando por uno u otro lado. La forma general es:

LSET var=expresión
RSET var=expresión

var es una variable alfanumérica y la *expresión* puede ser una variable del mismo tipo, una cadena, o una expresión de operaciones con cadenas.

Podemos comprobar el efecto de estas instrucciones con la siguiente secuencia, en la cual hemos escrito «e» para designar un espacio:

```
a$="eeeeeeeeee"
lset a$="Aquí"
print "/";a$;"/"
rset a$="Aquí"
print "/";a$"/"
```

Las barras inclinadas nos ayudan a «ver» los espacios. En la primera asignación, *Aquí* se ha colocado a la izquierda, y como la cadena tenía 10 caracteres, se ha llenado de espacios hasta llegar a esa cantidad. En el segundo ejemplo, se han puesto los espacios por la izquierda, para que *Aquí* quede exactamente justificado a la derecha.

En la siguiente secuencia vamos a asignar cadenas mayores que la longitud de la variable destino:

```
b$="eeeee"
lset b$="Cortado"
print b$
rset b$="Cortado"
print b$
```

En este ejemplo, se justifica por la izquierda, pero es necesario cortar por la «a» para ajustarse a los cinco caracteres de la variable destino. El resultado, por tanto, es *Corta.* **RSET** no tiene espacio suficiente ni para colocar la cadena, y se queda donde agota el espacio. El resultado también es *Corta.*

Estas dos instrucciones se utilizan para preparar campos en los ficheros. De todas formas, **RSET** nos puede servir para justificar por la derecha algún texto que así lo requiera, evitándonos lo que de otro modo habría que hacer midiendo la longitud del texto, etc. Incluimos un ejemplo.

```
10 REM PROGRAMA PARA COMPROBAR EL EFECTO DE 'ZONE' AL TABULAR
20 REM CON COMAS.
30 FOR A=4 TO 20
40 ZONE A
50 PRINT A,A,A
60 NEXT
70 ZONE 8
80 PRINT "ESO ES TODO"
```

CONVERSIONES

En algunas ocasiones los números no pueden almacenarse en los ficheros tal como están; hay que convertirlos en cadenas. Ya sabemos cómo hacerlo, con **STR$,** pero este método no es el más adecuado. Para empezar, la cadena resultado ocupa mucho espacio. Por ejemplo, el número 21234 pasa a ser «21234», cadena de 5 caracteres que por tanto ocupa 5 bytes, mientras que internamente sólo necesita dos. Además, existe el riesgo de perder precisión al efectuar los cambios.

Para hacer conversiones correctas, BASIC dispone de cinco funciones para pasar de número a cadena, y otras cinco para deshacer la conversión. Empezaremos por las de menor precisión.

MKI$(n) (de *make integer,* hacer entero) convierte un número entero en una cadena de dos bytes ilegible, por lo que debe guardarse

en una variable alfanumérica y no imprimirse nunca. *n* debe ser un número entero del margen − 32767 a 32768. Para obtener el número a partir de la cadena se utiliza **CVI(c),** donde *c* es la cadena producto de **MKI$.**

MKIK$(n) (hacer entero para clave) realiza la misma operación que **MKI$,** pero la cadena resultado tiene otras características distintas, destinadas a la ordenación. La función que recupera el número es **CVIK(c).**

MKUK$(n) (hacer entero sin signo para clave) es igual a las anteriores funciones excepto en el margen de *n,* el cual debe estar entre 0 y 65535. La función contraria es **CVUK(c).**

MKS$(n) convierte un número de precisión sencilla en una cadena de cuatro bytes. Si el número especificado entre paréntesis no está en precisión sencilla, se fuerza a ésta, perdiendo las correspondientes cifras significativas. La cadena resultado es también ilegible. Para obtener de nuevo el valor se utiliza la función **CVS(c),** donde *c* es la cadena resultado.

MKD$(n) convierte un número de doble precisión en una cadena de ocho bytes. *n* se fuerza a doble precisión si no se ha especificado así. De la cadena resultado, ilegible, se obtiene de nuevo el valor con **CVD(c).**

Los ejemplos del funcionamiento de estas funciones nunca serán muy claros, puesto que el resultado de las mismas es un conjunto de varios caracteres que no siempre es imprimible. Lo normal es almacenar estas cadenas en variables.

```
entero$=mki$(100)
print cvi(entero$)
doble$=mkd$(123.456789012)
valor=cvd(doble$)
print valor
```

El Octavo Bit de los Caracteres

La última función dedicada a convertir cadenas es **STRIP$(c)** (despojar) y su resultado es una cadena como la dada, pero suprimiendo el octavo bit de todos los caracteres que la forman (es decir, poniendo a cero ese bit). Se fuerza así a todos los caracteres a situarse en margen de la tabla ASCII (del 0 al 127).

En efecto, al eliminar el valor más significativo del byte, la expresión máxima alcanzable en binario es 1111111 (siete unos) equivalente

al valor 127. Normalmente no es de mucha utilidad, pero nosotros la manejaremos al hablar de **FIND$,** en el capítulo dedicado a las operaciones básicas de disco.

No es posible presentar un ejemplo claro de **STRIP$,** porque apenas hay caracteres accesibles por el teclado cuyo código sea superior al 127.

```
a$=chr$(161)
b$=strip$(a$)
print asc(b$)
```

el resultado será 161 − 128 = 33, que en la tabla ASCII es el signo de admiración (!). Partimos del carácter 161 (o elevada) que no es estándar ASCII.

CONTROL DE EJECUCIÓN

n este capítulo estudiaremos casi exclusivamente nuevas órdenes y órdenes ampliadas dedicadas a una mayor estética durante la ejecución del programa.

Empezaremos por indicar dos intrucciones no imprescindibles, pero que pueden resultar útiles durante la ejecución del programa. Se trata de **OPTION RUN** y **OPTION STOP.**

La primera impide que el programa pueda ser interrumpido (**ALT+S** y **ALT+C** no tendrán efecto). Esto puede servirnos para impedir una detención en una parte importante del programa (transferencia de datos, tratamiento de ficheros temporales...) o bien para que sea imposible acceder al listado del programa, con lo cual queda totalmente protegido.

Para desactivar este modo utilizaremos **OPTION STOP.** Hay que tener gran cuidado al impedir una interrupción, puesto que un programa erróneamente codificado, con algún bucle indefinido, no parará nunca y habrá que reinicializar el ordenador, perdiendo el programa si no tenemos copia de él.

TECLADO

En el primer libro de BASIC de esta colección, se introdujo la utilización de **INPUT** para captar valores numéricos y alfanuméricos durante la ejecución del programa, para asignarlo a una variable, la forma general indicada entonces será:

INPUT "comentario"; variable

En esta expresión se puede omitir el comentario (y también entonces el signo de punto y coma). En lugar de una variable, es posible indicar una lista de variables, en cuyo caso se teclean entre comas. Para introducir datos correctamente, se separarán también entre comas (todos en la misma línea); en caso contrario, se emitirá el error

?Redo from start

El comentario debe dejar bien claro el hecho de que es necesario separar los datos entre comas.

INPUT tiene su variante estética: cuando lo utilizamos como hasta ahora, aparece una interrogación delante del cursor. Por ejemplo

INPUT "dato 1 ";a

provoca como respuesta

Dato1 ?—

Para evitar la interrogación, en lugar de punto y coma (;) el separador que escribiremos será la coma (,) que, en este caso, no tiene el significado de tabulador, asociado a PRINT en otras ocasiones.

Dado que **INPUT** admite introducir más de una variable en la misma línea, la coma tiene un significado especial al teclear los datos. Esto representa un problema en el caso de que queramos introducir una coma en alguna variable alfanumérica.

Por ejemplo, si queremos introducir:

Come espinacas, hombre

bajo la instrucción **INPUT a$,** obtendremos el mensaje **?Redo from start,** debido a que BASIC entiende que se trata de dos cadenas: *Come espinacas y hombre.*

Para poder introducir comas, se utiliza **LINE INPUT,** que, por lo demás, admite la misma sintaxis que el **INPUT** normal. Esta vez no podremos introducir más de una variable a la vez, puesto que la coma ya no tiene el significado especial de separador. En caso de intentarlo (por ejemplo **line input a$,b$)** obtendremos el mensaje *syntax error.*

La otra alternativa a **INPUT** es **INPUT$(n)** donde **n** es un núme-

ro entero dentro del margen 1-255 (en el caso de superar este rango se provoca el error Improper argument). **INPUT&(n)** genera una cadena de n caracteres, que son los introducidos por el teclado en el momento de ejecutarse la instrucción. Para recoger el resultado de la función **INPUT$(n)** habrá que asignarlo a una variable alfanumérica (aunque podemos operar con el resultado de **INPUT$(n)** de cualquier forma, si es correcta). Por ejemplo,

a$=INPUT$(1)

Al teclearlo, el ordenador espera a que se pulse una tecla; hecho ésto, se asigna a **a$.** La expresión:

b$=INPUT$(10)

hace que el intérprete recoja las 10 pulsaciones siguientes, asignándolas a **b$.** Esta función es útil para introducir claves secretas, puesto que no se muestra la entrada en pantalla, tan sólo el cursor aparece en ella como señal de que es necesario introducir los datos.

CONTROL DE PANTALLA. COMANDOS

El PCW trabaja normalmente con 90 columnas en pantalla. Podemos limitar esa anchura por la derecha, por ejemplo a 30. La orden adecuada es WIDTH (ancho)

width 30

Tras teclear esto, listemos un programa en memoria, o escribamos algo. Cuando se llegue a la columna 30, se pasará a la fila siguiente. El resto de la pantalla se ignorará. También podemos hacer la anchura de la pantalla «infinita» con **WIDTH 255.** Esto hace que BASIC acepte muchos caracteres para la misma línea (más de 90) sin que baje a la siguiente línea por falta física de espacio para imprimirlos. Ello nos será de gran utilidad si enviamos una línea con caracteres de control (los trataremos más adelante) y texto, superando en conjunto los 90 caracteres. En realidad, dado que existen caracteres de control, no se sobrepasará la línea, pero eso BASIC no lo comprueba y pasa a la siguiente; WIDTH 255 lo evitará.

Muchas veces necesitaremos plasmar en la pantalla un cuadro con unas características determinadas. En la mayoría de las ocasiones, la división que efectúa PRINT en combinación con las comas para tabular, no nos será de gran ayuda. Necesitaríamos modificar la anchura de las columnas de tabulación, quizá estrechándolas. La instrucción

adecuada para ellos es **ZONE n,** donde **n** es un número entre 1 y 255 (se provocará el error **improper argument** si se sobrepasa).

n expresa la cantidad de columnas que debe separar **PRINT** cada vez que interpreta una coma.

```
10 REM PROGRAMA PARA JUSTIFICAR NUMEROS POR LA DERECHA
20 REM CON AYUDA DE RSET. PULSAR <STOP> PARA PARAR.
30 A$="          ": REM DIEZ ESPACIOS
40 RSET A$=STR$(INT(RND*RND*RND*1000000000))
50 PRINT A$
60 GOTO 40
```

Para controlar con más elasticidad la separación entre columnas, podemos utilizar **TAB(n),** donde *n* es un número entero mayor o igual que 1 (si es menor, se supone 1). **TAB** escribe los espacios necesarios para llegar hasta la columna especificada por *n* (si es necesario baja a la siguiente línea). A continuación tenemos un ejemplo de cómo trabaja **TAB,** con la ventaja de que no es necesario modificar continuamente las zonas de escritura con **ZONE.**

```
10 REM PROGRAMA PARA COMPROBAR COMO FUNCIONA TAB
20 FOR A=1 TO 30
30 PRINT TAB(A);"Arghs!"
40 NEXT
50 FOR A=30 TO 1 STEP -1
60 PRINT TAB(A);"Arghs!"
70 NEXT
80 END
```

En algunas ocasiones, necesitaremos un considerable número de espacios para colocar delante de un texto, para justificarlo de algún modo, separarlo, o simplemente para asignarlos a alguna cadena. Las funciones que generan espacios son **SPACE$(n)** y **SPC(n).** La primera está dedicada a crearlos «físicamente», es decir, de manera que pueden ser asignados a una cadena. **n** es un entero comprendido en el margen 0-255 (cualquier otro valor provoca un *Improper argument).*

a$=space$(30)

Esta línea asigna a la variable **a$** treinta espacios. Con **SPACE$** puede operarse normalmente, puesto que es una cadena.

Por el contrario, **SPC(n)** no genera los espacios físicamente, sino sólo para imprimirlos, es decir, **SPC** únicamente puede utilizarse junto con un **PRINT.** Tras **SPC** siempre se supone un punto y coma (;), aunque no se escriba, es decir, el elemento de la lista incluida en **PRINT** que siga a un **SPC** siempre aparece impreso tras los espacios que esta función genera.

Queda claro que no podemos asignar espacios a una cadena con ayuda de **SPC.** Expresiones del tipo

a$=spc(6)

siempre provocarán el error *Syntax error.* Un ejemplo de la correcta utilización de **SPC** es:

print «Valor:»;spc(8);va%

Si escribimos una coma en lugar de punto y coma, además de los espacios se pasará a la siguiente columna determinada por **ZONE** o a la correspondiente según las normas del tabulador.

Otra función relacionada con la disposición del texto en pantalla es **POS(a),** donde *a* es cualquier tipo de argumento (por ejemplo, 0). El resultado de esta función es el número de columna en la cual se encuentra situado el cursor de escritura (no se refiere al cursor con el que tecleamos nosotros sino al que lleva **PRINT).** Hagamos una prueba:

print pos(0)

BASIC devolverá 1. Si situamos el cursor de escritura previamente, el resultado será más interesante, por ejemplo:

print tab(20);pos(0)

El resultado será un 20, situado en la columna 20 porque es allí donde se encontraba el cursor de escritura cuando se ejecutó **POS.**

Esta función no es útil en general, excepto en aquellos programas en los cuales se mueva el cursor continuamente por la pantalla y se deba averiguar su posición exacta.

CONTROL DE PANTALLA. CÓDIGOS

Al contrario que otros ordenadores que se prestan más al juego y gráficos, el PCW no incorpora órdenes dedicadas a gestionar de modo directo la pantalla. Esto se debe, entre otras cosas, a que así Mallard BASIC es compatible con varias máquinas, y no sólo con Amstrad.

El modo de controlar las posibilidades de la pantalla es enviando códigos, identificados como modificadores del estado actual del monitor.

Todos los códigos se envían a través del comando **PRINT.** Podemos dividirlos en dos tipos, los directos, cuya expresión correcta incluye un solo código, y los compuestos, que se constituyen por código común y una secuencia que define la respuesta de la pantalla.

Estos códigos no pueden ser obtenidos por pulsaciones del tecla-

do, de manera que es necesario acudir a la función **CHR$.** Los códigos de control (así se les llama) están clasificados en la tabla ASCII por debajo del número 32. Algunos tienen un nombre específico, como por ejemplo **BEL.** Vamos a repasar uno por uno todos ellos.

Los códigos simples (un solo carácter) son:

CODIGO	FORMA DE ESCRIBIRLO	ACCION
BEL	PRINT CHR$(7)	Pitido
BS	PRINT CHR$(8)	Retroceso
LF	PRINT CHR$(10)	Avance
CR	PRINT CHR$(13)	Retorno carro
ESC 0	PRINT CHR$(27);«0»	Línea estado
ESC 1	PRINT CHR$(27);«1»	Línea estado
ESC 2 n	PRINT CHR$(27);«2»;CHR$(n)	Cambio ASCII
ESC A	PRINT CHR$(27);«A»;	Sube línea
ESC B	PRINT CHR$(27);«B»;	Baja línea
ESC C	PRINT CHR$(27);«C»;	Avance
ESC D	PRINT CHR$(27);«D»;	Retroceso
ESC E	PRINT CHR$(27);«E»;	Borra
ESC H	PRINT CHR$(27);«H»;	Coloca cursor
ESC I	PRINT CHR$(27);«I»;	Sube línea
ESC J	PRINT CHR$(27);«J»;	Borra
ESC K	PRINT CHR$(27);«K»;	Borra
ESC L	PRINT CHR$(27);«L»;	Inserta
ESC M	PRINT CHR$(27);«M»;	Borra
ESC N	PRINT CHR$(27);«N»;	Borra
ESC X a b c d	PRINT CHR$(27);«X»;CHR$(a); CHR$(b);CHR$(c);CHR$(d)	Define ventana
ESC Y f c	PRINT CHR$(27);«Y»; CHR$(32+f);CHR$(32+c);	Sitúa cursor
ESC b n	PRINT CHR$(27);«b»;CHR$(n)	Cambia colores
ESC c n	PRINT CHR$(27);«c»;CHR$(n)	Cambia colores
ESC d	PRINT CHR$(27);«d»;	Borra
ESC e	PRINT CHR$(27);«e»	Activa cursor
ESC f	PRINT CHR$(27);«f»	Inhibe cursor
ESC j	PRINT CHR$(27);«j»	Almacena cursor
ESC k	PRINT CHR$(27);«k»;	Recupera cursor
ESC l	PRINT CHR$(27);«l»;	Borra
ESC o	PRINT CHR$(27);«O»;	Borra
ESC p	PRINT CHR$(27);«p»	Vídeo inverso
ESC q	PRINT CHR$(27);«q»	Vídeo inverso
ESC r	PRINT CHR$(27);«r»	Subrayado
ESC u	PRINT CHR$(27);«u»	Subrayado
ESC v	PRINT CHR$(27);«v»	Continuidad

CODIGO	FORMA DE ESCRIBIRLO	ACCION
ESC w	PRINT CHR$(27);«w»	Continuidad
ESC x	PRINT CHR$(27);«x»	24x80
ESC y	PRINT CHR$(27);«y»	24x80

— **BEL** (7). Produce un pitido, único sonido disponible en el PCW. Si se manda a la pantalla sin un punto y coma al final, se pasa a la línea siguiente, aunque no sea imprimible.

— **BS** (8). Hace retroceder el cursor una posición a la izquierda. Si ya está en la primera, y **ESC v** permanece activo, pasa a la línea anterior, última columna.

— **LF** (10). Avanza una línea el cursor, y desplaza la pantalla si está en la última.

— **CR** (13). Retorno del carro: lleva el cursor a la primera columna de la línea actual.

Los códigos compuestos empiezan todos con un carácter, el 27 (llamado **ESC,** de escape). Los caracteres situados tras él, y que completan el código, se llaman en conjunto *secuencia de escape.* Estos son:

— **ESC «0».** Desactiva la línea de estado. Esta línea es la más baja de la pantalla, en la cual se indican los mensajes gestionados normalmente por CP/M (por ejemplo, «A:unidad no preparada...»). Al desactivarse dicha línea, resulta también utilizable. Los mensajes de CP/M, seguirán, por supuesto, mostrándose, pero en la línea en la cual correspondería imprimir al siguiente **PRINT.**

— **ESC «1».** Activa la línea de estado. Este código y el anterior no suelen ser de gran utilidad.

— **ESC «2» n.** Selecciona una de las variantes nacionales del juego de caracteres. Equivale a la orden de CP/M **language. n** es el número que indica esa variante.

— **ESC «A».** Mueve el cursor a la línea anterior. Si está en la primera, se ignora el código.

— **ESC «B».** Mueve el cursor a la línea siguiente. Si está en la última, se ignora el código.

— **ESC «C».** Mueve el cursor a la columna siguiente. Si está en la última, se ignora el código.

— **ESC «D».** Mueve el cursor a la columna anterior. Si está en la primera, se ignora el código.

— **ESC «E».** Borra la ventana (zona activa de la pantalla) sin modificar la posición del cursor.

— **ESC «H».** Mueve el cursor a la columna 0, línea 0 de la ventana actual.

— **ESC «I».** Mueve el cursor a la línea anterior. Si está en la primera, desplaza la pantalla hacia abajo.

— **ESC «J».** Borra desde donde se encuentra el cursor hasta el final de la ventana, sin modificar su posición.

— **ESC «K».** Borra desde donde se encuentra el cursor hasta el final de la línea. No se modifica la posición del cursor.

— **ESC «L».** Inserta una línea, desplazando aquella en la que está el cursor y todas las inferiores una línea hacia abajo. La actual queda en blanco.

— **ESC «M».** Borrar línea, eliminándola y desplazando las inferiores una hacia arriba.

— **ESC «N».** Borra el carácter que está bajo el cursor; los caracteres situados a su derecha se mueven una posición a la izquierda.

— **ESC «X» pf pc f c.** Define la posición y el tamaño de la ventana (zona activa de la pantalla). En principio, la ventana es toda la pantalla. **pf** es la coordenada de la primera fila más 32. **pc** es la coordenada de la primera columna más 32. **f** es el número de filas más 31, y **c** es el número de columnas más 31. Las coordenadas empiezan por 0.

— **ESC «Y» f c.** Lleva el cursor a las coordenadas especificadas. **f** es la fila destino más 32. **c** es la columna destino más 32.

— **ESC «b» n.** Invierte los colores de la pantalla (verde-negro). **n** es 9, 32, 96, 160 ó 224 para invertir, y los demás valores para restablecer al estado normal, es decir: caracteres verdes y fondo negro.

— **ESC «c» n.** La misma función que **ESC «b» n.** Los valores de **n** son de 1 a 8, de 10 a 31, de 65 a 95, de 129 a 159 y de 193 a 223 para invertir los colores. El resto de los valores los restablece. Recomendamos los valores 1 y 0 para invertir y restablecer, respectivamente.

— **ESC «d».** Borra desde el principio de la ventana hasta la posición del cursor.

— **ESC «e».** Activa el cursor, de manera que es visible. Esta es la situación normal.

— **ESC «f».** Inhibe el cursor. No puede verse en la pantalla. No se recomienda esta situación si se van a editar líneas de programa.

— **ESC «j».** Almacena la posición del cursor, es decir, fila y columna en la cual se encuentra.

— **ESC «k».** Restablece la posición del cursor almacenada por ESC «j».

— **ESC «l».** Borra los caracteres de la línea.

— **ESC «o».** Borra desde el principio de la línea hasta la posición del cursor.

— **ESC «p».** Activa el modo de imagen invertida, en el cual los caracteres aparecen en negro con fondo oscuro al ser impresos en la pantalla. Este código es distinto a los de inversión de colores, ya que aquellos cambian toda la pantalla, mientras que **ESC «p»** sólo invierte lo que se imprima a continuación.

— **ESC «q».** Desactiva el modo de imagen invertida.

— **ESC «r».** Activa el modo de subrayado. Los espacios también se subrayan en este estado.

— **ESC «u».** Desactiva el modo de subrayado.

— **ESC «v».** Activa la continuidad de líneas, es decir, cuando se llega a la última columna, los caracteres siguen siendo impresos en la fila siguiente. Esta es la situación normal.

— **ESC «w».** Desactiva la continuidad de líneas. En este modo, los caracteres que sobrepasen la última columna no serán impresos, aunque serán tenidos en cuenta igualmente.

— **ESC «x».** Equivale a la orden de CP/M **SET24×80.** Activa el modo de 24 filas por 80 columnas.

ESC «y». Desactiva el modo de 24 filas por 80 columnas.

La forma correcta de enviar estos códigos a la pantalla es la siguiente:

En el caso de que el código conste de un solo carácter, se escribe con la función **CHR$.** Por ejemplo, **BS,** retroceso del cursor, se escribe:

```
print chr$(7)
```

Si el código consta de **ESC** más otro carácter, este último se escribe entrecomillado. Por ejemplo, **ESC «q»** debe escribirse:

```
print chr$(27);«q»
```

Si el código consta de más de un carácter en la secuencia de escape, serán escritos con **CHR$** los posteriores al primero. Por ejemplo, **ESC «c» 1** (para invertir los colores) se escribe:

```
print chr$(27);«c»;chr$(1)
```

y para llevar el cursor a la fila 10, columna 10, utilizaremos **ESC Y 42 42:**

```
print chr$(27);«Y»;chr$(42);chr$(42);
```

Las líneas que desplacen el cursor por la pantalla deben llevar al final el signo de punto y coma (;), de lo contrario el cursor bajará a la fila siguiente, columna primera. Por ejemplo, para llevar el cursor a la fila 10, columna 10 e imprimir un texto, se escribe:

```
print chr$(27);«Y»;chr$(42);chr$(42);«Aquí!»
```

y no

print chr$(27);«Y»;chr$(42);chr$(42):print «Aquí no!»

La mejor forma de conocer y controlar adecuadamente estos códigos es practicar con ellos, e ir incluyéndolos gradualmente los programas. En los listados de este libro irán apareciendo normalmente comentados en una sentencia **REM.**

LA GESTIÓN DE ERRORES

Dentro del apartado sobre «control de ejecución» no podían faltar las instrucciones dedicadas a interceptar e identificar errores provocados durante la ejecución del programa, sin tener forzosamente que detener la misma, como hace BASIC automáticamente.

La orden de intercepción es **ON ERROR GOTO** (en caso de error ir a). Suele colocarse al principio del programa. Después del **GOTO** se especifica la línea a la cual debe dirigirse la ejecución en caso de que se produzca el error.

El conjunto de instrucciones que se ejecutan como consecuencia de **ON ERROR GOTO** deben ser especiales, es decir, han de averiguar cuál es el error producido, desviar la ejecución a una rutina donde pueda ser tratado o en caso contrario hacer que BASIC gestione el error.

Cuando se escribe un número de línea en **ON ERROR GOTO,** el intérprete entiende que los errores serán gestionados por el programa, y por tanto no emite los corespondientes mensajes, sino que desvía la ejecución a la línea mencionada.

Este estado de intercepción se mantiene hasta el final de la ejecución, o bien hasta que se interrumpa con la expresión **ON ERROR GOTO 0,** la cual hace que BASIC se encargue de nuevo de la gestión de errores, es decir, imprimir los correspondientes mensajes deteniendo el programa.

ON ERROR GOTO 0 es la instrucción que debe incluir el programa en caso de que el error producido no pueda ser remediado por el programa, o no haya sido previsto. De esta manera, BASIC emitirá el mensaje informativo adecuado para localizar el error y corregirlo, si es posible.

Pero aún no hemos dicho cómo «sabe» el programa el error que se ha cometido. Para empezar, debemos saber que todos los mensajes de error que puede emitir BASIC están clasificados y tienen un número asociado.

TABLA DE ERRORES Y MENSAJES CORRESPONDIENTES

COD.	Mensaje	Significado
1	Unexpected NEXT	NEXT inesperado
2	Syntax error	Error de sintaxis
3	Unexpected RETURN	RETURN inesperado
4	DATA exhausted	DATA agotada
5	Improper Argument	Argumento inapropiado
6	Overflow	Sobrepasamiento
7	Memory full	Memoria llena
8	Line does not exist	Línea no existente
9	Subscript out of range	Subíndice fuera de rango
10	Array already dimensioned	Matriz ya dimensionada
11	Division by zero	División por cero
12	Invalid direct command	Comando directo erróneo
13	Type mismatch	Error de tipo
14	String space full	Espacio para cadenas lleno
15	String too long	Cadena demasiado larga
16	String expression too complex	Expresión de cadena demasiado compleja
17	Cannot CONTinue	No se puede continuar
18	Unknown user function	Función de usuario desconocida
19	RESUME missing	Falta RESUME
20	Unexpected RESUME	RESUME inesperado
21	O/S depended error	Error dependiente del sistema operativo
22	Operand missing	Falta operando
23	Line too long	Línea demasiado larga
26	NEXT missing	Falta NEXT
29	WEND missing	Falta WEND
30	Unexpected WEND	WEND inesperado
50	Record overflow	Sobrepasamiento de registro
51	Internal error	Error interno
52	File number error	Error de número de fichero
53	File not found	Fichero no encontrado
54	File type error	Error de tipo de fichero
55	File already open	Fichero ya abierto
57	Disc I/O error	Error de entrada/salida en el disco
58	File already exists	Fichero ya existente
61	Disc full	Disco lleno
62	EOF met	Localizado fin de fichero

COD.	Mensaje	Significado
63	Record number error	Error de número de registro
64	File name invalid	Nombre de fichero incorrecto
66	Direct command found	Localizado comando directo
67	Directory full	Directorio lleno
68	Read past EOF	Lectura más allá del fin de fichero
69	Record is locked	El registro está bloqueado
70	Read-only disc	Disco de sólo lectura
71	Read-only file	Fichero de sólo lectura
72	Invalid drive	Unidad de disco incorrecta
73	File is locked	Fichero bloqueado
74	RESET denied	RESET denegado
113	Not a keyed file	No se reconoce el fichero como indexado
114	Record is not locked	El registro no está bloqueado
115	Inconsistent files	Fichero inconsistente

Existe una función, **ERR** la cual genera el número de error, si es que éste se ha producido. Cuando se pide el valor a **ERR** sin haberse producido ningún error, da 0:

```
print err
0
Ok
```

Si se ha producido alguno, devuelve el código correspondiente:

```
print chr$(800) (operación incorrecta)
Improper argument
Ok
print err
5
Ok
```

Esta función no es de gran utilidad directamente. Sin embargo, sí lo es dentro de un programa, ya que permite detectar el error. Por ejemplo, en el programa 1 se activa la intercepción de errores con **ON ERROR GOTO,** y luego se pide al usuario un número para operar con él. Hemos incluido operaciones que se prestan a error, como la raíz cuadrada.

Al final del programa, está la rutina encargada de detectar cuál ha sido el error, indicarlo y comenzar de nuevo. Si el código de error no se detecta, se desactiva la intercepción de errores para que BASIC emita el mensaje correspondiente.

Como puede observarse en el listado, aparece una nueva instrucción, **RESUME.** Esta orden equivale a **GOTO,** pero debe figurar la primera si estamos en una zona del programa ejecutada a causa de **ON ERROR GOTO.** Es decir, **RESUME** se encarga de reanudar la ejecución normal tras un error.

Las tres formas de utilizar **RESUME** son:

RESUME
RESUME (número de línea)
RESUME NEXT

La primera forma reanuda la ejecución al principio de la sentencia en la cual se produjo el error. Esta forma se utiliza cuando la rutina de gestión de errores ha corregido el o los parámetros adecuadamente, de manera que el error no volverá a repetirse. Un ejemplo de la utilización de este modo lo tenemos en el programa 2.

La segunda forma incluye un número de líneas. La ejecución se reanuda en otro punto, por ejemplo en el principio de la recogida de datos si los parámetros no pueden ser corregidos. Es lo que utiliza el programa 1.

RESUME NEXT es similar a la primera forma, pero en lugar de reanudar la ejecución en la instrucción que provocó el error, lo hace en la siguiente. El **NEXT** no tiene relación alguna con los bucles. Este modo de regresar al programa es adecuado cuando sólo hace falta advertir al usuario del error, aunque no se evite. Así funciona el programa 3.

```
10 REM ******  PROGRAMA 1 *******
20 REM Se reanuda la ejecucion con resume (no. de linea).
30 PRINT CHR$(27);"E"
40 REM Borrar la pantalla
50 PRINT CHR$(27);"H";
60 REM Colocar el cursor al principio de la pantalla
70 ON ERROR GOTO 200
80 INPUT "Numero para operar ";n
90 t=SQR(n)
100 REM Si el numero es negativo, hay error 5.
110 t2=10/t
120 REM Si t es cero, hay error 11.
130 PRINT "El resultado es ";t2
140 END
200 PRINT "Error!";
210 x=ERR
220 IF x=5 THEN PRINT "Error en la raiz cuadrada.":GOTO 260
230 IF x=11 THEN PRINT "Division por cero.":GOTO 260
240 ON ERROR GOTO 0
250 REM No hemos localizado el error. BASIC lo dira.
260 RESUME 30
270 REM Llevamos la ejecucion a la 30 para repetir la entrada de
      datos.
```

```
1  REM ********** PROGRAMA 2 **********
5  PRINT CHR$(27);"H";CHR$(27);"E";
7  REM Borramos pantalla.
10 REM Este programa utiliza RESUME para repetir la operacion,
       ya que corregimos el parametros para que sea correcto.
15 REM El usuario no notara el error ni la correspondiente
       gestion.
20 ON ERROR GOTO 200
30 DIM m(12)
40 DATA Enero,Febrero,Marzo,Abril,Mayo,Junio,Agosto
50 DATA Septiembre,Octubre,Noviembre,Diciembre
60 FOR a=1 TO 12
70 READ m(a)
80 NEXT
90 REM Vamos a escribir los meses repetidamente, aumentando una
       variable indice, x. Cuando x sobrepase el limite, se
       producira error (indice fuera de rango)
100 x=1
110 PRINT m(x)
120 x=x+1
130 GOTO 100
140 REM
200 REM Aqui el subindice es 13, no valido. Lo ponemos a 1 para
       continuar la ejecucion donde la dejamos, en el PRINT.
210 x=1: REM Todo arreglado.
220 RESUME
```

```
1  REM ******* PROGRAMA 3 **********
10 REM Gestion de errores utilizando RESUME NEXT. Vamos a
       provocar un rebosamiento (Overflow, error 6).
20 ON ERROR GOTO 120
30 INPUT "Escribe un numero grande ";n
40 PRINT n^1e10^1e-10
50 REM Seguramente el numero es muy grande para BASIC.
60 PRINT "Pulsa 's' para repetir"
70 a$=INPUT$(1)
80 REM INPUT$ se ha visto en este mismo capitulo.
90 IF a$="s" THEN GOTO 30
100 PRINT "Fin de programa."
110 END
120 IF ERR<>6 THEN ON ERROR GOTO 0
130 REM Si no es el 6, no sabemos de que error se trata. BASIC
       lo dira, e interrupira la ejecucion.
140 REM Ahora informamos del error conocido:
150 PRINT "Se ha sobrepasado el limite!"
160 PRINT "La operacion no es valida."
170 RESUME NEXT
```

En algunas ocasiones, sobre todo en grandes programas, comprobaremos que un mismo error puede producirse en varias zonas del mismo. Si el remedio a cada uno es distinto, por estar cada cual en una rutina muy determinada, debemos saber de cuál se trata.

Para averiguarlo disponemos de la función **ERL,** la cual retiene el número de líneas en el que se produjo el error. El programa 4 tiene

dos rutinas distintas, pero que pueden provocar el mismo error, *Improper argument.* Se utiliza **ERL** para determinar cuál es la acción apropiada.

Estas son las instrucciones más comunes en el tratamiento de errores «remediables». De todos modos, seguiremos hablando de errores.

```
10 REM XXXXXX  PROGRAMA 4 XXXXXXXX
15 PRINT CHR$(27);"H";chr$(27);"E":REM borra pantalla.
20 REM Utilizamos ERL para determinar el origen del error.
30 ON ERROR GOTO 200
40 INPUT "Dame un codigo valido ASCII, yo lo imprimiré.",n
50 REM Si se escribe un numero mayor que 255, error 5.
60 PRINT CHR$(n)
70 INPUT "Ahora dame un numero para calcular la raiz",n2
80 REM Si se escribe un numero negativo, tambien error 5.
90 PRINT SQR(n2)
100 PRINT "Empezamos otra vez."
110 GOTO 40:REM repetir el proceso
120 REM
200 IF ERR<>5 THEN ON ERROR GOTO 0
210 REM No conocemos el error, que BASIC lo gestione.
220 IF ERL=60 THEN PRINT "Por favor, escribe un numero entre  32
    y 255":RESUME 40
320  IF  ERL=90  THEN  PRINT  "Por  favor,  escribe  un  numero
    positivo":RESUME 70
```

GESTIÓN INTERNA

Vamos a ver en este capítulo instrucciones relacionadas con el hardware: comunicaciones, versiones y configuración de memoria. Esta última parte nos interesa especialmente, puesto que para manejar ficheros es necesario preparar determinadas zonas de la memoria destinadas a tratar los registros.

Los primeros grupos de instrucciones, dedicadas a la comunicación, etc. no serán de tanta utilidad. En concreto, no interesarán más que al programador con bastante experiencia, por lo que no se perderá la continuidad del libro si se omite la primera parte de este capítulo.

VERSIONES BASIC

Puesto que Mallard BASIC no está instalado sólo en los Amstrad PCW, es aconsejable que exista algún tipo de indicador para saber en qué máquina trabaja y sobre qué sistema operativo.

Esta es precisamente la labor de la función **VERSION(n). n** es un número entre 0 y 4 que indica el tipo de información que necesitamos (cualquier otro valor provocará el error *Improper argument).*

Parámetro	Valor generado	Significado
0	7	procesadores 8080/808
	8	procesador Z80
	16	procesadores 8088/8086
1	0	BASIC completo
	1	BASIC «sólo ejecución»
2	1	Uniusuario
	2	Multiusuario CP/M 86
3	1	CP/M
	2	MS-DOS
4	(dependiente del sistema operativo)	

En el caso particular del Amstrad PCW, las indicaciones que muestra BASIC con los distintos parámetros es:

Parámetro 0	Versión Z80	8
Parámetro 1	BASIC completo	0
Parámetro 2	Uniusuario	1
Parámetro 3	BASIC para CP/M	1
Parámetro 4	(Sistema CP/M)	49

MÁS ERRORES

Continuamos con los errores. Existe un comando, **ERROR,** dedicado a provocar la impresión de todos los mensajes de error disponibles. Hay que indicar el código de error. Existe una tabla donde se relaciona cada número, del 1 al 255 con un error, aunque existen números no asociados; si se especifica un código no asociado, aparece el mensaje *Unknow error* (error desconocido). Podemos provocar errores con **ERROR** durante la ejecución del programa, de manera que se detenga emitiendo el correspondiente mensaje. Por ejemplo, para provocar un error de sintaxis, escribiremos:

error 2

con lo cual obtendremos el mensaje correspondiente.

Vamos a fijar nuestra atención en un error muy determinado: el 21, *O.S. dependent error* (error dependiente del sistema operativo). Como ya hemos dicho, Mallard BASIC funciona en muchas máquinas distintas, y son varios los sistemas operativos que lo soportan. Evi-

dentemente, cada sistema operativo tiene su propio modo de gestionar el curso de ejecución, y BASIC por sí solo no puede «saber» qué ocurre con los errores de los distintos sistemas.

Para solucionar esto, cada sistema operativo entrega a BASIC un número entre 0 y 255 para designar el error que se ha producido. La función **OSERR** nos proporciona ese número. La forma de obtenerlo es escribir:

print oserr

Dado que es un error de sistema, es muy poco probable que nos veamos forzados a utilizar esta función debido a un error 21.

COMUNICACIÓN

Podemos utilizar las puertas de E/S (entrada y salida de datos). Para ello tenemos la función **INP(p)** y la orden **OUT p,d. p** indica una puerta de E/S, con valor comprendido entre 0 y 255. **d** es un dato también entre 0 y 255.

INP capta el valor presente en la puerta especificada. **OUT** envía el dato **d** a la puerta **p.** No se deben ejecutar estas dos instrucciones de entrada y salida sin un conocimiento previo de las características de la máquina, puesto que podemos cambiar ciertas condiciones del sistema. Tan pronto como empecemos a «investigar» con varios valores, comprobaremos que la unidad de disco acusa estos experimentos.

Existen las órdenes **INPW** y **OUTW,** pero trabajan con «palabras» en lugar de bytes, lo cual no se ajusta a las características del PCW. Por lo tanto, estas funciones se ha «condenado», de manera que BASIC acepta operar en principio con ellas pero emite el error *Improper argument* al interpretarlas.

ORGANIZACIÓN DE LA MEMORIA

BASIC está instalado en un bloque de 64K, del cual ocupa casi la mitad, hasta la dirección 31380 aproximadamente. A partir de ahí, es donde se instala el programa BASIC codificado (si lo hay), y las variables están por encima de él.

Pero ni el programa ni las variables pueden crecer indefinidamente. Existe un límite por encima del cual BASIC «se prohíbe» escribir, puesto que allí están otros datos imprescindibles (pila, etc...). Este

límite máximo es el que proporciona la función **HIMEM** (parte alta de la memoria). Podemos ver este valor:

print himem

Entregará aproximadamente el valor 62381. Este valor se toma de las direcciones 6 y 7 de memoria, por lo que podemos obtenerlo también con la expresión:

print peek(6)+256*peek(7)

Podemos cambiar este límite máximo (no es aconsejable subirlo demasiado, porque podemos «ahogar» a BASIC). Las dos instrucciones para ello son **CLEAR** y **MEMORY.** Ya hablamos de **CLEAR** en la primera parte dedicada a BASIC, pero como veremos ahora se trata de un comando múltiple.

Para cambiar el tope de memoria alta, se escribe

clear, m o bien **memory, m**

Podemos especificar más parámetros. La forma general completa de estos comandos es:

memory,ma,tp,nf,lr
clear,ma,tp,nf,lr

ma como ya hemos dicho es la memoria alta. **tp** es el tamaño de la pila. Es imprescindible asignar como mínimo 256 bytes a esta zona, puesto que contiene información necesaria para BASIC. Recomendamos no modificarla, si no es para agrandarla.

nf es el número máximo de ficheros a los que BASIC puede acceder a la vez. La situación normal es 3, y sólo debe aumentarse cuando sea necesario, puesto que se reserva bastante espacio para gestionar cada fichero.

lr es la longitud de registro máxima. El valor máximo que soporta CP/M es 128, y es el implícito. Hablaremos de registros más adelante, por ahora sólo debemos saber cómo modificar su longitud.

No es necesario especificar todos los parámetros cada vez que ejecutemos un **CLEAR** o un **MEMORY.** Podemos dejar vacío el espacio correspondiente. Por ejemplo, si queremos modificar el tope de memoria a 62000 y la longitud de registro a 40, escribiremos:

memory,62000,,,40

Recordaremos que para saber en cualquier momento la memoria disponible para variables y programa se obtiene con:

print fre(0)

expresión que normalmente se escribe después de borrar las variables con **CLEAR,** para saber exactamente lo que ocupa un programa.

LA IMPRESORA

Ya conocemos el modo directo de manejo de la impresora PCW: las teclas **IMPR, SAL,** +, ← (flecha izquierda), → (flecha derecha) y **STOP.** Su función es controlar la línea de estado de la impresora, situada en la fila 32 de la pantalla —cuando se pulsa **IMPR.**

Pero BASIC dispone además de cuatro instrucciones para manejar la impresora, y permite también enviar códigos de control para cambiar tipos de letra, etc...

Todas las instrucciones de impresora tienen asociada la letra «L», debido a una ya vieja historia que no nos interesa ahora. El periférico en sí se denomina **lpt** (CP/M) o bien **lprinter** (BASIC). Para escribir en ella, se utiliza el comando **LPRINT,** con la misma forma que **PRINT,** pero que saldrá impreso en el papel y no en la pantalla.

LLIST, como podemos suponer, lista el programa, sin que aparezca en la pantalla. BASIC tardará algún tiempo en devolver el mensaje «Ok», especialmente si el programa es largo. También es posible especificar los mismos parámetros que con **LIST** (conjuntos determinados de líneas, líneas sueltas, etc...).

Las otras dos instrucciones son **WIDTH LPRINT** y **LPOS,** que tienen el mismo efecto que **WIDTH** y **LPOS,** respectivamente, pero en la impresora (también se puede hacer la anchura de la impresora «infinita» con **WIDTH LPRINT 255).**

LOS CÓDIGOS DE CONTROL

Tal y como ocurre con la pantalla, la impresora reconoce unos determinados códigos de control, distintos totalmente a los anteriores, para controlar los distintos modos de funcionamiento.

Existen también códigos de un solo carácter, de dos y de más de dos. El carácter 27 (ESC) sigue siendo el primero de los que se especifica en los dos últimos tipos. Esta vez no los nombraremos por su longitud, sino por su función.

PASO DE LINEA:

Es el espacio que se inserta entre línea y línea. Lo normal es que sea de seis u ocho pulgadas, pero podemos ajustarlo para otras medidas:

— **ESC 0:** 1/8 de pulgada.
— **ESC 1:** 7/72 de pulgada.
— **ESC 2:** 1/6 de pulgada.
— **ESC 3 n:** n/216 de pulgada (n entre 0 y 255).
— **ESC A n:** n/72 de pulgada (n entre 0 y 85).

MARGENES:

— **ESC 1 n:** Define margen izquierdo. **n** es el número de caracteres de la posición (teniendo en cuenta el tamaño del actual).

— **ESC Q n:** Define margen derecho. **n** cumple las mismas condiciones que en el caso anterior.

La definición de márgenes está limitada según el tipo de carácter actual. En caso de enviar un código que exceda estas limitaciones se ignora.

TABULADORES:

Si no se han cambiado, los topes de tabulación están situados cada

ocho columnas en todos los tipos de letra. Podemos cambiar estos topes, pero hay que definirlos todos, en orden ascendente, puesto que al redefinir, se borran todos los anteriores.

Cada vez que se redefine el margen izquierdo se restauran todos los topes de tabulación a la situación inicial (un tope cada ocho columnas). Pueden definirse hasta 32 topes.

— **ESC D n 0:** Define tope de tabulación.

— **HT** (9): Avanza la cabeza impresora hasta el siguiente tope de tabulación.

Tipo letra	Caracteres por pulgada	Margen derecho	Margen izquierdo
Pica	10	2-81	0-78
Estrecha	17	4-139	0-133
Elite	12	3-96	0-93
Pica doble ancho	5	1-40	0-39
Estrecha doble ancho	8,5	2-69	0-66
Elite doble ancho	6	2-48	0-46
	(Proporcional igual que élite)		

MODIFICACION DE POSICIONES:

— **BS** (8): Hace retroceder la cabeza impresora una posición. Por ejemplo, la expresión **lprint «O»;chr$(8);«/»** imprime una «O» tachada por la barra inclinada.

— **CR** (13): Retorno del carro. La cabeza impresora pasa a la primera columna de la fila. Se baja a la siguiente fila, si está activado el avance automático.

— **FF** (12): Pasar a la siguiente página. La cabeza impresora pasa al margen izquierdo.

— **LF** (10): Avanza una línea, distancia definida por el «paso de línea».

— **ESC LF:** Activa el avance automático de línea. Cuando se efectúa un retorno del carro, se avanza también una línea.

— **ESC CR:** Se desactiva el modo anterior.

PAPEL:

— **ESC 8:** Desactiva la detección de fin de papel.
— **ESC 9:** Activa la detección.
— **ESC J n:** Avanza el papel n/216 pulgadas. **n** debe estar entre 0 y 255.
— **ESC C n:** Define la longitud de página en líneas del paso actual. **n** debe estar entre 1 y 127.
— **ESC C 0 n:** Define la longitud de página en pulgadas. n está entre 1 y 22.
— **ESC $:** Selecciona el modo de hojas sueltas: la impresora espera a que se cambie el papel cada vez que termina una página.
— **ESC c:** Selecciona el modo de papel continuo.
— **ESC N n:** Establece el número de líneas que se debe saltar al terminar una página, es decir, las líneas que no se imprimirán al final. **n** está entre 1 y 127.
— **ESC O:** Cancela la definición anterior.

INICIALIZACION DE LA IMPRESORA:

— **CAN** (24): Borra de la memoria temporal todos los caracteres pendientes de ser impresos desde el último CR, LF, FF o BS.
— **ESC:** Reinicializa la impresora, devolviendo los controles a la siguiente situación:
Paso pica (10 caracteres por pulgada) desactivando estrecha, doble ancho, élite, cursiva, negra, doble impresión, subíndices, subrayado, alta calidad, cero sin barra y caracteres especiales.
El margen izquierdo pasa a la columna 0, margen derecho a la 80; los topes de tabulación cada 8 columnas.
Paso de línea 1/6 de pulgada, longitud de página 70 líneas, salto de fin de página 3 líneas, CR sin avance automático.
Modo de papel de hojas sueltas, con sensor desactivado. Juego ASCII del Reino Unido
— **ESC d:** Hace que los parámetros definidos actualmente sean los implícitos, de manera que al reinicializar la impresora se pasará a los actuales y no a los descritos anteriormente.
— **SI** (15): Cambiar a letra estrecha.
— **DC2** (18): Volver a paso 10.
— **ESC M:** Cambiar a élite.
— **ESC P:** Volver a paso 10.

— **ESC p 1:** Cambiar a paso proporcional.
— **ESC p 0:** Volver a paso 10.
— **SO** (14): Establece doble ancho del paso actual.
— **DC4** (20): Vuelve al paso normal.

Se vuelve al paso normal cuando se ejecuta un FF o un LF. Si se desea un efecto continuo hasta que se especifique, se utiliza **ESC W 1,** y se desactiva con **ESC W 0.**

— **ESC 4:** Activar cursiva.
— **ESC 5:** Volver a letra normal.
— **ESC m 1:** Activar alta calidad.
— **ESC m 0:** Volver a calidad normal.

La alta calidad no es compatible con la letra estrecha, los subíndices o superíndices, y la doble impresión.

— **ESC G:** Activar la doble impresión.
— **ESC H:** Vuelve a impresión normal.

La doble impresión es incompatible con subíndices, superíndices y alta calidad.

— **ESC E:** Activar escritura en negra.
— **ESC F:** Volver a impresión normal.

El modo de doble impresión repite la pasada avanzando el papel. La escritura en negra hace los puntos de cada carácter más anchos, de manera que el resultado es más oscuro.

— **ESC S 0:** Activar superíndices.
— **ESC S 1:** Activar subíndices.
— **ESC T:** Desactivar ambos.
— **ESC — 1:** Activar subrayado.
— **ESC — 0:** Volver al modo normal.

CARACTERES ESPECIALES:

La impresora, como la pantalla, puede seleccionar entre uno de los 9 juegos de caracteres disponibles. El que en BASIC trabajemos con el juego español, no significa que la impresora también lo haga. El funcionamiento es independiente, y en caso de diferencia, se cambiará el código enviado por el correspondiente al juego actual de la impresora.

— **ESC R n:** Cambiar el juego de caracteres. **n** está entre 0 y 8.
— **ESC I 1:** Incluir los caracteres imprimibles con los códigos entre 0-31 y 128-159.
— **ESC I 0:** Cancelar la situación anterior.

— **ESC X:** Seleccionar cero con barra.
— **ESC o:** Seleccionar cero sin barra.

Cuadro de los distintos juegos de caracteres para impresora. Página 136 del primer tomo de instrucciones Amstrad PCW.

Dec	EE.UU.	Francia	Alem.	R.U.	Din.	Suecia	Italia	España	Japón
35	#	#	#	£	#	#	#	P_t	#
36	$	$	$	£	$	¤	$	$	$
64	@	à	§	@	@	É	@	@	@
91	[	°	Ä	[	Æ	Ä	°	¡	[
92	\	ç	Ö	\	Ø	Ö	\	Ñ	Ψ
93	]	§	Ü	]	Å	Å	é	¿	]
94	^	^	^	^	^	Ü	^	^	^
96	`	`	`	`	`	é	ù	`	`
123	{	é	ä	{	æ	ä	à	..	{
124	|	ù	ö	|	Ø	ö	ò	ñ	|
125	}	è	ü	}	å	å	è	}	}
126	~	¨	ß	~	~	Ü	ì	~	~

Cuadro ampliación de caracteres imprimibles. Página 137 del primer tomo de instrucciones Amstrad PCW.

DEC	DEC	DEC	DEC
0 à à	16 § §	128 *à à*	144 *§ §*
1 è è	17 ß ß	129 *è è*	145 *ß ß*
2 ù ù	18 *DC2*	130 *ù ù*	146 *DC2*
3 ò ò	19 æ æ	131 *ò ò*	147 *æ æ*
4 ì ì	20 *DC4*	132 *ì ì*	148 *DC4*
5 ° °	21 Ø Ø	133 ° °	149 *Ø Ø*
6 £ £	22	134 *£ £*	150 .. ..
7 *BEL*	23 Ä Ä	135 *BEL*	151 *Ä Ä*
8 *BS*	24 *CAN*	136 *BS*	152 *CAN*
9 *HT*	25 ü ü	137 *HT*	153 *ü ü*
10 *LF*	26 ä ä	138 *LF*	154 *ä ä*
11 *VT*	27 *ESC*	139 *VT*	155 *ESC*
12 *FF*	28 ü ü	140 *FF*	156 *ü ü*
13 *CR*	29 É É	141 *CR*	157 *É É*
14 *SO*	30 é é	142 *SO*	158 *é é*
15 *SI*	31 Ψ Ψ	143 *SI*	159 *Ψ Ψ*

EL MODO GRÁFICO

Además de las características descritas hasta ahora la impresora permite imprimir gráficos. Para ello, es necesario descomponer la imagen en filas de ocho puntos, puesto que al imprimir la cabeza se desplaza horizontalmente, y tiene ocho agujas.

El gráfico puede ser de simple o doble densidad. En el segundo caso, los puntos están mucho más próximos horizontalmente, con lo cual se consigue una mayor definición del dibujo.

La forma de preparar la impresora para los gráficos es:

ESC K n1 n2
ESC * 0 n1 n2: Establecen densidad normal.
ESC L n1 n2
ESC * 1 n1 n2: Establecen doble densidad.

n1 y **n2** son dos números entre 0 y 255 que indican el número de datos gráficos que han de enviarse a la impresora. **n1** es el número de datos MOD 256 y **n2** es el número dividido entre 256 (división entera).

Por ejemplo, si tenemos 300 datos o imágenes gráficas, y queremos imprimirlo en doble densidad, escribiremos:

```
lprint chr$(27);«K»;chr$(300 mod 256);chr$(200/256);
```

y a continuación los datos. El máximo número de imágenes que se pueden escribir en doble densidad es 960, y 480 en densidad normal.

Para codificar figuras, conviene dibujarlas sobre papel milimetrado, de manera que rellenemos por puntos todo el dibujo. Podemos distinguir así entre puntos llenos y vacíos, que tendrán los valores 1 y 0, respectivamente.

Ahora agrupemos los puntos de arriba abajo, en hileras verticales de ocho puntos de longitud. Tenemos grupos de ochos, unos y ceros, que codificados de binario a decimal dan un número entre 0 y 255, para mandar a la impresora.

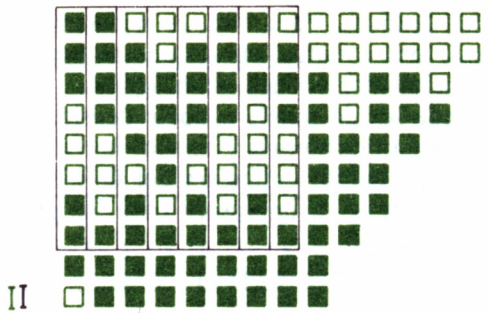
El modo de cambiar fácilmente de binario a decimal ya lo explicamos anteriormente. La forma de enviar los datos a la impresora es como caracteres, es decir, si tras la codificación de una imagen obtenemos el número 129, debemos enviarlo a la impresora así:

```
lprint chr$(129);
```

siempre debe haber un punto y coma detrás de cada dato.

El programa «generador de caracteres» ilustra perfectamente el modo de manejar la impresora en modo gráfico. Permite definir caracteres y luego imprimir ficheros de texto de acuerdo con esa defini-

ción. Veremos más adelante las operaciones de disco que este programa utiliza.



```
10 REM     ********************************************************************
20 REM     * GEN. CARACTERES-VER.3-1986 A.G.VERDUGO -GRAN BIBLIOTECA AMSTRAD *
30 REM     ********************************************************************
40 REM
50 OPTION BASE 0
60 WIDTH LPRINT 255
70 DEFSTR e
80 DEFINT q
90 esc=CHR$(27)
100 eh=esc+"E"+esc+"H"                                  :REM E y H son mayusculas
110 e1=CHR$(32)+esc+"C "                                :REM C mayuscula y un espacio
120 e2=CHR$(252)+esc+"C"+CHR$(253)                      :REM esta C tambien mayuscula
130 eoff=SPACE$(4)+CHR$(10)+STRING$(4,8)+SPACE$(4)
140 eon=esc+"p"+eoff+esc+"q"                            :REM 'p' y 'q' son minusculas
150 DIM echr(8)
160 DIM q(2048)
170 DIM w(255)
180 w(1)=3
190 FOR a=2 TO 255 STEP 2
200     w(a)=4*w(a-(a/2))
210     w(a+1)=w(a)+3
220 NEXT
230 FOR a=1 TO 8
240     echr(a)=STRING$(8,"0")
250 NEXT
260 DEF FN at$(x,y,t$)=esc+"Y"+CHR$(32+x)+CHR$(32+y)+t$:REM 'Y' mayuscula
270 PRINT eh
280 PRINT FN at$(10,20,"1  Definir / Modificar")
290 PRINT FN at$(12,20,"2  Cargar juego normal")
300 PRINT FN at$(14,20,"3  Imprimir fichero")
310 PRINT FN at$(16,20,"4  Cargar juego grabado")
320 PRINT FN at$(18,20,"5  Grabar juego")
330 PRINT FN at$(20,20,"6  Terminar")
340 h=1: stp=2: ev="123456": stv=8: sth=19
350 GOSUB 1380
360 ON h GOSUB 380,1720,2660,3700,3870,4030
370 GOTO 270
380 PRINT eh; "DEFINIR-MODIFICAR CARACTERES"
390 selec=0
400 IF FIND$("m:datos.gen")="" THEN 480
410 PRINT FN at$(2,10,"(10 segs.)")
420 OPEN "I",1,"m:datos.gen"
430 FOR a=0 TO 2048
440     INPUT #1,q(a)                  :REM El signo '#' se obtiene pulsando EXTRA y 3.
450 NEXT                               :REM Es igual en todas las instrucciones de este
460 CLOSE 1                            :REM programa  que  utilizan  algun  fichero.
470 PRINT FN at$(2,10,STRING$(10,32))
480 GOSUB 1550
490 PRINT FN at$(0,60,"Caracter: 0")
500 PRINT FN at$(5,13,CHR$(134)+STRING$(32,138)+CHR$(140))
510 FOR a=1 TO 14
520     PRINT FN at$(5+a,13,CHR$(133));FN at$(5+a,46,CHR$(133))
530 NEXT
540 PRINT FN at$(20,13,CHR$(131)+STRING$(32,138)+CHR$(137))
550 GOSUB 1480
560 PRINT FN at$(9,60,"0  Copiar")
570 PRINT FN at$(10,60,"1  Borrar tablero")
580 PRINT FN at$(11,60,"2  Seleccionar caracter")
590 PRINT FN at$(12,60,"3  Modificar caracter")
600 PRINT FN at$(13,60,"4  Inversion X")
610 PRINT FN at$(14,60,"5  Inversion Y")
620 PRINT FN at$(15,60,"6  Inversion f/p")
630 PRINT FN at$(16,60,"7  Terminar cambios")
640 h=2: stp=1: ev="01234567": stv=9: sth=59
650 GOSUB 1380
660 ON h+1 GOSUB 750,830,890,970,1140,1200,1300
670 IF h<7 THEN 650
680 PRINT FN at$(24,10,"(10 segs.)")
690 OPEN "O",1,"m:datos.gen"
```

```
700 FOR a=0 TO 2048
710     WRITE #1,q(a)
720 NEXT
730 CLOSE 1
740 RETURN
750 PRINT FN at$(28,0,"Numero destino: >");
760 INPUT "",selec2
770 IF selec<0 OR selec>255 THEN PRINT CHR$(7);: GOTO 750
780 PRINT FN at$(28,0,STRING$(22,32))
790 FOR a=1 TO 8
800     q(selec2*8+a)=q(selec*8+a)
810 NEXT
820 RETURN
830 FOR a=1 TO 8
840     echr(a)=STRING$(8,"0")
850 NEXT
860 GOSUB 1480
870 GOSUB 1630
880 RETURN
890 PRINT FN at$(28,0,"Numero: >");
900 INPUT "",selec
910 IF selec<0 OR selec>255 THEN PRINT CHR$(7);: GOTO 890
920 PRINT FN at$(28,0,STRING$(14,32))
930 PRINT FN at$(0,60,"Caracter: "+STR$(selec)+"  ")     :REM dos espacios al final
940 GOSUB 1550
950 GOSUB 1480
960 RETURN
970 GOSUB 1550
980 x=1: y=1
990 PRINT FN at$(4+y*2,10+x*4,"");
1000 a$=INKEY$: IF a$<>"" THEN 1050
1010 PRINT FN at$(4+y*2,10+x*4,eon)
1020 PRINT FN at$(4+y*2,10+x*4,eoff)
1030 IF MID$(echr(x),y,1)="1" THEN PRINT FN at$(4+y*2,10+x*4,eon)
1040 GOTO 1000
1050 IF a$=CHR$(1) THEN IF x>1 THEN x=x-1: GOTO 1000
1060 IF a$=CHR$(6) THEN IF x<8 THEN x=x+1: GOTO 1000
1070 IF a$=CHR$(31) THEN IF y>1 THEN y=y-1: GOTO 1000
1080 IF a$=CHR$(30) THEN IF y<7 THEN y=y+1: GOTO 1000
1090 IF a$=CHR$(22) THEN MID$(echr(x),y,1)="1": GOTO 1000
1100 IF a$=CHR$(28) THEN MID$(echr(x),y,1)="0"
1110 IF a$<>CHR$(13) THEN 1000
1120 GOSUB 1630
1130 RETURN
1140 FOR a=1 TO 4
1150     SWAP echr(a),echr(9-a)
1160 NEXT
1170 GOSUB 1480
1180 GOSUB 1630
1190 RETURN
1200 FOR a=1 TO 8
1210     FOR b=1 TO 3
1220             eot=MID$(echr(a),8-b,1)
1230             MID$(echr(a),8-b,1)=MID$(echr(a),b,1)
1240             MID$(echr(a),b,1)=eot
1250     NEXT
1260 NEXT
1270 GOSUB 1480
1280 GOSUB 1630
1290 RETURN
1300 FOR a=1 TO 8
1310     FOR b=1 TO 7
1320             IF MID$(echr(a),b,1)="1" THEN MID$(echr(a),b,1)="0" ELSE MID$(echr(a),b,1)="1"
1330     NEXT
1340 NEXT
1350 GOSUB 1480
1360 GOSUB 1630
1370 RETURN
1380 PRINT FN at$(stv+h*stp,sth,e2)

1390 a$=INKEY$
1400 PRINT FN at$(stv+h*stp,sth,e1)
1410 PRINT FN at$(stv+h*stp,sth,e2)
1420 IF a$="" THEN 1390
1430 IF INSTR(ev+CHR$(13),a$)=0 THEN 1390
1440 IF a$=CHR$(13) THEN RETURN
1450 PRINT FN at$(stv+h*stp,sth,e1)
1460 h=VAL(a$)
1470 PRINT FN at$(stv+h*stp,sth,e2): GOTO 1390
1480 FOR a=1 TO 8
1490     FOR b=1 TO 7
1500             PRINT FN at$(4+b*2,10+a*4,"");
1510             IF MID$(echr(a),b,1)="1" THEN PRINT eon ELSE PRINT eoff
1520     NEXT
1530 NEXT
1540 RETURN
1550 FOR a=1 TO 8
1560     mem=q(selec*8+a)
1570     FOR b=1 TO 8
1580             pow=2^(8-b)
1590             IF (mem/pow)>=1 THEN MID$(echr(a),b,1)="1": mem=mem-pow ELSE MID$(echr(a),b,1)=
"0"
1600     NEXT
1610 NEXT
1620 RETURN
1630 FOR a=1 TO 8
1640     res=0
1650     FOR b=1 TO 8
1660             pow=2^(8-b)
1670             IF VAL(MID$(echr(a),b,1))=1 THEN res=res+pow
1680     NEXT
1690     q(selec*8+a)=res
1700 NEXT
1710 RETURN
```

```
1720 PRINT eb;"CARGAR JUEGO PROPIO"
1730 PRINT FN at$(2,4,"(13 segs.)")
1740 OPEN "O",1,"m:datos.gen"
1750 FOR a=0 TO 256
1760    WRITE #1,0
1770 NEXT
1780 FOR a=32 TO 128
1790    FOR b=0 TO 7
1800          READ w
1810          WRITE #1,w
1820    NEXT
1830 NEXT
1840 FOR a=1032 TO 2048
1850    WRITE #1,0
1860 NEXT
1870 CLOSE 1
1880 RETURN
1890 DATA 0,0,0,0,       0,0,0,0          :REM Los datos se han escrito con  tabulador
1900 DATA 0,0,0,122,     0,0,0,0          :REM separandolos  en  dos  columnas  para
1910 DATA 0,0,96,0,      0,96,0,0         :REM hacerlos mas legibles y evitar errores.
1920 DATA 0,126,80,112,  8,62,10,0        :REM Se pueden escribir juntos.
1930 DATA 0,0,58,42,     127,42,46,0
1940 DATA 0,98,100,8,    16,38,70,0
1950 DATA 0,4,42,82,     42,4,10,0
1960 DATA 0,0,0,32,      64,0,0,0
1970 DATA 0,0,0,0,       60,66,0,0
1980 DATA 0,0,66,60,     0,0,0,0
1990 DATA 0,0,8,42,      28,42,8,0
2000 DATA 0,0,8,8,       62,8,8,0
2010 DATA 0,0,1,6,       0,0,0,0
2020 DATA 0,0,8,8,       8,8,8,0
2030 DATA 0,0,0,6,       6,0,0,0
2040 DATA 0,0,2,4,       8,16,32,0
2050 DATA 0,60,70,74,    82,98,60,0
2060 DATA 0,34,66,126,   2,2,0,0
2070 DATA 0,38,74,74,    74,74,50,0
2080 DATA 0,36,66,66,    82,82,44,0
2090 DATA 0,12,20,36,    126,4,4,0
2100 DATA 0,116,82,82,   82,82,76,0
2110 DATA 0,60,82,82,    82,82,12,0
2120 DATA 0,64,64,70,    72,80,96,0
2130 DATA 0,44,82,82,    82,82,44,0
2140 DATA 0,48,74,74,    74,74,50,0
2150 DATA 0,0,0,18,      0,0,0,0
2160 DATA 0,0,1,38,      0,0,0,0
2170 DATA 0,0,0,8,       20,34,0,0
2180 DATA 0,0,20,20,     20,20,20,0
2190 DATA 0,0,0,34,      20,8,0,0
2200 DATA 0,4,2,2,       82,10,4,0
2210 DATA 0,60,66,90,    106,90,56,0
2220 DATA 0,62,72,72,    72,72,62,0
2230 DATA 0,126,82,82,   82,82,44,0
2240 DATA 0,60,66,66,    66,66,36,0
2250 DATA 0,126,66,66,   66,36,24,0
2260 DATA 0,126,82,82,   82,82,66,0
2270 DATA 0,126,80,80,   80,80,64,0
2280 DATA 0,60,66,66,    74,74,44,0
2290 DATA 0,126,16,16,   16,16,126,0
2300 DATA 0,0,66,66,     126,66,66,0
2310 DATA 0,12,2,2,      2,2,124,0
2320 DATA 0,126,16,16,   40,68,2,0
2330 DATA 0,126,2,2,     2,2,2,0
2340 DATA 0,126,32,16,   16,32,126,0
2350 DATA 0,126,32,16,   8,4,126,0
2360 DATA 0,60,66,66,    66,66,60,0
2370 DATA 0,126,72,72,   72,72,48,0
2380 DATA 0,60,66,74,    70,66,60,0
2390 DATA 0,126,72,72,   72,76,50,0
2400 DATA 0,36,82,82,    82,82,12,0
2410 DATA 64,64,64,126,  64,64,64,0
2420 DATA 0,124,2,2,     2,2,124,0
2430 DATA 0,120,4,2,     2,4,120,0
2440 DATA 0,124,2,4,     4,2,124,0
2450 DATA 0,66,36,24,    24,36,66,0
2460 DATA 64,32,16,14,   16,32,64,0
2470 DATA 0,66,70,74,    82,98,66,0
2480 DATA 0,0,0,94,      0,0,0,0
2490 DATA 0,62,160,144,  136,132,62,0
2500 DATA 0,40,10,178,   2,2,4,0
2510 DATA 0,13,32,126,   32,16,0,0
2520 DATA 1,1,1,1,       1,1,1,1
2530 DATA 0,0,64,32,     0,0,0,0
2540 DATA 0,4,42,42,     42,30,0,0
2550 DATA 0,0,126,18,    18,18,12,0
2560 DATA 0,0,28,34,     34,34,0,0
2570 DATA 0,12,18,18,    18,126,0,0
2580 DATA 0,28,42,42,    42,18,0,0
2590 DATA 0,0,0,62,      80,64,0,0
2600 DATA 0,24,37,37,    37,62,0,0
2610 DATA 0,126,16,16,   16,14,0,0
2620 DATA 0,0,18,94,     2,0,0,0
2630 DATA 0,0,2,1,       1,94,0,0
2640 DATA 0,0,126,24,    36,2,0,0
2650 DATA 0,0,0,124,     2,2,0,0
2660 DATA 0,62,32,30,    32,30,0,0
2670 DATA 0,62,32,32,    32,30,0,0
2680 DATA 0,28,34,34,    34,28,0,0
2690 DATA 0,63,36,36,    36,24,0,0
2700 DATA 0,24,36,36,    36,63,1,0
2710 DATA 0,0,30,32,     32,32,0,0
2720 DATA 0,18,42,42,    42,4,0,0
2730 DATA 0,0,32,124,    34,2,0,0
2740 DATA 0,60,2,2,      2,60,0,0
2750 DATA 0,48,12,2,     12,48,0,0
```

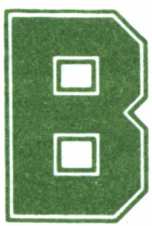
```
2760 DATA 0,60,2,28,     2,60,0,0
2770 DATA 0,34,20,8,     20,34,0,0
2780 DATA 0,56,5,5,      5,62,0,0
2790 DATA 0,34,38,42,    50,34,0,0
2800 DATA 0,0,128,0,     128,0,0,0
2810 DATA 0,158,80,144,  80,14,0,0
2820 DATA 170,85,170,85,170,85,170,0
2830 DATA 0,64,128,64,   128,0,0,0
2840 DATA 0,60,66,66,    66,60,0,0
2850 DATA 0,0,0,0,       0,0,0,0
2860 PRINT eh;"IMPRIMIR FICHERO"
2870 IF FIND$("m:datos.gen")<>"" THEN 2920
2880 PRINT:PRINT "No existen datos en memoria.";CHR$(7)
2890 FOR a=1 TO 2000
2900 NEXT
2910 RETURN
2920 PRINT FN at$(2,5,"(9 segs.)")
2930 OPEN "I",1,"m:datos.gen"
2940 FOR a=0 TO 2048
2950    INPUT #1,q(a)
2960 NEXT
2970 CLOSE
2980 LPRINT CHR$(27)+"@"
2990 PRINT FN at$(2,5,STRING$(9,32))
3000 PRINT FN at$(20,0,"Nombre del fichero: ");
3010 INPUT "",enm
3020 IF FIND$(enm)<>"" THEN 3070
3030 PRINT:PRINT"No existe ese fichero.";CHR$(7)
3040 FOR a=1 TO 2000
3050 NEXT
3060 RETURN
3070 PRINT eh
3080 PRINT FN at$(4,35,"IMPRESION")
3090 PRINT FN at$(10,30,"1  Normal")
3100 PRINT FN at$(12,30,"2  Normal + doble ancho")
3110 PRINT FN at$(14,30,"3  Doble")
3120 PRINT FN at$(16,30,"4  Doble + doble ancho")
3130 h=1: stp=2: ev="1234": stv=6: sth=29
3140 GOSUB 1380
3150 dba=0
3160 IF h<3 THEN elpt="K": lmt=480 ELSE elpt="L": lmt=960
3170 lca=lmt/6
3180 IF ABS(h-3)=1 THEN dba=1: lca=lca/2
3190 PRINT eh
3200 PRINT FN at$(14,30,"1  Altura normal")
3210 PRINT FN at$(17,30,"2  Doble altura")
3220 h=1: stp=3: ev="12": stv=11:sth=29
3230 GOSUB 1380
3240 PRINT eh
3250 IF h=2 THEN dbh=1 ELSE dbh=0
3260 OPEN "I",1,enm
3270 WHILE NOT EOF(1)
3280    LINE INPUT #1,a$
3290    GOSUB 3330
3300 WEND
3310 CLOSE 1
3320 RETURN
3330    WHILE a$<>""
3340            mrk=LEN(a$)
3350            b$=LEFT$(a$,lca)
3360            a$=RIGHT$(a$,mrk-LEN(b$))
3370            top=LEN(b$)*8*(dba+1)
3380            LPRINT CHR$(27)+elpt+CHR$(top MOD 256)+CHR$(top#256);    :REM ese mayuscula
3390            IF dbh THEN GOSUB 3510 ELSE GOSUB 3430                   :REM equivale   en
3400            LPRINT CHR$(13);CHR$(27)+"J"+CHR$(27);                   :REM version  7  a
3410    WEND                                                             :REM division
3420    RETURN                                                           :REM entera.
3430            FOR a=1 TO LEN(b$)
3440                    cde=ASC(MID$(b$,a,1))

3450                    FOR b=cde*8 TO cde*8+7
3460                            LPRINT CHR$(q(b));
3470                            IF dba THEN LPRINT CHR$(q(b));
3480                    NEXT
3490            NEXT
3500            RETURN
3510            FOR a=1 TO LEN(b$)
3520                    cde=ASC(MID$(b$,a,1))
3530                    FOR b=cde*8 TO cde*8+7
3540                            LPRINT CHR$(INT(w(q(b))/256));
3550                            IF dba THEN LPRINT CHR$(INT(w(q(b))/256));
3560                    NEXT
3570            NEXT
3580            LPRINT CHR$(13)+CHR$(27)+"J"+CHR$(24);
3590            LPRINT CHR$(27)+elpt+CHR$(top MOD 256)+CHR$(top#256);
3600            FOR a=1 TO LEN(b$)
3610                    cde=ASC(MID$(b$,a,1))
3620                    FOR b=cde*8 TO cde*8+7
3630                            co=w(q(b))
3640                            con=((INT(co/2) AND 127)*2):IF co/2-INT(co/2) THEN con=con+1
3650                            LPRINT CHR$(con);
3660                            IF dba THEN LPRINT CHR$(con);
3670                    NEXT
3680            NEXT
3690            RETURN
3700 PRINT eh;"CARGAR JUEGO GRABADO"
3710 PRINT FN at$(15,5,"Nombre del fichero: )");
3720 INPUT "",enm
3730 IF FIND$(enm)<>"" THEN 3780
3740 PRINT"No existe ese fichero.";CHR$(7)
3750 FOR a=1 TO 2000
3760 NEXT
3770 RETURN
3780 PRINT eh;"(Cargando ";enm;" en RAM...)"
```

```
3790 OPEN "I",1,enm
3800 OPEN "O",2,"m:datos.gen"
3810 FOR a=0 TO 2048
3820    INPUT #1,s
3830    WRITE #2,s
3840 NEXT
3850 CLOSE 1,2
3860 RETURN
3870 PRINT eh;"GRABAR JUEGO EN DISCO"
3880 PRINT FN at$(15,5,"Nombre del fichero: >");
3890 INPUT "",enm
3900 IF FIND$(enm)="" THEN 3970
3910 PRINT eh;"ESE FICHERO YA EXISTE"
3920 PRINT FN at$(10,20,"1  Conservar")
3930 PRINT FN at$(13,20,"2  Borrar")
3940 h=1: stp=3: ev="12": stv=7: sth=19
3950 GOSUB 1380
3960 IF h=1 THEN RETURN
3970 OPEN "O",1,enm
3980 FOR a=0 TO 2048
3990    WRITE #1,q(a)
4000 NEXT
4010 CLOSE 1
4020 RETURN
4030 PRINT eh
4040 END
```

```
10 REM subprogama 1
20 REM Capta veinte  nombres,  apellidos  y  direcciones
30 REM en las variables NOM$, AP$ y DOM$. Se  aprovechan
40 REM los codigos de control de pantalla para hacer mas
50 REM estetico el resultado.
60 DIM nom$(20),ap$(20),dom$(20)
70 PRINT CHR$(27)+"E"+CHR$(27)+"H":REM borra pantalla
80 PRINT "Subprograma 1: Entrada de datos"
90 FOR a=1 TO 20
100 PRINT CHR$(27);"Y";CHR$(34);CHR$(32);"Alumno numero";a
110 PRINT CHR$(27);"Y";CHR$(36);CHR$(36);"Nombre:      ";
120 INPUT "",nom$(a)
130 PRINT CHR$(27);"Y";CHR$(38);CHR$(36);"Apellidos:   ";
140 INPUT "",ap$(a)
150 PRINT CHR$(27);"Y";CHR$(40);CHR$(36);"Domicilio:   ";
160 INPUT "",dom$(a)
170 PRINT:PRINT "Algun error ( SI=<s> / NO=<INTRO> ) ";
180 a$=INPUT$(1)
190 IF a$="s" THEN PRINT CHR$(27)+"E"+CHR$(27)+"H":GOTO 100
200 IF a$<>CHR$(13) THEN PRINT CHR$(7);CHR$(27);"A";:GOTO 170
210 REM El caracter 13 es el resultado de la pulsacion de INTRO o RETURN.
220 PRINT CHR$(27)+"E"+CHR$(27)+"H"
230 NEXT
240 REM Proceso terminado.
```

OPERACIONES BÁSICAS DE DISCO



ASIC dispone de funciones para manejo del disco propias, además de las de CP/M, que permiten obtener directorios, mostrar ficheros de texto, cambiar nombres, borrar ficheros, etc...

Para mostrar el directorio, tenemos **DIR,** con las mismas posibilidades respecto a símbolos comodín que ofrece CP/M. La función equivalente propia de BASIC es **FILES,** y admite los mismos símbolos pero entre comillas. Estos son ejemplos válidos de búsqueda en el directorio de disco:

```
dir
files
dir *.bas
files "*.bas"
dir gba?.??t
files "gba?.??t"
```

Existe otra función para saber si un determinado fichero está o no en el disco. Es realmente útil cuando los programas manejan ficheros que pueden existir o no. Se trata de **FIND$(f,n),** donde **f** es una expresión de fichero con o sin símbolos comodín.

FIND$ busca en el directorio un fichero cuyo nombre coincida con lo indicado en el argumento. Si lo encuentra, genera una cadena de 12 caracteres conteniendo el nombre completo. Si se especifica **n** (es opcional), **FIND$** obtendrá el fichero número **n** del conjunto de aquellos que coincidan con el nombre especificado.

Por ejemplo, si en el disco tenemos los ficheros:

GENCAR.BAS, GENCHR.BAS, GENCAR.COM

y queremos obtener el segundo fichero que empiece por **GEN,** escribiremos:

a$=find$("gen*.*",2)

con lo cual **a$** contendrá **GENCHR.BAS.** Si escribimos un nombre que no existe, **a$** quedará vacía.

Otra orden de CP/M disponible en BASIC es **TYPE.** Ya se estudió esta utilidad en la Biblioteca Amstrad, por lo que no trataremos de ella ahora. La orden de BASIC equivalente es **DISPLAY,** y necesita, a diferencia de **TYPE,** que el nombre del fichero esté entre comillas.

Para cambiar de nombre disponemos de la orden **REN** de CP/M, y **NAME** propia del BASIC. La disposición de los nombres en **REN** es:

ren nombre nuevo=nombre antiguo

La orden **NAME** de BASIC, por el contrario, tiene la forma:

name nombre antiguo as nombre nuevo

as debe figurar tal y como se escribe; los nombres de fichero deben estar entre comillas.

Podemos borrar ficheros con otras dos órdenes. La de CP/M es **ERA,** y la forma de utilizarlo es igual a la del sistema. La orden de BASIC para borrar ficheros es **KILL,** y el nombre o expresión con símbolos comodín debe estar entre comillas. Ninguna de las dos instrucciones admite la expresión ***.***, la cual será rechazada con el mensaje *File name invalid.*

En CP/M podemos cambiar la unidad implícita simplemente mencionando la letra correspondiente. En BASIC, existe una orden para ello, que respeta en gran parte el método anterior. Se escribe:

option files u

donde **u** es el nombre de la unidad, entre comillas, por ejemplo, para cambiar la unidad implícita a M (disco RAM), escribiremos:

option files "m"

Existe también en CP/M un control destinado a informar al siste-

ma de que se ha cambiado de disco. Se trata de ↑ **C** (control C) y en BASIC hay que sustituirlo por **RESET.** Esta orden se debe ejecutar cuando se haya cambiado el disco antiguo y ya esté en su puesto el nuevo, de manera que **RESET** obligue a BASIC a leer el nuevo directorio.

GRABACIÓN Y RECUPERACIÓN DE PROGRAMAS

Para grabar un programa en el disco, se utiliza **SAVE nombre de fichero.** El *nombre de fichero* debe estar entre comillas, y respetar siempre las normas para los ficheros de CP/M. Al ejecutar esta orden, BASIC codifica de manera especial el programa entero, y a continuación graba la secuencia de códigos en el disco. El resultado es ilegible para **DISPLAY** o **TYPE,** pero se ahorra gran cantidad de espacio.

Muchas veces es necesario grabar el fichero en ASCII, es decir, de forma legible, como si se tratara de un fichero de texto. Esto es especialmente útil si se va a compilar un programa, o simplemente cuando se va a modificar con un procesador de texto para imprimir después.

La forma de obtener una copia del programa en ASCII es:

save "nombre programa",a

Un fichero de este tipo puede ser examinado con las órdenes **DISPLAY** o **TYPE.**

Otra opción de **SAVE** permite codificar el programa de manera que es imposible acceder al listado de ninguna manera una vez grabado. El fichero también es ilegible, por supuesto. No existe modo alguno de recuperar el listado de un programa grabado así. Para conseguir una copia de este tipo, escribiremos:

save "nombre programa",a

Las formas codificadas son propias de Mallard BASIC, y por tanto no son compatibles con otros intérpretes BASIC.

Para cargar un programa en memoria se utiliza la orden **LOAD nombre de fichero debe existir en el disco de lo contrario se emite el mensaje** *File not found.* Si en el nombre del fichero no se especifica distintivo, se supone implícitamente **.BAS,** igual que en **SAVE.**

Al cargar en memoria el programa indicado, se borra el anterior. Puede añadirse un parámetro, opcional, el cual indica a BASIC que debe ejecutar el programa en cuanto se cargue en la memoria. Dos ejemplos son:

load "miprog"
load "tuprog.tec",r

El primer ejemplo carga el fichero MIPROG.BAS en memoria. El segundo, carga el fichero TUPROG.TEC y a continuación lo ejecuta.

También pueden ejecutarse ficheros directamente con RUN, pero indicando el nombre de fichero.

run "nombre" es equivalente a **load "nombre" run.**
run "nombre",r es equivalente a **load "nombre",r.**

MEZCLAR Y ENCADENAR PROGRAMAS

Podemos «mezclar» programas. El resultado concreto de esta operación es un programa que contiene todas las líneas del último programa y las líneas del antiguo cuyo número no coincida con alguna de las del nuevo. La orden es **MERGE nombre fichero.** El «antiguo programa» es el que se encuentre en la memoria, y el «nuevo» es el que se halla en el disco.

Si se mezcla un programa protegido con **save "nombre",P** con otro desprotegido, el resultado está protegido.

Otra instrucción relacionada con el encadenamiento de programas es **CHAIN.** Esta orden permite que un programa cargue y ejecute a otro, con la opción de conservar el actual y las variables, en todo o en parte.

Las posibles expresiones de **CHAIN** son:

chain "nombre de fichero"

el resultado es la carga y ejecución del programa indicado, borrando el anterior y sus variables. El programa comienza a ejecutarse en la primera línea.

chain "nombre fichero",número de línea

igual que la expresión anterior, pero el programa empieza a ejecutarse en la línea indicada.

chain "nombre fichero",número de línea,all

igual que las anteriores, pero se conservan todas las variables. El número de línea es opcional, pero si no se indica se deben dejar las comas correspondientes.

chain merge "nombre fichero"

igual que **chain «nombre fichero»,** pero sin borrar el programa actual.

chain merge "nombre fichero",número de línea,all

igual que **chain «nombre fichero»,número de línea,all,** pero sin borrar el programa actual.

chain merge "nombre fichero",número de línea,all,delete margen

igual que **chain «nombre fichero»,número de línea,all,** pero conservando el programa actual excepto las líneas indicadas en el **margen.** La sintaxis de este margen es la misma que la aplicada para **DELETE** (tratado en el anterior número sobre BASIC).

Como vemos, utilizando **CHAIN** las únicas dos opciones respecto a las variables son conservarlas todas o borrarlas todas. Pero BASIC dispone de dos órdenes para especificar un conjunto de variables que no han de ser borradas. Se trata de **COMMON,** y la forma general de escribirlo es:

common (lista de variables)

La lista de variables puede contener las de cualquier tipo, dimensionadas o no.

Al margen de la actuación de **CHAIN** y **CHAIN MERGE,** podemos borrar las variables no «protegidas» por **COMMON.** Lo conseguiremos escribiendo:

COMMON RESET

cuyo efecto es similar al de **CLEAR** (tratado en el anterior número) pero sin afectar a las variables mencionadas en **COMMON.**

El proceso de mezcla y encadenamiento de programas no es un método muy usual. En ocasiones puede resultar especialmente complicado conservar determinadas variables; el problema se agrava si hay ficheros abiertos durante el proceso y el nuevo programa debe hacerse cargo de ellos.

De todos modos resulta interesante experimentar este intercambio de programas. La práctica, como siempre, nos permitirá dominar estas operaciones.

LOS FICHEROS

Los ficheros son conjuntos de datos estructurados de una manera determinada y almacenados en algún tipo de memoria auxiliar. En el caso del PCW se trata de discos de tres pulgadas.

El almacenamiento en sí no representa ningún problema, pues corre a cargo del sistema operativo; las instrucciones BASIC para ello son bien simples. La dificultad de la gestión de ficheros se encuentra en el modo a través del cual se accede a los datos que en ellos se guardan.

TIPOS DE FICHEROS

Mallard BASIC admite tres tipos de acceso: secuencial, aleatorio y por claves.

En el secuencial, los datos se graban y recuperan de una sola pasada, secuencialmente (uno tras otro). Los datos grabados pueden ser numéricos o alfanuméricos.

Este tipo de fichero se utiliza cuando existen pocos datos para

guardar, o bien cuando es necesario cargarlos todos en memoria para poder utilizarlos correctamente.

Su ventaja es la sencillez de operación. Su inconveniente es la obligación de leer TODOS los datos secuencialmente, de manera que si deseamos acceder al último dato, debemos leer previamente todos los grabados anteriormente.

Imaginemos un rollo de cinta de papel donde se escriben, uno por uno, los elementos de una lista; nos vemos obligados a pasar todo el rollo para llegar al último elemento, aunque los demás no interesen.

El acceso aleatorio permite obtener datos de cualquier parte del fichero, sin tener que acceder a todo lo anterior. Esto presenta un problema: debemos establecer una unidad de información básica que es a la que se accederá aleatoriamente.

Como ejemplo del acceso aleatorio tenemos un fichero convencional. El fichero completo es un cajón, y en su interior están las fichas. Cuando accedemos a una unidad de información básica, lo que hacemos es coger una tarjeta o ficha, que en BASIC se llama REGISTRO.

Suponiendo que el fichero sea de una biblioteca, cada ficha contendrá al menos la siguiente información:

— Número de libro.
— Clave de identificación.
— Título.
— Autor.

A estos cuatro apartados incluidos dentro de cada ficha o registro se les llama CAMPOS.

El sistema CP/M admite como máximo 65535 registros (es decir, fichas), y cada ficha contiene como máximo 128 caracteres (sin limitación en el número de campos).

Se puede grabar y escribir aleatoriamente, y la unidad de grabación es siempre la ficha o registro. Para acceder a uno de ellos, basta con indicar el número de orden que tiene dentro del fichero.

El acceso por claves permite acceder a un registro pero no sólo por su número de orden, lo cual en ocasiones resulta pesado, sino por una o varias CLAVES asociadas a dicho registro. Por ejemplo, tenemos un fichero de personal con la siguiente definición de registro:

— Primer apellido: 30 caracteres
— Segundo apellido: 30 »
— Nombre: 20 »
— Número: 2 »

— Puesto: 10 »
— Salario: 7 »

99 caracteres total.

Para buscar al empleado José García Verdugo, deberíamos leer uno por uno todos los registros; lo normal es que no se sepa qué número de orden tiene ese registro dentro del fichero, y un usuario ajeno a la programación no tiene porqué molestarse en saberlo.

Si el registro llevara como clave asociada el primer apellido, se indicaría éste en una búsqueda por claves y el proceso sería mucho más rápido, puesto que BASIC se encargaría de buscar la ficha automáticamente.

Estos son los tres tipos de ficheros disponibles; no puede accederse a ningún fichero con un tipo de acceso distinto a aquél en el cual fue creado.

BUFFERS

Pasemos a un detalle un poco más técnico dentro del tema de los datos grabados. Se trata del buffer: dispositivo utilizado en la gestión de ficheros por claves.

El buffer es una zona de memoria, destinada a albergar temporalmente los datos que van o vienen del disco.

Para comprender su objetivo, imaginemos a una directora de marketing y su secretario. Ella dicta un informe, y el secretario lo escribe. Es evidente que lo hablado es mucho más rápido que lo escrito, y sin embargo nada se pierde; el secretario lo escribe todo porque retiene en su memoria temporalmente parte de lo que escucha. Esta memoria temporal es lo que llamamos buffer.

En los ordenadores tiene un funcionamiento especial para conseguir una ejecución más rápida; mientras no esté totalmente lleno, no se descarga, a menos que se indique que no habrá más traslado de datos.

Podemos variar el tamaño del buffer con la instrucción **BUFFERS n.** **n** es un número que indica un bloque de 128 bytes que debe asignarse al buffer de disco. La orden **buffers 4** asigna 128*4 bytes de buffer.

Cuanto mayor sea este espacio, más rápido será el proceso de traslado, porque se reduce al mínimo el número de accesos físicos al disco para descargar el buffer. Para comprender esto, basta imaginar un secretario con un buffer de dictado de 14 folios. El proceso de escritura completo sería rapidísimo.

ACCESO SECUENCIAL

n ejemplo de fichero secuencial puede ser el que guarde el nombre y dirección de los 20 alumnos de una clase, y su domicilio. Dado el pequeño número de datos, es correcto el utilizar un acceso secuencial, puesto que cargarlos todos a la vez no supone un gasto excesivo de memoria.

Estableceremos para recoger los datos tres variables alfanuméricas dimensionadas: **NOM$, AP$,** y **DOM$.**

Primero, el programa debe captar los datos por el teclado. El proceso de recogida de datos ya lo conocemos, y puede ser, por ejemplo, algo parecido al subprograma 1 (habrá que teclearlo junto con los siguientes).

Ahora debemos crear el fichero. La orden de apertura y/o creación de ficheros es

OPEN "modo",número de fichero,"nombre"

modo es una letra (no importa si está en mayúsculas o en minúsculas) indicadora del modo de acceso. El número de fichero sirve después para identificarlo, puesto que pueden abrirse varios a la vez. A medida que vayamos abriéndolos, iremos dando los números: 1, 2, 3...

Como ya sabemos, en los ficheros secuenciales hay que leer o escribir de una sola vez, y no puede añadirse en un fichero ya escrito. Por lo tanto, la orden de abrir un fichero secuencial para escritura no abre, sino que crea uno nuevo, destruyendo el anterior que tuviera el mismo nombre.

El «modo» de escritura secuencial es **O** (de *output,* salida de datos). Puesto que es el primer fichero que abrimos, tendrá el número 1, y lo llamaremos CLASE.DAT. El fichero se abre así:

open "0",1,"clase.dat"

Ahora debemos escribir ordenadamente los datos, con la siguiente secuencia:

— Primer apellido
— Nombre
— Domicilio

de manera que la primera variable para escribir será **AP$,** luego **NOM$,** y por último **DOM$.**

La instrucción de escritura sobre ficheros secuenciales es **WRITE# número de fichero,lista de datos.** El signo «#» se obtiene en el PCW, teclado versión española con las teclas **EXTRA** y **4,** aunque el signo que aparezca es distinto (Pt).

El número de fichero sirve para indicar en cuál de los ficheros se efectuará la escritura. Vemos ahora lo útil que es asignar un número a cada uno; nos ahorramos mencionar su nombre cada vez que accedemos a él.

La lista de datos es un conjunto de uno o varios datos constantes (cadenas y números) o variables (numéricas y alfanuméricas) separadas entre comas. El efecto de **WRITE** es escribir esos datos en el disco, entrecomillados, y separados entre comas.

Un nuevo **WRITE** escribiría su lista de datos pero en la línea siguiente. Vamos a grabar en el disco los datos de cada alumno agrupados. El proceso de apertura y grabación puede ser como el del subprograma 2.

Una vez concluido el proceso de escritura es necesario cerrar el fichero. Recordemos que el buffer no se descarga hasta que esté lleno o se le indique que no hay más traslado de datos. Con **CLOSE** (de cerrar) indicaremos al buffer este hecho, y se descargará en el disco. Si no ejecutamos dicha instrucción, lo más seguro es que parte o todo el fichero quede en memoria sin llegar a grabarse.

La forma general de la instrucción es:

close lista de números de fichero

```
250 REM subprograma 2
260 REM Graba los veinte apellidos, nombres y direcciones
270 REM en este orden en un fichero  secuencial.  Durante
280 REM el proceso se imprimen en la pantalla los alumnos
290 REM que se van grabando en el disco.
300 PRINT "Subprograma 2: Grabacion de datos."
310 OPEN "O",1,"clase.dat"
320 FOR a=1 TO 20
330 WRITE #1,ap$(a),nom$(a),dom$(a)
340 PRINT "Grabando: Alumno";a;":"
350 PRINT TAB(22);ap$(a)
360 PRINT TAB(22);nom$(a)
370 PRINT TAB(22);dom$(a)
380 NEXT
390 CLOSE 1
400 PRINT CHR$(27)+"E"+CHR$(27)+"H"
410 REM Proceso terminado.
```

Si se escribe **CLOSE** sin más, se cierran todos los ficheros abiertos en ese momento. Nosotros en el subprograma especificamos el nombre del fichero porque, como veremos más adelante, conviene hacer las cosas bien. Si se va a indicar más de un número, deben separarse entre comas.

Durante la ejecución del subprograma 2 habremos comprobado que la grabación es realmente rápida, y que no siempre el disco se está moviendo; esto se debe a la acción del buffer.

Supongamos que dentro del proceso de grabación de alumnos, etc... hubiera una parte para la lectura. Debemos abrir el fichero, pero esta vez no para escribir, puesto que así lo borraríamos. El modo de lectura secuencial es **I.** La forma de abrir nuestro fichero sería:

open "i",1,"clase.dat"

Para recoger los datos se utiliza la instrucción **INPUT,** pero de un modo algo especial:

input# número de fichero, lista de datos

Igual que en **WRITE,** el signo «#» se obtiene con **EXTRA** y **4,** aunque aparezca **Pt.** Ya conocemos las condiciones de los parámetros incluidos, puesto que son similares a los de **WRITE.** El subprograma 3 es la última parte del programa completo y permite la lectura de los datos grabados. Cuando se termina ésta, se debe cerrar el fichero con **CLOSE.**

Como podemos observar, el proceso en sí de la escritura o lectura de datos del fichero es muy breve; la mayor parte del espacio del listado se dedica a presentación, menús, etc.

```
420 REM subprograma 3
430 REM Lee del disco los veinte grupos de datos a la vez,
440 REM los almacena en las correspondientes variables
450 REM dimensionadas y permite examinarlas una por una o
460 REM todas a la vez.
470 PRINT "Subprograma 3: Lectura de datos."
480 OPEN "I",1,"clase.dat"
490 FOR a=1 TO 20
500 PRINT CHR$(27);"Y";CHR$(35);CHR$(32);"Leyendo alumno";a
510 INPUT R1,ap$(a),nom$(a),dom$(a)
520 NEXT
530 CLOSE 1
540 PRINT CHR$(27)+"E"+CHR$(27)+"H"
550 PRINT "Opciones:"
560 PRINT
570 PRINT ,"1- Ver un alumno"
580 PRINT ,"2- Ver lista completa"
590 PRINT ,"3- Terminar"
600 a$=INPUT$(1)
610 IF INSTR("123",a$)=0 THEN PRINT CHR$(7);:GOTO 600
620 ON VAL(a$) GOTO 630,760,890
630 PRINT CHR$(27)+"E"+CHR$(27)+"H"
640 INPUT "Numero de alumno: ",n%
650 IF n%<1 OR n%>20 THEN PRINT CHR$(7);CHR$(27);"A";CHR$(13);:GOTO 640
660 PRINT
670 PRINT "Alumno numero";n%
680 PRINT
690 PRINT ,"Nombre:     ";nom$(n%)
700 PRINT ,"Apellidos: ";ap$(n%)
710 PRINT ,"Domicilio: ";dom$(n%)
720 PRINT
730 PRINT "Pulse una tecla"
740 a$=INPUT$(1)
750 GOTO 540
760 PRINT CHR$(27)+"E"+CHR$(27)+"H"
770 FOR a=1 TO 20
780 PRINT "Alumno numero:";a
790 PRINT
800 PRINT ,"Nombre:     ";nom$(a)
810 PRINT ,"Apellidos: ";ap$(a)
820 PRINT ,"Domicilio: ";dom$(a)
830 PRINT
840 PRINT "Pulse una tecla"
850 a$=INPUT$(1)
860 PRINT CHR$(27);"A";CHR$(27);"o";CHR$(13);
870 NEXT
880 GOTO 540
890 PRINT CHR$(27)+"E"+CHR$(27)+"H"
900 END
920 REM Programa terminado.
```

MODIFICACIÓN DE UN FICHERO

En realidad, no es posible modificar un fichero secuencial. Sabemos que al abrirlo para escritura se borra el anterior, y abriéndolo para lectura BASIC no admite **WRITE.** La forma de modificar el fichero es utilizando uno auxiliar.

El proceso es el siguiente: se carga el fichero secuencial en memoria, y se modifican o añaden los datos necesarios. A continuación, se crea un nuevo fichero secuencial auxiliar, al que llamaremos, por ejemplo, TEMP.$$$. En él escribiremos todos los datos ya modificados, y por último cerraremos el fichero. Si no ha habido errores, es el momento de borrar (con **KILL** o **ERA**) el fichero antiguo, y cambiar el nombre al fichero auxiliar. Le pondremos, por supuesto, el del original (con **REN** o **NAME).**

El resultado es una modificación, pero sin que en realidad se haya retocado; la verdad es que se ha escrito de nuevo todo el fichero.

También podemos copiar ficheros secuenciales, abriendo uno en lectura y uno o varios en escritura, donde se escribirán los datos leídos en el primero. Por supuesto, las distintas instrucciones **OPEN** que intervienen deberán tener un número distinto de fichero, o de lo contrario BASIC emitirá el error *File already open.*

El manejo de ficheros secuenciales es muy simple. La única precaución es abrirlo y cerrarlos correctamente y no confundir los modos (lectura y escritura).

FIN DE FICHERO

Hay muchas ocasiones en las que no sabemos con exactitud el tamaño de un fichero secuencial, es decir, el número de datos que contiene. Esto ocurre, por ejemplo, si tenemos un pequeño programa de contabilidad doméstica donde la lista de «movimientos» se guarda en un fichero secuencial. El número de líneas no se conoce, y aumenta a medida que se utiliza el programa.

Para saber cuándo termina de leerse un fichero secuencial, BASIC dispone de la función **EOF** *(End Of File,* final de fichero). Se escribe en general como:

eof(número de fichero)

y el resultado de la función es − 1 si se ha llegado al final del fichero cuyo número se indica, ó 0 si no es así.

La forma correcta de extraer datos de un fichero secuencial con ayuda de **EOF** es mediante un bucle **WHILE - WEND,** en el cual la condición es que **EOF** no gener el valor − 1. La expresión **NOT (EOF(num))** será cierta mientras no se llegue al final, y ésta es la que utilizamos en el programa ejemplo.

Al comenzar la ejecución, el programa pide un número, que será la cantidad de valores aleatorios que escriba en un fichero secuencial.

A continuación, sin utilizar esa cantidad, se leerá el fichero con la ayuda de **EOF** para no intentar leer más allá de su longitud.

Como prueba, se puede eliminar el bucle **WHILE-WEND** y ver lo que pasa cuando BASIC llega al final del fichero. También se puede probar a introducir 0 datos. **EOF** lo detectará inmediatamente y el error que se produce no tiene que ver con el disco, sino con la media aritmética que intenta realizarse *(Division by zero)*.

```
10 REM Programa para comprobar el funcionamiento de
20 REM EOF en un  fichero  secuencial.
30 PRINT CHR$(27)+"E"+CHR$(27)+"H"
40 INPUT "Introduzca el numero de datos para escribir ",n%
50 OPEN "O",1,"m:aleat"
60 FOR a=1 TO n%
70 WRITE #1,INT(RND*100)+1
80 PRINT CHR$(27);"Y";CHR$(37);CHR$(32);"Escribiendo dato";a
90 NEXT
100 CLOSE 1
110 PRINT "Fichero cerrado"
120 FOR a=1 TO 2000:NEXT:REM esto hace una pausa
130 n%=0:REM no vamos a hacer trampa; borramos el numero de datos.
140 tot=0
150 contador=0
160 PRINT "Abriendo fichero"
170 OPEN "I",1,"m:aleat"
180 WHILE NOT EOF(1)
190 INPUT #1,a
200 tot=tot+a
210 contador=contador+1
220 WEND
230 PRINT "Cantidad total: ";contador
240 PRINT "Suma de numeros:";tot
250 PRINT "Media:          ";tot/contador
```

EL ACCESO ALEATORIO

a hemos explicado en qué consistía el acceso aleatorio, y también la necesidad de definir una ficha o registro que será la unidad de acceso. Cada registro puede tener uno o vàrios campos.

Como ejemplo de acceso aleatorio tomaremos una lista de preguntas con sus respuestas correctas. Cada pregunta tendrá un número, que coincidirá con su número de orden dentro del fichero.

Los pasos para crear y escribir en un fichero aleatorio son: abrirlo con la instrucción **OPEN** adecuada, definir el registro asignando variables de campo, dar datos a las variables de campo, grabarlas y por último cerrar el fichero.

Para abrir un fichero aleatorio, no tenemos problemas sobre lectura y escritura, y por tanto, existe una sola letra para designar este modo de acceso:

open "R", número de fichero,"nombre de fichero"

Una instrucción correcta de apertura de fichero puede ser:

open "R",1,"m:cuentas.dat"

Una vez abierto el fichero, y antes de hacer cualquier otra cosa, se

deben definir las variables de campo, es decir, las variables dedicadas a contener los datos de cada registro. Por ejemplo, supongamos que en nuestro programa de preguntas y respuestas la definición de registro tiene este aspecto:

— Pregunta	50 caracteres
— Respuesta	30 »
— Valoración	2 »
	82 caracteres total

Tomaremos las variables PREG$, RESP$ y VALOR$ (recordemos que en ficheros aleatorio y por claves, los valores numéricos deben ser convertidos a cadenas).

La instrucción que define un registro es **FIELD** (campo) y la forma de escribirla es:

field número de fichero, lista de variables de campo

Las variables de campo son las que acabamos de mencionar. Dentro de la orden **FIELD,** cada una debe llevar la longitud máxima de caracteres que contendrá. Esto se escribe de la siguiente manera:

longitud máxima as nombre de variable

as debe aparecer tal y como se muestra. De manera que dentro de la lista de variables de campo, y dentro de nuestro ejemplo, el campo «preguntas» se definirá así:

50 as preg$

Vamos con la instrucción completa, es decir, la que define el registro completo. Los distintos campos se separan entre comas:

field 1,50 as preg$,30 as resp$,2 as valor$

Con esto hemos definido un registro de las características descritas. En una sola orden **FIELD** debe estar la definición de registro completa. No es correcto hacerlo en dos o más líneas. Tampoco es correcto definir un registro con más de 128 caracteres.

Para escribir en el disco un registro, se asigna el contenido a las variables de campo y a continuación se ejecuta la orden **PUT,** con la forma general

put número de fichero,número de registro

El número de fichero indica en cuál debe escribirse. BASIC sabe qué variables de campo corresponden a cada fichero abierto, si hay varios.

El número de registro es un parámetro opcional (si se omite no se escribe la coma) y al incluirlo, BASIC escribe en el registro con el

número de orden indicado. Si no se indica, BASIC escribe en el siguiente.

Un ejemplo para comprender esto es imaginar de nuevo el fichero clásico, con sus fichas. Su **PUT** contiene el parámetro de número de registro, entonces la ficha se meterá en el lugar correspondiente. Si no se indica, se meterá detrás, al final del fichero.

La manera de asignar contenido a las variables de campo es algo especial. Como tienen longitud limitada, pero DEBEN tener esa longitud, es necesario utilizar dos órdenes que ya hemos tratado anteriormente: las de asignación justificada. Podemos justificar por la derecha (con **RSET**) **o por la izquierda (con LSET).** Lo normal es que se utilice **LSET,** para dejar los espacios sobrantes, si los hay, a la derecha.

Para asignar una pregunta, la instrucción correcta podría ser:

lset preg(="Tengo 6 manzanas y quito una. ¿Cuántas quedan?"

La cadena tiene 46 caracteres, luego PREG$ tendría cuatro espacios al final para llegar a los 50. Las variables de los otros dos campos se asignarían de manera similar:

```
lset resp$="cinco"
lset valor$=mki$(299)
```

La última orden, tras haber asignado los valores es, simplemente, **put 1,** con lo cual un registro se añade al fichero, al final del mismo.

Una vez grabados todos los datos, debe ejecutarse la orden **CLOSE** para descarar el buffer y que todos los datos pasen físicamente al disco.

Para leer datos, el fichero se abre con la misma orden, es decir, con:

open "R",número de fichero,"nombre de fichero"

ya que no existe problema de reescritura. El proceso siguiente es definir el registro con la correspondiente orden **FIELD,** y por último leer datos. La orden de lectura es:

get número de fichero, número de registro

BASIC asigna a las variables de campo el contenido del registro. Si no se indica el número, se toma el primer registro (si aún no se ha leído ninguno) o el siguiente al último leído. Al terminar la lectura de datos, debe ejecutarse **CLOSE.**

La lectura de un registro de nuestro ejemplo anterior (el 23) se efectuaría así:

```
get 1,23
print preg$
print resp$
print cvi (valor$)
```

(recordemos las funciones de conversión cadena-número).

Incluimos un programa de preguntas y respuestas que utiliza dos ficheros de acceso aleatorio. Nos vemos obligados a tomar dos, puesto que la longitud de registro máxima es 128, y ésta es la cantidad de caracteres que destinamos a la pregunta. El otro fichero guarda las respuestas; una correcta, grabada en primer lugar, y otras tres incorrectas. A cada una se le destinan 32 caracteres, luego el total es de 128 también.

El programa plantea preguntas (si están grabadas) y ofrece una de las cuatro respuestas aleatoriamente. Deben rechazarse las incorrectas con la tecla «−» o aceptar la correcta con «+».

El hecho de que existan dos ficheros abiertos al mismo tiempo no dificulta en absoluto la relación entre pregunta y respuesta, puesto que se graban con el mismo número de orden dentro de su correspondiente fichero.

Gran parte del listado se dedica a un mejor acabado estético, con un tablero y una ficha. La puntuación está relacionada con el tiempo de respuesta del jugador, utilizando para ello la variable *time,* que va descendiendo continuamente.

Debe responderse el máximo número de preguntas, pasando y contestando correctamente en los casilleros que hacen esquina, para llegar así a la siguiente fase.

Las preguntas se dividen en cuatro temas muy amplios. Para clasificar éstos en el disco se opta por el siguiente criterio: el primer tema ocupa los registros 1, 5, 9, etc. (fórmula general 1+4*n, variando n desde 1 en adelante). El segundo tema ocupa los registros 2, 6, 10 (2+4*n), etc..., de manera que para encontrar preguntas del mismo tema habría que leer un registro y los siguientes que distarán cuatro del anterior.

Se utilizan en este programa determinados caracteres gráficos situados por encima de los códigos ASCII normales, es decir, por encima de 128. Son de gran ayuda para imprimir el tablero (LocoScript los utiliza para los menús).

```
10 REM ************************************************************************
20 REM * TESTER - 1986 - ANTONIO  G. VERDUGO  - GRAN BIBLIOTECA AMSTRAD *
30 REM ************************************************************************
40 REM
50 REM
60 REM
70 REM
80 REM
90 REM
100 REM
110 REM
120 REM
130 OPTION BASE 1
140 DIM resp2$(4)
150 DIM posn(28,2)
160 FOR a=1 TO 8
170 posn(a,1)=5:posn(a,2)=1+((a-1)*5)
180 posn(23-a,1)=19:posn(23-a,2)=1+((a-1)*5)
190 NEXT
200 FOR a=9 TO 14
210 posn(a,1)=5+2*(a-8):posn(a,2)=36
220 posn(37-a,1)=5+2*(a-8):posn(37-a,2)=1
230 NEXT
240 DIM tema$(4)
250 FOR a=1 TO 4
260 READ tema$(a)
270 NEXT
280 DEFSTR h
290 DIM hdado(6)
300 DATA Ciencias,Geografia e Historia,Cine y Espectaculos,Arte y Literatura
310 hesc=CHR$(27)
320 home=hesc+"E"+hesc+"H"
330 hon=hesc+"p"
340 hcursor.on=hesc+"e"
350 hcursor.off=hesc+"f"
360 hoff=hesc+"q"
370 hsub.on=hesc+"r"
380 hsub.off=hesc+"u"
390 hcab=SPACE$(35)+hsub.on+"T E S T E R"+hsub.off+STRING$(3,10)+CHR$(13)
400 hdel.line=hesc+"M"
410 hline.back=hesc+"I"
420 hcr=CHR$(13)
430 hpant1=STRING$(4,138)
440 hpant2=hpant1+CHR$(142)
450 hpant3=CHR$(134)+hpant2+hpant2++hpant2+hpant2+hpant2+hpant2+hpant2+hpant1+
CHR$(140)
460 hpant4=SPACE$(4)
470 hpant5=CHR$(133)+hpant4+CHR$(133)+hpant4+CHR$(133)+hpant4+CHR$(133)+hpant4+
CHR$(133)+hpant4+CHR$(133)+hpant4+CHR$(133)+hpant4+CHR$(133)+hpant4+CHR$(133)
480 hpant6=CHR$(135)+hpant1+CHR$(143)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+
CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(143)+hpant1+CHR$(141)
490 DEF FN at$(x,y,t$)=hesc+"Y"+CHR$(x+32)+CHR$(y+32)+t$
500 hpant7=CHR$(135)+hpant1+CHR$(143)+hpant1+CHR$(142)+hpant1+CHR$(142)+hpant1+
CHR$(142)+hpant1+CHR$(142)++hpant1+CHR$(142)+hpant1+CHR$(143)+hpant1+CHR$(141)
510 hpant8=CHR$(131)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+
CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(139)+hpant1+CHR$(137)
520 hcs=CHR$(150)+STRING$(5,154)+CHR$(156)
530 hci=CHR$(147)+STRING$(5,154)+CHR$(153)
540 hret=CHR$(10)+STRING$(7,8)
550 hdado(1)=hcs+hret+CHR$(149)+"     "+CHR$(149)+hret+CHR$(149)+"  "+CHR$(128)+
"  "+CHR$(149)+hret+CHR$(149)+"     "+CHR$(149)+hret+hci
560 hdado(2)=hcs+hret+CHR$(149)+CHR$(128)+"     "+CHR$(149)+hret+CHR$(149)+
"     "+CHR$(149)+hret+CHR$(149)+"     "+CHR$(128)+CHR$(149)+hret+hci
570 hdado(3)=hcs+hret+CHR$(149)+CHR$(128)+"     "+CHR$(149)+hret+CHR$(149)+"  "+
CHR$(128)+"  "+CHR$(149)+hret+CHR$(149)+"     "+CHR$(128)+CHR$(149)+hret+hci
580 hdado(4)=hcs+hret+CHR$(149)+CHR$(128)+"    "+CHR$(128)+CHR$(149)+hret+CHR$(149)+
"     "+CHR$(149)+hret+CHR$(149)+CHR$(128)+"    "+CHR$(128)+CHR$(149)+hret+hci
```

```
590 hdado(5)=hcs+hret+CHR$(149)+CHR$(128)+"    "+CHR$(128)+CHR$(149)+hret+CHR$(149)+
"  "+CHR$(128)+"  "+CHR$(149)+hret+CHR$(149)+CHR$(128)+"    "+CHR$(128)+CHR$(149)+
hret+hci
600 hdado(6)=hcs+hret+CHR$(149)+CHR$(128)+" "+CHR$(128)+" "+CHR$(128)+CHR$(149)+
hret+CHR$(149)+"      "+CHR$(149)+hret+CHR$(149)+CHR$(128)+" "+CHR$(128)+" "+
CHR$(128)+CHR$(149)+hret+hci
610 hdborra="       "+hret+"       "+hret+"       "+hret+"       "+hret+"       "
615 REM Linea 610: siete espacios entre las comillas.
620 PRINT home
630 PRINT hcab
640 a$(1)="Añadir fichas"
650 a$(2)="  Consultas  "
660 a$(3)="    Juego    "
670 a$(4)="     Fin     "
680 num%=4:s1=10:s2=33:vuelta%=1
690 GOSUB 3030
700 ON ptr GOTO 710,1290,1660,3190
710 OPEN "R",1,"b:krivial1"
720 OPEN "R",2,"b:krivial2"
730 FIELD 1,128 AS preg$
740 FIELD 2,32 AS resp$(1),32 AS resp$(2),32 AS resp$(3),32 AS resp$(4)
750 IF FIND$("b:krivial3")="" THEN 790
760 OPEN "I",3,"b:krivial3"
770 INPUT #3,numfich%
780 CLOSE 3
790 PRINT home
800 PRINT hcab
810 PRINT"Numero de fichas introducidas: >";numfich%
820 PRINT:PRINT "Siguiente ficha, numero ";numfich%+1;". Tema: ";tema$(((numfich%+1)
MOD 4)+1)
830 PRINT
840 hinput="Nueva pregunta"
850 houtput="pregunta"
860 max.out%=128
870 GOSUB 3140
880 LSET preg$=a$
890 hinput="Respuesta correcta"
900 houtput="respuesta correcta"
910 max.out%=32
920 GOSUB 3140
930 LSET resp$(1)=a$
940 hinput="Siguiente respuesta"
950 houput="respuesta"
960 max.out%=32
970 FOR a=2 TO 4
980 GOSUB 3140
990 LSET resp$(a)=a$
1000 NEXT
1010 PRINT home;hcab
1020 PRINT"Pregunta:"
1030 PRINT CHR$(252);preg$
1040 PRINT"Respuestas:"
1050 FOR a=1 TO 4
1060 PRINT CHR$(252);resp$(a)
1070 NEXT
1080 s1=20:s2=33
1090 a$(1)=" Grabar "
1100 a$(2)="Corregir"
1110 a$(3)="Abandono"
1120 num%=3
1130 GOSUB 3030
1140 IF ptr=2 THEN 790
1150 IF ptr=3 THEN CLOSE:GOTO 620
1160 PUT 1,numfich%+1
1170 PUT 2,numfich%+1
1180 numfich%=numfich%+1
1190 OPEN "O",3,"b:krivial3"
```

```
1200 WRITE #3,numfich%
1210 CLOSE 3
1220 PRINT home;hcab
1230 s1=10:s2=32
1240 a$(1)="   Continuar grabando   "
1250 a$(2)="Volver al menu principal"
1260 num%=2
1270 GOSUB 3030
1280 IF ptr=1 THEN 790 ELSE CLOSE:GOTO 620
1290 PRINT home;hcab
1300 PRINT"Consultas.";CHR$(10)
1310 OPEN "R",1,"b:krivial1"
1320 OPEN "R",2,"b:krivial2"
1330 IF FIND$("b:krivial3")="" THEN PRINT hon;"Sin datos.";hoff:a$=INPUT$(1):CLOSE:
     GOTO 320
1340 OPEN "I",3,"b:krivial3"
1350 INPUT #3,numfich%
1360 CLOSE 3
1370 FIELD 1,128 AS preg$
1380 FIELD 2,32 AS resp$(1),32 AS resp$(2),32 AS resp$(3),32 AS resp$(4)
1390 PRINT numfich%;" fichas disponibles en el disco."
1400 s1=10:s2=35
1410 a$(1)="Consultar"
1420 a$(2)="Abandonar"
1430 num%=2
1440 GOSUB 3030
1450 IF ptr=2 THEN CLOSE:GOTO 620
1460 PRINT home;hcab
1470 PRINT numfich%;" fichas disponibles."
1480 PRINT CHR$(10);"Numero de ficha para consultar ";CHR$(252);
1490 INPUT "",a%
1500 IF a%<1 OR a%>numfich% THEN PRINT hon;"No existe este numero. Reescriba, por
     favor";hoff;:a$=INPUT$(1):PRINT hdel.line;hline.back;hcr;:GOTO 1480
1510 GET 1,a%
1520 GET 2,a%
1530 PRINT home;hcab
1540 PRINT "Ficha numero";a%;"de";numfich%;". Tema ";CHR$(252);tema$((a% MOD 4)+1)
1550 PRINT"Pregunta:"
1560 PRINT CHR$(252)+preg$
1570 PRINT"Respuestas:"
1580 FOR a=1 TO 4
1590 PRINT CHR$(252);resp$(a)
1600 NEXT
1610 s1=20:s2=35
1620 a$(1)="Conforme"
1630 num%=1
1640 GOSUB 3030
1650 PRINT home;hcab:GOTO 1390
1660 PRINT home;hcab;hcursor.off
1670 OPEN "R",1,"b:krivial1"
1680 OPEN "R",2,"b:krivial2"
1690 FIELD 1,128 AS preg$
1700 FIELD 2,32 AS resp$(1),32 'S resp$(2),32 AS resp$(3),32 AS resp$(4)
1710 IF FIND$("b:krivial3")="" THEN 1760
1720 OPEN "I",3,"b:krivial3"
1730 INPUT #3,numfich%
1740 CLOSE 3
1750 IF numfich%>3 THEN 1820
1760 PRINT hon;"Imposible jugar, no existen datos suficientes. Retorno al menu
     principal";hoff
1770 s1=10:s2=34
1780 a$(1)="Conforme"
1790 num%=1
1800 GOSUB 3030
1810 CLOSE:GOTO 620
1820 PRINT numfich%;" fichas disponibles."
1830 IF numfich%<105 THEN vec%=numfich%-3 ELSE vec%=100
```

```
1840 PRINT:PRINT"El intervalo entre repeticiones es de";vec%;"preguntas distintas."
1850 DIM vector(vec%)
1860 PRINT:PRINT TAB(37);CHR$(252)+"<";hon;"RETURN";hoff;">";CHR$(253):a$=INPUT$(1)
1870 PRINT home;hcab
1880 cleft%=3
1890 PRINT hpant3
1900 PRINT hpant5
1910 PRINT hpant6
1920 FOR a=1 TO 5
1930 PRINT CHR$(133);TAB(6);CHR$(133);TAB(36);CHR$(133);TAB(41);CHR$(133)
1940 PRINT CHR$(135);hpant1;CHR$(141);TAB(36);CHR$(135);hpant1;CHR$(141)
1950 NEXT
1960 PRINT CHR$(133);TAB(6);CHR$(133);TAB(36);CHR$(133);TAB(41);CHR$(133)
1970 PRINT hpant7
1980 PRINT hpant5
1990 PRINT hpant8
2000 RESTORE 2050
2010 FOR a=1 TO 4
2020 READ x1,x2,t$
2030 PRINT FN at$(x1,x2,t$)
2040 NEXT
2050 DATA 5,1,"////",5,36,"////",19,1,"////",19,36,"////"
2060 plpos%=1
2070 quesos%=0
2080 PRINT FN at$(posn(plpos%,1),posn(plpos%,2),hon+"    "+hoff):REM Son cuatro
     espacios.
2090 a=0
2100 PRINT FN at$(18,50,STRING$(cleft%,"?")+"   ")
2110 REM
2120 a=a+1:PRINT FN at$(10,60,hdado(INT((RND*6)+1))):IF INKEY$="" THEN 2120
2130 RANDOMIZE a*100/SIN(RND)
2140 dd%=INT(RND*6)+1
2150 PRINT FN at$(10,60,hdado(dd%))
2160 op.pos1%=plpos%-dd%:IF op.pos1%<1 THEN op.pos1%=28+op.pos1%
2170 op.pos2%=plpos%+dd%:IF op.pos2%>28 THEN op.pos2%=op.pos2%-28
2180 PRINT FN at$(posn(op.pos1%,1),posn(op.pos1%,2),"----")
2190 PRINT FN at$(posn(op.pos2%,1),posn(op.pos2%,2),"++++")
2200 a$=INPUT$(1)
2210 IF a$<>CHR$(22) AND a$<>CHR$(28) THEN 2200
2220 PRINT FN at$(posn(op.pos1%,1),posn(op.pos1%,2),"    ")
2230 PRINT FN at$(posn(op.pos2%,1),posn(op.pos2%,2),"    "):REM Todo cuatro espacios
2240 PRINT FN at$(posn(plpos%,1),posn(plpos%,2),"    ")
2250 IF a$=CHR$(28) THEN plpos%=op.pos1% ELSE plpos%=op.pos2%
2260 PRINT FN at$(posn(plpos%,1),posn(plpos%,2),hon+"    "+hoff)
2270 FOR a=1 TO 28 STEP 7
2280 PRINT FN at$(posn(a,1),posn(a,2),"");
2290 IF plpos%=a THEN 2310
2300 IF pillao((a-1)/7+1)=0 THEN PRINT "////" ELSE PRINT "XXXX"
2310 NEXT
2320 PRINT FN at$(10,60,hdborra)
2330 play%=INT(RND*numfich%)+1
2340 a=1
2350 IF vector(a)=play% THEN GOTO 2330 ELSE a=a+1
2360 IF vec.ptr%>=a THEN 2350
2370 vec.ptr%=vec.ptr%+1
2380 vector(vec.ptr%)=play%
2390 IF vec.ptr%=vec% THEN FOR a=2 TO vec%:vector(a-1)=vector(a):NEXT:vec.ptr%=
     vec.ptr%-1
2400 PRINT FN at$(22,5,"Tema: "+tema$((play% MOD 4)+1))
2410 GET 1,play%
2420 GET 2,play%
2430 FOR a=1 TO 4
2440 resp2$(a)=resp$(a)
2450 NEXT
2460 cort%=0:a=128
2470 a$=MID$(preg$,a,1)
2480 WHILE a$=" ":REM Un espacio
```

```
2490 cort%=cort%+1:a=a-1
2500 a$=MID$(preg$,a,1)
2510 WEND
2520 hpreg=LEFT$(preg$,128-cort%)
2530 IF LEN(hpreg)<64 THEN hpreg1=hpreg:hpreg2="":GOTO 2600
2540 FOR a=50 TO 66
2550 IF MID$(hpreg,a,1)=" " THEN cort2%=a
2560 NEXT
2570 IF cort2%=0 THEN cort2%=LEN(hpreg)
2580 hpreg1=LEFT$(hpreg,cort2%)
2590 hpreg2=MID$(hpreg,cort2%,LEN(hpreg)-cort2%+1)
2600 hok=resp$(1)
2610 FOR a=1 TO 4
2620 a1=INT(RND*4)+1
2630 a2=INT(RND*4)+1
2640 WHILE a1=a2
2650 a2=INT(RND*4)+1
2660 WEND
2670 SWAP resp2$(a1),resp2$(a2)
2680 NEXT
2690 PRINT FN at$(23,5,"?"+hpreg1);:IF hpreg2="" THEN PRINT "?" ELSE PRINT FN
     at$(24,5,hpreg2+"?")
2700 time=150
2710 a=1
2720 time=time-1:IF time=0 THEN PRINT FN at$(25,1,"!Demasiado tarde! La respuesta
     correcta es "+hok):GOTO 2930
2730 PRINT FN at$(25,50,hon+STR$(INT(time/10))+" "+hoff)
2740 PRINT FN at$(25,1,resp2$(a))
2750 a$=INKEY$
2760 IF a$=CHR$(28) THEN IF hok<>resp2$(a) THEN a=a+1:GOTO 2720 ELSE GOTO 2920
2770 IF a$=CHR$(22) THEN IF hok<>resp2$(a) THEN 2920 ELSE 2790
2780 GOTO 2720
2790 PRINT FN at$(25,1,"!Correcto, continua asi!"+STRING$(10,32))
2800 FOR a=1 TO 1000:NEXT
2810 FOR a=1 TO 4
2820 IF (plpos%-1)/7+1=a AND pillao(a)=0 THEN pillao(a)=1:quesos%=quesos%+1
2830 NEXT
2840 IF quesos%=4 THEN vuelta%=vuelta%+1:PRINT home;hcab;"Muy bien! Vamos por la
     vuelta";vuelta%:ERASE pillao:a$=INPUT$(1):GOTO 1870
2850 puntos%=puntos%+time
2860 PRINT FN at$(16,50,"Puntos:"+STR$(puntos%))
2870 PRINT FN at$(23,5,STRING$(130,32))
2880 PRINT FN at$(25,50,"    ")
2890 PRINT FN at$(22,5,STRING$(30,32))
2900 PRINT FN at$(25,1,STRING$(42,32))
2910 GOTO 2090
2920 PRINT FN at$(25,1,"No es cierto, la respuesta correcta es "+hok)
2930 cleft%=cleft%-1
2940 FOR a=1 TO 2000:NEXT
2950 PRINT FN at$(25,1,STRING$(40+LEN(hok),32))
2960 PRINT FN at$(23,5,STRING$(130,32))
2970 PRINT FN at$(22,5,STRING$(30,32))
2980 IF cleft%>0 THEN 2090
2990 PRINT home;hcab;TAB(39);"F I N"
3000 PRINT:PRINT TAB(34);"Puntos: ";puntos%
3010 a$=INPUT$(1):PRINT hcursor.on
3020 ERASE vector,pillao:puntos%=0:CLOSE:GOTO 620
3030 PRINT hcursor.off
3040 FOR a=1 TO num%
3050 PRINT FN at$(s1+a,s2+1,a$(a))
3060 NEXT
3070 ptr=1
3080 PRINT FN at$(s1+ptr,s2,CHR$(252)+hon+a$(ptr)+hoff+CHR$(253))
3090 a$=INPUT$(1)
3100 IF a$=CHR$(30) THEN IF ptr<num% THEN PRINT FN at$(s1+ptr,s2," "+a$(ptr)+
     " "):ptr=ptr+1
```

```
3110 IF a$=CHR$(31) THEN IF ptr>1 THEN PRINT FN at$(s1+ptr,s2," "+a$(ptr)+
     " "):ptr=ptr-1
3120 IF a$=CHR$(13) THEN PRINT hcursor.on:RETURN
3130 GOTO 3080
3140 PRINT hinput;" ";CHR$(252);
3150 LINE INPUT "",a$
3160 IF LEN(a$)>max.out% THEN PRINT:PRINT hon;"La ";houtput;" supera el maximo de
     caracteres. Reescriba, por favor.";hoff;:a$=INPUT$(1):PRINT hdel.line;
     hline.back;hcr:GOTO 3140
3170 IF a$="" THEN PRINT hline.back;hcr;:GOTO 3140
3180 RETURN
3190 PRINT home
3200 END
```

EL ACCESO POR CLAVES

etsam es el sistema incluido en BASIC que permite manejar ficheros de acceso por claves. Todo el sistema Jetsam se compone de funciones. Esto significa que no manejaremos órdenes, sino funciones que generan un valor y provoca una acción a la vez. Normalmente, una expresión completa de Jetsam tiene la forma:

variable=función(parámetro)

La variable adquiere un valor, que indica si la operación se ha llevado a cabo con éxito o no. El valor que significa «éxito» es el 0. Trataremos de que sea siempre éste el resultado tras exponer el capítulo.

Para empezar, es necesario introducir nuevos conceptos que servirán para comprender y manejar este tipo de ficheros, llamados también indexados (puesto que tienen un índice).

Su funcionamiento es similar al de los aleatorios, pero podemos acceder a ellos no sólo por su número de orden de registro, sino también por una CLAVE. La gestión de claves y su correspondencia con los registros, se logra gracias a un segundo fichero paralelo (fichero

de índices), cuyas características y estructura no han de preocuparnos para nada, puesto que es BASIC quien se encarga de él.

Podemos asignar a cada registro un total de ocho claves. Para distinguir unas de otras, se graba cada una en un RANGO distinto, que se numeran del 0 al 7.

El ejemplo que nos ayuda a comprender todo esto es de nuevo el fichero normal. Imaginemos que las fichas tienen una solapa que sobresale por encima del cuerpo de la ficha. En esa solapa es donde escribiríamos algún dato para referencia, por ejemplo el apellido de la persona de la que se habla en tal ficha. Esa solapa es una clave: no hace falta que conozcamos el número que tiene; nos basta con dar un vistazo para localizarla.

Mallard BASIC admite ocho «solapas» de este tipo, cada una en su rango. Si indicamos al ordenador que busque la ficha cuya clave en la segunda solapa es «García», le decimos en realidad que busque el registro cuyo segundo rango contenga la clave «García».

Vamos ahora con la práctica. Un ejemplo de fichero cuyo acceso más adecuado es el indexado es una agenda de teléfonos, en la cual se pueda buscar a una persona por su nombre o por su dirección.

Admitiremos que las direcciones se puedan repetir, pero no los nombres. Esta prohibición se traduce en una orden de Jetsam.

La definición del registro será la siguiente:

— Apellido y nombre	50	caracteres
— Dirección	30	»
— Prefijo regional	3	»
— Número	7	»
	90	caracteres total

Tanto el prefijo como el número serán codificados como valores enteros, para lo cual utilizaremos la correspondiente función de conversión que ya estudiamos.

Lo primero, ya lo sabemos, es asignar un espacio de memoria a buffer de disco. Recordemos que esto se consigue con la orden **BUFFERS.** Es evidente que la memoria destinada a este dispositivo ya no será accesible por BASIC, de manera que debemos ajustarla para que uno y otro tengan suficiente. Mientras no se esté realmente apurado con la memoria (lo cual significa tener un programa de grandes dimensiones) lo mejor es asignar 6 bloques al buffer. La instrucción sería:

buffers 6

CREACIÓN DE UN FICHERO INDEXADO

Una vez hecho esto, hay que crear el fichero. En este caso no basta con abrirlo, porque como dijimos antes, existe un fichero de índices manejado por BASIC exclusivamente. Este debe ser creado la primera vez, y a partir de entonces ya se abrirá normalmente con **OPEN.**

La instrucción para crear un fichero indexado es:

create número de fichero,nombre 1,nombre2,bloqueo,registro

Vamos a explicar cada uno de los parámetros. **número de fichero** es, como siempre, el número que asignamos al fichero para manejarlo posteriormente sin tener que utilizar su nombre. Ya lo hemos empleado en secuenciales y aleatorios.

nombre 1 es el nombre del fichero que contendrá los datos en el disco. Es el que verdaderamente usamos, donde se encuentran los registros.

nombre 2 es el nombre del ficl ero de índices, de uso exclusivo para BASIC. Este fichero no debe retocarse nunca, puesto que cualquier desajuste puede romper la correspondencia entre cada registro y su clave o claves correspondientes.

bloqueo es un número que indica el tipo de bloqueo que ha de asignarse al fichero. Este detalle no nos interesa en absoluto, puesto que los bloqueos sirven para imponer preferencias en los sistemas multiusuario. Como vimos en la función **VERSION,** nuestro BASIC es uniusuario. Sólo debemos saber que el bloqueo en la orden **CREATE** es un 2.

registro es la longitud de registro en caracteres. La única precaución es que se deben sumar dos a la cantidad resultante de nuestra definición, porque BASIC los utiliza internamente.

De esta manera, nuestra orden de creación de fichero para la agenda sería parecida a esta:

create 1,"agenda.dat","agenda.ind",2,92

El proceso de definición de registro es igual que el de ficheros aleatorios, con **FIELD.** El máximo de caracteres en el registro es 126, puesto que BASIC se lleva 2 para gestión interna. En una sola instrucción **FIELD** debe estar toda la definición de registro.

En nuestro ejemplo, la orden **FIELD** tendría este aspecto:

field 1,50 as nombre$,30 as direc$,3 as pref$,7 as telef$

Se utilizan 90 de los 92 bytes (caracteres) dedicados a cada registro.

Para asignar valores a las variables de campo utilizaremos también **LSET** y **RSET.** Y aquí terminan los procesos comunes, porque al añadir el registro al fichero debemos también asignar una clave («solapa»). Un registro sin clave se pierde.

En nuestro ejemplo, hay dos claves para cada registro. Podríamos hacer una clave con 50 caracteres, para que fuera igual que el contenido del campo. Pero esto es un gasto excesivo de espacio en el disco, de manera que limitaremos la longitud de las claves:

— Clave del nombre 15 caracteres
— Clave de la dirección 10 caracteres

Limitar la longitud de las claves hace posible que se repitan dos sin ser iguales en su totalidad. Si por ejemplo, tenemos una clave limitada a 9 caracteres, «GARCIA LOPEZ» y «GARCIA LOZANO» serán la misma clave al recortarla, quedando «GARCIA LO». Sea cual sea la clave, no debe superar la longitud del campo al cual hace referencia. Si tuviéramos clave para el prefijo, debería ser de 3 caracteres. La longitud máxima de una clave es de 31 caracteres.

Es por esto que hemos fijado un límite mayor para el nombre, de mayor importancia y donde además vamos a prohibir repeticiones. No importa tanto, por el contrario, que en la búsqueda de direcciones se encuentren varias fichas.

Para limitar las repeticiones se utiliza **RANKSPEC.** La forma general es:

variable numérica=rankspec(número de fichero,rango,indicador)

rango es un número de 0 a 7 que indica en qué rango (en cuál de las solapas de la ficha) debe aplicarse la operación.

indicador puede ser 1 ó 0. En el primer caso, impide la repetición de claves. El valor 0 la permite. En nuestro fichero, el primer rango (rango 0) perteneciente a nombres no se debe repetir. La instrucción es:

res=rankspec(1,0,1)

res es una variable numérica que, como dijimos, indica el éxito o fracaso de la operación. En el apéndice 3 se detallan todos los errores posibles.

Otro detalle antes de añadir los registros en el fichero es el formato que van a tomar las claves. Si grabamos una clave en minúsculas y luego la intentamos buscar escribiéndola en mayúsculas, o simplemente, con una sola letra en mayúsculas, el ordenador no la encontrará, puesto que busca siempre variables idénticas.

La solución es, naturalmente, forzar el formato. En nuestro ejem-

plo, haremos que todas las cadenas contengan caracteres en mayúsculas con **UPPER$** (función tratada en el anterior número sobre BASIC).

Pasemos a añadir un registro. La forma general de la instrucción que efectúa esta labor es:

variable=addrec(número de fichero,bloqueo,rango,clave)

ADDREC viene de *add record,* añadir registro. El bloqueo para **ADDREC** es 0. **rango** es el número que indica a cuál de ellos (a qué solapa) se asignará la **clave,** expresión alfanumérica y último parámetro de la instrucción.

Vamos a asignar a nuestras variables de campo unos datos:

```
lset nombre$="García Veerdugo, Antonio"
lset direc$="Wyoming street, 41
lset pref$=mki$(999)
lset telef$=mks$(4579424)
```

Ahora asignamos a dos variables cualesquiera el contenido de las claves. Como la clave del rango 0 coincide con el nombre limitado a 15 caracteres, la asignación será:

clave0$=left$(nombre$,15)

como además hemos acordado que las claves sean siempre en mayúsculas, escribiremos:

clave0$=upper$(clave0$)

operaremos de manera similar con la clave del rango 1:

clave1$=upper$(left$(direc$,10))

Ya podemos utilizar **ADDREC.** La expresión sería:

res=addrec(1,0,0,clave0$)

con esto queda grabado el primer registro del fichero, con la clave de nombres en el rango 0. Hay que comprobar con **IF** el valor de **res.** Si es cero, todo ha ido bien.

Para añadir la segunda clave, no será necesario volver a escribir todos los datos. Existe una instrucción específica para añadir claves a un registro ya presente en el disco: **ADDKEY** (de *add key,* añadir clave).

Su forma general es **ADDKEY** es:

variable=addkey(número de fichero,bloqueo,rango,clave,número de registro

Se nos presenta un problema: el número de registro no lo conocemos. Podemos deducir que es 1, porque es el primer registro que

escribimos, pero eso no va a ser siempre así. Para saber el último número de registro que se ha mencionado, se utiliza otra función: **FETCHREC** (de *fetch record*, recoger registro). El modo de operar con él es:

variable=fetchrec(número de fichero)

La variable captará el número del registro al que se ha accedido por última vez, del fichero correspondiente. Como acabamos de grabar nuestra ficha, es precisamente la última en acceder. Obtengamos su número:

numreg=fetchrec(1)

Hecho esto, ya podemos añadir la segunda clave a nuestro registro, con **ADDKEY:**

res=addkey(1,0,1,clave1$,numereg)

El proceso de grabación de datos consiste en repetir los pasos que aquí hemos detallado.

Un detalle MUY importante es que BASIC utiliza el buffer de disco continuamente, y por tanto no siempre está grabando físicamente los datos. Ello significa que una modificación efectuada recientemente, aún no esté reflejado en el disco.

Para que todos los cambios se trasladen al disco disponemos de la función **CONSOLIDATE.** Tiene la forma:

variable=consolidate(número de fichero)

cuya función es descargar el buffer, de manera que todo quede en orden dentro de los ficheros de datos e índices. El caso es que la instrucción **CLOSE** sin parámetros, es decir, sin especificar número de fichero, no descarga el buffer adecuadamente, de manera que las claves y los registros se desfasan y no coinciden debidamente. Un fichero cerrado con **CLOSE** sin parámetros es inútil.

Por lo tanto, la función **CONSOLIDATE** debe ejecutarse después de importantes cambios en los datos. Análogamente, al terminar la sesión, se debe cerrar el fichero explícitamente, es decir, escribiendo su número de fichero correspondiente.

Terminamos aquí con las funciones Jetsam dedicadas a introducir fichas en un disco. Las demás funciones, y como era de esperar, las más numerosas, están destinadas a acceder y modificar la información almacenada.

LEER REGISTROS POR CLAVES

Una vez creado el fichero, ya no es necesaria la orden **CREATE.**

Lo correcto es emplear **OPEN,** el cual tiene otra letra para designar el modo de acceso: la «K», de *keyed* (indexado). De manera que la forma de abrir un fichero de acceso por claves es:

open número de fichero,nombre1,nombre2,bloqueo,variable

nombre1 es el nombre del fichero de datos, y **nombre2** el de índices. **bloqueo** es 2 (**CREATE** y **OPEN** tienen bloqueo 2, las demás instrucciones, 0). La **variable,** si se la indica, es del tipo numérico y su contenido es la longitud de registro especificada cuando se creó el fichero. En nuestro caso la longitud de registro era 92. La forma correcta de abrir nuestro fichero es, pues:

```
longreg=92
open "K",1,"agenda.dat","agenda.ind",2,longreg
```

A continuación, como ya sabemos, hay que destinar parte de la memoria al buffer de disco:

```
buffers 6
```

Y después definir las variables de campo del registro, tal y como lo hicimos al crear el fichero:

```
field 1,50 as nombre$,30 as direc$,3 as pref$,7 as telef$
```

Tras esto, ya estamos en condiciones de leer datos. Supongamos que nos dan un nombre cuyos datos deben obtenerse. Lo primero es adaptar esos datos al mismo formato con el cual obtuvimos las claves: 15 caracteres de longitud, y en mayúsculas.

```
input "Introduzca el nombre de búsqueda",a$
a$=upper$(left$(a$,15))
```

Una vez conseguido el formato, utilizaremos la instrucción de búsqueda del PRIMER registro con esa clave. Su forma general es:

```
variable=seekkey(número de fichero,bloqueo,rango,clave)
```

El bloqueo, como ya sabemos, es 0. El rango indica en qué «solapa» de las ocho disponibles se debe buscar la clave. La **clave** es una expresión alfanumérica que debe ajustarse al formato en el cual fueron grabadas. Esta expresión es la que se buscará entre las grabadas en el fichero.

En nuestro caso, si buscamos la ficha de una persona, primero hemos de pedirlo, y ajustarlo al formato de las claves:

```
input "Nombre ",a$
busclave$=upper$(left$(a$,15))
```

Ahora utilizaremos la función **SEEKKEY,** teniendo en cuenta que buscamos nombres, y por lo tanto están en el rango 0.

res=seekkey(1,0,0,busclave$)

A continuación debemos examinar el contenido de la variable **res.** Si es 0, entonces el registro se ha encontrado, y ahora es el último al que ha hecho referencia BASIC.

Podemos entonces asignar el contenido de dicho registro a las variables de campo. La orden a tal fin es **GET,** como en el caso de los ficheros aleatorios, y en nuestro ejemplo tiene la forma:

get 1

lo cual es bien sencillo, entre todas las nuevas funciones Jetsam que estamos viendo. Así, las variables ya contienen los datos del registro.

Como dijimos antes, es posible que la clave se repita en algunas ocasiones (y esto lo hemos admitido en el campo «domicilios»). En este caso, la forma correcta de operar no es continuar con **SEEKKEY,** puesto que la clave ya se ha encontrado, y lo que se pretende es localizar más claves iguales. Esta labor la realiza otra función, **SEEKNEXT** (de buscar siguiente). Su forma general es:

variable=seeknext(número de fichero,bloqueo,posición)

El bloqueo es 0. La posición es un parámetro opcional, y que nosotros no utilizaremos. Indica el número de orden dentro del conjunto de claves iguales que buscamos.

```
730 INPUT "Domicilio ",d$
740 buscdom$=UPPER$(LEFT$(d$,10))
750 res=SEEKKEY(1,0,1,buscdom$)
760 IF res<>0 THEN RETURN:REM No se ha encontrado ese domicilio
770 GET 1
780 (Aqui se muestra la ficha completa)
790 REM Ahora buscamos mas fichas con la misma direccion
800 res=SEEKNEXT(1,0)
810 IF res<>0 THEN RETURN:REM No hay mas fichas con ese domicilio
820 GET 1
830 GOTO 780
```

Vamos con nuestro ejemplo: buscamos un teléfono pero sólo conocemos la dirección del abonado. La búsqueda comenzaría tal como se indica en el listado.

Pero no es este el único proceso que realiza **SEEKNEXT.** En la tabla de errores de Jetsam podemos ver que esta función informa también cuando se encuentra una clave distinta a la anterior, dentro del mismo rango, o bien que se ha encontrado una clave en el siguiente rango que existe.

Resumiendo, **SEEKNEXT** permite leer secuencialmente, es decir, una tras otra, todas las claves de un fichero, pasando de un

rango al otro cuando se acaban las claves del anterior. En nuestro caso, con **SEEKNEXT,** podemos rastrear todo el fichero, empezando por los nombres y terminando por la última de las direcciones.

Una función similar para la búsqueda «en línea» es **SEEKPREV** (buscar anterior) y lo que hace es buscar claves pero hacia atrás, es decir, retrocediendo en el fichero y cambiando a un rango inferior si llega al principio del que sigue.

La forma general de escribir **SEEKPREV** es igual a la de **SEEKNEXT:**

variable=seekprev(número de fichero,bloqueo, posición)

El bloqueo también es 0. Los valores adoptados por la **variable** son los mismos que para **SEEKNEXT,** pero debe tenerse en cuenta que se está retrocediendo en el fichero.

Si nos interesa buscar registros con clave asociada siempre distinta, no tenemos que crear nuestra propia rutina; **SEEKSET** permite este tipo de búsqueda. Se escribe así:

variable=seekset(número de fichero,bloqueo,posición)

donde el **bloqueo** es 0 y la **posición** es un parámetro opcional que indica el número de orden de la clave a partir del cual se ha de buscar. El funcionamiento es similar al de **SEEKNEXT,** pero siempre busca claves distintas.

Otra función de gran utilidad es **SEEKRANK,** y permite conocer la primera clave de un determinado rango. La forma general es:

variable=seekrank(número de fichero,bloqueo,rango)

el bloqueo, como siempre, es 0 y el rango indicado es aquel en el cual deseamos obtener la primera clave. Si el contenido de la **variable** es 0, el registro que buscamos es el último mencionado y, por tanto, podemos recogerlo con **GET.**

MÁS FUNCIONES JETSAM

xisten además otras tres funciones Jetsam. Ya hemos hablado de una de ellas: **FETCHREC,** para captar el número del último registro mencionado.

La segunda es **FETCHRANK,** y nos permite conocer el rango al que pertenece la última clave mencionada (con éxito):

variable=fetchrank(número de fichero)

la **variable** contendrá el número de rango de esa clave.

La última función es **FETCHKEY$,** la cual genera una cadena que contiene la última clave mencionada.

variable$=fetchkey$(número de fichero)

La variable debe ser alfanumérica. No olvidemos que si se trataba de un número, debe ser convertido con la correspondiente función (**CVIK,** etc.) a un número manejable normalmente.

BORRAR REGISTROS DE ACCESO POR CLAVES

El método de borrado es un poco especial. Un registro queda borrado cuando quedan borradas todas las claves asociadas a él. Es de-

cir, una ficha queda borrada (eliminada) cuando no tiene solapas para buscarla.

Físicamente, el registro no es borrado, sino que permanece inaccesible hasta que un nuevo registro (añadido con **ADDREC**) ocupa su lugar en el disco.

La función de Jetsam que borra claves es **DELKEY** (de *delete key,* borrar clave):

variable=delkey(número de fichero,bloqueo,rango,clave,número de registro

La única precaución al utilizar **DELKEY** es adaptar la clave al formato en el cual fueror grabadas, y obtener el número de registro con **FETCHREC** antes de la operación.

La **variable** puede tomar cuatro valores calificados como éxito, y dependen del registro que se adopte como «último mencionado», y al cual afectarían las siguientes órdenes **GET,** etc.

DELKEY puede utilizarse en conjunción con **ADDKEY** para cambiar el valor de una clave. Primero se añade la versión nueva con **ADDKEY** y después se suprime la antigua con **DELKEY.** Es importante utilizar primero **ADDKEY,** porque en el caso de ser la clave afectada la única asociada al registro, se borraría éste.

Observaciones Sobre La Utilización De Jetsam

El hecho de que las instrucciones de manejo de ficheros por claves sean funciones permite gran variedad de expresiones BASIC útiles. En este capítulo hemos expuesto las del tipo:

variable=función(parámetros)

pero en realidad podemos utilizar expresiones igualmente válidas, sin tener que recurrir a una variable de paso:

if addrec(2,0,4,klave$) then goto 590

El problema de estas expresiones que no utilizan variables para almacenar resultados, sino que operan directamente con ellos, es precisamente que los valores no son reutilizables. En el ejemplo anterior, no podríamos volver a operar con el mensaje emitido por Jetsam, y que en otro caso sería asignado a una variable y, por tanto aún conocido.

Para almacenar en un fichero correctamente una clave numérica,

deben utilizarse las funciones de conversión adecuadas, que son sólo dos: **MKIK$** y **MKUK.** La forma apropiada de emplear estas funciones es:

— Si la clave es un número entre 0 y 255, generaremos una cadena con **CHR$.** Ocupa un solo byte.

— Si la clave está en el margen – 32768 a 32767, se utiliza **MKIK$(n).**

— Si la clave está en el margen 0 a 65535, se utiliza **MKUK$(n).**

Las dos últimas funciones generan cadenas de dos bytes, y las funciones sólo admiten valores enteros. En el caso de introducir un valor con decimales, será redondeado antes de efectuar la conversión. Los valores generados deben ser asignados a una variable numérica (la variable de campo correspondiente).

Existen ocho rangos distintos para guardar claves, pero esto no significa que el número de campos de la ficha esté limitado también a esta cifra. El número de campos, como dijimos, sólo está limitado por la longitud máxima de registro.

CÓDIGO MÁQUINA

No pretendemos en este capítulo exponer técnicas sobre el código máquina. Trataremos únicamente las funciones e instrucciones dedicadas a relacionar el BASIC con el código máquina.

Las dos primeras funciones relacionadas con el tratamiento directo con el C/M ya las conocemos: **PEEK(n)** devuelve el contenido de la dirección de memoria **n** (entre 0 y 65535). **POKE n,s** almacena en la dirección **n** (también entre 0 y 65535) el byte **s** (número comprendido entre 0 y 255).

Las instrucciones específicas de Mallard BASIC permiten invocar rutinas enviando parámetros, y conocer la localización de las variables del usuario dentro del bloque de memoria.

La orden **CALL** llama a una subrutina en C/M. La forma de escribirla es:

call variable (lista de argumentos)

Esta instrucción puede incluir una lista de parámetros opcionales. La dirección de memoria sólo puede estar expresada por medio de una variable, es decir, el contenido de la variable especificada es la dirección de memoria de salto.

Los parámetros han de ser igualmente variables, normales o términos de una dimensionada. Su contenido indica la dirección de memoria en la cual se encuentra el valor del parámetro.

Otro medio de invocar subrutinas es mediante la orden:

DEF USRnúmero=dirección

cuyo efecto es la definición de una función cuyo valor genera la rutina. **número** es una cifra (entre 0 y 9). La **dirección** indica el comienzo del programa.

La función se invoca cuando se la menciona como argumento de una expresión, de la siguiente forma:

USRnúmero (argumento)

sólo interviene un parámetro, y es obligatorio.

Por último, la función **VARPTR** permite conocer la dirección de una variable o de un buffer de fichero, para entregarla a una rutina o función definida. La forma general es:

variable=varptr(variable2)

o bien

variable=varptr(#número de fichero)

En la primera forma, la **variable2** es aquella cuya dirección queremos averiguar. El resultado es una dirección (de 0 a 65535), primer byte correspondiente a la variable.

En la segunda forma, se obtiene la dirección del buffer correspondiente al fichero, mencionado por su número. Si el fichero no está abierto, no existe buffer asignado, y por lo tanto no es correcto utilizar la función.

INTRODUCCIÓN AL ASIC CPC

Los objetivos planteados para el desarrollo del BASIC instalado en los CPC, difieren bastante de los correspondientes a Mallard BASIC para PCW. Se trata, en este caso, de ofrecer un intérprete fácil, pero a la vez potente y en definitiva capaz de abarcar las más variadas necesidades. El CPC se convierte así en un perfecto soporte para juegos, y en su versión 6128 alcanza ya la posibilidad de llevar a cabo tareas mucho más serias, como la contabilidad, etc...

Podemos afirmar que el BASIC de los CPC es uno de los mejores que se implementan en los ordenadores domésticos. Las capacidades de sonido y gráficos (en especial esta última) sobrepasan con creces la media normal. Tan peculiares son las características de gráficos y sonido que merecen un estudio aparte; los trataremos en próximos números de esta colección.

Estudiaremos, eso sí, las instrucciones básicas de cambio y selección de colores para modificar el aspecto de los escritos, y aquellas que faciliten de algún modo la disposición de texto en la pantalla.

Dado que existen modelos CPC con cinta y con disco, explicare-

mos todas las instrucciones relacionadas con estos soportes, lo cual incluye la gestión de ficheros secuenciales y la grabación de programas.

Dedicamos también un capítulo al teclado, puesto que nuestro ordenador posee unas características muy especiales en este periférico de entrada. En concreto, es especialmente útil la redefinición del teclado; posibilidad poco corriente en ordenadores de este tipo.

PANTALLA: COLOR Y TEXTO

Empezaremos por describir la estructura de la pantalla. Esta presenta tres anchuras distintas de los caracteres, controlables mediante la instrucción **MODE,** relacionada con los valores 0, 1 y 2.

En el modo 0, los caracteres son realmente anchos, y tan sólo caben 20 en cada línea. Para conseguir este aspecto en la pantalla, debemos teclear:

mode 0

Además de cambiar la anchura, se ha borrado la pantalla, igual que si hubiéramos tecleado **CLS.** Esto se debe a que el ordenador debe «cambiar de esquemas» internamente. No nos extenderemos en el sistema interno; baste decir que cuanto más grandes sean los caracteres, menos memoria se ocupa y, por tanto, más colores pueden aparecer a la vez en la pantalla. Así, en modo 0 es posible representar 16 colores.

Ahora tecleemos:

mode 1

Los caracteres adoptan una anchura un poco menor, más legible.

Ahora caben 40 caracteres en cada línea, y puede haber cuatro colores a la vez en la pantalla.

Con **mode 2** obtenemos la máxima anchura, adecuada para textos muy largos. Aquí el CPC sólo admite dos colores a la vez, pero conseguimos 80 caracteres en cada línea.

TINTEROS Y PLUMAS

Vamos a explicar ahora el manejo de los colores. Imaginemos 16 tinteros, numerados del 0 al 15. Podemos llenar cada uno de ellos con cualquiera de los 27 colores disponibles. Para llenar un tintero con un color, se utiliza la instrucción:

ink tintero,color

Análogamente, podemos obtener tinteros con *flash,* es decir, con dos colores que se alternan cíclicamente, para lo cual emplearemos:

ink tintero, color,color

La velocidad con que estos dos colores cambian no es fija y podemos variarla con la instrucción:

speed ink n1,n2

donde **n1** es un número entre 1 y 255. Si multiplicamos el número indicado por 0.02, nos dará los segundos de permanencia de cada color. **n1** señala cuánto tiempo debe aparecer el color 1, y **n2** fija el tiempo del segundo. Por tanto, para un *flash* uniforme se emplearán valores iguales.

Todavía no hemos dicho cómo cambiar realmente el color, puesto que lo explicado hasta ahora sólo permite variar las tintas de los tinteros. Para escribir, necesitaremos una pluma, que conseguiremos con la orden **PEN,** pudiendo indicar también el tintero en el cual queremos sumergir la pluma antes de escribir:

pen tintero,modo

El modo puede ser 1 (transparente) o 0 (opaco); o bien omitirse el modo:

pen tintero

Por otra parte, para cambiar el color del fondo también habrá que decirle de qué tintero ha de coger el tono:

paper tintero

Vamos ya con un ejemplo. Cambiemos a modo 0:

mode 0

Si queremos caracteres rojos sobre fondo verde, primero debemos saber los números a los cuales corresponden. El rojo brillante es el 6, y el verde el 9. Utilizaremos los tinteros 4 y 5. Para empezar, llenemos el 4 de rojo:

ink 4,6

... y el 5 de verde...

ink 5,9

Ahora asignemos el tintero 4 (rojo) a la pluma:

pen 4

... y el fondo al tintero 5 (verde)...

paper 5

Asimismo, podemos cambiar la tinta de un tintero que esté siendo utilizado en ese momento. El resultado es que todo lo que esté escrito con ese tintero en la pantalla, cambiará de color. Por ejemplo, para cambiar el rojo que queda ahora en la pantalla, podemos asignar al tintero 4 un tono distinto:

ink 4,26

con lo cual se convierte en blanco brillante.

También se puede, tal y como indicábamos antes, asignar un efecto *flash* al tintero. Así, por ejemplo, para que las letras salgan en blanco y negro a intervalos, teclearemos:

ink 4,26,0

Podemos variar la velocidad del parpadeo. Por ejemplo, con

speed ink 100,30

conseguiremos que el blanco aparezca mucho más tiempo que el negro.

ALREDEDOR DE LA PANTALLA

Cambiando colores de fondo y de escritura (la pluma), el contorno de la pantalla no cambia. Para manejar éste existe una instrucción determinada, que como cabía esperar, se llama **BORDER** (en inglés, marco, borde o contorno). La forma general de escribirlo en el ordenador es:

border color

donde *color* es el número correspondiente. Para conseguir un borde azul pastel, escribiremos:

border 14

También es posible obtener un flash en esta zona de la pantalla. Para lo cual actuaremos tal y como lo hacíamos en **INK:**

border color1,color2

Asimismo, este modo también resulta afectado por la instrucción **SPEED INK.**

Y para finalizar con este tema, un buen consejo: mientras se muestre texto en la pantalla, conviene que el color del borde tenga un tono apagado, de manera que resulte fácil y cómodo leer. Por el contrario, si se trata de un juego, lo mejor es un color original, o bien el mismo que el del «escenario» de la acción, para «aumentar» (aunque sólo sea una sensación óptica) las dimensiones del mismo.

Situar Y Localizar El Cursor

Cuando escribimos con **PRINT,** normalmente el texto se presenta a continuación del último mensaje mostrado. No obstante, podemos situar el comienzo de la escritura donde queramos: en cualquier punto de la pantalla.

El comando para ubicar el cursor es **LOCATE,** indicando a continuación las coordenadas absolutas de la posición deseada, contando en filas y en columnas (y empezando por 1), considerándose como origen de coordenadas (1,1) la esquina superior izquierda. Por ejemplo, para escribir «HOLA» en la fila 10, columna 5, ejecutaremos:

locate 5,10:print "HOLA"

Es muy importante que nada más ejecutar un **LOCATE** aparezca el correspondiente **PRINT,** ya que de otro modo el cursor bajaría a la línea siguiente, primera columna.

Como era de suponer, si estamos en modo 0, sólo hay 20 columnas, y por tanto podremos utilizar **LOCATE** dentro de este límite. De forma similar, debemos adaptarnos cuando la pantalla se encuentra en modo 1 (40 columnas) y en modo 2 (80 columnas).

Para averiguar la posición del cursor existen dos funciones: **VPOS** (coordenada vertical) y **POS** (coordenada horizontal).

La forma de utilizarlas es bien sencilla:

a=vpos(#0)
b=pos(#0)

Donde *a* y *b* contendrán las coordenadas del cursor.

Para terminar, existe otra función relacionada con el estado de la pantalla. Se trata de **COPYCHR$,** y permite conocer el carácter que se encuentra en una determinada posición de la pantalla. Para ello, debemos situar el cursor en el lugar deseado, y a continuación obtener lo que allí se encuentra con **COPYCHR$.**

Como ejemplo, tomemos un carácter nada más reinicializar el ordenador. Aparece el mensaje *Amstrad 128K microcomputer* (en el caso del 6128). Podemos teclear:

locate 2,2:a$=copychr$(#0):cls

... y a continuación...

print a$

lo cual mostrará el carácter situado en la posición (2,2) en el mensaje de inicialización. En este caso se trata de la *A* de *Amstrad.* Tengamos en cuenta que esta función no se halla implementada en la ROM de los 464/472.

EL TECLADO

El teclado de la serie CPC permite modificar lo escrito de una manera muy sencilla, mediante el manejo simultáneo de dos cursores.

Gracias a este sistema, podemos corregir las líneas de un programa sin necesidad de utilizar la orden **EDIT.** El método es bien sencillo: el segundo cursor (denominado «cursor de copia») se sitúa allí donde deseemos copiar texto, y a continuación se pulsa la tecla **COPIA.**

Vamos directamente a comprobarlo con un ejemplo. Acabamos de escribir una línea del programa, la 140, y contiene un error:

```
. . .
. . .
140 prnit "Ultima fase del cálculo"
```

Para desplazar el segundo cursor (el de copia), se utilizan las mismas teclas que para el principal, pero con mayúsculas. Normalmente, el segundo cursor está escondido tras el primero.

Pulsemos ahora la mayúscula de la flecha hacia arriba. Aparecerá por encima un segundo cursor, que estará ahora sobre él.

Ahora, pulsaremos **COPIA,** con lo cual el cursor de copia remite

al principal aquello que lee, trasladándose. Cuando lleguemos a *prnit,* dejaremos el segundo cursor sobre la *r,* y escribiremos **in.**

El aspecto de la pantalla ahora es el siguiente:

```
. . .
. . .
140 prnit "Ultima fase del cálculo"
140 prin_
```

Sólo queda ahora llevar el segundo cursor por encima del error hasta la *t,* y de nuevo pulsar **COPIA** hasta el final. Por último, **RETURN,** como es normal, y la línea estará corregida.

Realmente, explicar el proceso es mucho más lento que llevarlo a cabo. Con un mínimo de experiencia comprobaremos la facilidad y rapidez de este método de edición.

DETECTAMOS ENTRADAS

Ya conocemos el funcionamiento de **INKEY$,** "especialista" en detectar las pulsaciones del teclado sin detener por ello la ejecución

Cuadro de entradas de la función INKEY. Página 3/34 del manual de CPC 6128.

Valor	SHIFT	CONTROL	Tecla
– 1			No pulsada
0	No pulsada	No pulsada	Pulsada
32	Pulsada	No pulsada	Pulsada
128	No pulsada	Pulsada	Pulsada
160	Pulsada	Pulsada	Pulsada

Cuadro de bits activados en la función JOY. Página 3/37 del manual de CPC 6128.

Bit	Desplazamiento	Valor
0	Arriba	1
1	Abajo	2
2	Izquierda	4
3	Derecha	8
4	Disparo 2	16
5	Disparo 1	32

del programa. El CPC ofrece, además, otras dos funciones de recogida de datos inmediata, por teclado y joystick.

INKEY permite determinar si una tecla en concreto ha sido pulsada. Para ello, es necesario indicar entre paréntesis el número de la tecla. Por ejemplo, si queremos comprobar el estado de la tecla *Q* (código 67), escribiremos:

```
a=inkey(67)
```

El valor de a depende de si la tecla está o no pulsada, y en el primer caso, si intervienen también las teclas **MAYS** y **CONTROL.**

La función dedicada al joystick es **JOY,** y detecta la situación de cualquiera de los dos joysticks acoplables a la vez en el CPC. Corresponden a los números 0 y 1. Por ejemplo, para comprobar el estado del joystick 0, podemos ejecutar lo siguiente:

```
mov=joy(0)
```

El contenido de *mov* depende de cuántos pulsadores estén cerrados, recogiéndose la información bit a bit; es decir, el valor resultado es aquél que se obtiene tras asignar a los bits 0 a 5 un determinado estado del joystick.

La Instrucción «Clear Input»

Seguramente, ya habremos tenido oportunidad de comprobar que, aunque el ordenador se encuentre realizando una tarea, recuerda las teclas pulsadas durante el transcurso de ésta, y emite en la pantalla los caracteres correspondientes cuando la finaliza.

Si durante la ejecución de dicha tarea existe un **INKEY$** o **INKEY,** recogerá la primera de las pulsaciones memorizadas, y quedarán pendientes las siguientes.

Es posible que en alguna ocasión debamos recoger un dato con **INKEY** o **INKEY$** pero supongamos que hay pulsaciones pendientes que no interesan. La orden que borra de la memoria todas las pulsaciones pendientes es **CLEAR INPUT** (no implementada en la ROM 464/472).

Teclas Rápidas y Teclas Lentas

El fenómeno de la autorrepetición de las teclas queda muchas ve-

ces tan asumido que no le damos importancia. Pero se trata, sin lugar a dudas, de un complemento más para facilitar el manejo del teclado, principal fuente de datos para el ordenador.

Pulsemos una tecla. Inmediatamente se imprime en la pantalla el carácter correspondiente. Si mantenemos la presión, se imprime continuamente, hasta que soltemos la tecla.

En esta sucesión podemos distinguir dos tiempos: el existente entre la primera impresión y las siguientes, y el que separa los de los repetidos automáticamente.

El primer tiempo es bastante más largo. El segundo es lo suficientemente largo como para poder detener la repetición con una precisión de un carácter.

No obstante, para controlar estas dos velocidades se utiliza **SPEED KEY.** La forma general de escribirlo es:

speed key t1,t2

donde *t1* es la demora inicial y*t2* la transcurrida entre repeticiones. Probemos a variar estos tiempos, que inicialmente son 30 y 2, respectivamente.

Los valores que hacen inmanejable el teclado son los próximos al 1, de manera que si escribimos

speed key 1,1

el CPC se convertirá en algo inútil (recomendamos no hacer esto si hay algún programa importante en memoria, puesto que esta modificación significa casi con toda seguridad renunciar a él).

DEFINIR EL TECLADO

Dentro de la definición de teclas, podemos distinguir dos acciones diferentes: la asignación de un código a una tecla determinada, y la definición de códigos de expansión.

La instrucción que asigna códigos a una tecla es:

KEY DEF nt,rp,n1,n2,n3

nt es el número de tecla al cual se refiere la definición. **rp** indica si el carácter debe repetirse en una pulsación continuada o no. Colocaremos un 1 o un 0, respectivamente.

Donde **n1** es el código que debe generarse al pulsar la tecla sola. **n2** es el carácter que debe generarse si se pulsa junto con **MAYS,** y **n3** con **CONTROL.** Pueden omitirse **n2** y **n3,** o bien todas las **n** para definir tan solo la repetición.

Por ejemplo, para hacer que la tecla **TAB** genere el carácter «A» (código 65) sin autorrepetición, escribiremos:

key def 68,0,65

Para que **TAB** genere «A», y «B» con mayúsculas, y autorrepetición, se escribe:

key def 68,1,65,66

Por otra parte, existen unos determinados códigos, denominados de expansión, que pueden almacenar más de un carácter (al contrario que los normales). Están comprendidos entre los números 128 y 159.

Para definir uno de estos caracteres se utiliza la instrucción:

key cd,ce

Cuadro de asignaciones implícitas de los códigos expansibles. Página 7/72 del manual de CPC 6128

donde **ce** es el número del código de expansión y **ce** es la cadena de expansión que se le asigna. Por ejemplo, podemos teclear:

key 159,"Estamos probando la instrucción KEY"

Ahora asignaremos a la tecla = el código expansible 159:

key def 25,1,159

Comprobemos, pulsando la tecla afectada, que todo ha funcionado correctamente.

VARIABLES Y EXPRESIONES

amos a tratar ahora las instrucciones específicas del CPC dedicadas al manejo de variables, y las funciones dedicadas a manejar números de acuerdo con las posibilidades de esta máquina.

Empezaremos por los tipos de variables. El CPC trabaja con dos precisiones: entera y real. La entera no admite decimales, y la real trabaja con ellos hasta una precisión máxima de unas siete cifras.

Podemos indicar a BASIC que trabajamos con una variable entera utilizando el sufijo **%**. Implícitamente, las variables son de tipo «real», aunque también tienen sufijo: el signo **Pt.**, en el teclado castellano.

Para variar la situación implícita de las variables, se utilizan las instrucciones

DEFINT lista

para definir como enteras y

DEFREAL lista

para definir como reales.

Donde **lista** es una o varias variables, especificadas del siguiente modo:

— Si se trata de una sola, se escribe tal cual.
— Si es un conjunto determinado, se especifican entre comas.
— Si es un fragmento determinado del alfabeto, se pondrán las variables primera y última separadas por un guión (signo menos).

El resultado de estas definiciones es que podremos utilizar las variables, sin especificar sufijo, y BASIC entenderá que se trata de uno u otro tipo, de acuerdo con las definiciones efectuadas. Recordamos que el modo implícito es tipo real.

Para ahorrarnos el sufijo de las variables alfanuméricas **($)**, podemos utilizar **DEFSTR,** con la sintaxis de las anteriores. Por ejemplo, la secuencia:

```
defint a-h
defreal f
defstr p,t,v
```

hace que las variables a, b, c, d, e, g y h sean enteras, f real; p, t y v alfanuméricas.

De este modo, serán correctas las siguientes asignaciones:

```
a=4%
f=4.29873
p="hola"
```

CONVERSIÓN DE TIPOS

Cuando interese asignar el contenido de una variable entera a una real, o viceversa, podremos hacerlo gracias a dos funciones de conversión.

CINT(n) convierte la variable o expresión numérica **n** en un valor entero comprendido entre − 32767 y 32768.

CREAL efectúa la operación contraria, es decir, toma una variable o expresión numérica de cualquier tipo y la convierte a precisión real.

GRADOS

Como el CPC tiene características adecuadas para la educación, es todo un acierto ofrecer la posibilidad de operar en grados sexagesimales y en radianes (este último sistema de medida puede resultar complicado hasta ciertas edades).

La instrucción que ordena el cálculo en grados es **DEG.** Una vez

ejecutada, todas las funciones trigonométricas asumirán el argumento indicado en grados. Para invertir este modo, se utiliza **RAD.**

El ordenador trabaja implícitamente en radianes. Además, y aun en el caso de que se le haya indicado la operación en grados, volverá a la situación por defecto cuando se ejecute alguna de las siguientes sentencias: **NEW, CLEAR, LOAD** y **RUN.**

Operar Con Otras Bases

BASIC ofrece también dos funciones para la operación con números en base 2 (binario) y en base 16 (hexadecimal). Además, dispone de dos prefijos para manejar dichas notaciones directamente.

Para poder expresar números binarios directamente, debemos incluir el prefijo **&X.** Por ejemplo, el 129 debe escribirse como **&×10000001.** Los números en hexadecimal necesitan el prefijo **&,** de manera que **&ff** es 255 en decimal.

La función **BIN$** genera una cadena con el argumento expresado en binario. La forma de utilizarla es:

bin$(num,nb)

donde **num** es la cifra cuya expresión deseamos en binario, y **nb** es el número de bits que debe utilizarse para representarla. Este último argumento puede omitirse. Si se indica, pero es menor que la cantidad necesaria de bits, se ignora. Si es mayor, se añaden ceros a la izquierda hasta completar la cantidad exigida.

Con **HEX$** se genera también una cadena, pero conteniendo la forma hexadecimal del argumento. Tiene la misma forma que la función anterior:

hex$(num,nd)

donde **nc** es el número de dígitos del resultado, entre 0 y 16. Tal y como ocurre con **BIN$, nd** puede omitirse; si es demasiado pequeño se ignora, y si es más grande de lo necesario, se añaden ceros.

DISCO Y CINTA

ara poder guardar nuestros programas, disponemos de diversos soportes. En el caso de Amstrad, cintas (464 y 472) y discos (6128). El primer medio es bastante más lento, pero sin duda más económico. Ambos medios permiten la creación de un tipo de ficheros llamados secuenciales, los cuales serán tratados también en este capítulo.

CONSERVAR PROGRAMAS

La instrucción de grabación de programas es **SAVE.** Las formas generales de utilizarlo son:

save"nombre",tipo

y

save"nombre",B,din,long,com

La primera forma graba programas. Los tipos pueden ser **A** y **P.** Si no se especifica ninguno, el programa se graba en forma codificada (irreconocible). Si se añade la **A,** se graba en formato ASCII, es de-

cir, como un fichero de texto. Por último, si añadimos la **P** se graba de manera que no será posible listarlo de ninguna forma, para que quede protegido.

La segunda forma permite la grabación de fragmentos de la memoria del CPC. Seguido del nombre y la letra **B,** se indican **din** (dirección inicial del bloque), **long** (su longitud) y **com** (dirección de comienzo de ejecución). Este último parámetro es opcional, e indica la dirección de memoria a la cual debe saltar la ejecución en caso de llamada directa.

Para cargar un programa en memoria sin ejecutarlo, se utiliza **LOAD.** Esta instrucción no necesita los parámetros de **SAVE,** puesto que reconoce el formato de los programas grabados, o los bloques de memoria, en su caso.

La forma general de escribirla es

load"nombre"

y como parámetro opcional puede añadirse, tras una coma, la dirección de memoria a partir de la cual ha de cargarse el bloque de memoria (si de eso se trata).

Si el programa cargado estaba en forma protegida, entonces será borrado inmediatamente. La única manera de cargarlos es ejecutándolos directamente, con la orden **RUN.**

Con **RUN** podemos cargar y ejecutar inmediatamente un programa grabado. La forma de escribirlo es:

run "nombre"

con lo cual, el ordenador busca en el disco o cinta el programa especificado, lo carga y ejecuta.

Los sistemas con cinta permiten la instrucción

RUN"

(se puede obtener pulsando **CONTROL** + **INTRO**) y busca, carga y ejecuta el primer programa que se encuentre en la cinta.

MEZCLAR Y ENCADENAR

Podemos mezclar un programa existente en la memoria con otro grabado. El criterio de «mezcla» es el siguiente: todas las líneas del programa grabado pasan al actual. Las del programa en memoria cuyo número coincida con alguno del grabado, son sustituidas. Se trata, en definitiva, de una sustitución.

La instrucción de mezcla es

merge "nombre"

efectiva con cualquier programa que no esté protegido.

Si queremos mezclar un programa con otro, pero sin detener la ejecución (**MERGE** lo hace), podemos utilizar la instrucción **CHAIN MERGE,** tras la cual se indica el nombre del programa a mezclar.

CHAIN MERGE permite incluir el número de línea en el cual debe comenzar la ejecución una vez efectuada la mezcla. Entonces lo escribiremos así:

chain merge "nombre",450 (por ejemplo)

Si queremos borrar algunas líneas, también esta instrucción lo permite. En este caso se escribe así:

chain merge "nombre",450,delete 700-800

En este ejemplo, se mezclarían el programa en memoria con el grabado, llamado «nombre»; se borrarían las líneas comprendidas entre la 700 y la 800 y se reanudaría la ejecución en la línea 450.

CHAIN permite cargar un programa en memoria desde otro, sustituyéndolo, para ser ejecutado inmediatamente. Tras él, se indica el nombre del programa para cargar y el número de línea en el cual ha de comenzar la ejecución. Si se omite el número de línea, comienza la ejecución en la primera que tenga el programa recién almacenado.

CHAIN y **CHAIN MERGE** son las instrucciones adecuadas para utilizar en el raro caso de que nuestro programa sea tan grande que no quepa entero en la memoria. Nos veríamos obligados entonces a grabarlo por partes, que se ejecutarían secuencialmente (en el caso de una cinta) o sin seguir un criterio ordenado (para discos).

La Velocidad De Grabación

Los usuarios de cinta pueden variar la velocidad de grabación de sus programas y ficheros. La instrucción al efecto es **SPEED WRITE,** que lleva un solo parámetro: 0 ó 1. El 1 indica que se debe grabar a 2000 bits por segundo (baudios). El 0, 1000 baudios. Al ser más fiable la velocidad lenta, ésta es la que utiliza implícitamente el ordenador.

Esta instrucción afecta a todas las órdenes de manejo de cinta y no es necesario saber cómo fue grabado un programa puesto que el ordenador selecciona automáticamente la velocidad correcta.

SPEED WRITE no afecta al funcionamiento en disco.

¿DISCO A CINTA O CINTA A DISCO?

Si disponemos de un sistema con cinta y disco, podemos seleccionar dónde vamos a grabar o desde dónde vamos a cargar los programas. Los comandos apropiados empiezan todos por el signo | (barra vertical).

Para seleccionar la transmisión de datos en disco, se escribe:

|disc

Y para trabajar con el cassette:

|tape

Para seleccionar las salidas y entradas en concreto, disponemos de otras instrucciones. Permiten seleccionar uno u otro medio para grabar o cargar, de manera que pueden transferirse programas o datos de un soporte a otro.

Instrucción	Entrada	Salida
DISC	Disco	Disco
TAPE	Cinta	Cinta
TAPE.OUT	*	Cinta
DISC.OUT	*	Disco
TAPE.IN	Cinta	*
DISCO.IN	Disco	*

(*: dependiente de la última instrucción | TAPE, | DISC o bien del modo implícito del ordenador.)

LOS FICHEROS SECUENCIALES

El proceso general de creación y lectura de un fichero secuencial es el siguiente.

Primero debe crearse el fichero, con la instrucción **OPENOUT.** A continuación, se escriben los datos, numéricos o alfanuméricos, en cualquier orden, pero que después debemos recordar. Una vez concluido el proceso de escritura, debe cerrarse el fichero **(CLOSEOUT).**

Para leerlo, debe abrirse con la instrucción **OPENIN,** y proceder a la lectura de los datos, teniendo cuidado de no sobrepasar el final del fichero, para lo cual utilizaremos la función **EOF.** Finalizada la lectura, se cierra el fichero **(CLOSEIN).**

Por otra parte, no es posible corregir un fichero secuencial, hay

que crearlo de nuevo, y seguramente borrar el anterior, que queda entonces desfasado.

Supongamos un fichero muy pequeño, en el cual se guardan día, mes y año de la última actualización de un programa (el propio programa se encargaría de grabar cada vez la nueva fecha). El día y el año lo escribiremos en número; el mes en letra.

Para crear el fichero será válida la instrucción:

openout "fecha"

tras la cual escribiremos los datos, que supondremos que están almacenados en las variables **día, me$** y **year,** respectivamente.

Para escribir datos se utiliza la instrucción **WRITE;**

write #9,lista-de-datos

La *lista-de-datos* es una o más expresiones o variables, separadas por comas.

En nuestro caso, grabaremos los datos con:

write #9,dia,me$,year

y a continuación cerraremos el fichero, con **CLOSEOUT.** Aunque el ordenador ejecute un **WRITE,** el disco o la cinta no tienen por qué moverse; lo harán sólo cuando haya una cantidad determinada de datos pendientes o bien aparezca la instrucción **CLOSEOUT.**

Para leer nuestro fichero, utilizaremos la expresión:

openin "fecha"

y para leer los datos almacenados —ya sabemos que son un número, una cadena, y otro número— utilizaremos:

input#9,a,b$,c

con lo cual **a** contendrá el día, **b$** el mes en letra, y **c** el año.

Si utilizamos el fichero de manera que no es conocida o fija la cantidad de datos introducidos, debemos saber cuándo acaba el fichero al leerlo. Para ello, se utiliza la función **EOF** (End Of File, final de fichero). En el caso de intentar leer más allá del final del fichero, se produce el error *EOF met.*

EOF genera el valor − 1 (verdadero) si se ha llegado al final del fichero, y 0 (falso) en caso contrario. El programa adjunto muestra el modo de utilizarlo correctamente.

DIRECTORIOS EN DISCO Y CINTA

Para ver todos los ficheros que contiene un disco o una cinta, se

utiliza la orden **CAT** (la cinta habrá que rebobinarla por completo). En el caso de sistemas con dos tipos de soporte, **CAT** está afectado por las órdenes **|TAPE, |DISC,** etc...

```
1 REM PROGRAMA 1
10 REM Programa para comprobar como funciona EOF
20 REM Primero se graba un numero de datos especificado
30 REM por el usuario, y a continuacion se leen si  so-
40 REM brepasar el final del fichero.
50 CLS
60 INPUT "Cuantos datos escribo",n
70 OPENOUT "prueba"
80 FOR a=1 TO n
90 WRITE#9,INT(RND*100)+1
100 NEXT
110 CLOSEOUT
120 n=0:REM Ya no conocemos los datos que hay en el fichero.
130 PRINT "Comienza la lectura"
140 contador=0
150 OPENIN "prueba"
160 WHILE NOT EOF
170 INPUT#9,x
180 contador=contador+1
190 PRINT contador;"dato: ";x
200 WEND
210 CLOSEIN
220 END
```

La orden específica para sistemas de disco es **|DIR,** independiente de la selección de soporte. Siempre muestra el contenido de discos.

CAT, sobre disco, muestra los ficheros ordenados alfabéticamente e indica su longitud aproximada en Kilobytes. Por el contrario, **|DIR** muestra los ficheros según los encuentra, sin ninguna clase de indicación adicional.

INTERRUPCIONES

na característica casi única del BASIC Amstrad es la posibilidad de controlar el microprocesador directamente a través de interrupciones. Esto significa que podemos pedir que dos tareas se lleven a cabo «simultáneamente», requiriendo a intervalos la atención del microprocesador para una y otra. Estos intervalos son de 1/50 de segundo.

Existe una interrupción que funciona continuamente en el CPC. Se trata de aquella que explora el teclado para ver si se ha pulsado alguna tecla, y en ese caso, recordarla o bien emitir el carácter correspondiente, según el estado en el cual se encuentre el ordenador.

Pues bien, nosotros también podemos generar nuestras propias rutinas de interrupción, desde el propio BASIC. Será útil también para inhibir temporalmente éstas; la instrucción al efecto es **DI** (sin parámetros). Y la que permite reactivarlas es **EI.** Así, será conveniente que nuestras rutinas de interrupción comiencen y terminen con **DI** y **EI,** respectivamente, con el fin de evitar que durante la ejecución de la rutina de interrupción, se produzca nuevamente una llamada a la misma, con el consiguiente efecto de entrar en un ciclo sin fin.

CUATRO RELOJES EN EL AMSTRAD

Efectivamente, nuestro ordenador posee cuatro relojes independientes. Podemos basarnos en ellos para establecer hasta cuatro interrupciones propias. Los relojes están numerados del 0 al 3, y todos ellos llevan un ritmo de 1/50 de segundo (es decir, 0,02 segundos).

El reloj número 3 tiene la máxima prioridad. Esto significa que si se requieren varias acciones a la vez, la que dependa de este reloj tendrá preferencia. El 0 es el de menor prioridad.

La primera instrucción que utiliza estos relojes es **EVERY** (cada). La forma general es:

every c,r gosub n

donde **c** es una cantidad de ciclos de tiempo de un reloj, **r** @ es el número de dicho reloj, y **n** es el número de línea al cual debe saltar la ejecución del programa (en forma de subrutina) cada vez que se cumplan los **c** ciclos del reloj **r.** Leída en castellano, esta instrucción significa: *cada c ciclos del reloj r, ve a la subrutina de la línea n.*

Si sólo queremos que la acción se ejecute una vez, podemos utilizar **AFTER** (después), cuya forma general es:

after c,r gosub n

```
10 REM Demostracion de EVERY
20 EVERY 50,1 GOSUB 70
30 REM Cada 50X0.02 segundos, o sea, cada segundo, se
40 REM invoca a la subrutina de la linea 70
50 PRINT "Estoy en el programa principal"
60 GOTO 50
70 PRINT "Estoy en la subrutina"
80 RETURN
```

también están aquí el número de ciclos de espera, el número de reloj y el de la línea en la cual comienza la subrutina.

En este caso, la instrucción completa significa: *después de que hayan pasado c ciclos del reloj r, ve a la subrutina n.* Se trata, en definitiva, de una «acción retardada». Ofrecemos también un ejemplo.

Aunque los temporizadores estén activados por **AFTER** o **EVERY,** no tenemos por qué desconocer su estado en un momento determinado de la ejecución. Si queremos detenerlos, podemos hacerlo con la función **REMAIN.** Debe indicarse entre paréntesis el número de reloj:

REMAIN(r)

```
10 AFTER 250,0 GOSUB 70:CLS
20 PRINT "Adivine una letra en 5 segundos"
30 a$=INKEY$:IF indicador=1 THEN END
40 IF a$<>CHR$(INT(RND*26)+97)) THEN 30
50 PRINT a$;" es correcto. Usted gana."
60 SOUND 1,478:SOUND 1,358:END
70 SOUND 1,2000:indicador=1:RETURN
```

El efecto de esta función es detener el reloj mencionado (**r**) y generar el valor en ciclos que le quedaban a dicho reloj para requerir una subrutina.

La utilización de interrupciones puede resultar muy útil en los programas de trato directo con el usuario, donde es necesario cuidar los tiempos de respuesta, de presentación en pantalla, etc... Es una herramienta realmente valiosa y no muy frecuente en este tipo de ordenadores.

ERRORES

abemos que los errores durante la ejecución de un programa no siempre son evitables. BASIC nos ofrece un sistema de gestión de errores, gracias al cual el propio programa puede hacerse cargo de remediarlo, o bien devolver el control al sistema operativo para que emita el mensaje de error correspondiente.

La instrucción de intercepción de errores es **ON ERROR GOTO.** Le sigue un número de línea, que indica el grupo de instrucciones hacia las cuales debe dirigirse la ejecución del programa en caso de producirse algún error.

Dentro de este conjunto de líneas dedicadas a la gestión del error, puede informarse al usuario sobre el tipo de problema (por ejemplo, número demasiado grande, etc...).

Para averiguar el error producido podemos utilizar la función **ERR,** la cual genera un código correspondiente al mensaje de error (ver tabla). Gracias a esta función, y a **ERL,** que genera el número de línea en la cual se produjo el error, disponemos de un método muy eficaz para localizar los temibles fallos del programa.

Otra función muy especial es **DERR,** dedicada exclusivamente a

facilitar información acerca de los errores de disco. Puede resultar muy útil si el programa soporta algún fichero o simplemente maneja la unidad de disco.

Tabla de mensajes de error de BASIC. Página 7/27 del manual de CPC 6128.

COD.	Mensaje	Significado
1	Unexpected NEXT	NEXT inesperado
2	Syntax error	Error de sintaxis
3	Unexpected RETURN	RETURN inesperado
4	DATA exhausted	DATA agotada
5	Improper Argument	Argumento inapropiado
6	Overflow	Sobrepasamiento
7	Memory full	Memoria llena
8	Line does not exist	Línea no existente
9	Subscript out of range	Subíndice fuera de rango
10	Array already dimensioned	Matriz ya dimensionada
11	Division by zero	División por cero
12	Invalid direct command	Comando directo erróneo
13	Type mismatch	Error de tipo
14	String space full	Espacio para cadenas lleno
15	String too long	Cadena demasiado larga
16	String expression too complex	Expresión de cadena demasiado compleja
17	Cannot CONTinue	No se puede continuar
18	Unknown user function	Función de usuario desconocida
19	RESUME missing	Falta RESUME
20	Unexpected RESUME	RESUME inesperado
21	Direct command found	Localizado comando directo
22	Operand missing	Falta operando
23	Line too long	Línea demasiado larga
24	EOF met	Localizado fin de fichero
25	File type error	Error de tipo de fichero
26	NEXT missing	Falta NEXT
27	File already open	Fichero ya abierto
28	Unknown command	Comando desconocido
29	WEND missing	Falta WEND
30	Unexpected WEND	WEND inesperado
31	File not open	Fichero no abierto
32	Broken in	Interrumpido en

Tabla de mensajes de error de disco. Página 7/31 del manual de CPC 6128.

COD. DERR	Significado
0 ó 22	Pulsación de ESC
142	El canal no se halla en situación correcta
143	Final físico de fichero
144	Error de sintaxis, generalmente, por un nombre incorrecto de fichero
145	Fichero ya existente
146	Fichero inexistente
147	Directorio lleno
148	Disco lleno
149	Se ha efectuado un cambio de disco con ficheros abiertos en él
150	Fichero de sólo lectura
154	Localizado fin de fichero (lógico, no físico)

Si queremos provocar un error sin necesidad de acudir a la instrucción que lo genera, podemos utilizar **ERROR.** Esta orden va seguida de un número, que representa el mensaje de error que debe emitirse. Su utilidad es muy limitada, excepto si tenemos en cuenta que esta instrucción también está afectada por la intercepción de errores. Esto significa que si hemos ordenado dicha intercepción con **ON ERROR GOTO** y a continuación utilizamos **ERROR,** la ejecución saltará donde se haya indicado.

Cuando la rutina invocada al ocurrir cualquier error ha terminado, debe devolver la ejecución al programa principal. Para ello puede utilizarse **RESUME** o **RESUME NEXT.**

La primera instrucción se escribe delante de un número de línea, al cual debe dirigirse la ejecución para reanudar el programa. **RESUME NEXT** indica al ordenador que continúe con el programa en la instrucción siguiente a la que provocó la interrupción.

PROGRAMAS INTOCABLES

Sabemos que al pulsar la tecla **ESC** dos veces, el programa se detiene. Podemos impedir que esto ocurra, con las instrucciones adecuadas.

ON BREAK CONT indica al ordenador que debe ignorar todas las

pulsaciones de la tecla **ESC,** de manera que el programa no pueda interrumpirse de ninguna forma. Antes de añadir esta instrucción al programa, conviene grabarlo primero, ya que si contiene algún fallo y su ejecución no tiene fin, no podremos interrumpirlo y, por lo tanto, lo perderemos. La orden que anula este efecto es **ON BREAK STOP.**

En alguna ocasión será interesante ejecutar una determinada acción cuando se intenta detener el programa. Esto es lo que permite **ON BREAK GOSUB,** que se escribe delante de un número de línea, el cual es el principio de una subrutina. En esta subrutina podemos indicar al usuario que no debe intentar interrumpir la ejecución, etc...

Al llegar al RETURN correspondiente, la ejecución volverá al punto en el que se intentó detener el programa. Esta orden también se anula con **ON BREAK STOP.**

MEMORIA

En este último capítulo veremos instrucciones dedicadas a estructurar la memoria, utilizar el reloj interno del Amstrad y modificar la zona destinada a los caracteres definibles.

Comenzaremos con **TIME.** Esta función genera el valor en 1/300 de segundo del tiempo transcurrido desde que se encendió o reinicializó el ordenador. Dado el pequeñísimo intervalo de tiempo que sucede entre incremento e incremento del valor de esta función, resulta muy apropiada para desordenar la lista de números aleatorios con **RANDOMIZE.**

Este desorden está garantizado con expresiones del tipo

randomize time

o cualquier otra en la cual **RANDOMIZE** reciba algún valor en función de **TIME.**

La exactitud de este reloj es bastante fiable; podemos construirnos uno aprovechando la ventaja de poder presentar la hora como más nos guste.

```
10 CLS:REM Reloj
20 INPUT "Hora: ",hora
30 INPUT "Minuto: ",minuto
40 INPUT "Segundo: ",segundo
50 CLS:referencia=INT(TIME/300)
60 WHILE hora<13
70 WHILE minuto<60
80 WHILE tiempo<60
90 tiempo=(INT(TIME/300)-referencia)+segundo
100 LOCATE 1,1
110 PRINT hora;":";minuto;":";tiempo
120 WEND
130 tiempo=0:segundo=0:minuto=minuto+1
140 GOTO 50
150 WEND
160 minuto=0:hora=hora+1
170 WEND
180 hora=1
190 GOTO 60
```

EL TOPE DE MEMORIA

BASIC utiliza un banco de 64k para los programas. En realidad, no lo ocupa por entero un programa, puesto que el intérprete necesita una pequeña cantidad para sus variables internas. Por lo tanto, la dirección de memoria máxima utilizable no es 65535, sino un valor variable determinable mediante la función **HIMEM** *(HIgh MEMory)*.

Evidentemente, este tope puede ser modificado. No explicaremos aquí los detalles sobre memoria; nos limitaremos a decir que para modificar este tope se utiliza **MEMORY.** Por ejemplo,

memory 40000

fija la dirección más alta de memoria utilizable por BASIC para variables y programa en 40000.

DEFINIR CARACTERES

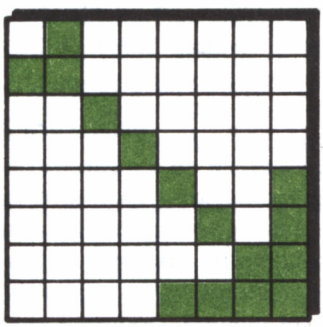
Nuestro Amstrad permite definir caracteres, además en número variable: desde 1 a 223. No se trata de caracteres independientes, sino de los propios de la máquina. De esta manera, podremos redefinir caracteres ASCII que luego serán utilizados por el ordenador con normalidad.

Pero, como podemos suponer, la definición ocupa cierta memoria, lo cual significa que puede haber problemas con programas de gran extensión.

Para definir un carácter necesitaremos un papel cuadriculado con un tamaño de 8×8 cuadros. Tenemos que rellenar por completo cada cuadro que consideremos que deba ser negro. Por ejemplo, para definir una flecha inclinada tomaríamos el esquema adjunto.

Una vez diseñado el carácter a nuestro gusto, debemos codificarlo. Supongamos para ello ocho filas horizontales de ocho cuadros cada una. En nuestro dibujo, encontraremos que dentro de esa fila hay cuadrados negros y blancos. Escribiremos un cero para cada blanco y un uno para cada negro. En nuestro ejemplo, quedaría así:

00001111
00000011
00000101
00001001
00010000
00100000
11000000
01000000



Ahora basta con convertir cada fila a decimal, teniendo en cuenta que éstas son representaciones binarias de la figura. Si queremos ahorrarnos el trabajo, podemos expresarlas con el prefijo de números binarios, de manera que la primera fila sería:

&X1111

(recordemos que los ceros a la izquierda no valen ni en decimal ni en binario).

Vamos ya con la parte práctica. La instrucción que permite definir caracteres es **SYMBOL AFTER.** Se expresa como

symbol after nc

donde **nc** es el número de carácter a partir del cual (y hasta el final) podrán ser definidos.

Por ejemplo, tras teclear

symbol after 250

los caracteres 250 a 255 incluidos podrán ser redefinidos. Al encender o reinicializar el ordenador, se ejecuta automáticamente la instrucción **SYMBOL AFTER 240,** con lo cual son redefinibles directamente los últimos 16 caracteres.

Si queremos redefinir todos, habrá que escribir **SYMBOL AFTER 32,** y para no poder definir ninguno, **SYMBOL AFTER 255.**

La orden de definición de un carácter es:

symbol c,d1,d2,d3,d4,d5,d6,d7,d8

donde **c** es el número del carácter al que afecta la definición y los **d** son las ocho filas codificadas, en binario o decimal. Si queremos definir nuestra flecha en el carácter 255, escribiremos:

symbol 255, &x1111, &x11, &x101, &x1001, &x10000, &x100000, &x11000000, &x1000000

o bien, si codificamos las imágenes en binario:

symbol 255,15,3,5,9,16,32,192,64

A partir de ahora, para obtener el carácter, basta con teclear:

print chr$(255)

Podemos basarnos en los caracteres incluidos en la propia máquina para hacer los nuestros. Es posible que sólo queramos modificarlos ligeramente. Así, es posible combinar varios caracteres definidos para formar una sola figura, ubicando cada uno convenientemente con **LOCATE.** Los caracteres serán afectados por la anchura impuesta con **MODE;** de esta manera, podemos ensanchar y estrechar nuestras figuras.

Pero no es esta la única posibilidad gráfica del CPC, como podremos comprobar el número destinado a su estudio en esta misma colección.

APENDICE 1

INICIALIZACIÓN DE BASIC

BASIC se carga desde el sistema operativo CP/M. La línea de órdenes puede contener varios parámetros. En general, la expresión completa de llamada a BASIC es la siguiente:

BASIC nombre de fichero /F:número de ficheros /M:dirección de memoria /S:longitud

nombre de fichero indica algún fichero que contenga un programa BASIC codificado, protegido o en ASCII. Este programa se cargará tras la instalación de BASIC, y a continuación se ejecutará.

Los demás parámetros, si se incluyen, comienzan con la barra inclinada (/). Pueden aparecer en cualquier orden, e indican lo siguiente:

/F: número de ficheros: indica el máximo número de ficheros manejable al mismo tiempo por BASIC. Tiene que estar comprendido en el margen 0-255, aunque la memoria no sea suficiente para la cantidad máxima. Si no se especifica este parámetro, se establece el máximo en 3.

/M: memoria: asigna la dirección más alta utilizable por BASIC. Si no se especifica, se toma la máxima memoria disponible.

/S: longitud: indica la longitud máxima de los registros, expresada en caracteres. Si no se incluye este parámetro, la longitud se fija en 128.

Todos los efectos de estos parámetros pueden ser modificados ya durante la ejecución de BASIC mediante las instrucciones **MEMORY** y **CLEAR.**

APENDICE 2

PALABRAS CLAVE DE BASIC

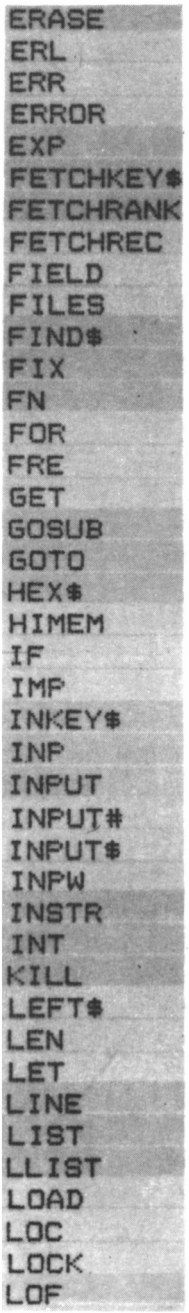
CPC 464/472

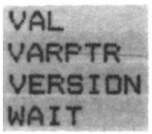
ABS
AFTER
AND
ASC
ATN
AUTO
BIN$
BORDER
CALL
CAT
CHAIN
CHR$
CINT
CLEAR
CLG
CLOSEIN
CLOSEOUT
CLS
CONT
COS
CREAL
DATA
DEF
DEFINT
DEFREAL
DEFSTR
DEG
DELETE
DI
DIM
DRAW
DRAWR
EDIT
EI
ELSE
END
ENT
ENV
EOF
ERASE
ERL
ERR
ERROR
EVERY
EXP
FIX
FN
FOR
FRE
GOSUB
GOTO
HEX$
HIMEM
IF
INK
INKEY
INKEY$
INP
INPUT
INSTR
INT
JOY
KEY
LEFT$
LEN
LET
LINE
LIST
LOAD
LOCATE
LOG
LOG10
LOWER$
MAX
MEMORY
MERGE
MID$
MIN
MOD
MODE
MOVE
MOVER
NEXT
NEW
NOT
ON
ON BREAK
ON ERROR GOTO
ON SQ
OPENIN
OPENOUT
OR
ORIGIN
OUT
PAPER
PEEK

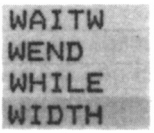
PEN
PI
PLOT
PLOTR
POKE
POS
PRINT
RAD
RANDOMIZE
READ
RELEASE
REM
REMAIN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROUND
RUN
SAVE
SGN
SIN
SOUND
SPACE$
SPC
SPEED
SQ
SQR
STEP
STOP
STR$
STRING$
SWAP
SYMBOL
TAB
TAG
TAGOFF
TAN
TEST
TESTR
THEN
TIME
TO
TROFF
TRON
UNT
UPPER$
USING
VAL
VPOS
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR
XPOS
YPOS
ZONE

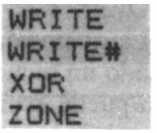
PCW 8256/8512

ABS
ADDKEY
ADDREC
ALL
AND
AS
ASC
ATN
AUTO
BASE
BUFFERS
CALL
CDBL
CHAIN
CHR$
CINT
CLEAR
CLOSE
COMMON
CONSOLIDATE
CONT
COS
CREATE
CSNG
CVD
CVI
CVIK
CVS
CVUK
DATA
DEC$
DEF
DEFDBL
DEFINT
DEF SEG
DEFSGN
DEFSTR
DELETE
DELKEY
DIM
DIR
DISPLAY
EDIT
ELSE
END
EOF
EQV
ERA


ERASE
ERL
ERR
ERROR
EXP
FETCHKEY$
FETCHRANK
FETCHREC
FIELD
FILES
FIND$
FIX
FN
FOR
FRE
GET
GOSUB
GOTO
HEX$
HIMEM
IF
IMP
INKEY$
INP
INPUT
INPUT#
INPUT$
INPW
INSTR
INT
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCK
LOF
LOG
LOG10
LOWER$
LPOS
LPRINT
LSET
MAX
MEMORY
MERGE
MID$
MIN
MKD$
MKI$
MKIK$
MKS$
MKUK$
MOD
NAME
NEXT
NEW
NOT
OCT$
ON
ON ERROR GOTO 0
OPEN
OPTION
OR
OSERR
OUT
OUTW
PEEK
POKE
POS
PRINT
PRINT#
PUT
RANDOMIZE
RANKSPEC
READ
REM
REN
RENUM
RESET
RESTORE
RESUME
RESUME 0
RETURN
RIGHT$
RND
ROUND
RSET
RUN
SAVE
SEEKKEY
SEEKNEXT
SEEKPREV
SEEKRANK
SEEKREC
SEEKSET
SGN
SIN
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
STRIP$
SWAP
SYSTEM
TAB
TAN
THEN
TO
TROFF
TRON
TYPE
UNT
UPPER$
USING
USR







APENDICE 3

MENSAJES DE ERROR Y POSIBLE ORIGEN

Incluimos a continuación una lista con todos los errores generados por BASIC. En primer lugar, se indica el código (número de error que se obtiene con la función **ERR**) después el mensaje correspondiente y un breve comentario sobre el posible origen del error.

ERRORES DE BASIC ORDINARIO

1 Unexpected NEXT
Se ha encontrado un NEXT que no corresponde a ningún FOR, o bien se ha especificado tras el NEXT una variable que no es la del FOR.

2 Syntax error
Orden, expresión, parámetros o puntuación no reconocidos por el intérprete.

3 Unexpected RETURN
Se ha encontrado un RETURN sin que haya pendiente de retorno ningún GOSUB.

4 DATA exhausted
Se produce siempre en una línea con READ, y no hay datos (o no suficientes) para asignar contenido a las variables especificadas en READ.

5 Improper argument
Este es un error no específico. El error está en un argumento o un parámetro, seguramente por estar fuera del rango admisible.

6 Overflow

La operación aritmética procesada en ese momento da como resultado un número más grande del que BASIC puede procesar. También puede ocurrir si se intenta convertir un número en simple o doble precisión a entero con signo de 16 bits.

7 Memory overflow

El programa y/o variables ocupan demasiada memoria, o existe un anidamiento demasiado profundo de GOSUB, WHILE o FOR. Puede intentarse hacer sitio con CLEAR o MEMORY.

8 Line does not exist

La orden hace referencia a un número de línea que no existe.

9 Subscript out of range

Se ha indicado un subíndice fuera de margen en una variable dimensionada.

10 Array already dimensioned

Se ha intentado dimensionar dos veces la misma variable, o se ha intentado ejecutar OPTION BASE tras una instrucción de dimensión (o del propio OPTION BASE).

11 Division by zero

División por cero, producida en cualquiera de las divisiones posibles (entera, normal, MOD) y en la exponenciación.

12 Invalid direct command

Se ha intentado ejecutar una instrucción directamente y sólo permite serlo dentro de un programa.

13 Type mismatch

Desajuste en los tipos. Se relacionan valores, variables o funciones de un tipo determinado con otro que no concuerda.

14 String space full

Se han creado demasiadas cadenas alfanuméricas y no hay espacio para más.

15 String too long

Tras una operación o asignación, la cadena supera los 255 caracteres.

16 String expression too complex

Expresión de operación de cadenas demasiado compleja. Debe separarse en varias partes para ser procesada.

17 Cannot CONTinue
Se ha intentado ejecutar CONT para reanudar el programa pero se ha modificado éste.

18 Unknown user function
Se ha invocado una FN sin haberla definido, o bien después de haber modificado el programa o borrado las variables.

19 RESUME missing
Se ha llegado al final de un programa estando dentro de la rutina invocada por ON ERROR GOTO, pero no se ha encontrado un RESUME para reanudar la ejecución.

20 Unexpected RESUME
El programa ha encontrado un RESUME pero sin haber recibido la desviación de ejecución de ON ERROR GOTO.

21 O/S dependent error
Error dependiente del sistema operativo.

22 Operand missing
Expresión incompleta.

23 Line too long
Línea demasiado larga para ser codificada por BASIC. Hay que escribirla en varias partes.

26 NEXT missing
BASIC ha encontrado un FOR al que no corresponde ningún NEXT. Normalmente BASIC detecta esta falta antes de empezar a ejecutar el interior del bucle.

29 WEND missing
BASIC ha encontrado un WHILE al que no corresponde ningún WEND.

30 Unexpected WEND
BASIC ha encontrado un WEND al que no corresponde ningún WHILE.

ERRORES DE DISCO Y FICHEROS

50 Record overflow
La orden FIELD ha sobrepasado el límite de longitud expresada para el registro.

52 File number error
Se ha indicado un número de fichero fuera del margen admitido, o se intenta leer o escribir sobre un fichero que no está abierto.

53 File not found
No se ha encontrado el fichero especificado.

54 File type error
Se ha intentado realizar alguna operación de fichero en un modo distinto a aquel en que fue creado.

55 File already open
Se ha intentado abrir un fichero que ya lo estaba, o con un número de fichero que ya correspondía a otro.

57 Disc I/O error
Error físico de lectura/escritura.

58 File already exists
Existe un fichero cuyo nombre coincide con uno al que se cambia el nombre (NAME, REN) o bien CREATE intenta crear fichero que ya existen.

61 Disc full
Disco lleno. Este error puede provocar desajustes en los ficheros Jetsam.

62 EOF met
Se ha intentado leer más allá del final de un fichero secuencial.

63 Record number error
Número de registro fuera del margen 1-32767. Número de registro no existente (Jetsam).

64 File name invalid
El nombre de fichero para grabar, crear o abrir no es un nombre correcto de CP/M.

66 Direct command found
Al cargar un programa BASIC ha encontrado una instrucción sin su correspondiente número de línea. Puede ser debido a que se intenta cargar un fichero que no es un programa BASIC.

67 Directory full
El número de fichero en el disco ha llegado al máximo.

ERRORES ESPECÍFICOS DE JETSAM

113 Not a keyed file
El fichero abierto con la orden OPEN «K» no es un fichero de acceso por claves.

115 Inconsistent files
Hay un desajuste entre índices y datos, que no se remedió con CONSOLIDATE ni con un CLOSE explícito. El fichero es inútil y debe utilizarse la copia de seguridad más moderna de que se disponga.

APENDICE 4

ORDENES DE TECLADO

La pulsación de **ALT** en combinación con otra tecla, provoca determinadas reacciones por parte del intérprete. En algunas ocasiones, las teclas dedicadas al procesador (**BUSC,** etc.) tienen el mismo efecto.

ALT A
Equivale a la tecla de cursor atrás (←). Si el cursor está en la primera columna, entonces se recupera la misma línea escrita y enviada con RETURN.

ALT C
Detiene la ejecución del programa, emitiendo el mensaje *break.* Es equivalente a la tecla STOP.

ALT F
Equivale a la tecla de cursor a la derecha (→) y CARC.

ALT G
Borra el carácter que hay a la derecha del cursor. Equivale a la tecla ←BORR.

ALT H
Borra hacia la izquierda. Equivale a la tecla CAN.

ALT I
Pasa el cursor al siguiente tope de tabulación. Equivalente a TAB.

ALT J

Baja el cursor a la siguiente fila, pasando a ésta los caracteres que haya en ese momento a la derecha del cursor.

ALT M

Retorno del carro. Equivale a RETURN e INTRO.

ALT S

Detiene temporalmente la ejecución del programa, hasta que se pulse otra tecla. Si se pulsa cuando no se ejecuta ningún programa, BASIC lo rechaza.

ALT U

Orden de borrado. Tras esta pulsación, se presionará el carácter hasta el cual queremos que se borre, hacia la derecha de la línea y a partir de la posición del cursor. Si el carácter pulsado no se encuentra en la línea, se borra hasta el final. Equivale a CORT.

ALT V

Control de inserción-sobreescritura. Normalmente, si situamos el cursor entre dos caracteres y escribimos, los que están a su derecha se desplazan para dejar sitio. Al pulsar **ALT V,** se cancela este modo, y el cursor borrará los caracteres sobre los que vaya escribiendo. Para volver a la situación normal, se pulsa de nuevo **ALT V.**

ALT /

Orden de búsqueda de caracteres. Tras esta pulsación, se tecleará el carácter sobre el cual deseamos que se sitúe el cursor (debe estar a su derecha). Si no se encuentra, el cursor se coloca al final de la línea. Equivale a BUSC.

APENDICE 5

ORDENES EXCLUSIVAS DE CPC 664/6128

CLEAR INPUT	DERR	GRAPHICS PEN
COPYCHR$	FILL	ON BREAK CONT
CURSOR	FRAME	MASK
DEC$	GRAPHICS PAPER	

Ha llegado el momento de perfilar nuestros conocimientos sobre el lenguaje BASIC, ya esbozado en el octavo volumen de esta GRAN BIBLIOTECA AMSTRAD, estudiando en más detalle aquellas instrucciones propias de cada modelo, motivo por el cual este libro se halla dividido en dos partes bien diferenciadas: una de ellas dedicada por entero a los PCW y otra a los CPC.

GRAN BIBLIOTECA AMSTRAD

450 ptas.
(incluido IVA)
Precio en Canarias, Ceuta y Melilla: 435 ptas.