CURSO AUTODIDACTICO DE BASIC II

AMSTRAD

CPC·464

AMSTRAD

BASIC

Curso Autodidáctico de Basic II

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

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

I.S.B.N.: 84-86176-33-6

Depósito Legal: M-18276-1985

AMSTRAD BASIC

Curso Autodidáctico de Basic II

Parte 2

MAS SOBRE BASIC

por Ian Padwick y George Tappenden

First edition 1985

Amstrad Consumer Electronics plc
Brentwood House
169 Kings Road
Brentwood
Essex

Amstrad BASIC
A Tutorial Guide
Part 2: More BASIC

SOFT 156
ISBN 1 85084 001 6

CONTENIDOS

PREFACIO

Esta es la Parte 2 de un curso autodidáctico sobre programación en BASIC usando el Ordenador Personal a Color Amstrad CPC464. Mientras que la Parte 1 estaba pensada para el principiante, **Más sobre BASIC** es aconsejable para cualquiera que esté familiarizado con los fundamentos de la programación y que entienda ya los comandos BASIC más habituales.

Las dos datocassettes que acompañan este texto escrito contienen programas de ordenador que son una parte integral del curso.

La datocassette A contiene:

- ☐ Programas que ayudan a ilustrar las técnicas descritas en el texto.
- ☐ Juegos para tu diversión, y sobre todo para demostrar las facultades del CPC464.

La datocassette B contiene:

- ☐ Pruebas de **autoevaluación** para asegurarte que has comprendido el contenido del capítulo precedente.

La tercera parte de esta serie cubre las facetas más avanzadas del Amstrad BASIC y está pensada para programadores expertos.

Capítulo 1
AQUI ESTAMOS DE NUEVO

Si leíste la Parte 1 de esta serie, ya habrás aprendido unos cuantos comandos del Amstrad BASIC y tendrás una idea muy buena sobre las posibilidades del Ordenador Personal a Color Amstrad CPC464. Esperamos que también hayas ensayado los nuevos conocimientos adquiridos escribiendo algunos programas por ti mismo y hayas tenido la satisfacción de ver que funcionan adecuadamente -quizás no al principio, aunque a estas alturas sería preocupante... Esta parte del curso intenta consolidar lo que has aprendido en la Parte 1 facilitándote incluso un vocabulario más extenso de comandos Amstrad BASIC.

Si te has saltado la Parte 1, eres merecedor de que repitamos el hecho de que Amstrad BASIC no funcionará necesariamente sobre otros ordenadores -y viceversa. Eso es porque contiene muchos comandos genuínos de Amstrad y funciones que no están disponibles en equipos menos avanzados.

Cada vez que una nueva **palabra clave,** o una ampliación o matiz de una palabra clave previamente descrita, se presenta en el texto, la imprimimos también en el margen externo de manera que puedas repasar fácilmente las páginas cuando necesites refrescar tu memoria.

COMO USAR ESTE LIBRO

Aunque este libro ha sido escrito con un mínimo de 'jerga' informática, sería muy molesto si continuaramos llamando a una espada un "instrumento de jardín con mango de madera y una hoja plana de metal". Dentro de cualquier campo de estudio especializado es inevitable que algunas palabras adquieran significados especiales, y la informática no va a ser la excepción. Así que comencemos con una mirada a los términos principales usados para describir los comandos del Amstrad BASIC.

Cada capítulo de este libro representa aproximadamente una o dos tardes de trabajo.

Habitualmente consistirá en:

- ☐ Explicación escrita
- ☐ Trabajo práctico sobre el ordenador
- ☐ Ejemplos para que tú mismo los programes

Hay ejercicios que refuerzan lo que has aprendido y además una prueba de autoevaluación programada en cada capítulo.

No te saltes capítulos. La nueva información se presenta progresivamente a lo largo del libro y se fundamenta en el conocimiento obtenido en los capítulos anteriores. Si crees que un capítulo o una sección de un capítulo, parece ser un poco complicada, basta con que la leas rápidamente una o dos veces y luego pases a desarrollarla concienzudamente. Los ejercicios y los programas que esperamos teclees y compruebes, están pensados para ayudarte a comprender los principios involucrados en el tema que se está comentando, así que no los eludas: hazlos. Asegúrate que has comprendido aprovechando las pruebas de autoevaluación.

Después de completar esta parte del curso serás capaz de escribir tus propios juegos o entretenimientos y programas de gestión -todavía no demasiado avanzados, ya que eso sólo se adquiere con la experiencia. La Parte 3 de este curso te instruirá adicionalmente sobre los detalles más precisos del Amstrad BASIC.

DESCRIPCIONES DE COMANDOS

En la Parte 1 vimos que un comando está siempre constituido por una palabra clave -un **verbo** en forma imperativa- al que acompañan uno o más parámetros -lingüísticamente son **nombres sustantivos**- tal y como en el comando para hacer que TRACE una línea:

```
DRAW x,y
```

donde los parámetros 'x' e 'y' dan las coordenadas gráficas de la posición en la pantalla hasta la que debe trazarse el segmento. Como ya sabrás, si tecleas caracteres erróneos o dejas fuera espacios y caracteres cuando tecleas el comando, se te producirá el mensaje de 'ERROR SINTACTICO'. Pero ¿qué es eso de **sintaxis**? ¿Y por qué necesitas una coma en la mitad del comando? Todo esto quedará claro al acabar este capítulo.

Sintaxis

En el lenguaje ordinario, la frase 'La cola fue movida el pequeño perro por', es ridícula totalmente. No hay muchos de nosotros (esperemos) que sueñen con romper las reglas de nuestro lenguaje de esa manera. La sintaxis, i.e. el orden en que las palabras se colocan en una frase, es tremendamente importante. Incluso en el ejemplo dado, es una medida de la superioridad del **cerebro humano** sobre ese **cerebro mecatrónico**, ya que una persona es capaz de comprender el sentido de esa frase y compensar su disconformidad con las reglas sintácticas.

Un caso más extremo es cuando hay una conexión imprecisa entre las diversas partes gramaticales de una frase:

La cola/el pequeño perro/fue movida

Si la palabra **conectiva**, que en el lenguaje ordinario son preposiciones y conjunciones, no se incluye la palabra 'por' en el lugar correcto hay una duda considerable sobre "quién movió qué". En BASIC las reglas sintácticas son incluso más rígidas que en el lenguaje normal. No hay ninguna duda sobre si una instrucción IF-THEN-ELSE es equivalente a la cambiada IF-ELSE-THEN ya que la segunda simplemente no tiene ningún sentido en BASIC (IF-ELSE-THEN no está admitida). De la misma manera, si olvidas dejar un espacio entre una palabra y otra en el lenguaje normal, puede que sea un poco más difícil de leer, pero todavía es fácilmente inteligible para nosotros. Pero si omites un espacio en blanco en un punto crítico cuando escribes en BASIC, el CPC464 es totalmente incapaz de determinar lo que estás intentando decirle, y a menudo llanamente rehusa a trabajar con ese programa hasta que hayas corregido el error.

Existen gran cantidad de nuevos comandos, y algunas veces son complicados, que vamos a aprender en este libro, así que para ahorrar tiempo y espacio, usaremos una clase de 'notación taquigráfica' para describir la sintaxis de cada uno de ellos. Eso implica algunos términos poco familiares y algunos signos de puntuación que requieren una breve explicación. Estudia el siguiente glosario cuidadosamente. Puede que tengas que consultarlo continuamente hasta que las palabras usadas se te hagan familiares.

Paréntesis.- Ciertos comandos y funciones exigen que los parámetros estén encerrados entre paréntesis '()', e.g. CHR$(97). Estos paréntesis también se usan para establecer el orden -la **prioridad**- en que han de llevarse a cabo las operaciones aritméticas de una expresión.

En este libro también usaremos otras dos clases de signos similares para describir los comandos. El primero, **corchetes angulados** '< >', significa que lo que está encerrado entre ellos se describe en alguna otra parte, y además ha de ser sustituido por el nombre o dato que el programador requiera. Los segundos, **corchetes cuadrados** '[ ]', muestran que lo que haya dentro de ellos es totalmente **opcional**, y puede por tanto prescindirse de especificarlo.

Preceptuado.- Cuando se enciende el CPC464 o cuando se **restaura** a sus condiciones iniciales, hay algunos parámetros, tales como los colores de pantalla, reborde y texto, cuyo valor está 'prefijado' automáticamente. Además de eso, hay comandos de trazar (DRAW) y pintar (PLOT) que una vez que se ha especificado la tinta (INK) correspondiente, pueden tener el valor de este parámetro **omitido** al dar los subsiguientes comandos. Parámetros como esos se conocen como de valores **preceptuados** (o bien **prescritos** en el programa interpretador) y veremos más adelante que hay muchos comandos donde los parámetros opcionales tienen un determinado valor preceptuado.

Expresión.- Véase Expresión Entera, Expresión Numeral y Expresión Literal.

Expresión Entera.- Cualquier fórmula explícita con operandos y operadores numéricos, que puede contener constantes y variables reales o enteras, y cuyo resultado se **redondea** al número entero más próximo antes de entregarlo.

Numeral entero.- Son constantes y valores de variables que no tienen parte fraccionaria (también llamada 'decimal').

Clave.- Una palabra reservada para ser usada en comandos BASIC tales como los verbos RUN, MOVE y PRINT, o los nombres que representan funciones BASIC tales como RND e INT.

Expresión Numeral.- Puede tener una de estas cuatro formas:

- ☐ Una constante numérica, e.g. 20473,611
- ☐ Una variable numeral, e.g. TOTAL
- ☐ El resultado de una expresión numeral, e.g. total/252*por ciento.
- ☐ Una función numeral, e.g. LOG (x)

Numerales reales.- Aquéllos que tienen parte fraccionaria.

Separadores.- Usados en BASIC para delimitar dónde comienza y termina una cláusula o un parámetro de un comando. Los usuales son:

coma	(,)
punto-y-coma	(;)
espacio	( )

El signo **dos-puntos** (:) también se usa pero para separar un comando de otro dentro de una misma línea.

Expresión literal.- Una serie de caracteres alfanuméricos tomados como un solo dato, y también llamados sartas, cadenas y series, que puede venir dada en una de las cuatro formas siguientes:

- ☐ Una constante literal (encerrada entre comillas), e.g. "Lourdes"
- ☐ Una variable (con el nombre terminado en $), e.g. Nombre$
- ☐ El resultado de una expresión literal, e.g. "Lourdes + Apelli$"
- ☐ Una función de resultado literal, e.g. LEFT$(Nombre$,1)

Ejemplo típico de sintaxis

Mira atentamente lo siguiente:

```
DRAW <coordenada x>,<coordenada y> [, <tinta>]
```

donde los parámetros han de ser:

<coordenada X>	= expresión entera
<coordenada Y>	= expresión entera
<tinta>	= expresión entera

Dado que los parámetros **coordenada X** y **coordenada Y** han de tener valores que sean enteros, están descritos en la sintaxis como **expresiones**, sabemos que podemos sustituir en esas posiciones o bien una constante numeral, o una variable numeral, o una combinación de constantes, variables y operadores cuyo resultado sea una cantidad numérica. Ambas de estas expresiones enteras deben incluirse en el comando. Si una de ellas, o ambas, o el separador (,) se omiten cuando se ejecute esa línea, el CPC464 dará el correspondiente mensaje de "Sintax error".

Pero en el caso del parámetro **tinta**, que está encerrado entre corchetes, si dejamos fuera esa parte del comando, el CPC464 supone que **implícitamente** estás haciendo referencia a la tinta más recientemente usada -la **preceptuada** en ese momento- y no te dará ningún mensaje de error. Como regla general, cuando un parámetro es opcional es porque hay para él un valor preceptuado de alguna manera.

Así, cada comando del Amstrad BASIC puede tener descrita su sintaxis. Usando estos paréntesis, corchetes y separadores, y ahora que ya sabes las diferentes maneras de las expresiones, no necesitamos emplear mucho más tiempo en las sutilezas de cada comando. Observa, por ejemplo, que nuestro comando DRAW requiere expresiones enteras para las coordenadas. Eso no significa que el programador solamente pueda usar números enteros como valores de esos parámetros, sino que el CPC464 redondeará el valor que se le dé. Así que no tienes por qué preocuparte sobre la parte fraccionaria en los valores de estos parámetros.

Si consultas tu ejemplar de la Guía del Usuario Amstrad CPC464, verás que estamos usando exactamente el mismo convenio. De esta manera puedes comprender rápidamente el uso de **cualquier** comando que no hayamos cubierto en los siguientes capítulos.

TIEMPO-OCIO

Carga el primer programa de la datocassette A llamado PASTOR. Puedes luego pasar una media hora disfrutando de este nuevo oficio, e intentando lograr que tu perro pastor consiga que dos ovejas estúpidas entren en su redil. Diviértete.

CONTROL DEL ESTUDIO

Para aquéllos que no habéis pasado por la Parte 1 de este curso, **Primeros Pasos,** sería una buena idea que tuviérais vuestro primer CONTROL autoevaluativo, llamado TAE1, para asegurarte que no hay ningún aspecto del Amstrad BASIC que necesites repasar antes de pasar al Capítulo 2. Incluye aunque ya hubieras completado la Parte 1, una pequeña revisión no te hará ningún daño.

Capítulo 2
CAMBIO DE DOMICILIO

Nos encontraremos algunos viejos amigos de la Parte 1 del curso en este capítulo: la CASA para comenzar. Nuestro diseño original fue un poquito largo y pesado con todos esos MOVEs, así que ya es hora de aprender cómo hacerlo más breve y claro. Vamos pues a echar una mirada más detallada a las **variables.**

Aunque antes de todo eso, veamos un grupo de comandos que son ayudas para la confección de programas.

ENCIMA Y DEBAJO DE LA LINEA

Cuando has usado LIST para hacer que te **liste** en pantalla un programa largo, probablemente te has cansado un poco de esperar que el trozo que tú querías apareciera. De hecho, este comando tiene un parámetro opcional que acelera las cosas bastante. Su sintaxis es:

LIST

```
LIST[<numero de linea>]
```

donde la **banda de números de línea** puede venir dada de cuatro formas. La mejor manera de comprender lo que traemos entre manos es ensayarlo tal y como sigue. Carga el siguiente programa de la datocassette A, llamado CASA (aunque parezca un 'castillo'), pero no hagas que se ejecute. Luego teclea:

```
LIST 100
```

Como puedes ver, **sólo** la línea 100 es listada en pantalla.

Ahora ensaya con este nuevo parámetro:

```
LIST -100
```

Esta vez el CPC464 te da el listado con todos los números de línea hasta el 100 inclusive. Ahora ya no tienes que ser rápido de reflejos para pulsar ESCape y detener el listado cuando lo deseas.

Como tercer ejemplo teclea una **banda** de números:

```
LIST 100-200
```

Para conseguir todas las líneas entre las 100 y la 200, ambas inclusive. Si imaginas que tienes una 'pifia molestora' en esa parte del programa, puedes ver fácilmente lo útil que es mostrar sólo esa sección en la pantalla cada vez que tengas que hacer un cambio.

Y finalmente, ensaya con una banda sin cota superior:

```
LIST 700-
```

lo que te listará todas las líneas desde la 700 inclusive hasta el final del programa, es especialmente útil si sigues la práctica normal de colocar todas tus subrutinas comenzando en números de línea concretos y fáciles de recordar.

El siguiente comando es para que **tache** (DELETE) una banda de números de línea. Su sintaxis es:

DELETE

```
DELETE <numero de linea>
```

Observarás el parámetro que define la banda a eliminar no es opcional, pero por lo demás acepta las formas de designar a la banda de manera idéntica al comando LIST. Dando por ejemplo el comando:

```
DELETE 100-200
```

se suprimirán del programa que hay en memoria las líneas con número del 100 al 200, ambas inclusive. Y lo puedes comprobar haciendo que lo liste después de darlo.

Ten cuidado al usar este comando. Si no has hecho que guarde el programa en cassette puede que accidentalmente pierdas un montón de trabajo. Donde es muy aprovechable es cuando usas algunas rutinas de un programa existente, ya que basta cargar el programa y tachar las líneas que no quieres.

El siguiente comando útil en este grupo es el de hacer que **renumere** (RENUM) el programa que hay en memoria. Ahora apreciarás la necesidad de usar la notación taquigráfica para explicar la sintaxis:

RENUM

```
RENUM [<numero de linea nuevo>]
  [, [<numero de linea antiguo>] [, <incremento>]]
```

Estos parámetros opcionales te permiten cambiar los números asociados a las líneas del programa corriente, para que comiencen a partir de la dada por el número designado como **viejo nurolin,** le den a ése el **nuevo nurolin** y a partir de él vaya en saltos dados por **incremento.** Si no especificas alguno de los argumentos, se tomarán los valores preceptuados.

Dichos valores son equivalentes a dar el comando:

```
RENUM 10,,10
```

Y el parámetro que falta, observa que se debe dar la coma separadora, para que la interpretación del comando sea la correcta; y en este caso se refiera al primer número de línea que haya en el programa.

Si todavía te queda algo del programa CASA después de haber practicado con los comandos anteriores, puedes ensayar con los diversos parámetros opcionales, dando simplemente RENUM por sí solo, de manera que aprecies los valores preceptuados. Observa que RENUM automáticamente actualiza las referencias numéricas de los saltos GOTO y desvíos GOSUB.

Finalmente, hay otro comando que te libera de la tediosa tarea de teclear el número de línea en cada instrucción. Con él (AUTO) consigues que automáticamente **numere** el propio BASIC, y se usa en la forma siguiente:

AUTO

```
AUTO [<numero de linea>][, <incremento>]
```

El parámetro **número de línea** es el número con el que deseas comenzar, y el parámetro **incremento** es el tamaño del salto entre dos líneas de programa consecutivas. Así por ejemplo, si mandas:

```
AUTO 100,5
```

el CPC464 escribirá el 100 al comienzo de la siguiente línea, dejará un espacio en blanco y situará el cursor en la posición adecuada para que puedas teclear la línea que desees. Cuando concluyas la línea pulsando la tecla ENTER volverá a hacer lo mismo, exceptuando que el número de línea será ahora 105, como corresponde.

La numeración automática se termina pulsando la tecla ESCape.

Si hubiera ya en memoria una línea con el mismo número que el recién generado, se expondría un asterisco (*) como advertencia. Si no se toma ninguna otra acción y simplemente se pulsa la tecla ENTER, esa vieja línea se mantendría en memoria, en los demás casos sería sustituida por la tecleada nuevamente.

VARIABLES PORQUE VARIAN

La manera en que las variables se describieron en la Parte 1 - lugares de la memoria especialmente rotulados y adecuadamente preparados según sean numerales o literales- es la parte esencial de la historia. Realmente en el caso de las variables numerales, hay dos categorías: **reales** y **enteras**, pero además, tanto las variables numerales como las literales pueden formar **colectivos** de datos, con un nombre identificativo general y con el uso de **subíndices** para indicar cada uno de los elementos individuales de ese colectivo.

Variables numerales reales

Los números reales son aquéllos que pueden tener una parte fraccionaria, o pueden incluso ser únicamente fracciones (valores menores de uno). Uno de los adjetivos para ellos es el de **coma flotante** dado que el ¡punto! que separa la parte entera de la fraccionaria no está en una posición **prefijada**. Los siguientes son constantes numéricas reales:

```
1234862.01
0.000000154768119
2.3
```

Cuando especificas el valor de una variable, bien sea mediante una instrucción de asignación LET o por estar implicada en una expresión, está preceptuado que se adopte como real si no se indica lo contrario. Para mayor claridad en tus programas, puedes indicar explícitamente que son reales si usas en el nombre de la variable el sufijo admiración (!), como por ejemplo en:

```
LET finallong!=medida*coef*0.0001
```

Variables numerales enteras

Si quieres trabajar con valores numéricos que sean únicamente enteros, i.e. que no puedan tener parte fraccionaria, es necesario que especifiques el nombre de la variable añadiéndole el sufijo **porcentaje** (%), como por ejemplo en:

```
LET porcien%=100*benficio/coste
```

La variable **porcient%** sólo podrá contener valores enteros, y cualquier parte fraccionaria de cualquier valor que intentara asignarse a esa variable sería automáticamente redondeado al valor entero más cercano.

Variables subindicadas

Las variables subindicadas, forman lo que se llama **variable múltiple**, y también se conocen como tablas, ristras, ringlas, matrices, etc., y reflejan siempre una colección de variables **homogéneas** y cada una identificable por su **refrencia.** Los elementos de la tabla pueden ser datos numerales reales, enteros o datos literales, y cada uno de ellos se describe mediante:

```
<nombre variable> (<listado:<subindices>)
```

donde los **subíndices** son expresiones enteras, separadas por comas. De esta manera se pueden conservar datos de una colección bajo un nombre común, e identificarlos mediante diferentes valores de los subíndices.

Toma por ejemplo los primeros seis equipos de fútbol en el campeonato:

Equipo	**Puntos**
At.Madrid	69
Coruña	67
Palencia	60
R.Vallecano	57
Valencia	56
B. Linense	55

Supongamos ahora que queremos reflejar esta colección de datos de manera que pueda ser actualizada semana a semana, y luego sacar la relación de acuerdo con la puntuación alcanzada. Si numeramos las posiciones en la liga desde **cero** hasta **cinco**, los puntos de cada equipo de acuerdo con la posición, pueden reflejarse mediante una variable denominada **punto** y añadirle luego entre paréntesis el valor de **n** si representamos con ese subíndice la posición en la liga. Tendríamos:

Variable	**Contenido**
punto(0)	69
punto(1)	67
punto(2)	60
punto(3)	57
punto(4)	56
punto(5)	55

Una tabla de datos como la anterior se suele denominar **ringla,** para destacar la diferencia con las tablas normales en las cuales los datos no tienen por qué ser todos exactamente **del mismo género.**

(En inglés se usa el vocablo **array** y en español también el de **arreglo** y el de **ristra,** para indicar que los elementos de la tabla siguen una determinada **regla.** Estrictamente hablando, esa variable múltiple sería **mono-dimensional**).

Probablemente también desearías tener otra tabla literal para reflejar los nombres de los equipos en cada posición de la liga.

CASAS

Habiendo estropeado el programa en la primera parte de este capítulo, sería aconsejable volver a rebobinar el cassette y cargar de nuevo el programa en memoria. La intención ahora es mostrar un programa que puede mejorarse progresivamente, y así nuestro programa original CASA habrá sufrido diversas reconstrucciones para convertirlo en un castillo o mansión de lujo.

Así que, habiendo cargado el programa haz que se ejecute. Parece exactamente el mismo que el conocido con sus ventanas y su valla, ¿no es así? Sin embargo, si observas el listado que te presentamos, verás que el programa ha sido completamente re-escrito.

```
100 ' Casa
110 ' (MANSION mejorada)
120 '
130 ' DA 17/9/84
140 '
150 MODE 0 : BORDER 12
160 INK 0,12 : INK 1,3 : INK 2,6 : INK 3,17
170 PAPER 0
180 '
190 ' Bordes
200 '
210 READ x,y
220 IF y=-1 THEN 290
230 IF x=0 THEN colour=y : GOTO 210
240 IF x<0 THEN MOVE -x,-y ELSE DRAW x,y,colour
250 GOTO 210
260 '
270 ' Cerca
280 '
290 FOR F=0 TO 620 STEP 20
300 MOVE F,0:DRAW F,60
310 NEXT F
```

```
320 MOVE 0,45:DRAW 620,45
330 '
340 ' Marcos ventanas
350 '
360 size=18
370 FOR i=1 TO 16
380 READ x,y:MOVE x,y
390 DRAWR 0,size : DRAWR size,0
400 DRAWR 0,-size: DRAWR -size,0
410 NEXT
420 '
430 END
440 '
450 ' Data para orillas
460 ' por pares : negative = move
470 '           : 0,x      = colour change
480 '           : 0,-1     = end
490 '
500 DATA    0,   1
510 DATA -100,-50,  100,250, 400,250, 400,50,  100,50
520 DATA -400,-250, 600,250, 600,50,  400,50,  400,250
530 DATA 500,350,   600,250, 400,250
540 DATA -100,-250, 200,350, 500,350
550 '
560 ' Puerta
570 '
580 DATA 0,2
590 DATA -225,-50, 225,140, 275,140, 275,50
600 '
610 ' Ventanas grandes
620 '
630 DATA 0,3
640 DATA -120,-70,  120,130, 180,130, 180,70,  120,70
650 DATA -120,-170, 120,230, 180,230, 180,170, 120,170
660 DATA -320,-170, 320,230, 380,230, 380,170, 320,170
670 DATA -320,-70,  320,130, 380,130, 380,70,  320,70
680 DATA 0,-1
690 '
700 ' Data para ventanas pequenas
710 '
720 DATA 130,78,  156,78,  130,103, 156,103, 130,178
730 DATA 156,178, 130,203, 156,203, 330,78,  356,78
740 DATA 330,103, 356,103, 330,178, 356,178, 330,203,
    356,203
```

La primera cosa que observarás es el uso del **apóstrofe** (') en lugar de la palabra clave REM. Opera exactamente de la misma manera -todo lo que va detrás del apóstrofe es únicamente un **memo**randum. Como puedes ver, la ventaja es que es en la presentación del listado que es mucho más claro y fácil de leer, con líneas postizas entre explicaciones y notas.

Y hay otras dos nuevas e importantísimas instrucciones además, para reflejar **listas de datos** y hacer que **tome** sucesivamente los datos para apuntarlos como valores de las variables. Son las que tienen como palabras clave DATA (datos) y READ (leer). En nuestro programa de la Parte 1 había unos cuantos comandos sucesivos MOVE para **situar** el cursor de gráficos en el lugar correcto antes de hacer que se desviara a una subrutina. Una manera más pulcra de conseguir ese mismo efecto es colocar todas las coordenadas para situación del cursor en un mismo lugar y luego tomarlas como valores de las variables que usaremos en los comandos MOVE y DRAW. Eso es exactamente lo que la pareja de instrucciones READ y DATA te permite hacer.

La sintaxis para reflejar una lista de datos, es:

DATA

```
DATA <listado:<constante>
```

Se pueden usar constantes numerales o literales, separadas por comas unas de otras. De hecho, casi todo es aceptable dado que una línea de texto con ninguna coma dentro de ella, sería tomada como una mera constante literal.

La sintaxis del comando para hacer que **tome** un dato de la lista y lo 'apunte' como valor de una variable es:

READ

```
READ <listado:<variable>
```

Y cada **variable** puede ser literal o numeral, en exacta **correlación** con la lista de **constantes;** y al igual que ellas, deben estar separadas por comas.

El siguiente programa ilustrará cómo estos dos comandos trabajan 'sintonizadamente':

```
10 FOR x=1 TO 4
20 READ N$
30 PRINT N$
40 NEXT
50 DATA Juan,Bruno,Borja,Julio
```

Cada vez que el programa ejecuta un comando READ, la siguiente constante en la instrucción DATA es colocada como valor de la variable N$. Así, cuando x=1 se **tomará** "Juan" como valor de N$; cuando x=2 se tomará "Bruno"; cuando x=3 se tomará "Borja" y cuando x=4 se tomará "Julio".

Te puedes estar preguntando si hay alguna otra manera de evitar tener que escribir las largas instrucciones DATA una y otra vez, cuando necesitas exactamente que los mismos valores sean leídos más de una vez. La palabra clave que puede ayudarte en eso corresponde a la acción de cambiar la dirección señalada por el **puntero** interno que lleva la cuenta de las variables tomadas, para hacer que **repunte** (RESTORE) a un determinado número de línea. Su sintaxis es:

RESTORE

```
RESTORE [<numero de linea>]
```

Para comprender cómo funciona el parámetro **número de línea,** en esta instrucción hay que analizar lo que sucede cada vez que el CPC464 ejecuta un comando READ. No le importa cuántas instrucciones DATA haya, ni tampoco dónde están dentro del programa. Cada uno de los elementos es **tomado** uno a uno en el orden en que aparecen y cuando se encuentra la instrucción READ. Después de que se ha ejecutado una instrucción READ, el CPC464 señala que la ha **tomado** y corre el **puntero** interno que le sirve para indicar la siguiente constante que va a ser usada cuando se encuentre con la siguiente instrucción READ.

El parámetro opcional de la instrucción RESTORE ('repunte') permite que cambies la dirección señalada por ese puntero hacia la primera constante de la instrucción DATA que haya en el programa detrás o exactamente en **número de línea.** De esta manera, puedes volver a utilizar o puedes elegir dinámicamente durante la ejecución de un programa, la lista de datos sobre la que quieres trabajar. Omitiendo este parámetro hará que el puntero mencionado señale a la **primera** constante de la **primera** instrucción DATA que haya en el programa.

TABLAS (ARRAYS)

Como prometimos, vamos a echar una mirada a lo que conocemos como **tablas** (e insistimos que en informática han de ser elementos **homogéneos,** por lo que se prefiere hablar de ristras, ringlas, arreglos, 'arrays',...). Anteriormente vimos cómo podemos confeccionar tablas usando **variables subindicadas.** Obtenemos así el efecto de un almacén distribuido en compartimentos y con un nombre común.

Así, en una tabla mono-dimensional, llamada **chisme** y designando a cada uno de sus elementos como **chisme(a)** obtendríamos lo que está alojado en ese compartimento concreto. Haciendo que exprese **chisme(3)** obtendríamos lo que hubiera almacenado en el compartimento 3. Podemos subdividir aún más nuestro almacén colectivo haciando una tabla bi-dimensional en que cada cajón de cada compartimento estaría designado por **chisme(b,a)** donde **a** es el número identificativo de nuestros compartimentos originales, y **b** el número del cajoncito dentro de ese compartimento. Así, sacando **chisme(15,3)** obtendríamos lo que hubiera almacenado en el cajón 15 del compartimento 3.

Y antes de que me hagas la pregunta, la respuesta es que sí: efectivamente **puedes** tener tablas **tri-dimensionales**, pero no te vamos a fastidiar con eso en este libro. De hecho, el CPC464 te permite usar tablas (y ¡ojo! son ringlas porque sus elementos han de ser homogéneos) de tantas dimensiones como desees (dependiendo de la longitud de la línea y de la memoria disponible) como puedes ver en nuestros ejemplos.

Así que, ¿cómo usamos una tabla? Bueno, teclea lo siguiente y lo verás:

```
10 chismes(5)=47
20 for n=0 to 20
30 print chismes(n);
40 next
```

Observa que cada uno de los elementos de la tabla va designado por un **subíndice**, y que la numeración comienza a partir de **cero**. La línea 10 asigna el valor 47 al elemento 5 de la tabla chisme, i.e. el sexto compartimento en nuestro almacén; y las líneas restantes solamente las hemos incluido para exponer el contenido de toda la tabla. Lo que consigues en pantalla es:

```
 0  0  0  0  0  47  0  0  0  0  0
Subscript out of range in 30
```

Y ese mensaje de error que te dice "subíndice fuera de rango en 30" es porque hay un nuevo comando que tenemos que aprender y que te presentaremos dentro de un minuto. Ahora, lo que puedes ver es que el CPC464 establece como valor inicial de toda variable subindicada numérica el valor de cero, así que los primeros cinco ceros expuestos representan el contenido de los elementos 0, 1, 2, 3 y 4 de la tabla, y los últimos cinco ceros los valores de los elementos del 6 al 10. Pero ¿qué le ha sucedido al elemento 11 y a los siguientes?

Pues verás, el CPC464 a no ser que **explícitamente** le digas lo contrario, supone que cualquier tabla que menciones tiene **once** elementos o menos (que van del 0 al 10), lo que explica por qué te ha lanzado el mensaje de error.

Si tú deseas tablas con más de 11 elementos tendrás que decirle previamente que **ocupe** el espacio necesario en memoria para alojar esos elementos, y lo haces mediante el comando de clave DIM, abreviatura de 'dimensione' que determina el tamaño máximo de una tabla. Su sintaxis es como sigue:

```
DIM <listado:<variable subindicada>
```

DIM

donde **variable subindicada** es desde luego:

```
<nombre variable> (<listado:<expresion entera>)
```

donde cada **expresión entera** establece el máximo valor permitido para el subíndice correspondiente.

Podemos por tanto, quitarnos de encima el mensaje de error de dos maneras. La más simple es hacer que el bucle termine en 10 en lugar de en 20. Pero si realmente insistes en una tabla con 21 elementos, **debes** hacer que ocupe el espacio pertinente, mediante el comando:

```
5 DIM chismes (20)
```

Ensáyalo.

TIEMPO DE JUEGO

El siguiente programa, CIUDAD, te presenta no solamente un casa, sino una 'ristra' entera de casas.

CONTROL DEL ESTUDIO

Asegúrate que has comprendido completamente toda esta nueva e importantísima información, y aprovecha para ello la posibilidad de hacerte una autoevaluación mediante TAE2 antes de pasar al siguiente capítulo.

Capítulo 3
PINTANDO CON NUMEROS

PEN

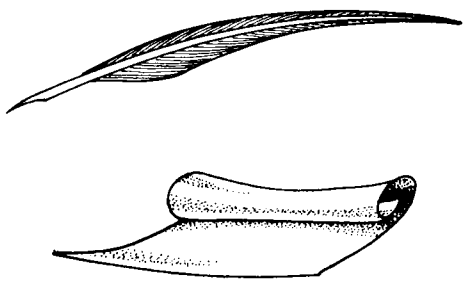
PAPER

INK

Si te has visto confundido por las relaciones entre **pluma, papel** y la **tinta** de ambos, continúa leyendo. Si no es así, léetelo de todas maneras, dado que es probablemente mucho más sutil de lo que crees.

El CPC464 puede mostrar 27 colores diferentes (véase tabla al dorso), que pueden ser colocados en 16 diferentes **tinteros.** Sin embargo, el número de tinteros disponibles para dibujar imágenes depende del MODE en que se esté trabajando.

En el **MODO 0** dispones de un surtido de 16 tinteros; en el **MODO 1** tienes 4, y en el **MODO 2** solamente dispones de 2 tinteros. Imagina ahora que siempre que mandas que se **exponga** texto estás usando una "plumilla" y una hoja de "papiro" que previamente han sido "tintadas" aprovechando uno de los tinteros disponibles.



Enciende el ordenador o fuerza un **restauro** (SHIFT/CTRL/ ESC); el ordenador se coloca en el **MODO 1** y puedes tener por tanto 4 tintes diferentes y simultáneamente en la pantalla. Ya se han rellenado los 16 tinteros con los colores preceptuados inicialmente. La pantalla es azul porque se ha prefijado para ella un PAPER cuyo color está preceptuado que sea el correspondiente a INK 0 (color 1). El texto está en amarillo porque la PEN con que se ha escrito, previamente se ha mojado en el valor preceptuado de INK 1 (color 24).

Número	Color	Número	Color
0	negro	14	azul pastel
1	azul	15	naranja
2	azul brillante	16	rosa
3	rojo	17	magenta pastel
4	magenta	18	verde brillante
5	malva	19	verde marino
6	rojo brillante	20	ciano brillante
7	púrpura	21	verde lima
8	magenta brillante	22	verde pastel
9	verde	23	ciano pastel
10	ciano	24	amarillo brillante
11	azul cielo	25	amarillo pastel
12	amarillo	26	blanco brillante
13	blanco		

Teclea ahora PEN 2. Ahora el nuevo texto se escribe en ciano brillante (color 20). Teclea PEN 3 y el nuevo texto se pone en rojo (color 6). Teclea PEN 4 y sin embargo, verás que el nuevo texto es azul, lo mismo que el papel, y entonces... Eso es lo mismo que teclear PEN 0 dado que aunque hay 16 tinteros diferentes, sólo contienen 4 colores de tinta cuando se trabaja en este **MODO** 1. De hecho, la pluma realmente se ha tintado con INK 0. En este modo, sucede que el CPC464 simplemente rehusa emplear tintas fuera de la banda 0 a 3: ha convertido tu solicitud de INK 4, que está fuera de esa banda, en la solicitud INK 0, que sí está dentro de la banda. Igualmente, INK 5 se convertiría en INK 1, INK 6 en INK 2, INK 7 en INK 3, INK 8 en INK 0 de nuevo, etc.

Todavía tenemos azul sobre azul, así que teclea PAPER 1 con cuidado, porque no ves ninguna de las equivocaciones que pudieras cometer. Si lo has hecho correctamente verás ahora la palabra 'Ready' en azul sobre un fondo amarillo. Eso es debido a que el CPC464 cuando expone un carácter en la pantalla, ocupa un **cuadratín** como fondo del carácter y con el espacio justo para ese carácter, con el valor correspondiente al **tinte** del **papel** que se eté usando. Sobre ese cuadratín es donde se escribe con el **tinte** de la **pluma** corriente, de manera que el amarillo que rodea a la palabra 'Ready' es el resultado de trazar un carácter sobre ese fondo recientemente definido.

Si hacemos que **limpie** la pantalla, tecleando CLS, conseguiremos que toda la pantalla (exceptuando el reborde o limbo de la imagen) aparezca con el **tinte** amarillo que acabamos de darle al **papel.**

Ahora pasemos a la parte de trucos. Digamos que no queremos amarillo, sino que queremos en su lugar blanco. Eso obliga a usar el comando de **tinta** en la forma:

```
INK 1,26
```

¡Oh, maravilla! Todo lo que estaba en amarillo, está ahora en blanco.

Esto es lo que ha sucedido. Para cambiar de amarillo a blanco teníamos que vaciar el tintero 1 (que contenía **tinte** amarillo) y volverlo a rellenar con blanco, que es el color 26. De manera que INK 1,26 significa 'cambia el **tinte** en el frasco 1 a otro de color 26'. Lo que no significa es 'cambia el color 1 al color 26'.

Recordarás que INK 4 estaba preceptuado inicialmente a blanco, incluso aunque estuviera fuera del surtido de colores del MODO 1. Ahora tenemos además tinte blanco en INK 1. Eso es perfectamente correcto -podemos tener el mismo color en tantos tinteros como deseemos.

Desde luego, el CPC464 está constantemente sacando la tinta de los tinteros para la pluma y el papel que usa en la pantalla (de hecho, 50 veces por segundo), de manera que todo lo que antes era amarillo se convierte ahora en blanco. Trata de cambiar el color de cualquier otro tintero, digamos de azul en naranja. El azul está en el tintero 0 y el naranja es el color 15, así que teclearemos:

```
INK 0,15
```

Y si ahora queremos que el tintero 0 vuelva a tener tinta azul, basta teclear:

```
INK 0,1
```

En los **MODOS** 1 y 2 puedes colocar cualquier número en la banda 0 a 15 como parámetro de los comandos PEN y PAPER, pero el CPC464 cambiará los valores que uses para que correspondan a las **INK** de número inferior, i.e. en MODE 2 cualquiera que sea la INK que intentes fijar para la PEN o el PAPER, sólo las **tintas** 0 y 1 están dentro de la gama permitida, y así sucesivamente. Sin embargo, eso no es cierto para el comando INK, que fija la **tinta** de cada tintero. Los colores de todos los 16 "tinteros" pueden verse afectados, aunque no apreciarás el efecto a no ser que cambies los modos.

Fuerza un **restauro** y luego teclea:

```
MODE 0:PEN 13
```

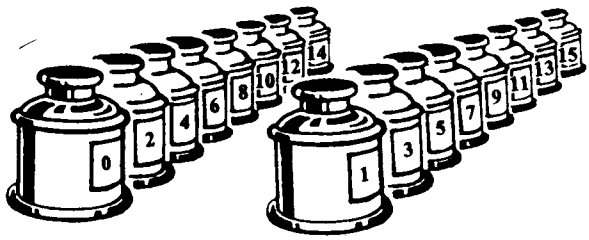
Observa que el color del texto es "verde pastel". Ahora teclea:

```
MODE 1
```

y luego teclea:

```
INK 13,26
```

Nada cambia, dado que INK 13 está muy fuera de la banda permitida en MODE 1. Ahora teclea de nuevo MODE 0:PEN 13. El texto es ahora blanco, y no verde pastel. Lo que te ha demostrado es que puedes cambiar el color de la tinta de un tintero incluso aunque ese color esté fuera de la banda permitida dentro del modo de pantalla corriente en cada momento.



DIBUJOS COLOREADOS

Aquí están las descripciones sintácticas de los comandos ya conocidos para que **pinte** motas y **trace** rayas:

PLOT

DRAW

```
PLOT <coordenada x>,<coordenada y> [,<ink>]
DRAW <coordenada x>,<coordenada y> [,<ink>]
```

COLORES DE TINTA PREFIJADOS

Tinta	Color
0	1
1	24
2	20
3	6
4	26
5	0
6	2
7	8
8	10
9	12
10	14
11	16
12	18
13	22
14	1,24
15	16,11

La **coordenada X** y la **coordenada Y** deben ser expresiones numerales. Si no se incluye el parámetro de **tinta,** la raya o la mota se trazará de acuerdo con la tinta que en ese momento corresponda a la pluma de gráficos.

Eso está preceptuado inicialmente a INK 1, pero cada vez que se da un comando que incluya el parámetro de **tinta,** la mota o raya se traza de acuerdo con esa nueva tinta y la pluma de gráficos queda a partir de ese momento tintada con ese color.

Observa que cambiando la tinta de la pluma para textos no afecta a la tinta de la pluma con que haces los gráficos, en los comandos PLOT y DRAW. Teclea:

```
DRAW 640,400
```

Debieras obtener una **raya** trazada a través de la pantalla con el mismo color que el texto. Teclea ahora:

```
MOVE 0,0:DRAW 640,400,3
```

La línea volverá a ser trazada, pero ahora usando INK 3.

Teclea el siguiente programa, después de haber forzado un restauro de las condiciones iniciales.

```
 10 CLS
 20 r=40:y=200:c=1
 30 FOR x=40 TO 600 STEP 40
 40 FOR i=-r TO r STEP 2
 50 h=SQR(r*r-i*i)
 60 PLOT x-h,i+y,c
 70 DRAW x+h,i+y
 80 NEXT i
 90 c=c+1
100 NEXT x
```

Teclea el comando MODE 0, y luego ejecuta ese programa. Verás una serie de discos de diversos colores trazados a través de la pantalla, cada uno parcialmente cubierto por el siguiente. No te preocupes sobre cómo hemos trazados los discos: son los colores los que nos conciernen ahora.

Recuerda primeramente que en el **MODE 0** tienes los 16 **tinteros** disponibles como máximo y rellenos con los colores que puedes ver. En segundo lugar, observa que cambias el color con el que trazas una **raya** si incluyes el número de **tinte** después de las coordenadas. Ahora observa el listado. En la línea 60 tenemos un tercer parámetro -la variable 'c'- que especifica la INK que ha de usarse. En la línea 90 vemos que 'c' se incrementa en uno cada vez que se completa una ronda del bucle controlado por 'i'. La variable 'c' tiene un valor inicial de uno, asignado en la línea 20. Prueba a ejecutar el programa en cada uno de los modos posibles, para confirmar todo lo que hemos estado comentando en este capítulo.

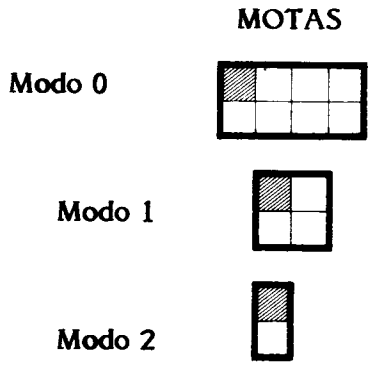
Finalmente, ejecuta el programa en el **MODO 0** y luego experimenta por ti mismo tecleando algunos comandos de **tinte** tales como INK 6,12 e INK 2,15.

ADORNO DE LAS VENTANAS

Una vez que hayas dominado el funcionamiento de la **pluma** (PEN), el **papel** (PAPER) y de los **tintes** (INKs) que puedes dar a cada uno de ellos, serás capaz de crear algunos efectos interesantes. Incluso pueden conseguirse imágenes en pantalla mucho más elaboradas, una vez que hayas aprendido unos cuantos detalles más sobre la habilidad del CPC464 -que es única en el mercado- para mezclar textos y gráficos.

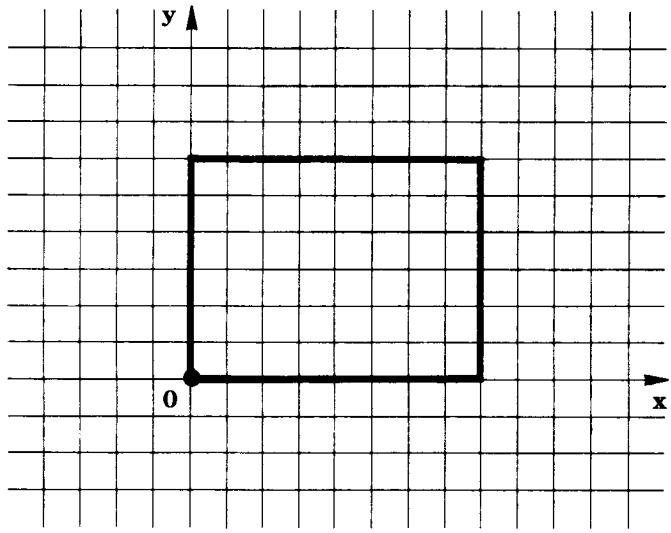
Primeramente, vamos a estudiar el comando que establece el **origen** de coordenadas. Aquéllos de vosotros que estéis familiarizados con las matemáticas sabréis que el origen de un gráfico es el punto donde se cruzan los ejes **horizontal** y **vertical**, y que por tanto tiene como coordenadas x=0, y=0.

La imagen del CPC464 está formada a partir de 256.000 **puntos** individuales. Hay 640 **puntos a lo ancho** de la pantalla (numerados del 0 al 639 desde la izquierda hasta la derecha) y hay 400 **puntos a lo "largo"** de la pantalla (numerados del 0 al 399 desde abajo hasta arriba). Esos **puntos** no son lo mismo que las **motas** ("pixels"), ya que es el mínimo elemento que se puede manejar para formar una imagen, están constituidas por un número diferente de puntos dependiendo del modo en que se trabaje. Cada **mota** (pixel=elemento pictórico) puede ser designado por las coordenadas de **cualquiera de los puntos** que lo componen.



Cada uno de esos puntos puede mencionarse en un comando de forma individual con el propósito de trazar rayas, etc. Haciendo que pinte el punto 0,0 haremos que dicho punto 0,0 **-y todos los otros puntos pertenecientes a la misma mota-** sea iluminado en pantalla con el color que corresponde a la **pluma** de gráficos.

Si al pintar una mota damos valores de las coordenadas x e y que caen fuera de los límites mencionados (x=0 a 639 e y=0 a 399) no provocaremos ningún error. Eso es debido a que el CPC464 recuerda tu posición cuando te sales de la pantalla y continúa ejecutando los comandos PLOT y DRAW, incluso aunque tú no puedas ver su efecto en la pantalla.



ORIGIN

El comando de **sitúe origen** (ORIGIN) te permite trasladar los ejes de coordenadas a cualquier parte que desees de la pantalla, o ¡incluso fuera de ella! Ser capaz de hacer eso es muy conveniente cuando se está dibujando.

Fuerza un **restauro**, y luego teclea el siguiente programa:

```
 5 CLG
10 PLOT 0,0
20 DRAW 200,200
30 DRAW 400,200
40 DRAW 200,350
50 DRAW -100,350
60 DRAW -100,100
70 DRAW 0,100
80 DRAW 0,0
```

Ejecuta el programa y verás cómo se dibuja un polígono irregular que no tiene nada especial, salvo que parte de él no puede verse porque se sale por el borde izquierdo de la pantalla. Teclea ahora el comando ORIGIN 150,0 y vuelve a pasar el programa. ¡Es inteligente! Claro, justamente hemos mandado al CPC464 que considere el punto de coordenadas 150,0 en relación con la esquina **inferior izquierda** de la pantalla, como nuevo origen de coordenadas para ejecutar todos los comandos PLOT y DRAW de acuerdo con las coordenadas en relación a ese nuevo origen. Así la esquina inferior izquierda de la pantalla tiene ahora por coordenadas -150,0.

No te confundas sobre lo que está sucediendo. Imagínate una hoja de papel sobre un tablero de dibujo: puedes dibujar todo lo que quieras sobre la totalidad del tablero, pero la hoja de papel sólo registra lo que realmente pasa por encima de ella. El resto pudiera muy bien no haber sucedido nunca.

Ensaya y experimenta colocando diferentes valores para las coordenadas del origen tales como -50,0 o 200,-50. Una vez que hayas hecho eso, añade algunas líneas más a tu programa:

```
 5 FOR o=0 TO 100 STEP 10
 6 ORIGIN 100+o,o
90 NEXT o
```

y ponlo en marcha, comprobando lo que sucede.

Ya te habrás tropezado con el comando **limpie pantalla de gráficos** de clave CLG, que tiene un efecto similar al de **limpie pantalla**, de clave CLS. Aquí está la sintaxis:

CLG

```
CLG [<ink>]
```

CLG hace que se **aclare** la ventana de gráficos y mueve el cursor de gráficos a la posición correspondiente a las coordenadas 0,0, donde quiera que estén según el origen dado. El parámetro opcional de **tinte** rellenará la ventana de gráficos con la tinta especificada. Pero espera un minuto, ¿me puedes decir qué es eso de la ventana de gráficos? Bueno, es toda la pantalla hasta que le digas al CPC464 explícitamente otra cosa. Y lo puedes hacer incluyendo los parámetros pertinentes dentro del comando ORIGIN. Teclea por ejemplo:

```
ORIGIN 100,0,0,300,200,0
```

Eso le dice al CPC464 que mueva el **origen** a las coordenadas absolutas 100,0, y luego que limite la ventana de gráficos en **anchura** a todos los puntos con coordenadas 0 a 300, y en **altura** a los **puntos** desde el 0 al 200.

Haciendo esto, recuadramos toda la pantalla excepto el 'cuartel' situado en la esquina inferior izquierda de la pantalla.

Ya es hora de que miremos la sintaxis del comando para fijar el origen de coordenadas:

```
ORIGIN <x>,<y>[, <izq>, <derc>, <arb>, <abj>]
```

Todos los parámetros de este comando corresponden a coordenadas de pantalla **absolutas** y deben ser expresiones numéricas acordes. Las coordenadas de pantalla **absolutas** son similares a las coordenadas especificadas en los comandos DRAW o PLOT, exceptuando que siempre son posiciones con respecto a la esquina inferior izquierda real de la pantalla. No importa si se ha dado o no previamente un comando ORIGIN que haya desplazado el punto tomado como coordenadas 0,0; el siguiente comando ORIGIN siempre vuelve a ser con respecto a la esquina inferior izquierda. En otras palabras, el nuevo **origen** o la nueva **ventana NO** son coordenadas relativas a la última fijación de la coordenada 0,0; sino que son coordenadas absolutas, tomadas con respecto a la esquina inferior izquierda de la pantalla.

Teclea CLS para que se limpie la pantalla y luego RUN para que se ejecute el programa. Verás que la imagen gráfica está ahora limitada al cuarto inferior izquierdo de la pantalla. Lista el programa dos o tres veces y luego tecleas CLG para que se limpie dicho cuarto de pantalla para gráficos. Luego teclea lo siguiente:

```
CLS: PAPER 3: CLS
```

Deberás obtener una pantalla en rojo; y si ahora tecleas CLG observarás que la ventana de gráficos es azul. Ahora vuelve a ejecutar el programa para completar el efecto.

Ensaya valores diferentes para el origen y para la ventana, e.g.

```
ORIGIN 100,0,301,600,400,201
```

trasladará el recuadro para gráficos a la parte superior derecha de la pantalla. Para regresar al modo normal basta teclear:

```
ORIGIN 0,0,0,639,399,0
```

PALABRAS Y FIGURAS

El siguiente listado es del programa en la datocassette A denominado SENCOS que es el que vamos a usar para ilustrar algunas de las nuevas palabras clave y presentarte otros temas.

```
10 '
20 ' Graficas de Seno y Coseno
30 '
40 ' By Ian Padwick
50 ' Amended by DA 21/9/84
60 '
70 ' Montar pantalla
80 '
90 MODE 1:BORDER 20
100 PRINT CHR$(23);CHR$(0); ' graficas normales
110 INK 0,23:INK 1,20:INK 2,0:INK 3,26
120 ORIGIN 144,200,108,584,310,56 : CLG 2
130 DEG ' uso grados en sen/cos
140 PEN 2:LOCATE 4,4
150 PRINT' Graficas de las funciones Seno y Coseno'
160 '
170 PLOT 0,-100,3 : DRAW 0,100
180 PLOT 360,-100 : DRAW 360,100
190 PLOT 0,0      : DRAW 360,0
200 PLOT 90,7     : DRAWR 0,-14
210 PLOT 180,7    : DRAWR 0,-14
220 PLOT 270,7    : DRAWR 0,-14
230 TAG
240 MOVE -20,104  : PRINT '1';
250 MOVE -20,4    : PRINT '0';
260 MOVE -36,-94  : PRINT '-1';
270 MOVE -4,-110  : PRINT '0';
280 MOVE 78,-110  : PRINT '90';
290 MOVE 158,-110 : PRINT '180';
300 MOVE 248,-110 : PRINT '270';
310 MOVE 338,-110 : PRINT '360';
320 MOVE 50,-128  : PRINT ' Angulo en grados';
330 '
340 ' Dibujar curva seno
350 '
360 FOR x=0 TO 360
370 PLOT x,100*SIN(x),1
380 NEXT
390 MOVE 132,90
400 PRINT'SENO';
```

```
410 '
420 ' Dibujar curva coseno
430 '
440 FOR x=0 TO 360
450 PLOT x,100*COS(x),3
460 NEXT
470 MOVE 40,-80:PRINT'COSENO';
480 '
490 ' Arreglar
500 '
510 TAGOFF : LOCATE 1,1
520 WHILE INKEY$="" : WEND
530 END
```

Después de observar el funcionamiento del programa, estudia el listado anterior. Primeramente, en la línea 180 el reborde se prefija al color 20. En la línea 200 se sitúa el origen de manera que se prefija una ventana para gráficos en el centro de la pantalla. Nuestro nuevo origen para los comandos gráficos es el punto absoluto 144,200 y la ventana está limitada a todos los puntos dentro del recuadro formado horizontalmente desde el 108 al 534 y verticalmente desde el 56 al 310.

En la línea 10 decimos al CPC464 que todos los cálculos trigonométricos sean usando **grados** (DEGrees) en lugar de **radianes** (véase Capítulo 6). Y luego, en las líneas 250 a 300 le mandamos que trace los ejes que usaremos en los gráficos. La palabra clave en la línea 310, TAG (Text At Graphics) nos permite hacer algunas cosas interesantes con **texto en gráficos.** Cuando expones normalmente el texto, los caracteres ocupan los **cuadratínes** que forman la gradilla de pantalla, y a los que podemos referirnos usando el comando LOCATE para situar el carácter en un cuadratín determinado. El cursor de texto señala hacia ese cuadratín donde aparecerá expuesto el siguiente carácter de texto.

TAG

TAG corresponde a la abreviatura de **texto en cursor de gráficos** y cuando se ha dado previamente ese comando, todo el texto que se exponga en pantalla mediante PRINT se comenzará a escribir a partir de la posición del cursor de gráficos, y no a partir de la posición del cursor de textos. Dado que ahora sabemos que el cursor de gráficos puede estar en cualquier parte dentro o fuera de la pantalla, eso permite una enorme flexibilidad para exponer textos. En este ejercicio, vamos a usar el comando TAG para rotular los ejes y las curvas de los gráficos.

La razón de que el CPC464 tenga un comando especial para conseguir este efecto, es que sería mucho más lento que tuviera que trazar un carácter en la pantalla si cae a medias entre cuadratínes consecutivos.

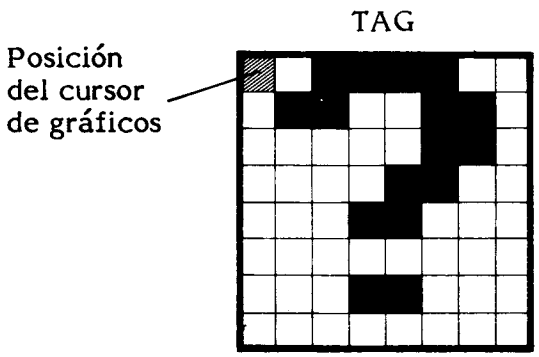
El cursor de gráficos se desplaza por la pantalla mediante los comandos MOVE o MOVER. Lista de la 310 a la 400 y verás cómo efectuamos la exposición de texto en pantalla. Observa que el punto-y-coma después de cada pareja de comillas se usa para suprimir un par de flechas que en caso contrario saldrían expuestas en pantalla.

Teclea lo siguiente:

```
CLS:TAG:MOVE 50,50:PRINT"HOLA"
```

y verás las flechas mencionadas. Corresponden a las marcas de retorno de carro y avance de línea; y si no estuvieramos usando el comando TAG, harían que el cursor se moviera hasta el comienzo de la siguiente línea. Pero dado que estamos usando TAG no tienen el efecto normal, sino que son representadas en pantalla.

Para cada carácter el cursor de gráficos toma el punto superior izquierdo del cuadratín de 8 x 8 motas que ocupa, y se desplaza 8 motas hacia la derecha para exponer el siguiente carácter del texto. Observa que si ahora tecleas PRINT"LOLO", los caracteres vuelven a aparecer expuestos una vez más en la posición señalada por el cursor de textos. Eso indica que la acción del comando TAG queda cancelada inmediatamente después de haber usado la tecla ENTER; y esa es la razón de por qué los comandos anteriores los tecleamos como una sola línea. Teclea ahora TAG:PRINT"LOLO", y observarás otra vez que los caracteres se exponen en la posición señalada ahora por el cursor de gráficos (que se desplazó en la acción primera de exponer "HOLA").



Observa también que la tinta usada para el texto expuesto comandado por TAG, es la de gráficos y no la tinta de la pluma usada para textos. Para cambiar la tinta tendrías que haber usado el tercer parámetro de los comandos PLOT o DRAW, dado que no existe ningún comando similar a PEN para cambiar directamente la pluma de gráficos. Igualmente, el color para el fondo de los caracteres es el que corresponde a la ventana de gráficos, y no el que correspondería al PAPER de texto.

En las líneas 440 a 550 se llevan a cabo los cálculos y el trazado de los gráficos (que puede o no serte familiar). En el Capítulo 6, la utilización de las funciones SIN y COS se explicará completamente y haremos mención a este programa. En la línea 590 aparece la palabra clave TAGOFF que exactamente **quita** lo hecho por TAG -hace que la exposición de texto vuelva a corresponder con el cursor de texto. Observa que cuando está puesto TAG, el comando LOCATE todavía sitúa el cursor de texto en la parte de la pantalla que se le mande, incluso aunque no suceda nada en esa posición hasta que lances un comando TAGOFF.

TAGOFF

GRAFICOS DE PASTEL

El siguiente programa en la datocassette A es el popular **gráficos de 'pastel'.** Es un ejemplo adicional de un programa que usa tanto textos como "grafos" sobre la misma pantalla.

Estos diagramas son incluso más populares con los directivos comerciales, investigadores de mercado y sociólogos, que los **diagramas de barras.** La razón es que pueden mostrar los porcentajes como una porción del pastel total, en lugar de hacerlo en relación de uno con otro. Este programa sólo te pide que teclees los números que desees ver representados de esta manera, ¡e incluso calcula los porcentajes! Ensaya con él antes de estudiar el listado pertinente.

```
100 ' Graficas de pastel
110 '
120 ' DA 30/9/84
130 '
140 ' Pantalla
150 '
160 CLEAR
170 DIM name$(9),amount(9) ' opcional
180 MODE 2   : BORDER 19
190 INK 0,23 : INK 1,0
200 PAPER 0  : PEN 1
```

```
210 PRINT CHR$(23);CHR$(0);
220 '
230 GOSUB 500 ' encabezado
240 '
250 q$="Y"
260 WHILE UPPER$(q$)="Y"
270 '
280 GOSUB 650 ' obtener data
290 CLS
300 GOSUB 930 ' Dibujar pastel
310 '
320 LOCATE 1,19
330 INPUT "Otra vez  (S/N) ",q$
340 WEND
350 END
360 '
370 ' SUBRUTINA : Dibujar un circulo
380 '     Centre : x,y
390 '     Radius : r
400 '
410 MOVE x,y+r
420 s=0.2 : 'Finura del dibujo PLOT
430 FOR i=0 TO 2*PI+s STEP s
440 DRAW x+SIN(i)*r,y+COS(i)*r,1
450 NEXT
460 RETURN
470 '
480 ' SUBRUTINA : Imprimir pag. encabez.
490 '
500 CLS
510 LOCATE 32,2 : PRINT "Graficas de Pastel"
520 MOVE 236,360:DRAW 400,360:DRAW 400,390:DRAW 236,390
:DRAW 236,360
530 WINDOW 5,80,5,25
540 PRINT "    Este programa muestra una subrutina senc
illa que dibuja graficas de"
550 PRINT "    pastel partiendo de un determinado conju
nto de datos. Estos deberas"
560 PRINT "  sumnistrarlos en forma de lista dandole, s
i lo deseas, un nombre a cada"
570 PRINT "  asiento. El programa aceptara solamente di
ez de estos asientos, aunque"
580 PRINT "   la rutina trabajara con cuantos asientos
tu quieras, siempre que se"
585 PRINT TAB(18);" modifique la entrada del programa."
590 LOCATE 23,9:PRINT"Pulsa ESPACIO para continuar"
600 WHILE INKEY$<>" " : WEND
610 RETURN
```

```
620 '
630 ' SUBRUTINA : Obtener datos 10 art.
640 '
650 CLS
660 PRINT"       Por favor, escribe la cantidad y la de
scripcion de cada cosa."
670 PRINT"La lista terminara despues de diez asientos o
 si da una cantidad negativa."
680 PRINT
690 LOCATE 18,4 : PRINT " Cantidad" : LOCATE 35,4 : PRI
NT "Descripcion"
700 entry=0:count=0
710 WHILE ( entry>=0 )  AND ( count<10 )
720 LOCATE   9,5+count : PRINT "Art.";count+1
730 LOCATE  19,5+count : INPUT "",entry : amount(count)
=entry
740 LOCATE  35,5+count : INPUT "",name$(count)
750 count = count + 1
760 WEND
770 IF entry>=0 THEN count=count+1
780 count = count - 2 ' now=no de arti. validos
790 sum=0
800 FOR i=0 TO count
810 sum=sum+amount(i)
820 NEXT i
830 PRINT : PRINT TAB(13);"Gracias. Ahora pulsa ESPACIO
 para que veas la grafica."
840 WHILE INKEY$ <> " " : WEND
850 RETURN
860 '
870 ' SUBRUTINA : Dibujar el  pastel
880 ' Entrada variables : x,y,r - centro y radio del ci
rculo
890 '                   : contar - numero de art.
900 '                   : amount(0) a amount(count) - art
s.
910 '                   :
920 '
930 x=320 : y=180 : r=130
940 GOSUB 410 ' Dibujar circulo
950 '
960 p = 0 : oldp = 0
970 '
980 FOR i=0 TO count ' bucle principal
990 cum=0 ' acumulador ceros
1000 '
1010 ' Contar hasta barra actual
1020 '
```

```
1030 FOR j=0 TO i : cum=cum+amount(j) : NEXT
1040 '
1050 oldp = p ' guardar ultima barra
1060 p = ( cum/sum )*2*PI  ' calcular siguiente
1070 MOVE x,y ' centro del circulo
1080 DRAWR SIN(p)*r,COS(p)*r,1 ' dibujar hasta el borde
1090 GOSUB 1160 ' imprimir descripcion
1100 NEXT i
1110 RETURN
1130 '
1140 ' SUBRUTINA : imprimir descripcion pastel
1150 '
1160 middle=(p+oldp)/2  ' mitad del segmento
1170 outer=r+25         ' circulo exterior
1180 MOVE x+SIN(middle)*outer,y+COS(middle)*outer
1190 wid = LEN(name$(i))
1200 IF middle > PI THEN MOVER -8*wid,0 'si a la izq. m
ueva izq.
1210 TAG : PRINT name$(i); : TAGOFF
1220 RETURN
```

CONTROL DEL ESTUDIO

Cuando compruebes lo aprendido aprovechando el control TAE3, sigue con el libro a mano para que puedas examinar las respuestas. No estás ante un exámen oficial -estos controles sólo implican una ayuda para que puedas autoevaluarte y confirmar lo que has comprendido bien, antes de pasar al siguiente capítulo.

Capítulo 4
EL TIEMPO EN TUS MANOS

BUCLES DENTRO DE BUCLES

En la Parte 1 se presentaron las palabras clave FOR, NEXT y STEP para mostrar cómo una ristra de postes pueden trazarse en la pantalla simulando una valla. Para ahorrarte la consulta, la rutina que llevaba a cabo esa tarea era:

FOR-NEXT

```
650 FOR f=0 TO 620 STEP 20
660 MOVE f,0:DRAW f,60,15
670 NEXT f
```

Tecléala ahora si te apetece para comprobarla. Ya se explicó que el CPC464 asigna valores sucesivos al contador 'f', incrementado en el valor dado como parámetro STEP, cada vez que se tropieza con la instrucción NEXT; comenzando con el 0 y finalizando con el 620. Así, el primer valor de 'f' es el 0, el **otro** es el 20, el **otro** el 40, y así sucesivamente. Debiera mencionarse que el valor que marca el final de las rondas del bucle, no es necesariamente el último valor que ha adoptado 'f'; eso depende del **paso** especificado en STEP. Por ejemplo, si tecleas como nueva línea 650, la siguiente:

```
650 FOR f=0 TO 635 STEP 20
```

el último valor de 'f' será el de 620 porque el **otro** valor que adoptaría al encontrarse con la instrucción NEXT sería el de 640, y eso ya se sale fuera del intervalo permitido. El paso especificado en STEP puede virtualmente ser cualquier cantidad incluyendo fracciones y números negativos, por ejemplo:

```
FOR f=0 TO 10 STEP 0.15
```

daría sucesivamente a 'f' los valores 0, 0.15, 0.3, 0.45, 0.6, etc. hasta 9.9; y observa que el 10 no es el último valor.

```
FOR f=10 TO -7 STEP -2
```

daría sucesivamente a 'f' los valores 10, 8, 6, 4, 2, 0, -2, -4 y -6, y también aquí observa que -7 que marca el final de la banda de valores, no es el último valor para 'f'.

En el último capítulo, el programa que trazaba una ristra de discos a través de la pantalla tenía dos bucles **preconfinados** FOR-NEXT, uno **dentro** del otro. El bucle interno (variable contadora i) dibujaba los discos, mientras el bucle externo (variable contadora x) marcaba la posición del centro de cada disco.

Estos bucles preconfinados FOR-NEXT pueden usarse de tres maneras:

- ☐ Como en el ejemplo anterior, si queremos una sucesión concreta de valores para la variable que cuenta las rondas del bucle, y que va a ser usada para exponer o pintar o recorrer una tabla.
- ☐ Si queremos repetir una serie de comandos un número preconfinado -conocido de antemano- de veces.
- ☐ Si queremos introducir en el programa una **demora** para disminuir la velocidad a la que suceden los eventos.

El siguiente programa ilustra estos tres principios:

```
10 FOR 1=1 TO 3
20 FOR n=10 TO 100 STEP 5
30 PLOT 0,n:DRAWR 100,100
40 FOR d=1 to 500:NEXT d
50 NEXT n
60 CLS
70 NEXT 1
```

El bucle más interno de la línea 40 es el bucle de demora que obliga al CPC464 a contar desde el 1 hasta el 500 sin hacer absolutamente nada más. Intenta cambiar ese valor de 500 para que sea 50 o 5000 y apreciarás la diferencia; luego suprime completamente esa línea para ver la ejecución a máxima velocidad.

Anidamiento de bucles

El siguiente bucle comprende desde la línea 20 hasta la 50 y usa como contador la variable 'n' para situar el cursor de gráficos. El bucle externo, desde la línea 10 hasta la 70, asegura que todo el proceso de repite tres veces. Los bucles en este programa están encajados uno dentro de otro, y decimos normalmente que están **anidados** para indicar este hecho. Se pueden anidar cualquier número de bucles siempre y cuando recuerdes que no deben 'solaparse', es decir, deben estar totalmente incluidos uno dentro del siguiente.

Intercambia las líneas 50 y 70 para ver lo que sucede. Obtendrás el mensaje "Unexpected NEXT in 70", para indicar que se ha encontrado una instrucción **otra** que **no era la que esperaba.** El CPC464 se ha dado cuenta de los bucles controlados por 'l' y 'n' que estaban solapados y por tanto ha rehusado ejecutar el programa. Si no incluyes en la función NEXT el nombre de la variable, el CPC464 automáticamente hará que sea la que se menciona en la instrucción FOR más recientemente ejecutada. Pero eso significa que si tu programa es erróneo, simplemente funcionará un poquito más antes de sufrir el colapso, así que no te aconsejamos que prescindas de indicar la variable dentro de la instrucción NEXT.

Observa que cuando el CPC464 sale de un bucle, la variable usada para contar las rondas del bucle tendrá un valor igual al primer valor que caiga **fuera** de la banda especificada, y no el último valor que cae dentro de dicha banda. Prueba con el programa anterior otra vez y cuando haya finalizado teclea:

```
PRINT n;l
```

Verás los números 105 y 4 expuestos en pantalla; esos no son el 100 y el 3 que puede que pensaras debían exponerse. El CPC464 también te permitirá que des un salto fuera del bucle antes de que hayan concluido todas las rondas del mismo, y desde luego, los parámetros de valor de comienzo, de finalización y de paso son numerales, i.e. pueden ser sustituidos por variables:

```
10 LET a=7:LET b=73: LET s=1.2
20 FOR n=a TO b STEP s
30 IF n>50 THEN GOTO 60
40 PLOT 5,n
50 NEXT n
60 PRINT n
```

MIENTRAS QUE SE CUMPLA

Este bucle preconfinado FOR...NEXT no es la única clase de bucle que se puede usar en el CPC464. También se dispone del bucle condicionado WHILE...WEND. La sintaxis de la instrucción **mientras...** (WHILE) es como sigue:

WHILE-WEND

```
WHILE <expresion logical>
```

Estos bucles condicionados también pueden anidarse igual que los bucles preconfinados FOR...NEXT y se aplican las mismas reglas para marcar la entrada y salida del bucle. Es decir, por cada instrucción WHILE debe haber una instrucción WEND.

El bucle condicionado WHILE...WEND es uno cuya realización depende de la certeza o falsedad de la condición logical especificada en la instrucción WHILE. Todas las instrucciones del programa que están por detrás de la instrucción **mientras** y por delante de la correspondiente instrucción WEND, se ejecutarán siempre y cuando (mientras que) la condición especificada en la instrucción WHILE sea **cierta.** Por ejemplo:

```
10 WHILE s<1000
20 n=n+1
30 s=n*n*n
40 PRINT s;
50 WEND
60 PRINT
```

Ensaya con eso y verás en tu pantalla diez números que son los cubos de los números 1 a 10. Al programa se le impidió pasar de eso porque se le mandó que diera rondas **mientras** ése fuera menor que 1000, y esa es la condición que se comprueba antes de efectuar cada ronda. Cuando el CPC464, al tropezarse con la instrucción WEND, pasa a comprobar si 's' es o no menor que 1000, efectúa **otra** ronda del bucle si es menor, y se sale del bucle en cuanto es igual o mayor, pasando a la línea siguiente, en este caso la 60.

Puede que estés pensando cuáles son las ventajas de usar un bucle condicionado WHILE...WEND en lugar de lo siguiente, por ejemplo:

```
10 IF s>=1000 GOTO 60
20 n=n+1
30 s=n*n*n
40 PRINT s;
50 GOTO 10
60 REM fin del bucle
```

Después de todo ¡ambos hacen lo mismo!

La respuesta es que con el bucle WHILE...WEND no tienes que preocuparte de los números de línea a los que quieres que **vaya,** y por tanto, es más conveniente y cómodo, especialmente si en el bucle se incluye un gran número de instrucciones o si es probable que muevas y cambies líneas con el riesgo de que olvides aquélla a la que se debe saltar.

Puedes hacer que se **salga** de un bucle WHILE...WEND antes de que la condición se haga falsa, si lo deseas y usas las instrucciones pertinentes. Eso queda a tu discreción, al CPC464 no le causas ninguna confusión.

RELOJ DIGITAL

Si echamos ahora una mirada al listado de RELOJDIG, verás que hay siete bucles preconfinados FOR...NEXT, seis de los cuales están anidados. También hay un bucle condicionado WHILE...WEND muy sencillo, en la línea 170. Hay varios temas de programación en este programa que puede que sean nuevos para ti, pero uno muy importante son las líneas **multi-comando condicionales** como las líneas 1000 y 1180 de RELOJDIG.

```
10 INK 1,26:PEN 1:MODE 1
20 FOR n=48 TO 57
30 n$(n-48)=CHR$(n)
40 NEXT n
50  PRINT TAB(14);'RELOJ DIGITAL'
60 PRINT :PRINT
70 INPUT 'Escriba hora de comienzo:-';h
80 ht=INT(h/10)
90 x=h-ht*10
100 PRINT
110 INPUT 'Escriba minutos de comienzo:-';m
120 y=INT(m/10)
130 z=m-y*10
140 LOCATE 1,12:PRINT'El reloj esta listo para comenzar
a las':PRINT:PRINT TAB(16);STR$(ht*10+x);':';n$(y);n$(z)
;':00'
150 PEN 2:INK 2,1,24:SPEED INK 12,38
160 LOCATE 6,22:PRINT ' Pulsa una tecla para comenzar'
170 WHILE INKEY$='':WEND
190 MODE 0:PEN 2:INK 2,24
200 PRINT '   RELOJ  DIGITAL'
210 LOCATE 9,10:PRINT ':  :'
1000 IF ht=0 THEN v=9 ELSE v=2:LOCATE 7,10:PRINT n$(ht)
1010 FOR hu=x TO v
1020 LOCATE 8,10:PRINT n$(hu)
1030 FOR mt=y TO 5
1040 LOCATE 10,10:PRINT n$(mt)
1050 FOR mu=z TO 9
1060 LOCATE 11,10:PRINT n$(mu)
1070 FOR st =0 TO 5
1080 LOCATE 13,10:PRINT n$(st)
1090 FOR su=0 TO 9
1100 LOCATE 14,10:PRINT n$(su)
1110 FOR p=1 TO 929:NEXT p
1120 NEXT su
1130 NEXT st
```

```
1140 NEXT mu:z=0
1150 NEXT mt:y=0
1160 NEXT hu:x=0
1170 LOCATE 7,10
1180 IF ht=1 THEN ht=0:x=1:PRINT ' ':ELSE ht=1:x=0:PRIN
T n$(ht)
1190 GOTO 1000
```

En el Capítulo 7 de la Parte 1, te presentamos el comando IF-THEN-ELSE ¿Lo reconoces y lo recuerdas?

IF-THEN-ELSE

```
IF pasta>0 THEN PRINT 'Rico'
           ELSE PRINT 'pobrecillo'
```

En este comando se expondría la palabra "Rico" si la condición fuera cierta en ese momento, i.e. la variable 'pasta' tuviera un valor mayor que cero; en los demás casos, es decir, si la pasta fuera cero o negativo, se expondría la palabra "pobrecillo". En el Capítulo 6 de la Parte 1 también se te dijo cómo podrías incluir varios comandos en una única línea siempre y cuando estuvieran separados unos de otros por el símbolo **dos-puntos.** Eso ya lo aprovechamos antes en este capítulo.

Hasta ahora puede que hayas pensado que no hay ninguna diferencia entre incluir retahílas de comandos en una sola línea o en líneas separadas. Eso es correcto siempre y cuando uno de esos comandos no sea el comando **condicional** IF... La ejecución de cualquier comando que vaya detrás de la palabra clave IF en una línea multi-comando depende completamente de si la condición era cierta o falsa.

Considera por ejemplo:

```
IF pasta>0 THEN PRINT 'Rico' ELSE PRINT
'pobrecillo':PRINT'Prestame algo de dinero por favor'
```

y este otro ejemplo casi similar:

```
IF pasta>0 THEN PRINT 'Rico' ELSE PRINT
'pobrecillo'
PRINT 'Prestame algo de dinero por favor'
```

Puedes pensar que no hay ninguna diferencia entre estas dos frases condicionales, pero eso sería pensar erróneamente. En el ejemplo inferior se expondría en pantalla "préstame algo de dinero por favor" cualquiera que fuera el valor de la variable 'pasta'.

Pero en el ejemplo superior no sucedería así, sólo se expondría "préstame algo de dinero por favor" cuando la 'pasta' fuera cero o inferior, igual que sucede con la exposición de "pobrecillo". Este es un tema importantísimo para que lo recuerdes bien y puede expresarse en la forma: **Cuando la condición IF es cierta, únicamente los comandos después del THEN y hasta llegar a ELSE son ejecutados; cuando la condición no es cierta, únicamente se ejecutan los comandos después de la palabra ELSE.**

Consultando la línea 1180 de RELOJDIG podemos ver que si la variable 'ht' tiene un valor de 1 se cambiará ese valor a 0, se fijará la variable 'x' al valor 1 y se expondrá un espacio en la columna 7, línea 10 de la pantalla. Sin embargo, si el valor de 'ht' no es 1, entonces se colocará al valor 1, la variable 'x' se pondrá a 0, y el valor de la variable subindicada n$(ht) será el que se exponga en la columna 7, línea 10 de la pantalla.

Las líneas multi-comando condicionales son muy útiles cuando se usan correctamente, y ahorran un montón de líneas en un programa.

Otro tema que concierne al RELOJDIG es el uso de las variables literales subindicadas para la exposición de los números con que se designa la hora del reloj. ¿Por qué no se usan los valores de las propias variables numerales del bucle FOR? La respuesta es que cuando el CPC464 expone un dato numeral en pantalla siempre coloca un espacio en blanco por delante y un espacio después del número. Así no puedes exponer un número directamente después de otro ya que el primero será borrado parcialmente por el espacio blanco que precede al segundo. Cuando se expone una variable **literal** no hay ningún espacio blanco delantero ni trasero. Es por tanto mucho más fácil estipular la variable múltiple como literal y usar esos caracteres para mostrar números (considerados como meras sartas o retahílas de cifras, y no como cantidades) que quitar los espacios blancos de los verdaderos números (lo cual también puede hacerse, como puedes ver en el Capítulo 7).

Observa que el programa RELOJDIG no efectúa ninguna comprobación de los datos que ingresas. Aceptará horas imposibles tales como 10:76 o 35:15. En un capítulo ulterior explicaremos cómo **atrapar errores** mediante la inserción de líneas que comprueban los datos ingresados por teclado. Finalmente, puedes estar preguntándote por qué este reloj no se aprovecha del propio reloj incorporado internamente en el CPC464 (que es accesible aplicando la función de **crono** de clave TIME).

CONTROL DEL ESTUDIO

Y aquí viene otra oportunidad de comprobar tu progreso a lo largo de este curso. No te preocupes si todavía tienes que repasar algunas veces las cosas presentadas -de lo que se trata, es de que aprendas. Así que carga TAE4 y efectúa tu propio control.

Capítulo 5
TODO TRABAJO Y NADA DE JUEGO

En los primeros días de los ordenadores, éstos eran comprados casi todos por organizaciones comerciales y utilizados casi exclusivamente para el tratamiento de tales cosas como niveles de inventarios, facturación y nóminas. Hubo una compañía que intentó hacer que todo su personal fuera consciente de eso insertando el siguiente mensaje en la pantalla cada vez que alguien comenzaba a trabajar con el ordenador:

Los ordenadores de la compañía son únicamente para propósitos de la empresa

La idea era hacer que la gente dejara de divertirse con juegos de ordenador durante el tiempo de trabajo. Obviamente, no funcionó.

Dado que los ordenadores están ahora disponibles a un precio asequible, el problema ya no existe. Pocos ordenadores de gestión tienen algo ni siquiera parecido a la capacidad de gráficos en color del CPC464; y no digamos de sus comandos de sonido.

La ironía real estriba en que el CPC464 que en principio ha sido comprado para jugar y para disfrutar programando juegos de ordenador, también puede utilizarse para trabajar con programas de gestión. Pudiera ser el único ordenador que un pequeño industrial necesitara durante toda su vida.

LA PROGRAMACION DE GESTION

Muy pocos usuarios industriales saben, o desde luego desean saber, nada sobre programación. Para ellos el ordenador es simplemente otra parte de sus equipos de gestión y no precisamente el principal. Los decoradores profesionales tampoco necesitan conocer la composición química de las pinturas que usan para producir un buen decorado; y no tendrán que construirse sus propias brochas.

Por esta razón, el diseño de tales programas es considerablemente distinto de los que los aficionados producen para su propio placer.

La primera diferencia está en el diseño de las instrucciones de pantalla para el usuario. Para un usuario industrial deben ser claras y directas, y cualquier instrucción para ingreso de datos deberá especificar la naturaleza y la banda permitida de los datos precisos.

La otra diferencia es la necesidad de una "Guía de Usuario" para el programa. Incluso aunque el diseño en pantalla ayudará a los usuarios durante las etapas de introducción de datos, los usuarios lo encontrarían tedioso si tuvieran que sentarse y repasar una larga explicación de lo que se requiere, cada vez que el programa está funcionando. Mucha gente prefiere también tener una rápida perspectiva sobre la descripción del programa antes de realmente sentarse ante el teclado. Además de eso, encuentran muy provechoso disponer de alguna forma de material de consulta, apoyado en las rodillas, sobre lo que ellos están haciendo.

La mayor parte del resto de este capítulo es sobre un programa llamado PRESUPuestos; eso te dará una idea de cómo son los programas administrativos e industriales. No te asuste la idea. Aunque Amstrad nunca ha reclamado pomposamente que el CPC464 puede hacer funcionar una planta de energía nuclear, hay muchísimas maneras en que puede ayudarte en tu hogar, en la escuela y en el trabajo.

DISEÑO DE IMAGEN

Introducción de datos

INPUT

Aquí hay otra vieja amiga nuestra de la Parte 1 del curso, la instrucción para **ingreso** de datos:

```
INPUT [<literal entrecomillado>;]
  [<listado:<variable>]
```

Como puedes ver, es posible incluir más de una variable numeral o literal en una única instrucción INPUT. Eso se hace usando una coma como separador, como en el siguiente ejemplo:

```
INPUT 'Introduzca cuatro nombres';a,b,c,d
```

De esta forma, el comando expone un **signo de interrogación** al final del literal que sirve como mensaje de aviso, y deja el cursor situado una posición a la derecha de él.

Cuando ahora se introducen por el teclado los datos pedidos, separados por comas, el CPC464 intentará **in-ponerlos** sucesivamente a la variables mencionadas en el comando. Si los datos tecleados no son de la misma clase, e.g. literales en lugar de numerales, o si no se introducen tantos datos como se requieren en el comando, aparecerá el mensaje:

```
Redo from start
```

para indicar que "vuelve a hacerlo desde el principio", por lo que repite la solicitud de datos de entrada.

Si sustituyes el **punto-y-coma** después del literal de aviso con una **coma**, se suprimirán el signo de interrogación y el espacio que le sigue.

Sacando datos

PRINT

Un amigo incluso más viejo es el que usamos para **exponer** datos en pantalla, de clave PRINT. Mientras que INPUT es la manera **primordial** que tiene el usuario para **ingresar** datos en el programa, PRINT es la **única** manera que el programa tiene de **expresar** los resultados al usuario. Aparte de lograr que el cursor se **sitúe** en la posición deseada, mediante los comandos LOCATE o TAG, el uso del comando de exposición de datos PRINT es lo que puede hacer que la presentación en pantalla sea óptima o... pésima.

La sintaxis habitual es:

```
PRINT <dato a exponer>
      [<separador><dato a exponer> ... ][separador]
```

Como recordarás de la Parte 1, el signo **separador** puede ser la coma, que hace que el cursor salte hasta el comienzo de la siguiente zona de exposición en pantalla antes de sacar el siguiente **dato a exponer** (y veremos el comando ZONE posteriormente); o el **punto-y-coma**, que adosará al anterior el **dato a exponer.** Cuando hay un separador al final de los datos que se han de exponer, hace también que no se envíe a pantalla un "retorno de carro" y que por tanto el siguiente comando PRINT no exponga sus datos comenzando a partir de una nueva línea.

El **dato a exponer** puede ser uno de los tres objetos siguientes:

SPC

TAB

```
<expresion>
SPC(<expresion entera>)
TAB(<expresion entera>)
```

Puede que a primera vista los **adverbios** de TABulando (TAB) y SeParando con espaCios (SPC) en blanco parezcan la misma acción, dado que ambos se usan para generar automáticamente cuadratínes en blanco entre los datos que se exponen. La diferencia estriba en que SPC insertará tantos espacios en blanco como se especifiquen en el parámetro **expresión entera** y contados a partir de la posición actual del cursor de texto; mientras que TAG cuenta el número de espacios en blanco a partir del **comienzo** de cada línea.

Si el número de espacios especificados en SPC es mayor que la anchura de línea, el CPC464 automáticamente le restará el número de posiciones por línea (y lo restará tantas veces como sea necesario) para conseguir el número efectivo de espacios que va a usar en la separación. SPC no necesita ir seguido de un punto-y-coma ya que automáticamente se presupone ese signo separador, incluso aunque SPC aparezca como el último elemento en la lista de datos a exponer.

Cuando la posición o columna especificada en TAB es mayor que la posición corriente del cursor, se generan los espacios necesarios para alcanzar dicha posición. (No sucede nada si el cursor ya está en la posición requerida). Si se hubiera sobrepasado la posición requerida, se comenzaría a contar en la línea siguiente y se generarían los espacios suficientes hasta alcanzar dicha posición. Automáticamente se supone que hay un signo separador punto-y-coma, al igual que ocurría con SPC.

Disponemos también de dos **funciones literales** que obviamente pueden usarse como **expresión** en la lista de datos a exponer, mediante un comando PRINT. Son las que tienen de clave SPACE$ y STRING$.

SPACE$

STRING$

SPACE$ es una función que cumple una labor similar a la del adverbio SPC, siempre que el número especificado como argumento de esa función sea mayor que el número restante de columnas en la línea corriente, ya que se genera una nueva línea y se siguen contando los espacios en blanco a lo largo de ella. El máximo número posible de espacios es 255.

Estas retahílas de espacios en blanco se usan a menudo para borrar texto sin necesidad de dar el comando CLS de limpie pantalla. Es más fácil teclear:

```
PRINT SPACE$(40)
```

que la constante literal equivalente:

```
PRINT "
```

Finalmente, está la **función** STRING$ que dará una retahíla o sarta de caracteres formada completamente por el mismo carácter. La sintaxis de esta función literal es:

```
STRING$ <expresion entera>
```

Y el resultado de aplicarla es un **literal** con tantos caracteres iguales al segundo argumento como indique el primer argumento **expresión entera.** Observa que como segundo argumento puede darse el **código** ASCII que corresponde al carácter que deseas, o puedes colocar un literal cualquiera, aunque únicamente se tomará para la repetición la **inicial** del mismo. He aquí algunos ejemplos:

```
PRINT STRING$(10,90)       el resultado es zzzzzzzzzz
PRINT STRING$(16,'=')      el resultado es ================
PRINT STRING$(7,'CPC464') el resultado es CCCCCCC
```

Como sabrás la pantalla tiene varias **zonas** de exposición, de una anchura determinada. El comando que trata de eso es:

ZONE

```
ZONE(<expresion entera>,<expresion>)
```

Realmente no es muy complicado, y su acción es simplemente reponer la anchura de las zonas de exposición a lo ancho de la pantalla al valor dado por el parámetro **expresión entera** (y recuerda que el valor preceptuado para omisiones es de 13 columnas). Es un comando muy aprovechable cuando se trata de expresar tablas de números.

Molde para expresar datos

Las anteriores facetas del comando de exposición PRINT tratan todas de la colocación en pantalla de los literales y numerales. Aun con eso, todavía no se pueden manejar todos los aspectos de la imagen en pantalla, sin un extra de programación. Esa es la razón de por qué en el Amstrad BASIC tienes una de las opciones más potentes, que actúa en la forma de **adverbio** de PRINT, **usando** un molde para dar forma a los datos que expone. La sintaxis completa es:

```
PRINT USING <expresion literal>;<datos a exponer>
```

Los **datos a exponer** van separados unos de otros por el signo **coma,** aunque no tiene el mismo efecto que en una instrucción PRINT ordinaria; pero la "horma" o "molde" es la **expresión literal** que se menciona en la instrucción (también se la conoce como "plantilla de formato"), y se usa por ejemplo para que el CPC464 **exprese así** las "pesetas":

PRINT USING

```
PRINT USING '###.##';pesetas
```

Si **pesetas** tiene por valor el de 87.4473, el número que aparecerá en la pantalla será:

```
87.45
```

Esto sucederá siempre y cuando el dato a exponer sea de la naturaleza adecuada (numeral en este caso), el CPC464 lo expondrá con el punto fraccionario en la posición señalada en el "molde" y redondeándolo al número de posiciones fraccionarias especificadas en el "molde".

En este contexto, el punto (.) y la almohadilla (#) se conocen como signos **especificadores del formato del campo.** Como esos hay un total de once, pero en estos momentos suponemos que preferirás volver al teclado, así que dejaremos la explicación de ellos para más adelante (la lista la puedes ver en el Capítulo 8, página 71 de la Guía del Usuario).

PRESUPUESTOS

PRESUP es un ejemplo de un sencillo programa para estimaciones que puede ser usado por un pintor o decorador profesional o un entusiasta arreglalotodo. Realmente es mucho más avanzado que el método habitual utilizado profesionalmente, que es dar un precio fijo por habitación (basado en la experiencia, desde luego) y esperar que todo quede equilibrado, pero no es tan avanzado como para acabar con la pequeña industria. El principio es que todos los materiales y tiempos de trabajo están relacionados con el tamaño medio de las habitaciones de una casa y con el número de habitaciones -incluso el trabajo externo.

Para usar la jerga informática, diremos que este programa está **comandado por menú;** eso significa que el usuario -al igual que en algunos restaurantes- puede elegir entre diferentes tareas a realizar de las presentadas en la "carta" o menú. La rutina de presentación del menú se ve que comienza en la línea 640 detrás de las rutinas de condiciones iniciales, pantalla y presentación de títulos. Este es el bucle principal del programa y dirige al usuario a las rutinas pertinentes **según** la opción elegida entre las cinco. Aparte de la opción de "finalice programa", las opciones ofrecidas son las de realizar las tareas de ingreso de datos, que piden al usuario que teclee una serie de valores o las tareas de expresión de resultados, que le muestran los datos obtenidos como transformación de los datos introducidos.

Observa que los valores preceptuados para omisiones y utilizados para calcular el presupuesto están contenidos en la **lista de datos** que comienza en la línea 3740. Esos valores constantes son sugerencias, y desde luego, no son inmutables. Comprueba por favor esas cifras contra los precios corrientes y los salarios involucrados antes de que te puedas comprometer a decorar un palacio de 57 habitaciones en la Costa del Sol.

Aquí está el listado dispuesto para tus experimentos.

```
100 '
110 ' Presupuesto
120 '
130 ' DA 29/9/84  DGC 28/11/84
140 '
150 ' Inicializar
160 '
170 CLEAR
180 tb%=45 ' columna para entrada de respuestas
190 true=-1:false=0
200 beep$=CHR$(7):bs$=CHR$(8):lf$=CHR$(10):cr$=CHR$(13)
210 x$=" "+bs$+CHR$(24)+" " : x1$=" "+CHR$(24)
220 s1$=CR$+lf$+SPACE$(5) : s2$=SPACE$(10)
230 cp.roll=12 'un rollo cubre 12 m. cuadra.
240 DEF FNcentre$(a$)=SPACE$(40-LEN(a$)/2)+a$
250 DEF FNeven(x,y)=ROUND( x/y ) * y  'redondear x a y
mas proxima
260 '
270 ' Inicializar datos
280 '
290 DIM m$(7),m(7),mu$(7),c$(4),c(4),a$(6),a(6),l$(1),l
(1) ' opcional
300 RESTORE 3790
310 FOR i=0 TO 7 : READ m$(i),m(i),mu$(i) : NEXT
320 FOR i=0 TO 4 : READ c$(i),c(i) : NEXT
330 FOR i=0 TO 6 : READ a$(i),a(i) : NEXT
340 FOR i=0 TO 1 : READ l$(i),l(i) : NEXT
350 READ rph$,rph
360 '
370 ' Pantalla
380 '
390 MODE 2  : BORDER 0
400 INK 0,0 : INK 1,22
410 PAPER 0 : PEN 1
420 '
430 h$="Presupuesto de Decoracion"
440 GOSUB 1860 ' encabezado
```

```
450 '
460 ' Menu
470 '
480 RESTORE 3760
490 FOR i=1 TO 5
500 READ t$
510 LOCATE 28,i*2+5
520 PRINT i;'. ';t$
530 NEXT
540 LOCATE 32,20 : PRINT CHR$(18);
550 INPUT 'Seleccione (1-5) : ',i$ : i=VAL(i$)
560 IF i<1 OR i>5 THEN PRINT beep$; : GOTO 540
570 ON i GOSUB 670,2360,2660,1670,630
580 '
590 GOTO 430
600 '
610 ' Fin programa
620 '
630 LOCATE 1,22 : END
640 '
650 ' SUBRUTINA : Entrada detalles obra
660 '
670 h$='Entrada detalles'
680 GOSUB 1860 ' encabezado
690 LOCATE 1,5
700 '
710 ' General
720 '
730 hd$='Nombre del cliente':F%=79-tb%:GOSUB 2140:clien
t$=v$
740 hd$='Direccion (sin comas por favor)':F%=79-tb%:GOS
UB 2140:address$=v$
750 GOSUB 1810 : total.rooms = rooms
760 PRINT
770 '
780 ' Data pormenores
790 '
800 hd$=x$+'DECORACION INTERIOR'+x1$:GOSUB 2240:id.flag
=v%
810 IF id.flag=false GOTO 1150 ' no dec interior
820 '
830 ' Techos
840 '
850 hd$=s1$+x$+'Techo'+x1$:GOSUB 2240:ce.flag=v%
860 IF ce.flag=false GOTO 930
870 hd$=s2$+'Techos empapelados+emulsion':GOSUB 2040:ce
.pe=v%
```

```
880 hd$=s2$+'Techos solo con emulsion':GOSUB 2040:ce.e=
v%
890 ce.number=ce.pe+ce.e
900 ce.area.pap=ce.pe*a(0) : ce.area.emul=(ce.pe+ce.e)*
a(0)
910 ce.cost=(ce.area.pap * ( m(1)/cp.roll ) ) + (ce.are
a.emul * (m(2)/c(2)) )
920 ce.time=(ce.area.pap / l(0)) + (ce.area.emul / l(1)
)
930 '
940 ' Paredes
950 '
960 hd$=s1$+x$+'Paredes'+x1$:GOSUB 2240:wa.flag=v%
970 IF wa.flag=false GOTO 1040
980 hd$=s2$+'Habit. empapeladas solamente':GOSUB 2040:w
a.p=v%
990 hd$=s2$+'Habit. solo con emulsion':GOSUB 2040:wa.e=
v%
1000 wa.number=wa.p+wa.e
1010 wa.area.pap=wa.p*a(1) : wa.area.emul=wa.e*a(1)
1020 wa.cost=(wa.area.pap *  (m(0)/cp.roll ) ) + (wa.ar
ea.emul * (m(2)/c(2)) )
1030 wa.time=(wa.area.pap / l(0)) + (wa.area.emul / l(1
))
1040 '
1050 ' Carpinteria
1060 '
1070 hd$=s1$+x$+'Carpinteria'+x1$:GOSUB 2240:ww.flag=v%
1080 IF ww.flag=false GOTO 1150
1090 hd$=s2$+'Dos manos pintura habitaciones':GOSUB 204
0:ww.ut=v%
1100 hd$=s2$+'Solo una mano pintura habitaciones':GOSUB
 2040:ww.t=v%
1110 ww.number=ww.ut+ww.t
1120 ww.area.under=ww.ut*a(2) : ww.area.top=(ww.ut+ww.t
)*a(2)
1130 ww.cost=(ww.area.under * m(3)/c(0) ) + (ww.area.to
p * m(4)/c(1) )
1140 ww.time=(ww.area.under+ww.area.top) / l(1)
1150 '
1160 ' Decoracion Exterior
1170 '
1180 PRINT : PRINT
1190 hd$=x$+'DECORACION EXTERIOR'+x1$:GOSUB 2240:od.fla
g=v%
1200 IF od.flag=false GOTO 1520 ' no dec ext
1210 '
1220 ' Ventanas y Puertas
```

```
1230 '
1240 hd$=s1$+x$+'Ventanas y Puertas'+x1$:GOSUB 2240:wd.
flag=v%
1250 IF wd.flag=false GOTO 1340
1260 wd.cost=0:wd.time=0
1270 doors.and.wind=total.rooms * a(3)
1280 hd$=s2$+'.Solo Barniz a todo?':GOSUB 2240
1290 IF v% THEN wd.cost=doors.and.wind * m(6)/c(3) : wd
.time=doors.and.wind / l(1):GOTO 1340
1300 hd$=s2$+'.Todo con dos manos de pintura?':GOSUB 22
40
1310 IF v% THEN wd.cost=doors.and.wind *( m(3)/c(0) +m(
5)/c(1) ):wd.time=doors.and.wind*2 / l(1):GOTO 1340
1320 hd$=s2$+'.Todo con una mano de pintura?':GOSUB 224
0
1330 IF v% THEN wd.cost=doors.and.wind * m(5)/c(1) :wd.
time=doors.and.wind / l(1)
1340 '
1350 ' Tuberias & Canalones
1360 '
1370 hd$=s1$+x$+'Tuberias y Canalones'+x1$:GOSUB 2240:g
d.flag=v%
1380 IF gd.flag=false GOTO 1450
1390 gd.cost=0:gd.time=0
1400 gd.length=total.rooms * a(4)
1410 hd$=s2$+'.Todo con dos manos de pintura?' : GOSUB
2240
1420 IF v% THEN gd.cost=gd.length *( m(3)/c(0) + m(5)/c
(1) ):gd.time=gd.length*2 / l(1) :GOTO 1450
1430 hd$=s2$+'.Todo con una mano de pintura?' : GOSUB 2
240
1440 IF v% THEN gd.cost= gd.length * m(5)/c(1) :gd.time
=gd.length / l(1)
1450 '
1460 ' Paredes exteriores
1470 '
1480 hd$=s1$+x$+'Paredes exteriores'+x1$:GOSUB 2240:ow.
flag=v%
1490 IF ow.flag=false THEN  1520
1500 ow.cost = total.rooms * a(5) * ( m(7)/c(4) )
1510 ow.time = total.rooms * a(5) / l(1)
1520 '
1530 ' Redondeo de variables
1540 '
1550 ce.cost=FNeven(ce.cost,0.01) : ce.time=FNeven(ce.t
ime,0.25)
1560 wa.cost=FNeven(wa.cost,0.01) : wa.time=FNeven(wa.t
ime,0.25)
```

```
1570 ww.cost=FNeven(ww.cost,0.01) : ww.time=FNeven(ww.t
ime,0.25)
1580 wd.cost=FNeven(wd.cost,0.01) : wd.time=FNeven(wd.t
ime,0.25)
1590 gd.cost=FNeven(gd.cost,0.01) : gd.time=FNeven(gd.t
ime,0.25)
1600 ow.cost=FNeven(ow.cost,0.01) : ow.time=FNeven(ow.t
ime,0.25)
1610 '
1620 PRINT : PRINT : GOSUB 1950 ' barra espaciadora
1630 RETURN
1640 '
1650 ' SUBRUTINA : Preparar salida de presupuesto
1660 '
1670 h$='Mostrar Presupuesto' : GOSUB 1860 ' Encabezado
1680 ' Enviar a pantalla
1690 '
1700 str = 0 : GOSUB 3100
1710 '
1720 ' Enviar a impresora
1730 '
1740 REM:  str = 8 : GOSUB 3100  (disabled)
1750 '
1760 RETURN
1770 '
1780 ' SUBRUTINA : Preguntar numero de habitac.
1790 '                da respuesta en 'habitaciones'
1800 '
1810 hd$='Numero total de habitaciones' : F% = 2
1820 GOSUB 2040 : rooms=v% : RETURN
1830 '
1840 ' SUBRUTINA : Imprimir encabezado
1850 '
1860 CLS
1870 temp1=LEN(h$):temp2=(80-temp1)/2
1880 PRINT TAB(temp2);CHR$(24);STRING$(temp1+2,131);CHR
$(24)
1890 PRINT TAB(temp2);CHR$(24);' ';h$;' ';CHR$(24)
1900 PRINT TAB(temp2);CHR$(24);STRING$(temp1+2,140);CHR
$(24)
1910 RETURN
1920 '
1930 ' SUBRUTINA : Esperar barra esp. regrese pos. norm
al
1940 '
1950 LOCATE 1,25
1960 PRINT FNcentre$('Pulsa '+CHR$(24)+' ESPACIO '+CHR$
(24)+' para continuar');
```

```
1970 WHILE INKEY$ > "" : WEND
1980 WHILE INKEY$<>" " : WEND
1990 RETURN
2000 '
2010 ' SUBRUTINA : Entrada rutina (numero)
2020 '                numero respuesta en V%
2030 '
2040 PRINT hd$;TAB(tb%);
2050 PRINT"[";
2060 PRINT STRING$(F%," ");"] numero";STRING$(F%+8,bs$)
;
2070 INPUT "",V%
2080 RETURN
2090 '
2100 ' SUBRUTINA : Rutina (string)
2110 '                regresa string en V$
2120 '                Longitud campo en F%
2130 '
2140 PRINT hd$;TAB(tb%);
2150 PRINT "[";
2160 LOCATE tb%+F%,VPOS(#0):PRINT"]";
2170 LOCATE tb%+1 ,VPOS(#0)
2180 INPUT "",V$
2190 RETURN
2200 '
2210 ' SUBRUTINA :  Rutina Si/No
2220 '                si v%=verdadero, no v%=falso
2230 '
2240 PRINT hd$;TAB(tb%);
2250 v%=1
2260 PRINT"[ ]  S - si  o N - no"STRING$(35,8);
2270 g$=INKEY$:IF g$="" GOTO 2270
2280 IF UPPER$(g$)="S" THEN v%=true : PRINT"S";STRING$(
1,bs$);:GOTO 2320
2290 IF UPPER$(g$)="N" THEN v%=false: PRINT"N";STRING$(
1,bs$);:GOTO 2320
2300 IF v%<1 AND g$=cr$ THEN PRINT:RETURN
2310 PRINT beep$;
2320 GOTO 2270
2330 '
2340 ' SUBRUTINA : Mostrar detalles
2350 '
2360 GOSUB 2610
2370 PRINT : PRINT"MATERIALES";TAB(40);"COSTE";TAB(55);
"UNIDAD" : PRINT
2380 FOR i=0 TO 7
2390 PRINT " ";m$(i);
2400 PRINT TAB(40);USING "##.##";m(i);
```

```
2410 PRINT TAB(55);mu$(i)
2420 NEXT
2430 PRINT : PRINT"RENDIMIENTO";TAB(40);"METROS CUAD./L
ITRO": PRINT
2440 FOR i=0 TO 4
2450 PRINT " ";c$(i);TAB(40);USING "###.#";c(i)
2460 NEXT
2470 GOSUB 1950 ' Siguiente pantalla
2480 GOSUB 2610
2490 PRINT : PRINT"SUPERFICIES";TAB(40);"METROS CUAD./H
ABIT." : PRINT
2500 FOR i=0 TO 5
2510 PRINT " ";a$(i);TAB(40);USING "###.#";a(i)
2520 NEXT
2530 PRINT : PRINT "HOMBRES-HORAS";TAB(40);"METROS CUAD
./HORA" : PRINT
2540 FOR i=0 TO 1
2550 PRINT " ";l$(i);TAB(40);USING "###.#";l(i)
2560 NEXT
2570 PRINT : PRINT "INDICE / HORA";TAB(40);"LIBRAS/HORA
" : PRINT
2580 PRINT " ";rph$;TAB(40);USING "##.##";rph
2590 GOSUB 1950 ' space
2600 RETURN
2610 h$="Coste" : GOSUB 1860 ' encabezado
2620 RETURN
2630 '
2640 ' SUBRUTINA : Corregir Coste Materiales
2650 '
2660 GOSUB 3000
2670 PRINT : PRINT"MATERIALES";TAB(40);"COSTE";TAB(50);
"NUEVO COSTE";TAB(65);"UNIDAD" : PRINT
2680 FOR i=0 TO 7
2690 PRINT " ";m$(i);TAB(40);USING "##.##";m(i);
2700 PRINT TAB(65);mu$(i)
2710 LOCATE 55,VPOS(#0)-1 : INPUT "",n
2720 IF n>0 THEN m(i)=n
2730 NEXT
2740 PRINT : PRINT"RENDIMIENTO";TAB(40);"METROS CUAD./L
ITRO";TAB(60);"NUEVO VALOR": PRINT
2750 FOR i=0 TO 4
2760 PRINT " ";c$(i);TAB(40);USING "###.#";c(i);
2770 PRINT TAB(60);:INPUT "",n
2780 IF n>0 THEN c(i)=n
2790 NEXT
2800 GOSUB 1950 ' Siguiente pantalla
2810 GOSUB 3000
```

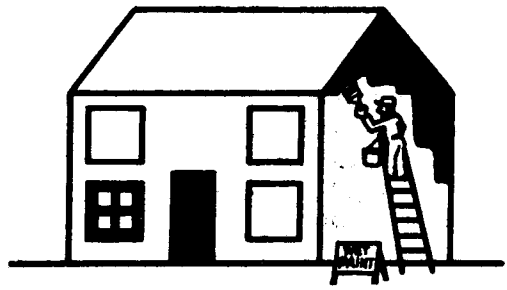
```
2820 PRINT : PRINT'SUPERFICIES';TAB(40);'METROS CUAD./H
ABIT.';TAB(60);'NUEVO VALOR' : PRINT
2830 FOR i=0 TO 5
2840 PRINT ' ';a$(i);TAB(40);USING '###.#';a(i);
2850 PRINT TAB(60); : INPUT '',n
2860 IF n>0 THEN a(i)=n
2870 NEXT
2880 PRINT : PRINT 'INDICE TRAB.';TAB(40);'METROS CUAD.
/HORA';TAB(60);'NUEVO VALOR'
2890 PRINT
2900 FOR i=0 TO 1
2910 PRINT ' ';l$(i);TAB(40);USING '###.#';l(i);
2920 PRINT TAB(60); : INPUT '',n
2925 IF n>0 THEN l(i)=n
2930 NEXT
2940 PRINT : PRINT 'INDICE / HORA';TAB(40);'LIBRAS/HORA
';TAB(60);'NUEVO INDICE'
2950 PRINT
2960 PRINT ' ';rph$;TAB(40);USING '##.##';rph;
2970 PRINT TAB(60); : INPUT '',n
2975 IF n>0 THEN rph=n
2980 GOSUB 1950 ' espacio
2990 RETURN
3000 h$='Reajustar Coste' : GOSUB 1860 ' emcabezado
3010 PRINT'     Pulse ENTER para saltar'
3020 RETURN
3030 '
3040 ' SUBRUTINA : Imprimir presupuesto
3050 '             : str=stream para imprimir to
3060 '             : ta - te = posic. tabulador
3070 '
3080 ' Encabezado principal
3090 '
3100 ta=0 : tb=5 : tc=25 : td=45 : te=65 ' posiciones d
e columna en informe
3110 IF (id.flag = false) AND (od.flag = false) THEN PR
INT beep$; : RETURN ' si n cifras
3120 PRINT #str,FNcentre$('Presupuesto de tiempo y mate
riales para obra de redecoracion en:')
3130 PRINT #str
3140 PRINT #str,FNcentre$(address$) : PRINT #str
3150 PRINT #str,FNcentre$('para '+client$) : PRINT #str
3160 '
3170 ' Obra interior
3180 '
3190 id.cost=0 : id.time=0
3200 IF id.flag=false GOTO 3350
3210 PRINT #str,TAB(ta);'Obra interior :' : PRINT #str
```

```
3220 PRINT #str,TAB(tb);'TRABAJO';TAB(tc);'NO DE HAB.';
TAB(td);'COSTE MATERIALES';
3230 PRINT #str,TAB(te+1);'TIEMPO'
3240 PRINT #str,TAB(tb);'----';TAB(tc);'----------';TA
B(td);'--------------';
3250 PRINT #str,TAB(te+1);'----' : PRINT #str
3260 IF ce.flag THEN n=0 : n1=ce.number : c1=ce.cost :
t1=ce.time : GOSUB 3670 : id.cost=id.cost+ce.cost : id.
time=id.time+ce.time
3270 IF wa.flag THEN n=1 : n1=wa.number : c1=wa.cost :
t1=wa.time : GOSUB 3670 : id.cost=id.cost+wa.cost : id.
time=id.time+wa.time
3280 IF ww.flag THEN n=2 : n1=ww.number : c1=ww.cost :
t1=ww.time : GOSUB 3670 : id.cost=id.cost+ww.cost : id.
time=id.time+ww.time
3290 PRINT #str
3300 PRINT #str,TAB(td);'-------'; TAB(te-2);'--------'
3310 PRINT #str,TAB(tc);'Totales:';
3320 n=6 : n1=0 : c1=id.cost : t1=id.time : GOSUB 3670
3330 PRINT #str,TAB(td);'======='; TAB(te-2);'========'
3340 PRINT #str : PRINT #str
3350 '
3360 ' Exterior
3370 '
3380 od.cost=0 : od.time=0
3390 IF od.flag=false GOTO 3520
3400 PRINT #str,TAB(ta);'Obra exterior :' : PRINT #str
3410 PRINT #str,TAB(tb);'TRABAJO'; TAB(td);'COSTE MATER
IALES'; TAB(te);'TIEMPO'
3420 PRINT #str,TAB(tb);'----'; TAB(td);'--------------
'; TAB(te);'----'
3430 PRINT #str
3440 IF wd.flag THEN n=3 : n1=0 : c1=wd.cost : t1=wd.ti
me : GOSUB 3670 : od.cost=od.cost+wd.cost : od.time=od.
time+wd.time
3450 IF gd.flag THEN n=4 : n1=0 : c1=gd.cost : t1=gd.ti
me : GOSUB 3670 : od.cost=od.cost+gd.cost : od.time=od.
time+gd.time
3460 IF ow.flag THEN n=5 : n1=0 : c1=ow.cost : t1=ow.ti
me : GOSUB 3670 : od.cost=od.cost+ow.cost : od.time=od.
time+ow.time
3470 PRINT #str,TAB(td);'-------'; TAB(te-2);'--------'
3480 PRINT #str,TAB(tc);'Totales:';
3490 n=6 : n1=0 : c1=od.cost : t1=od.time : GOSUB 3670
3500 PRINT #str,TAB(td);'======='; TAB(te-2);'========'

3510 PRINT #str : PRINT #str
3520 '
```

```
3530 t.cost=id.cost+od.cost : t.time=id.time+od.time
3540 PRINT#str,TAB(tb);'Gran Total';
3550 n=6 : n1=0 : c1=t.cost : t1=t.time :GOSUB 3670
3560 PRINT #str,TAB(tb);'Mano de Obra';TAB(td);USING '#
###.##';t1*rph;
3570 PRINT #str
3580 PRINT #STR,TAB(td-2);'=========';
3590 PRINT #str,TAB(tb);'Total Presupuesto';TAB(td);USI
NG '####.##';t.time*rph+t.cost
3600 PRINT #str : PRINT #str
3610 '
3620 GOSUB 1950 : RETURN
3630 '
3640 ' SUBRUTINA : Linea salida para impresora
3650 '               necesita n para a$,n1,c1,t1 : if n1
=0 then no imprime
3660 '
3670 IF a$(n)>'' THEN PRINT #str,TAB(tb);a$(n);
3680 IF n1>0 THEN PRINT #str,TAB(tc);USING '##';n1;
3690 PRINT #str,TAB(td);USING '####.##';c1;
3700 PRINT #str,TAB(te);USING '###.##';t1;
3710 PRINT #str,' hrs.'
3720 RETURN
3730 '
3740 ' Lista de Data
3750 '
3760 DATA Entrada detalles,Mostrar costes,Reajustar cos
tes
3770 DATA Imprimir presupuesto cliente,Fin de programa
3780 '
3790 DATA papel de pared,3.00,rollo (12x1m)
3800 DATA papel de techo,3.00,rollo (12x1m)
3810 DATA emulsion,2.72,litro
3820 DATA una mano,2.81,litro
3830 DATA dos manos (interior),3.05,litro
3840 DATA dos manos (exterior),3.05,litro
3850 DATA barniz,11.20,litro
3860 DATA pintura pared,2.72,litro
3870 '
3880 DATA primera mano,14.5,segunda mano,15,emulsion,14
3890 DATA barniz,22,pintura de pared ,17
3900 '
3910 DATA techos,9,paredes interiores,34,carpint. inter
ior,3
3920 DATA ventanas y puertas exteriores,3,canalones etc
.,0.5
3930 DATA paredes exteriores,16,'',0
```

```
3940 '
3950 DATA empapelado,6,pintura,1.5
3960 '
3970 DATA Indice por hora,5.50
```



GUIA DEL USUARIO PARA PRESUPuestos

Al principio de este capítulo decíamos que la guía del usuario era parte esencial de cualquier programa de gestión; ¿o no lo dijimos? Sin elaborar este tema demasiado, los siguientes párrafos son consejos que podrían ayudar a un no-programador en el uso del programa PRESUP sin que se vuelva loco intentando atar cabos.

Comienzo

Cuando mandas al CPC464 que **cargue** y **ejecute** el programa PRESUP, presenta un menú en la pantalla del que puedes elegir la tarea que requieres. Vuelves a regresar a ese menú al final de las tareas 1 a 4 para efectuar una nueva elección.

Observa que este programa sólo conserva la información durante el tiempo en que el sistema está trabajando. Ten cuidado antes de elegir la tarea 5 de finalizar programa, en que no haya nada que desees registrar, dado que poniendo en marcha el programa de nuevo hará que se olviden todas las condiciones anteriores.

Introduciendo los detalles del trabajo

Primero teclea el nombre del cliente y la dirección tal y como te lo solicitan los avisos en pantalla. Luego se te pide el número de habiaciones de la casa para las que se requiere el presupuesto. Incluye la cocina, el cuarto de baño, los lavabos y los grandes salones, pero puedes excluir los armarios de la cocina y similares.

Después de responder sí a las preguntas 'sí-no', teclea el número de habitaciones que exigen cada clase concreta de mano de obra. Puedes teclear o bien 0 o pulsar la tecla ENTER solamente si el oficio requerido no se aplica a una habitación concreta.

Costes

La opción 2 muestra las cantidades, precios, horas, etc., sobre las que se basará el presupuesto.

Remendando los costes

Esta opción (Nº 3) presenta en pantalla cada coste, uno después de otro, de manera que puedas imponer un nuevo valor. Puedes saltarte aquéllos que no quieres cambiar, pulsando simplemente la tecla ENTER. Estos cambios sólo quedan retenidos durante el tiempo en que el programa esté en marcha.

Expresando el presupuesto

Puedes exponer el presupuesto alcanzado si eliges la opción 4. Es un resumen de los costes estimados, calculado a partir de las informaciones que diste sobre los detalles de los trabajos y los costes actuales. Observa que sólo se puede manejar un presupuesto a la vez, y que ingresando una nueva serie de detalles de los trabajos, se borrará completamente el presupuesto previo.

Es posible sin embargo, regresar a la opción 3 (sin hacer que finalice el programa) y cambiar algunos de los costes -un papel pintado más barato, por ejemplo- y exponer la nueva versión del mismo presupuesto.

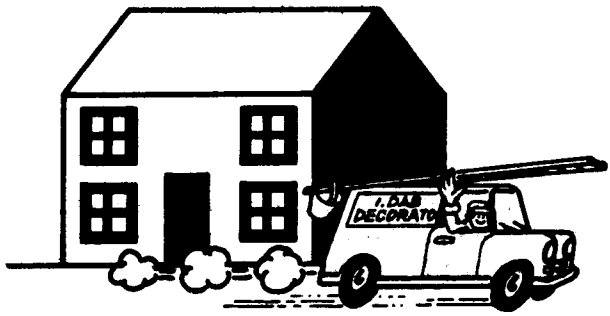
MEJORAS ADICIONALES

Todavía no se ha escrito un programa que no pueda ser mejorado o ampliado de alguna manera. PRESUP no es ninguna excepción. Además de eso, los programas de gestión se están modificando y adaptando constantemente para amoldarlos a las necesidades cambiantes de sus usuarios. Sin ninguna duda, también tú querrás hacer lo mismo con este programa.

Hay unas cuantas cosas que te sugerimos consideres como mejoras adicionales al programa:

- ☐ Permitir la clase de casa (chalet, semi-adosada, etc.)
- ☐ Registrar el esquema de papel y color por habitación.
- ☐ Calcular las necesidades totales de pintura y papel.
- ☐ Añadir la labor de quitar la pintura y el papel que hubiera.
- ☐ Añadir el coste de alquilar escaleras u otros accesorios.

Probablemente puedas pensar por ti mismo otra docena por lo menos, aunque la primera prioridad es programar una nueva opción que te exprese el resultado mediante la impresora matricial de puntos Amstrad DMP-1. Alguna guía sobre eso la daremos en el Capítulo 12.



Aunque no debes ser demasiado ambicioso. Los ordenadores a menudo han adquirido una mala reputación simplemente porque un programador demasiado entusiasta ha hecho el sistema tan complicado de usar que es mucho más rápido elaborar los presupuestos en el dorso de un sobre cualquiera.

CONTROL DEL ESTUDIO

Esto realmente ha sido todo el trabajo y no ha habido nada de juego ¿o sí? Por esta razón, somos amables y no te damos un control autoevaluador en este capítulo.

Capítulo 6
REGRESO A LA ESCUELA

Aquéllos de vosotros lo suficientemente afortunados como para haber conseguido un sobresaliente en 5º de E.G.B. y cursos sucesivos, habréis aprendido todas esas cosas sobre senos, cosenos, tangentes, radianes, logaritmos y exponenciales. Para el resto de vosotros, ahora llega la hora de disfrutar de ellos.

Este capítulo no pretende ser una lección de matemáticas - después de todo, puedes comprar algunos libros muy buenos de matemáticas- pero será necesario que nos disculpes por la explicación que te vamos a dar de las funciones matemáticas del CPC464. El Amstrad CPC464 es un sirviente mecatrónico inteligente y puede hacer tratamientos numéricos bastante rápidos, pero hay un montón de programas de ordenador que no requieren realmente funciones matemáticas en absoluto, sólo +, -, * y /. Sin embargo, necesitarás saber cosas sobre funciones como la de SIN, LOG y EXP con el fin de aprender cuándo usarlas (y cuándo no).

TRIGONOMETRIA

De modo que vamos con ello... Las palabras clave SIN, COS, TAN y ATN tratan todas con ángulos y son funciones trigonométricas respectivamente del **seno, coseno, tangente** y el **arco cuya tangente** corresponde al argumento. En cualquier cálculo en que utiliza ángulos, el CPC464 presupone -si se omite especificar explícitamente lo contrario- que tus ángulos han de ser medidos en **radianes** y no en **grados**. Un radián es una medida de la abertura angular y corresponde a un ángulo igual a 57,295 grados; en una circunferencia hay por tanto 2*PI radianes (6,2838). Aunque parezca extraño, trabajar en radianes hace muchas veces las cosas más fáciles. Es más probable que quieras trabajar en grados, ya que estás más acostumbrado y así te ahorras un montón de conversiones tediosas. El CPC464 dispone del comando DEG que ya nos encontramos en el Capítulo 3. Este le dice al CPC464 que trabaje en **grados**.

SIN

COS

TAN

DEG

RAD

Hay otro comando de palabra clave RAD, que le dice que debe trabajar en **radianes.**

Teclea el siguiente programa:

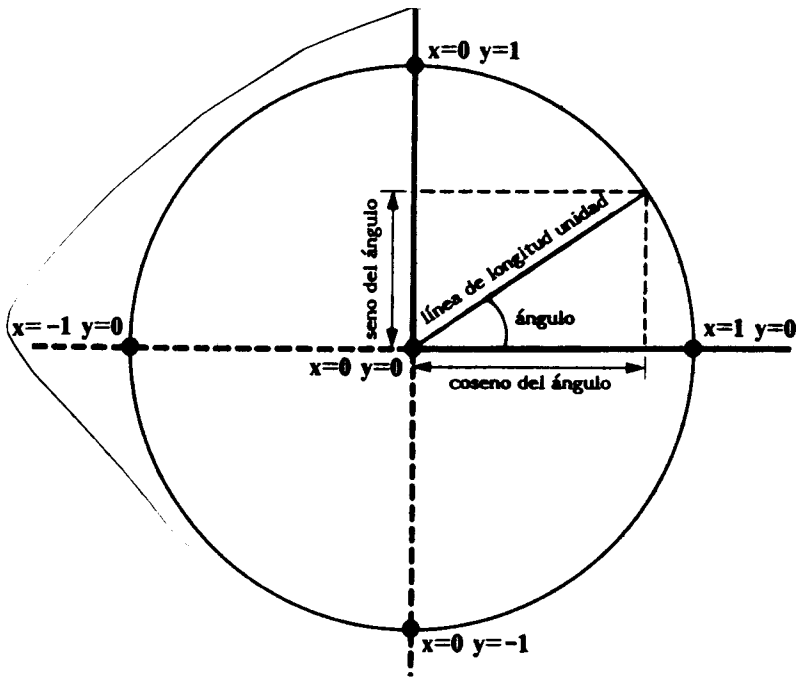
```
5 DEG
10 ORIGIN 300,200
FOR n=1 TO 360
30 x=SIN (n)
40 y=COS (n)
50 PLOT 100*x,100*y
60 NEXT n
```

Vamos ahora a aprovechar este programa para demostrar unos cuantos temas.

El comando ORIGIN mueve el punto crucial 0,0 al centro de la pantalla más o menos (como se explicó en el Capítulo 3). La variable contadora 'n' va a ser incrementada en unidades, comenzando por uno y finalizando por 360. Los senos y cosenos de cada valor de 'n' (recuerda SENCOS) se multiplican por 100 y se usan como coordenadas para que se **pinte** en pantalla una mota.

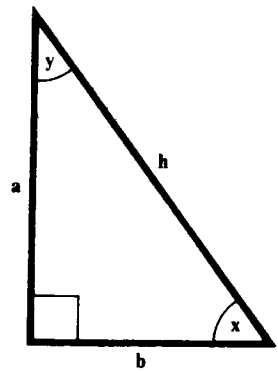
Ahora pasa el programa y verás el círculo pintado "grado a grado".

Para aquéllos de vosotros que os estéis preguntando cómo funciona nuestro programa de circunferencias, el siguiente diagrama os ayudará.



Resolviendo triángulos

Las funciones trigonométricas de seno, coseno y tangente, son las usadas más a menudo para la solución de triángulos, i.e. teniendo algo de información sobre un triángulo, puedes encontrar con ellas el resto de las características. Estas funciones trigonométricas representan las **razones** entre las longitudes de los diferentes lados. En un triángulo rectángulo (uno de los ángulos = 90º) el seno de cualquiera de los otros dos ángulos se halla dividiendo la longitud del lado opuesto a ese ángulo entre la longitud del lado más largo, que normalmente se llama hipotenusa.



SQR

En el Amstrad BASIC, como de hecho en los libros de matemáticas, en lugar del 'seno del ángulo x', escribimos SIN(x), de igual manera que decimos 'la raíz cuadrada de x' como SQR(x).

Teniendo como referencia el diagrama, vemos que:

sen(x)=a/h

y de igual manera,

sen(y)=b/h

El coseno del ángulo se halla dividiendo la longitud del lado adyacente al ángulo por la longitud de la hipotenusa:

cos(x)=b/h y cos(y)=a/h

Observa que el seno de x es igual al coseno de y, al ser ángulos complementarios; y viceversa. Eso es un caso especial de los triángulos rectángulos que los hace más fáciles de resolver.

La tangente se halla dividiendo el lado opuesto por el lado adyacente:

tg(x)=a/b y tg(y)=b/a

Para resolver cualquier triángulo rectángulo, necesitaremos saber como mínimo dos lados, o un lado y cualquier otro ángulo distinto del recto. De manera que si a=20 y h=50, podemos calcular el seno de x y el coseno de y. Si tecleas:

```
LET a=20: LET h=50:PRINT a/h
```

Obtienes la respuesta 0.4; y 0.4 será pues el seno(x) y el cos(y).

```
SIN(x)=COS(y)=0.4
```

Ahora que sabemos eso, podemos hallar fácilmente la abertura de los ángulos 'x' e 'y' en grados. En una calculadora científica de mesa tendrías probablemente que pulsar alguna tecla especial para la conversión y luego pulsar la de seno o coseno para obtener la respuesta. Lo que realmente estás haciendo es usar las funciones **inversas** del seno y el coseno. Son las que llamamos arco cuyo seno es... y arco cuyo coseno es... También existe **arco cuya tangente es...**

ATN

De estas funciones que dan el **arco**, o sea, el ángulo, el CPC464 sólo dispone de la que calcula el arco cuya tangente es... su clave es ATN. La función se aplica en la forma:

```
ATN (<expresion numerica>)
```

De modo que para hallar el arcoseno y el arcocoseno tenemos que saber un poquito de transformaciones, pero no es muy difícil dado que todas las funciones están relacionadas entre sí. Simplemente convertiremos los senos y cosenos en tangentes y luego aplicaremos la función ATN para hallar el ángulo. Para ahorrarte el repaso de páginas y páginas de tus apuntes de matemáticas o de colocarte como esclavo durante horas intentando determinar las fórmulas, te damos directamente la solución:

$$\tan(x) = \frac{\sin(x)}{\sqrt{1 - \sin^2(x)}}$$

$$\tan(x) = \frac{\sqrt{1 - \cos^2(x)}}{\cos(x)}$$

RAICES CUADRADAS

El CPC464 tiene una función para calcular la raíz cuadrada de un número.

Su clave es SQR (Square Root=Cuadrada Raíz). Y si usamos por ejemplo PRINT SQR(64) da como respuesta 8, dado que 8 elevado al cuadrado (y en BASIC la notación de esta operación sería 8^2) da como resultado 64. Teclea lo siguiente:

```
LET tangente=0.4/SQR(1-(0.4^2))
PRINT ATN(tangente)
```

Obtendrás la respuesta 23.578, que aproximadamente son 23½ grados. Ahora haz lo mismo para el coseno:

```
PRINT ATN(SQR(1-(0.4^2))/0.4)
```

Deberás obtener 66.421, que son aproximadamente 66½ grados. Si ahora sumas estos dos valores obtenidos, el resultado es 90 porque los ángulos de cualquier triángulo siempre suman 180; y en un triángulo rectángulo ya hay uno de 90º, por lo que los otros deben sumar los otros 90º.

```
23.578+66.421+90=180
```
(suficientemente aproximado)

Ahora que conocemos los ángulos, todo lo que nos queda saber del triángulo es el lado 'b'. Sabemos por ejemplo que sin(y)=b/h. Podemos fácilmente transformar eso para que nos dé b=h*sin(y). Ahora podemos hacer la misma cosa para las otras funciones y así poder producir una tabla:

sin(y)=b/h se convierte en b=h*sin(y)
cos(x)=b/h se convierte en b=h*cos(x)
tan(x)=a/b se convierte en b=a/tan(x)
tan(y)=b/a se convierte en b=a*tan(y)

Así que tenemos cuatro maneras de calcular la longitud del lado 'b' y cada una de ellas debe dar la misma respuesta. Ensáyalo y a ver si es verdad. Teclea lo siguiente:

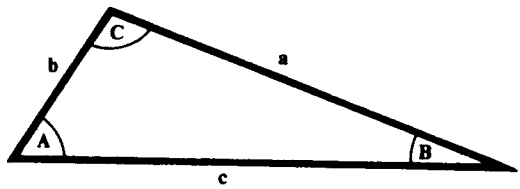
```
PRINT 50*SIN(66.421)      (da 45.825...)
PRINT 50*COS(23.578)      (da 45.825...)
PRINT 20/TAN(23.578)      (da 45.826...)
PRINT 20*TAN(66.421)      (da 45.824...)
```

Desde luego que sacamos los mismos resultados, junto con algo más. Este fue un ejemplo de cómo resolver un triángulo rectángulo. Como hemos dicho antes, con dos lados, o con un ángulo distinto del recto y un lado cualquiera, nos es suficiente para hallar el resto de las características.

Si en lugar de eso tenemos un triángulo cualquiera, la situación es un poquito más complicada; pero conociendo la suficiente información siempre podemos encontrar todo lo que necesitamos. Sin embargo, los cálculos son más complicados.

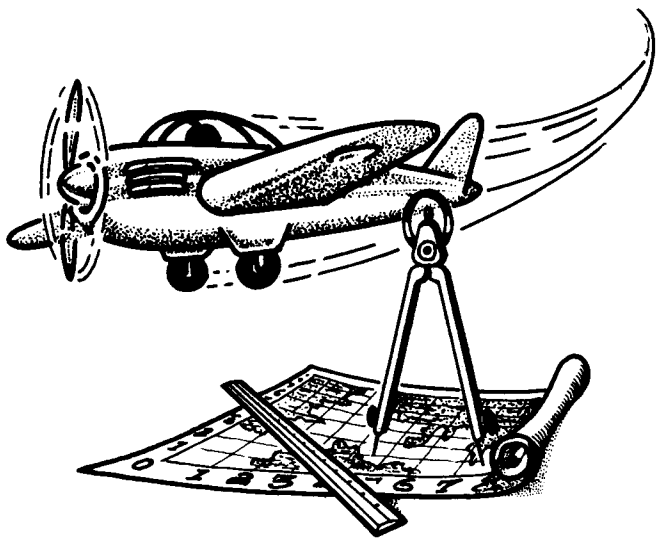
Podemos utilizar dos conocidas reglas:

- ☐ **La regla del seno:** para cualquier triángulo se cumple que: a/seno(A) = b/seno(B) = c/seno(C)
- ☐ **La regla del coseno:** para cualquier triángulo se cumple que:
  $a^2 = b^2 + c^2 - 2bc\,\text{coseno}(A)$



DESPEGUE

En este momento puede que estés pensando "Todo eso está muy, pero ¿cómo lo puedo aplicar yo?"... Muy bien, supongamos entonces que eres el piloto de un aeroplano que quieres poner rumbo norte para llegar a otro aeropuerto. En un día sin viento simplemente pondrás rumbo norte para llegar allí. Sin embargo, en un día borrascoso, si vuelas derecho al norte acabarías, dependiendo de la dirección del viento, en cualquier otro aeropuerto porque éste te desviaría de tu curso.



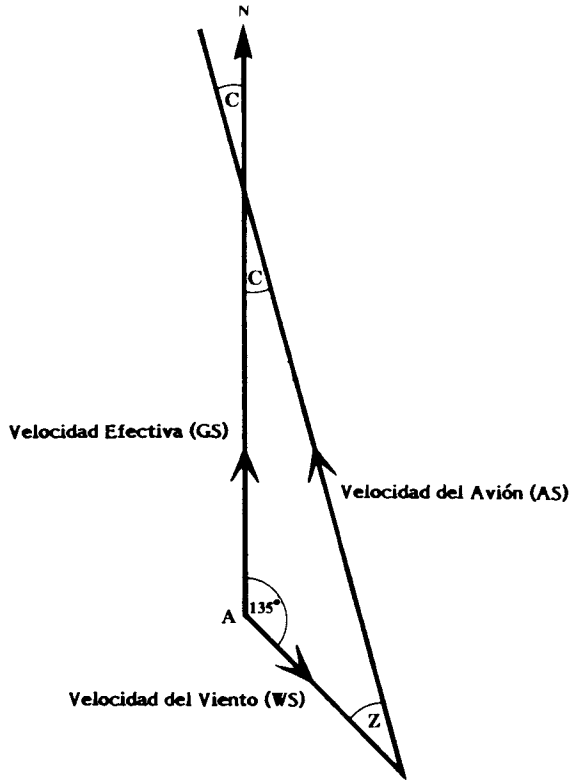
Corrigiendo el rumbo

Lo que necesitas es saber alguna forma de calcular el rumbo necesario para compensar el arrastre del viento. ¿La respuesta? Los triángulos.

¿Has oido alguna vez hablar de vectores? Un vector es una magnitud física representada por una línea que tiene una cierta dirección, sentido y un cierto valor. El viento puede representarse por un vector; tiene una dirección, e.g. sudeste, y una magnitud, e.g. fuerza 6. La velocidad de un avión también se representa por un vector, con su dirección, e.g. rumbo 156 grados, y su magnitud, e.g. velocidad 315 nudos.

Lo que tenemos que preparar es un triángulo de vectores cuyas longitudes representen velocidades y no distancias.

Observa el diagrama... El primer vector representa la dirección que queremos lograr, derecho al norte, o sea, rumbo 0 grados. Podemos trazar un segmento directamente desde nuestro punto de arranque A, pero no sabemos lo largo que debe ser, ya que no conocemos la velocidad que vamos a conseguir.



Por ejemplo, si nosotros volamos a 200 nudos estando sometidos a una racha de viento de 50 nudos que nos da en el morro directamente, sólo podremos conseguir 150 nudos con relación al terreno. Esta velocidad con respecto a la tierra se llama lógicamente la velocidad efectiva para distinguirla de la velocidad de vuelo, que solamente dependerá de la fuerza desarrollada por los motores de nuestro avión.

El segundo vector representa al viento, trazado a partir del punto A. Lo que decimos es que el viento proviene del noroeste, lo que es equivalente a decir que va hacia el sudeste. Dado que estamos más interesados en hacia dónde vamos que de dónde venimos, esa es la variable que nos interesa. Supongamos una velocidad del viento de 20 nudos. El tercer vector para completar el triángulo va a representar la dirección que debemos dar como rumbo al avión, pero que todavía no sabemos. Todo lo que sabemos es la velocidad del avión, que digamos es de 200 nudos.

Así que tenemos un triángulo y queremos hallar dos valores: primero nuestro rumbo determinado por el ángulo C, y la longitud del vector velocidad efectiva de manera que podamos calcular cuánto se tardará en el viaje. Así que para calcular el ángulo C podemos aplicar la regla del seno. Pondremos valores en la fórmula:

SENO(C)/20 = SENO(135)/200

de la cual deducimos:

SENO(C) = 20*SENO(135)/200

A continuación, usamos **grados** y calculamos la variable senc:

```
LET sinc = 20*SIN(135)/200: PRINT sinc
```

Ya tenemos la respuesta -7.07107E-02, sin preocuparnos de lo que significa. Puede que ya sea el momento de que aprendamos cómo muestra el CPC464 los números grandes y pequeños.

Cuando un número se hace muy grande o muy pequeño, se hace muy tedioso escribirlo, de manera que empleamos una forma taquigráfica que denominamos **notación científica.** En esa notación, los números se representan como multiplicados por ciertas potencias de 10, e.g. 1.000.000 se escribe 10^6 (porque 10*10*10*10*10*10 = 1.000.000); 3.000 se escribe como 3*10^3. Una potencia negativa de diez significa dividir por diez ese número de veces. De manera que 10^-6 es 1/10/10/10/10/10/10, o sea, 0.000001, y 0.000002 se convierte en 2*10^-6. El CPC464 mostraría estas cantidades como 1E6, 3E3, 1E-06 y 2E-06.

La E significa **exponente**, y en este caso la expresión es un número multiplicado por 10 elevado a la potencia señalada por el número que va detrás de la E.

De modo que nuestro número 7.07E-02 significará 7.07*10^-2 que es lo mismo que 7.07*0.01, o sea, 0.0707; sin embargo, todavía nos referimos a él simplemente mediante 'senc'. Ahora debemos convertir ese 'senc' en un ángulo. ¿Recuerdas la fórmula mágica que te dimos anteriormente? Teclea pues:

```
LET tanc = sinc/SQR(1-sinc^2):PRINT tanc
```

El valor de 'tagc' es 7.08881E-02, o sea, 0.0708881. Ahora ya podemos aplicar la función ATN, en la forma:

```
PRINT ATN(tanc)
```

y sacar como respuesta 4.05480723. El ángulo C es por tanto de aproximadamente 4.05 grados, de manera que nuestro rumbo debe ser 4.05 grados hacia el oeste de la dirección norte de referencia, o sea 355,95 grados tal y como se establece convencionalmente (los rumbos se miden en grados a partir del norte y yendo en el sentido de las agujas del reloj).

Plan de vuelo

Dado que el viento está soplando en alguna medida contra la dirección del movimiento de avance del avión, podemos esperar que nos retarde un poquito. ¿Cuánto? Debemos hallar la longitud que corresponde al vector velocidad efectiva, el lado vertical de nuestro triángulo. También aquí podemos usar la regla del seno:

gs/sen(Z) = 200/sen(135)

cambiando la igualdad para que sea:

gs = sen(Z)*200/sen(135)

Espera un momento, que todavía no conocemos el ángulo 'Z'. Claro que podemos hallarlo fácilmente porque tenemos los otros dos, y sabemos que los tres ángulos sumados han de ser 180 grados. Por lo que:

Z = 180 - (135 + 4.05)

Y por tanto:

Z = 40.95

Si ahora hacemos que PRINT SIN(40.95)*200/SIN(135) tendremos 185.375162, de manera que nuestra velocidad efectiva sobre el terreno es un poco superior a 185 nudos. El viento frena al avión en aproximadamente 15 nudos. (Un nudo es una milla náutica por hora, que es un poquito más rápido que las millas por hora ordinarias y establecidas). Podemos usar la regla del coseno para calcular la velocidad efectiva:

$$gs = \sqrt{as^2 + ws^2 - 2as.ws.\cos(Z)}$$

O dicho en BASIC sería:

```
LET GS=SQR(200^2+20^2-(2*200*20*COS(40.95)))
```

lo que nos da 185.358424, que es aproximadamente lo mismo que antes. En la línea anterior hay un aspecto que parece muy complejo. Podemos desglosarlo en varias operaciones más cortas. Sin embargo, la regla es recordar que los **paréntesis** pueden **anidarse**, igual que los bucles preconfinados FOR...NEXT. Cada paréntesis de apertura debe tener un paréntesis de cierre asociado. Los paréntesis más internos son sobre los que se opera en primer lugar y luego el CPC464 va buscando su camino hacia la expresión total, con los paréntesis más y más externos. Nuestro ángulo C podría calcularse en la forma:

$$c = \arctan \frac{\left(\frac{ws.\sin(Z)}{as}\right)}{\sqrt{\left[1 - \left(\frac{ws.\sin(Z)}{as}\right)^2\right]}}$$

O usando el BASIC:

```
LET C = ATN((20*SIN(135)/200)/SQR(1-
        (20*SIN(135)/200)^2)
```

Esta línea combina todas las operaciones que hemos repasado. Compruébala para ver si funciona. Debes obtener la misma respuesta, 4.05480723.

Si ahora cargas el programa titulado PLANVUEL, que es el siguiente en la datocassette A y lo ejecutas, comprobarás que todo eso que hemos comentado hasta ahora está incorporado dentro de ese programa. El listado se muestra a continuación.

```
100 '
110 ' Plan de Vuelo
120 '
130 ' por Ian Padwick
140 ' amended by DA 21/9/84
```

```
150 '
160 ' Pantalla
170 '
180 MODE 2:BORDER 20
190 INK 0,20:INK 1,0
200 LOCATE 34,3:PRINT CHR$(24) ; STRING$(17,32)    ; CHR
$(24)
210 LOCATE 34,4:PRINT CHR$(24) ; '  Plan de Vuelo  ' ; C
HR$(24)
220 LOCATE 34,5:PRINT CHR$(24) ; STRING$(17,32)    ; CHR
$(24)
230 WINDOW 5,79,8,24
240 '
250 DEG
260 CLS
270 '
280 ' Velocidad y direccion al azar
290 '
300 ws=ROUND(RND*50)
310 wd=ROUND(RND*359)
320 '
330 ' Obtener velocidad
340 '
350 PRINT:PRINT'Velocidad del viento:';ws;'nudos'
360 PRINT:PRINT'Direccion del viento:';wd;'grados'
370 PRINT: INPUT'Escribe la velocidad de tu avion:',as
380 IF as<12 THEN PRINT '/Muy baja!' : GOTO 370
390 '
400 ' Hacer calculos
410 '
420 IF wd>=180 THEN wd=wd-180 ELSE wd=wd+180
430 sinc=ws*SIN(wd)/as
440 c=ATN(sinc/SQR(1-sinc^2))
450 IF c<0 THEN co=ABS(c) ELSE co=360-c
460 co=ROUND(co,2)
470 IF wd=180 OR wd=0 THEN gs=as+ws*COS(wd):GOTO 510
480 IF wd>180 THEN z=180-((360-wd)+co)ELSE z=180-(wd+c)
490 gs=ABS(SIN(z)*as/SIN(wd))
500 gs=ROUND (gs,1)
510 t=100/(gs/60)
520 t=ROUND (t,1)
530 '
540 ' Imprimir resultados
550 '
560 PRINT:PRINT'El rumbo del vuelo es';co;'grados'
570 PRINT:PRINT'La velocidad sera de';gs;'nudos'
580 PRINT:PRINT'El vuelo tardara';t;'minutos'
590 '
```

```
600 ' Volverlo a hacer
610 '
620 PRINT:PRINT:PRINT "Pulsa una tecla....."
630 WHILE INKEY$="":WEND
640 GOTO 260
```

También aquí hay algunas palabras clave nuevas. El programa genera un vector aleatorio para el viento y te pide que le des como entrada la velocidad de vuelo del avión. Luego él te dice el rumbo de vuelo y cuánto se tardará en un trayecto de 100 millas. **Aviso...** No teclees una velocidad de vuelo inferior a la velocidad del viento, ya que obviamente nunca llegarás a alcanzar tu destino. Nuestra nueva palabra clave es ABS, que es abreviatura de 'ABSoluto' y que representa realmente la **magnitud** de un número, lo que significa que es el mismo valor prescindiendo del signo negativo o positivo que pueda tener ese número.

ABS

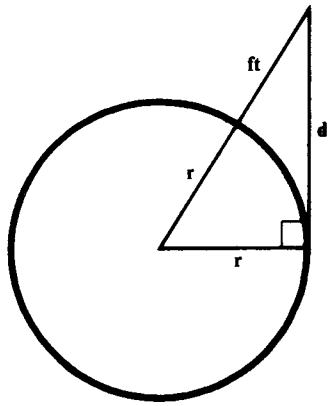
Las únicas líneas que necesitan algo de explicación son la 470 y la 480. La línea 470 trata dos casos especiales. Son cuando la dirección del viento es exactamente el norte o el sur 0 ó 180 grados). En estos casos no podemos permitir que el cálculo continúe ya que se produciría una **división por cero** al ser nulo el seno de 0 grados y el seno de 180 grados. En lugar de ese cálculo simplemente tenemos que añadir o restar la velocidad del viento. La función COS(wd) de la línea 470 sólo puede ser 1 ó -1 y por tanto la aprovechamos de manera elegante para hacer eso.

La línea 480 calcula el ángulo 'Z'. Si la dirección del viento 'wd' es mayor que 180, i.e. un ángulo convexo, entonces debemos restar ese ángulo de 360 grados para hallar el ángulo cóncavo que le corresponde.

Hasta donde puede alcanzar la vista

Pasemos ahora a otro problema. Supongamos que habiendo despegado queremos saber lo lejos que podemos ver, i.e. la distancia hasta el horizonte. Por ejemplo, ¿cuán alto tendríamos que volar para ser capaces de ver nuestro destino que está a 100 millas de nosotros? Teclea el siguiente programa y lo verás. Estudia el listado en conjunción con el diagrama y trata de determinar cómo funciona.

```
10 r=3960      'radio de la tierra en millas
20 INPUT 'Teclea altitud en pies:',ft
30 mi=ft/5280
40 d=mi+r)^2-r^2
50 d=SQR(d)
60 d=ROUND (d,2)
70 PRINT:PRINT'La distancia al horizonte es';d;'millas'
80 PRINT:PRINT:PRINT
90 GOTO 20
```



Los senos y cosenos aparecen en las matemáticas de muchos juegos. En los choques de bolas por ejemplo, gobiernan la dirección en que saldrán después de golpear una sobre la otra, y la dirección de la bola después de rebotar en el borde de la mesa de juego. También aparecen siempre que hay cosas que están vibrando o con movimientos alternativos -la velocidad de un pistón en un coche, el péndulo o balancín, o el peso en el extremo de un muelle elástico, todos esos movimientos se describen mediante senos y cosenos.

Si quieres usar un ordenador para hacer un modelo de lo que sucede en la realidad (y eso es lo que hay detrás de los mejores programas de juegos), las funciones trigonométricas son un gran aliado.

Desde luego, hay algunas otras funciones matemáticas que podríamos explicarte como LOG, LOG10, EXP, SGN, pero están muy bien explicadas en el manual.

TIEMPO DE PRIMOS

Y finalmente, aquí tienes un programa que no parece hacer absolutamente nada útil. Restaura el CPC464 y teclea lo siguiente:

```
10 MODE 0
20 DIM a(2000)
30 a(0)=2: x=0
40 FOR n=3 TO 10001 STEP 2
50 p=0
60 IF a(p)^2>n THEN x=x+1: a(x)=n: PEN 1: GOTO 90
70 IF n/a(p)=INT(n/a(p)) THEN PEN 3: GOTO 90
80 p=p+1: GOTO 60
90 PRINT TAB(8);n
100 NEXT n
```

Expone lo que podría parecer un chorro sin fin de números extraños. Aquéllos que son **primos** se exponen en amarillo; los otros en rojo. Un número primo es aquél que no puede ser exactamente dividido por ningún otro excepto por sí mismo y por la unidad. Hay un montón enorme de ellos como puedes ver, pero son más difíciles de encontrar a medida que van siendo mayores.

Aunque el programa expresa todos los valores de 'n', sólo aquéllos que resultan ser números primos son los que se cargan en la ristra o tabla de 2.000 elementos llamada 'a'. El programa usa luego esos números de la tabla... Después de todo, si un número no es divisible por ningún número primo anterior debe ser por sí mismo un numero primo también.

Cuando el programa haya finalizado, o antes si te has aburrido, puedes hacer que exponga los valores de la tabla tecleando lo siguiente:

```
FOR n=0 TO 2000: PRINT a(n): NEXT n
```

No necesita un número de línea ya que lo das como **comando.** ¿Cuántos números primos hay entre 1 y 10.000? ¿Hasta qué valor de 'n' he tenido que ir antes de haber llenado la tabla 'a'? Si dicha tabla llenara por completo la memoria del CPC464, ¿qué número primo tendríamos en el extremo final?

CONTROL DEL ESTUDIO

Esto fue bastante bonito aunque no lo pareció. Repasa los puntos esenciales de este capítulo otra vez, antes de que efectúes tu control autoevaluativo mediante TAE6.

Capítulo 7
JUGANDO CON PALABRAS

El CPC464 no es un equipo procesador de textos por sí mismo, pero sí puede tratar las **sartas** o retahílas de caracteres que al fin y al cabo son en esencia las palabras y los textos, y hacer con ellas toda clase de operaciones inteligentes. Debes recordar que cualquier dato que uses en el programa cuya vida transcurra **entre comillas** (""), o cuyo nombre termine con el identificativo dólar ($) es un dato cuya naturaleza es **literal**, por contraposición a los datos de naturaleza **numeral.**

Un **literal** no es necesariamente una palabra. Es cualquier **serie de caracteres** que esté encerrada entre comillas. Y los caracteres pueden ser letras, cifras o cualquiera del repertorio. Cuando asignas el nombre a una variable literal debe tener el sufijo dólar ($) para indicarle al CPC464 que los valores de esa variable son meras **sartas de caracteres,** y que los trate como tales. Aquí tienes algunos ejemplos, tecléalos:

```
LET a$='Amstrad'
LET c$='CPC464'
LET r$='45^%) 1ko+-CF#!} '
```

Ahora teclea:

```
PRINT a$,c$,r$
```

Dentro de un literal puedes incluir **cualquiera** de los 256 caracteres del CPC464, excepto precisamente las comillas que lo delimitan. Obviamente, no puedes hacer operaciones matemáticas directamente sobre los literales aunque sí puedes colocar entre ellos el signo más, pero es para que el resultado sea el **empalme** o concatenado de los que intervienen en la operación. Ensaya con esto.

```
LET z$=a$+'    '+c$: PRINT z$
```

En pantalla se expondrá **Amstrad CPC464.** Los contenidos de a$ y c$ que han sido empalmados con un blanco entre medias para formar z$, son ahora **subliterales** componentes de z$, i.e. literales dentro de un literal mayor que los contiene.

¿CUAN LARGO ES UN LITERAL?

Así que ¿qué podemos hacer con literales? Bien, usando las palabras clave de este capítulo serás capaz de **trocearlos** en literales más pequeños, **canjearlos, invertirlos, escrutarlos** en busca de determinados caracteres, y mucho más.

LEN

Lo primero que podemos hacer con un literal es aplicar una función para que nos diga su **longitud**, i.e. la cantidad de caracteres que componen la sarta o retahíla. Hacemos eso usando la función LEN (abreviatura de length=longitud) que tiene como sintaxis:

```
LEN (<expresion literal>)
```

Por ejemplo, PRINT LEN(a$) dará 7 porque hay siete caracteres en la sarta "Amstrad". PRINT LEN(c$) también da 7. PRINT LEN(z$) da 15. Podemos necesitar conocer la longitud de un literal por diversas razones, entre otras para permitir que un programa exponga un **letrero** (que al fin y al cabo, quizá debieramos llamar así a los **literales**) centrado en la pantalla. Por ejemplo:

```
10 INPUT "Teclea tu nombre: ",nome$
20 IF LEN(nome$)>40 THEN PRINT "Nombre muy largo":GOTO10
30 l=LEN(nome$)
40 PRINT TAB(20-l/2);nome$: PRINT
50 GOTO 10
```

Habrás observado que cuando expones una lista de números que tienen diversa "longitud", o sea, diverso número de cifras, dichos números aparecen expuestos a partir de la izquierda, de manera que las primeras cifras quedan alineadas. Sin embargo, cuando escribimos columnas de números habitualmente queremos que las unidades, decenas, centenas, queden **alineadas.** Por ejemplo, lo que se consigue es:

```
1
12
97
4032
255
```

y normalmente lo que se quiere es:

```
   1
  12
  97
4032
 255
```

Una posible forma de realizar esto sería hallar la longitud de la **sarta de cifras** con que vemos el número, y luego calcular la posición de TABulación que necesitamos, como en el ejemplo anterior. Pero observa que si mandas al ordenador que PRINT LEN(n), donde 'n' es el **número** que necesitas, te da el mensaje:

```
Type mismatch
```

para recordarte que hay una **discordancia de tipo.** Eso es debido, insistimos, a que numerales y literales son dos clases de datos de diferente índole o naturaleza (aunque en ciertos literales que son meras sartas de cifras, puedan parecer a primera vista como números). De hecho, los **números** no se almacenan en memoria de la misma manera que los **"líteros"**, ya que para lo números no tienes celdillas individuales para cada una de las cifras que lo componen, mientras que para los **líteros** a cada carácter le corresponde una celdilla. Para solventar eso debemos cambiarle la naturaleza a nuestro **numeral** y convertirlo en un **literal** (y cuando los expongas en pantalla no se observará la diferencia). La palabra clave para la función que efectúa esta conversión es STR$, que es abreviatura del inglés 'string' que significa cuerda como en las guitarras, o sartas como en los collares de perlas (pero no cadena, que sería 'chain'). La sintaxis de esta función es pues:

STR$

```
STR$(<expresion numerica>)
```

Ahora podemos asignar n$=STR$(n) y mandar PRINT LEN(n$) o podemos hacerlo todo de una sola vez, PRINT LEN(STR$(n)). Recuerda que cuando el CPC464 expone un dato **numeral** siempre coloca un espacio precediendo a las cifras de ese numeral y otro espacio inmediatamente detrás. El espacio delantero realmente es la posición del **signo,** y no sale en pantalla cuando se trata de positivos. Los números negativos tienen desde luego el correspondiente signo menos. La aplicación de la función STR$ automáticamente prescinde del espacio delantero que va detrás, pero sigue manteniendo el espacio previsto para el signo; por lo tanto cuando halles la **longitud** de un literal que es conversión de un numeral, la respuesta siempre sobrepasa en una unidad al número de cifras que tú aprecias en el número. Por ejemplo:

```
PRINT LEN( STR$(464))
```

dará como respuesta el cuatro y no el tres.

Eso no causará ningún problema, siempre y cuando lo tengas en cuenta. Más adelante ya veremos cómo también podemos quitar el espacio de ese signo delantero.

Ahora podemos escribir un breve programa para conseguir nuestra meta:

```
10 INPUT "Teclea un numero: ",n
20 l=LEN( STR$(n))
30 PRINT TAB(20-l);n
40 GOTO 10
```

NUMEROS ENSARTADOS

Queda claro que podemos cambiar la naturaleza de un dato numeral y considerarlo como **literal.** ¿Qué pasa con la conversión en el sentido contrario? Desde luego la podemos hacer.

VAL

La función de clave VAL, abreviatura de 'value'=valor, nos permite convertir un literal, que es una sarta de cifras, a un dato **numeral,** que puede tratarse realmente como un **valor** numérico cuantificable, sumable, restable, etc. La sintaxis de esta función es:

```
VAL(<expresion literal>)
```

Apliquemos esa función a nuestra variable a$="Amstrad". ¿Qué **valor** puede tener ese literal? Inténtalo tecleando:

```
PRINT VAL(a$)
```

Era obvio, no puede ser considerado como un numeral en ningún aspecto. Eso nos recalca que la **expresión literal** debe estar formada por caracteres numéricos -por **cifras**- o tener caracteres numéricos por lo menos al principio de la retahíla de caracteres, y son esos los únicos que consideraremos en la conversión a numeral, al aplicar la función **valor.**

Teclea ahora:

```
n$="65536"
```

y luego:

```
PRINT n$:PRINT VAL(n$)
```

En pantalla aparece un 65536 expuesto en el borde izquierdo y por debajo de él un 65536 simplemente con un espacio en blanco por delante. La diferencia a simple vista es muy pequeña, pero son dos cosas completamente diferentes: la primera es un **literal,** la segunda es un **numeral.** VAL(n$) aplicado a un dato literal adecuado se convierte en un **numeral** con el que podemos operar. Teclea:

```
PRINT VAL(n$)/256*4
```

y obtendrás naturalmente 1024 como respuesta.

Ensaya aplicando la función VAL sobre estas constantes literales: "CPC464", "464 CPC", "-4 64Am".

Convertidos a numerales son respectivamente, 0, 464, -464.

¿Cómo podemos usar esta función? Supongamos que tenemos un programa que pide se le ingrese un número por teclado y el operador por equivocación coloca una letra (o quizás deliberadamente). Aparecerá el mensaje:

```
?Redo from start
```

para avisarnos que **vuelve a hacerlo desde el principio,** aunque nos haya destrozado la imagen. Teclea:

```
10 INPUT "Teclea un numero (1 al 9): ",n
20 IF n<1 OR n>9 GOTO 10
30 PRINT n
40 GOTO 10
```

Ejecuta este programa y cuando te lo solicite teclea una letra. ¿Ves lo que pasa? Ahora, si en la instrucción INPUT mencionáramos en lugar de eso una **variable literal,** podríamos atrapar el error cometido en la operación, usando la función VAL. Cambia pues la variable numeral 'n' en las líneas 10 y 30 para que sea una variable literal 'n$', y luego cambia la línea 20 para que sea:

```
20 IF VAL(n$)<1 OR VAL(n$)>9 GOTO 10
```

Ahora al teclear una letra no se estropeará el programa.

Supongamos que la banda de números no fuera del 1 al 9, sino del 0 al 9. Tecleando una letra obtendríamos al aplicar VAL(n$) el valor 0, que en estas circunstancias debiera ser válido. ¿Cómo podemos decir entonces si se ha tecleado un 0 o una letra? Para eso disponemos de otra palabra clave, ASC, que nos da el **código** ASCII que está íntimamente asociado a cada carácter del repertorio (ASCII corresponde a American Standard Código para Información Intercambio).

Esa es una de las pocas cosas que la mayoría de los micros tienen en común: un repertorio estándar de caracteres según la norma ASCII. Cada carácter del CPC464 tiene asociado un código ASCII, y viceversa. Desde los códigos ASCII del 32 al 126, los caracteres son **visivos** y son las mismas cifras, letras, signos que se usan en la mayoría de los ordenadores.

ASCII Y ASCII

Al aplicar pues la función ASC a un **carácter** nos entrega el valor del **código** ASCII de ese carácter (y si lo aplicamos a una expresión literal, nos entrega el código del primer carácter -de la **inicial**- de esa expresión. La sintaxis es pues:

```
ASC(<expresion literal>)
```

Teclea por ejemplo:

```
PRINT ASC("0")
```

y obtendrás 48, que es el **código** ASCII del carácter 0. Teclea ahora:

```
PRINT ASC("Amstrad")
```

y obtienes 79 que es el código ASCII de la letra **mayúscula** O, que es diferente obviamente al de la letra minúscula o, cuyo código es el 111 -basta sumar 32-. Así que cambiemos el 1 en la línea 20 por un 0 y añadamos:

```
25 IF VAL(n$)=0 AND ASC(n$)<>48 GOTO 10
```

Si ahora trabajas con ese programa encontrarás que se aceptan los números 0 al 9 únicamente pero que todo lo demás se rechaza. Eso no es exactamente cierto, yo creo... tienes razón, si tecleas el literal "7Amstrad" todavía será aceptado el 7. ¿Cómo podemos impedir eso? Bien, únicamente queremos que se teclee un **único** carácter, así que podríamos poner esa condición en la línea 20, en lugar de la que hay, es decir, teclea:

```
20 IF LEN(n$)>1 THEN GOTO 10
```

O desde luego, podemos usar la función INKEY$ que **indaga** la tecla que se haya pulsado y nos entrega como resultado un **carácter**. Tú mismo puedes comprobar cómo se hace eso.

AND

OR

XOR

Aquí se han usado otras dos palabras clave más que son **conectivos** (y en este caso, **conjunciones**) con el sentido de **...y si...** (AND) y la **...o si...** (OR). Su uso es así autoexplicativo, pero hay que hacer una observación sobre el OR. Normalmente decimos "si a=10 o si b=20 entonces c=30" queriendo decir que haremos c=30 cuando a=10 o también cuando b=20, **pero no** cuando a=10 y también b=20. Esa condición es lo que se conoce como una conjunción **o exclusiva** (XOR=exclusive OR). Esta nueva palabra clave XOR que también es una conjunción, nos permitiría decir "si lo uno o si lo otro, pero no ambos a la vez, entonces...". Así la línea:

```
IF n>50 XOR n<73 THEN PRINT n
```

sólo expondría el valor de 'n' si una y solamente una de las condiciones fuera cierta, pero no lo haría si ambas fueran ciertas.

La conjunción OR nos permite decir "si lo uno o si lo otro, o si los dos a la vez, entonces...". Así pues, la línea:

```
IF n>50 OR n<73 THEN PRINT n
```

expondría el valor de 'n' si la primera de las condiciones, la segunda o ambas a la vez eran ciertas.

Finalmente sobre este tema, supongamos que sólo queremos aceptar los números del 1 al 5, por ejemplo como elección en un menú. Las siguientes cuatro líneas de programa efectúan todas la misma comprobación pero de diferentes maneras:

```
20 IF VAL(n$)<1 OR VAL(n$)>5 THEN...incorrecto
20 IF VAL(n$)>0 AND VAL(n$)<6 THEN...  correcto
20 IF ASC(n$)<49 OR ASC(n$)>53 THEN...incorrecto
20 IF ASC(n$)>48 AND ASC(n$)<54 THEN...  correcto
```

LO CONTRARIO DE NUMERALES

CHR$

La función opuesta a ASC es la de clave CHR$ con la que te has encontrado antes. La función CHR$ se aplica a un argumento **numeral** y da como resultado el **carácter** literal cuyo código ASCII es dicho numeral. Su sintaxis es:

```
CHR$(<expresion numerica>)
```

El siguiente programa expresa por pantalla todo el repertorio de caracteres:

```
10 FOR n=0 TO 255
20 PRINT CHR$(1);CHR$(n);
30 NEXT n
```

Mandar que se exponga CHR$(1) por sí solo, no tiene ningún efecto. Es lo que se llama un **carácter de control**, (que efectúan acciones y no son visivos) y permite que sean expuestos los caracteres cuyos códigos van del 0 al 31. Si no lo usáramos, esos códigos serían inaccesibles. Una tabla de esos caracteres de control, puede verse en la página 2 del Capítulo 9 de la Guía del Usuario Amstrad CPC464.

IZQUIERDA, DERECHA Y EN EL MEDIO

En el Capítulo 4 de este libro, hay un programa llamado RELOJDIG que expone las decenas y unidades de las horas y minutos separadamente, pero lo hace de manera que parecen iguales a números de dos cifras. Recuerda que de ordinario no se puede exponer un número inmediatamente adosado al otro, debido a que el espacio para el signo del segundo hace que se deje un cuadratín entre ellos. Sabemos que podemos usar la función STR$ para convertirlos a literales y eliminar así el espacio trasero al exponer números, pero ¿cómo eliminamos ese espacio delantero para el signo?

LEFT$

RIGHT$

MID$

Hay tres funciones literales que el Amstrad BASIC usa para resolver ese tipo de cosas: las tres se usan para **segregar** partes de un literal. La de clave LEFT$ lo hace tomando los caracteres de la **izquierda**, la de clave RIGHT$ lo hace tomándolos por el extremo **derecho**, y la de clave MID$ los segrega de la parte de en**medio**, que puede llegar hasta el principio y hasta el final. La sintaxis para RIGHT$ es la misma que para LEFT$:

Con LEFT$ y RIGHT$ la **expresión entera** define el número de caracteres que quedarán incluidos en el subliteral segregado, contándolos desde la izquierda o desde la derecha del correspondiente literal matriz dado por la **expresión literal.**

La sintaxis para segregar un sub-literal, con los caracteres **medianeros**, es:

Con MID$, la primera **expresión entera** define el primer carácter del subliteral segregado, y la segunda -que es opcional- define la cantidad de caracteres contados a partir de ese primer carácter. Aquí tienes algunos ejemplos de este método de definición, suponiendo que z$="Amstrad CPC464":

```
PRINT RIGHT$(z$,3)     es el subliteral 464
  '   RIGHT$(z$,6)     es el subliteral CPC464
  '   RIGHT$(z$,14)    es el subliteral Amstrad CPC464
  '   LEFT$(z$,3)      es el subliteral Ams
  '   LEFT$(z$,9)      es el subliteral Amstrad C
  '   LEFT$(Z$,14)     es el subliteral Amstrad CPC464
  '   MID$(z$,4)       es el subliteral trad CPC464
  '   MID$(z$,4,4)     es el subliteral trad
  '   MID$(z$,9,3)     es el subliteral CPC
```

Se puede extraer cualquier porción de un literal usando estas funciones ¿y de qué vale? Bien, eso depende de lo que quieras hacer. Veamos cómo quitarnos ese espacio delantero para el signo. Si 'n' es un número de una sola cifra, podríamos aplicar cualquiera de estas funciones:

```
PRINT RIGHT$(STR$(n),1)
```

o bien

```
PRINT MID$(STR$(n),2)
```

Si 'n' es un número de varias cifras, deberemos aplicar además la función de **longitud.** Ensaya este programa:

```
10 CLS: PAPER 3
20 INPUT "Teclea un numero: ",n
30 PRINT n
40 PRINT RIGHT$(STR$(n),LEN(STR$(n))-1)
50 GOTO 20
```

El propósito de cambiar el papel es el de que puedas ver los espacios más fácilmente. En la línea 40, STR$(n) **es** la expresión literal sobre la que segregar y LEN(STR$(n))-1 es la expresión entera. Le restamos 1 dado que LEN(STR$(n)) sería mayor en una unidad que el número de cifras que hay en 'n' - ¿lo recuerdas?.

Las siguientes líneas de programa pueden reemplazar las que con los mismos números ya están en RELOJDIG. Así que carga el programa, y luego teclea:

```
180 t=TIME
1070 LOCATE 13,10:PRINT "0"
1080 WHILE s<60
1090 s=INT((TIME-t)/300)
1100 s$=STR$(s)
1110 LOCATE 14+(s>9),10
1120 PRINT RIGHT$(s$,LEN(s$)-1)
1130 WEND:s=0:t=TIME
```

TIME

Estas líneas sustituyen al bucle de los segundos y aprovechan el propio reloj interno del CPC464, que es accesible a través de la palabra clave TIME (tiempo horario).

Puedes ver que la línea 1120 elimina el espacio delantero y que s$ ha sido previamente definida como igual a STR$(s) en la línea 1100. La variable 's' representa la diferencia entre el TIME actual y el valor de 't' que tenía TIME cuando se comenzó el **crono**metraje (línea 180). La diferencia se incrementa en 300 por cada segundo transcurrido, de aquí la línea 1090.

El bucle condicionado WHILE...WEND asegura que 't' se actualiza cada 60 segundos con el nuevo valor de TIME.

Observa la línea 1110. Aquí se usa un concepto interesante. La condición entre paréntesis s 9, se evalúa para ver si es cierta, o bien si es falsa: o bien 's' es mayor que 9 o no lo es. Si es cierta, el valor de la condición es -1, si es falsa su valor es 0. Ese valor se usa entonces para mover el cursor hasta la columna 14 si 's' es un número de una sola cifra; o hasta la columna 13 si es de dos cifras. Todas las condiciones reciben valores de 0 ó -1, dependiendo de si son falsas o ciertas. Para comprobarlo teclea lo siguiente:

```
v=17
```

y luego:

```
PRINT v>4;v<20;v<2;v>50;v=17;v*2>50;SQR(v)>
      4;v/2<v/3;v+3=20
```

Obtendrás:

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

Estos **valores lógicos** pueden ser muy aprovechables cuando conozcas un poco más de programación, pero no te preocupes demasiado sobre ellos por ahora.

ESCRUTINIO DE PALABRAS

INSTR

Aquí hay otra nueva función para ti, de clave INSTR que corresponde a "in string"="(está) en literal". Eso te permite escrutar un literal para ver si un determinado subliteral pertenece o no al literal escrutado. El número que resulta de aplicar la función es la **posición** del primer carácter del subliteral cuando éste pertenece totalmente al literal escrutado. La sintaxis es:

```
INSTR([<expresion entera>,]<expresion literal>,
     <expresion literal>)
```

La **expresión entera** es opcional y determina la posición dentro del literal a partir de la cual ha de empezar el escrutinio. Si se omite, se comienza desde el principio del literal. La primera **expresión literal** es el literal que se ha de examinar, mientras que la segunda es el **sub-literal** que va a comprobarse si pertenece al literal primero. De manera que si es z$="Amstrad CPC464" entonces PRINT INSTR(z$,"d") nos entrega el 7, dado que al **ponerlos en correlación** hay coincidencia a partir del séptimo carácter del literal escrutado.

```
PRINT INSTR(z$,"CPC")
```

Eso expondría un 9 porque la posición donde coinciden aparece a partir de la posición 9 del literal escrutado.

```
PRINT INSTR(z$,"4")
```

Nos daría un 12, que es la posición del primer "4"; y que luego podemos aprovechar para hacer que se pongan en relación a partir de esa posición y conseguir así el segundo "4" mediante:

```
PRINT INSTR(13,z$,"4")
```

Así obtendríamos como respuesta el 14, porque se pusieron en **correlación** los dos literales a partir del decimotercer carácter de z$.

Esta función literal se aprovecha mucho en el programa del "Juego del Ahorcado" que vimos en la Parte 1 de este Curso Autodidáctico. El siguiente programa es un ejemplo de la manera en que se usó la función INSTR:

```
10 CLS
20 LOCATE 1,1: INPUT "Escribe una palabra: ",w$
30 w$=UPPER$(w$)
40 LOCATE 1,3: PRINT "Adivina una letra: "
50 l$=INKEY$
60 IF l$="" GOTO 50
70 l$=UPPER$(l$)
80 LOCATE 1,3: PRINT SPACE$(16)
90 p=1
100 p=INSTR(p,w$,l$)
110 IF p=0 GOTO 40
120 LOCATE 10+p,10: PRINT MID$(w$,p,1)
130 p=p+1: GOTO 100
```

Observa que este programa no se detendrá incluso aunque hayas acertado todas las letras. Puedes desarrollar cómo conseguir que pare. Aquí hay también otra nueva palabra clave, correspondiente a la función que nos permite convertir todos los caracteres alfabéticos (A-Z) dentro del literal sobre el que se aplica a **mayús**culas, y tiene por clave UPPER$ (para reflejar que están en la "caja de más **arriba**" en las antiguas linotipias).

UPPER$

Los últimos serán los primeros

El programa final que te presentamos ahora demuestra algunas operaciones posibles con literales. Esta vez, todo lo que se teclee aparecerá expuesto **revirado.**

Seguro que eres capaz de comprender cómo lo efectúa.

```
10 PRINT 'Escribe un literal: '
20 INPUT w$
30 FOR n=LEN(w$) TO 1 STEP -1
40 b$=b$+MID$(w$,n,1)
50 NEXT
60 PRINT '  ';b$: PRINT: b$=''
70 GOTO 10
```

Ensaya -aunque quizá no lo compruebes del todo- con la conocida frase "dábale arroz a la zorra el abad".

TIEMPO DE JUEGO

Nuestro juego esta vez es muy apropiado. Es una ayuda para los juegos de palabras en los periódicos donde tienes que construir montones de palabras diferentes a partir de las letras de una muy larga. Así que carga SOPALEtras y disfruta durante un rato.

CONTROL DEL ESTUDIO

Ahora toca el control TAE7. Nunca se sabe, pero puede que ya estés capacitado para escribir un programa que resuelva los crucigramas del periódico de tu pueblo.

Capítulo 8
FIGURAS ANIMADAS

El principio de hacer **figuras que se mueven** ha estado rondando durante un montón de tiempo. Uno muestra una figura quieta, luego la reemplaza rápidamente con otra que tiene ligeras diferencias, luego esa la reemplaza con otra, y así sucesivamente. Si eso se hace rápida y uniformemente, las leves diferencias entre cada una de las sucesivas exposiciones de esa figura se hacen imperceptibles y aparentemente se consigue una figura **animada.**

TITILANTE Y CENTELLEANTE

Una de las maneras más simples de añadir animación a un mensaje en pantalla o a una imagen gráfica es usar un comando de **tinte** que especifique dos colores. La sintaxis del comando es:

```
INK <numero tinte>,<color>[,<color>]
```

Cuando se incluye el segundo parámetro de **color** el CPC464 alternativamente usará la tinta de esos dos colores.

Ensaya tecleando directamente después de un encendido o un restauro general:

```
INK 1,24,1
```

Te da una imagen parpadeante, ¿o no? Eso es porque el segundo color es el mismo que el usado como tinta preceptuada implícitamente para el **papel.** No es que la imagen se apague durante la mitad del tiempo -es simplemente que estás escribiendo con una **pluma** azul sobre un **papel** azul. Se pueden especificar dos colores cualesquiera, así que ensaya mandando diferentes parejas de colores de tu propia elección y será obvio lo que está sucediendo.

La rapidez con la que se alternan los dos colores puede cambiarse mediante un nuevo comando de "aceleración de tintes". Su sintáxis es:

SPEED INK

```
SPEED INK <expresion entera>,
<expresion entera>
```

El primer parámetro especifica el intervalo de tiempo en que aparece en pantalla con el primer color de tinta, y el segundo corresponde al segundo color. Estas **expresiones enteras** representan intervalos múltiplos de 0,02 segundos.

Aquí hay un breve programa que ilustra lo anterior:

```
10 rem Mensaje Parpadeante de programa
20 cls
30 mode 0
40 ink 1,24,1
50 for a=50 to 1 step -1
60 locate 3,12
70 speed ink a,a
80 print 'MENSAJE PARPADEANTE'
90 for x=1 to 200:next
100 next a
110 goto 10
```

Hacer que un texto o un gráfico **parpadee** es inmensamente útil cuando quieres atraer la atención a una parte concreta de la imagen de pantalla.

En el MODO 0, las tintas de números 14 y 15 están preceptuadas como de colores dobles y alternando: la 14 lo hace entre azul y amarillo brillante, y la 15 entre azul cielo y rosa.

ANIMACION DE FIGURAS

Cuando estás escribiendo programas para juegos, el ingrediente realmente esencial es el movimiento. También pueden usarse gráficos y textos centelleantes, pero la esencia completa de esta clase de programas es la batalla entre elementos mostrados gráficamente que el jugador puede controlar y otros que él no puede controlar. Así que necesitamos saber cómo conseguir que nuestras **figuras** sean auténticamente "objetos movibles" por toda la pantalla.

Para animar una figura, no es necesario sustituir toda la imagen completa cada vez. Unicamente aquellas partes que se mueven necesitan cambiarse entre las sucesivas escenas del juego. La animación por ordenador trabaja de esta manera. La imagen global no se vuelve a trazar cada vez que alguna parte de ella cambia, en lugar de eso sólo se cambia la parte que se mueve. El movimiento se consigue **borrando** la vieja posición ocupada por el objeto, y luego volviéndolo a **exponer** en una nueva posición.

Otra manera de conseguir la impresión de movimiento es trazar las figuras que se mueven en todas sus posiciones, pero hacerlo de manera que queden ocultas, que no puedan verse, e.g. pintarlas con el mismo color del **fondo.** Luego, cuando se requiere el movimiento, se puede **conmutar** el color de esa posición sucesivamente entre el color del **frente** y el color del **fondo;** es decir, no volviéndola a dibujar, sino cambiando el color de la tinta usada en el momento original. Eso puede quedar demostrado si ahora tecleas el programa que aparece en la página 32 del Capítulo 3 de este curso.

Asegúrate que trabajas en el MODE 0 y luego añade las siguientes líneas extra (más fácil en MODE 1 ó 2).

```
110 FOR n=2 TO 15
120 INK n,1
130 NEXT n
140 FOR n=1 TO 15
150 INK n,24
160 INK n-1,1
170 FOR x=1 TO 80:NEXT x
180 NEXT n
190 INK 15,1
200 GOTO 140
```

Asegúrate que estás en MODE 0 cuando ejecutes el programa. Verás los discos trazados como es usual, pero todos excepto el primero desaparecerán y luego el primero aparentará que se mueve a través de la pantalla rápidamente una y otra vez. Cuando ya estés harto -y pulses ESCape por dos veces, es lo más probable que quedes con texto azul sobre papel azul. Manda cuidadosamente INK 1,24 y volverá a aparecer el texto. Ahora estudia el listado. Deberías ser capaz de comprender cómo funciona, pero si no es así, simplemente cambia los valores en donde creas oportuno y observa lo que sucede. Haciéndolo así las cosas quedarán más claras.

Dada la manera en que el CPC464 organiza sus colores, puedes hacer eso con cualquier carácter o símbolo expuesto mediante PRINT o pintado mediante PLOT.

Pies danzantes

El único problema con el método de animación conseguido por el cambio de color es que está limitado en alguna manera por el número de tintes disponibles. También debes tener cuidado en no trazar una figura sobre otra oculta, que vayas a usar posteriormente.

Por esta razón, el método preferido para hacer que las figuras se muevan es **exponerlas** y **borrarlas** alternativa y sucesivamente.

Comencemos con la clase más sencilla de movimiento: un hombrecillo danzando en la pantalla. El CPC464 está provisto de cuatro símbolos que pueden usarse directamente para este propósito. Vienen dados por los códigos numéricos 248-251, inclusives, y puedes ver los caracteres correspondientes en la Guía del Usuario Amstrad CPC464 Apéndice 3, páginas 12 y 13. Aquí hay un ejemplo extremadamente sencillo de cómo pueden utilizarse:

```
10 rem Programa Danzarin
20 cls:mode 0
30 data 248,250,249,251
40 for a=1 to 4:read danza(a):next
50 b=1
60 for c=1 to 4
70 locate b,12
80 print chr$(danza(c));
90 for x=1 to 500:next
100 locate b,12:print " ";
110 b=b+1
120 next c
130 if b<20 then goto 60
140 goto 50
```

La primera cosa que hacer es cargar una tabla, de nombre "danza" con el valor ASCII de esos caracteres. Eso se hace en el mismo orden en que luego van a ser usados. Luego, dentro del bucle condicionado por 'b', que mediante la línea 130 puede variar desde 1 hasta 20, tenemos anidado un bucle preconfinado que contado por 'c' efectúa la animación del hombrecillo. Este se expone en la siguiente coordenada 'x' de la pantalla usando el valor corriente de 'b' como parámetro del comando de situar LOCATE.

Después de exponer los caracteres dados por el valor corriente de la tabla, hay un tiempo de **demora** mediante un bucle del 1 al 500. A continuación se expone en la misma posición un **blanco** para quitar el carácter de la pantalla y antes de moverlo al siguiente cuadratín.

Cuando hayas tecleado y ejecutado el programa, quita la línea 100 y ejecútalo de nuevo. Eso te mostrará en primer lugar por qué tuvimos que incluir esa línea. También puedes experimentar con la duración del bucle de demora estipulado en la línea 90.

¿De dónde provienen todos esos?

La Parte 1 de este curso tenía un juego de acción que trataba de bombarderos, estaba diseñado para enseñarte los fundamentos de la geometría de coordenadas, al pedirte que estimaras la posición de un invasor alienígena. Si recuerdas, esa pequeña criatura aparecía continuamente en diferentes sitios de la pantalla aprovechando la función de azar RND. Esta función es esencial para incluir un elemento no predecible en los movimientos de caracteres y gráficos.

RND

Las siguientes líneas harán que aparezca en la pantalla la mayor flota especial que puedas haber visto.

```
10 rem Invasion Espacial
20 cls:mode 0
30 locate 19*rnd+1,24*rnd+1
40 print chr$(239);
50 for demora=1 to 999*rnd
60 next
70 goto 30
```

La cosa interesante es que los navíos espaciales también aparecen según intervalos aleatorios de tiempo, merced a la inclusión de la función RND en el bucle de retardo de la línea 50.

LOCATE

Observa que has tenido que añadir un **uno** a ambos parámetros del comando de situación LOCATE. Si no lo hicieras así, el producto de 19*rnd o 24*rnd podría algunas veces ser menor que uno y el CPC464 pararía el programa, dando un mensaje de error de 'inapropiado argumento', dado que LOCATE sólo puede aceptar **parámetros** mayores o iguales a uno.

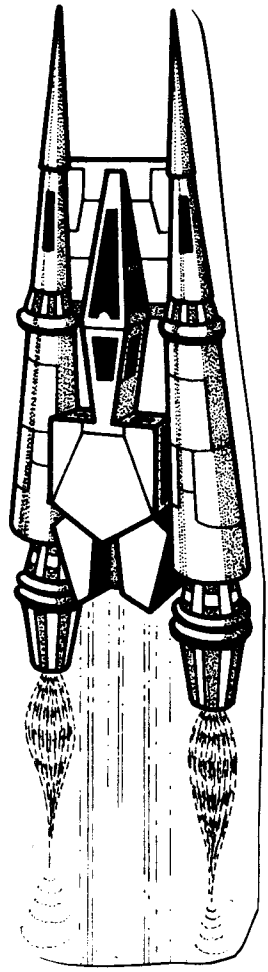
Uno de cada ocho

Supongamos que nuestro pequeño danzarín se ha pasado un poquito, pero todavía intenta mantener una línea recta a través de la pantalla. Podemos simular eso añadiendo al programa:

```
65 d=int(3*rnd+5)
```

Como recordarás, la función INT asegura que el resultado de la expresión numeral que toma como argumento es un número **entero** obtenido mediante el **truncamiento** de cualquier parte fraccionaria. Las manipulaciones de la función de azar RND al multiplicarla por 3 y añadirle 5 sólo están ahí para que las expresiones resultantes estén dentro de la banda deseada.

INT



Esta nueva variable 'd' puede añadirse ahora al programa para que refleje el segundo parámetro de los dos comandos de situación LOCATE, tal y como sigue:

```
10 rem Programa Ebrio
20 cls:mode 0
30 data 248,250,249,251
40 for a=1 to 4:read danza(a):next
50 b=1
60 for c=1 to 4
65 d=int(3*rnd+5)
70 locate b,d
80 print chr$(danza(c));
90 for x=1 to 500:next
100 locate b,d:print " ";
110 b=b+1
120 next c
130 if b<20 then goto 60
140 goto 50
```

Si ahora pasas esta versión modificada del programa, te encontrarás que nuestro danzarín no se sostiene muy firmemente sobre sus pies.

OTRO LADRILLO EN LA PARED

Uno de los juegos primitivos para ordenador consistía en "derribar muros de ladrillos". El jugador tenía que mover un "bate" adelante y atrás por la parte inferior de la pantalla para golpear hacia arriba una "pelota rebotante" contra una "pared" para irse cargando los "ladrillos" que la componen. Desarrollaremos ahora este programa en bastante extensión para demostrar algunos de los principios de programación de juegos y presentar a medida que lo hacemos algunas nuevas palabras clave.

Asegúrate que has restaurado las condiciones preceptuadas inicialmente en tu CPC464 y tienes alguna cinta en blanco o a estrenar para guardar el programa.

Lo primero que hay que hacer es trazar los límites de la **cancha** de juegos. Podemos simplemente usar el **reborde**, pero por razones que serán más claras posteriormente y porque queremos, no lo haremos. Los números de línea están puestos a propósito, así que no los cambies todavía:

```
10 MODE 1:CLS:GOTO 200
200 LOCATE 1,1: PRINT CHR$(136);
210 FOR c=2 TO 39
220 LOCATE c,1: PRINT CHR$(140);
230 NEXT
240 LOCATE 40,1: PRINT CHR$(132);
250 FOR 1=2 TO 24
260 LOCATE 1,1: PRINT CHR$(138);
270 LOCATE 40,1: PRINT CHR$(133);
280 NEXT
300 LOCATE 1,25: PRINT CHR$(130);
310 FOR c=2 TO 39
320 LOCATE c,25: PRINT CHR$(131);
330 NEXT
340 LOCATE 40,25: PRINT CHR$(129);
350 LOCATE 1,1
```

La línea 350 se ha incluido como una medida temporal para hacer que la imagen no se **deslice** con respecto a la pantalla.

No debieras tener problemas con este programa. Por el momento, el mensaje de **Preparado** "Ready" se interpondrá en el camino, pero no lo hará posteriormente. Lo que queremos ahora es una pelota que rebote...

```
20 x=1: y=-1
30 c=INT(RND*37+2)
40 l=INT(RND*22+2)
50 IF c+x>39 OR c+x<2 THEN x=-x
60 IF l+y>24 OR l+y<2 THEN y=-y
80 LOCATE c,l: PRINT " ";
90 c=c+x:l=l+y
100 LOCATE c,l: PRINT CHR$(224);
110 GOTO 50
350 GOTO 20
```

Cuando ejecutes el programa, verás una pelota que rebota dentro de la cancha establecida. Bastante rápida ¿no? Si quieres bajarle la velocidad un poquito, puedes insertar temporalmente la línea:

```
105 FOR d=1 TO 50: NEXT
```

Las líneas 20 a 40 prefijan los valores iniciales para las variables 'x' e 'y' (usadas para desplazar la bola), y 'c' y 'l' (la posición de la bola). Las líneas 50 y 60 computan si la siguiente posición de la bola va a caer dentro o fuera de los límites. (Estamos usando las columnas 2 a 39 y los renglones 2 a 24). Si así fuera, entonces el incremento pertinente de 'x' o de 'y', sería **invertido.**

La línea 80 **sitúa** el cursor en la vieja posición ocupada por la pelota y **expone** un espacio en blanco en esa posición, con lo que se borra la figura; la línea 90 calcula la nueva posición y la línea 100 expone la pelota después de situarse en esa nueva posición. La línea 110 completa el bucle. Bastante simple.

Si eres capaz ahora de hacer que se **guarde** este programa en cinta, hazlo porque vamos a cambiarlo un poquito, pero volveremos de nuevo sobre él posteriormente.

Suavizando los modales

Cada nueva posición de la pelota está necesariamente sobre un **cuadratín** completo, de los ocupados por cada carácter, a partir de la posición anterior. Aunque la velocidad de la bola ayuda a que el movimiento parezca suave y sin espasmos, no lo es tanto como uno pudiera desear. Para obtener movimientos más suavizados necesitaríamos hacer que se expusiera la bola en intervalos de un **semi-cuadratín**, o incluso menores. Eso puede hacerse aprovechando el comando TAG, para "pegar" la figura al cursor de gráficos.

Recuerda que TAG te permite exponer en pantalla un carácter o símbolo gráfico en la posición señalada por el cursor de gráficos, en lugar de hacerlo en la señalada por el cursor de textos. El cursor de gráficos puede moverse horizontalmente en cualquiera de los 640 **puntos** y verticalmente en cualquiera de los 400 **puntos** (y no podemos moverlo más allá porque nos salimos de la pantalla). Recuerda también que en el MODE 1 cada **mota** (o "pixel") ocupa una colección de **cuatro puntos** contiguos, dos verticalmente y dos horizontalmente; y es por tanto la misma mota para los cuatro pares de coordenadas posibles. (Consulta el Capítulo 3).

Eso significa que para mover una figura gráfica un paso equivalente a un **cuadratín de carácter** -que ocupa 8 x 8 motas- debemos desplazar el cursor de gráficos 16 **puntos** o coordenadas en uno u otro sentido.

Vamos ahora a cambiar el programa. Lo primero a hacer es **activar** la posibilidad de TAG, para lo que basta insertar la línea:

```
45 TAG
```

La banda de valores al trabajar con los puntos gráficos como coordenadas para exponer símbolos, es ahora mucho mayor; así que debemos cambiar las líneas 50 y 60 apropiadamente:

```
50 IF c+x>609 OF c+x<16 THEN x=-x
60 IF 1+y>383 OR 1+y<30 THEN y=-y
```

Las líneas 30 y 40 también deben cambiarse para dar una posición válida inicial de la pelota:

```
30 c=INT(RND*592+16)
40 1=INT(RND*337+42)
```

Cuando está en vigor el **texto en gráficos**, el comando LOCATE para situar el cursor de textos, no tiene ningún efecto; así que debemos cambiarlo por el correspondiente que **mueve** el cursor de gráficos:

```
80 MOVE c,1:PRINT " ";
100 MOVE c,1: PRINT CHR$(224);
```

Si anteriormente incluiste la línea 105, ahora es el momento de excluirla.

La primera cosa que te puede maravillar es lo lento que es. Sucede así porque el programa hace referencia a cada mota dos veces. Podemos doblar la velocidad sin que afecte a la suavidad del movimiento si duplicamos los incrementos 'x' e 'y':

```
20 x=2: y=-2
```

Ejecuta ahora el programa de nuevo para ver las mejoras. Si no estás seguro de lo que se hace en una línea concreta o de lo que representa una variable, cámbiala y ensaya para comprobar el efecto que tiene.

Y con el "bate" batimos la bola

Cuando hayas acabado de experimentar con esta versión dos del programa, si **guardaste** la original, **cárgala.** Si no lo hiciste, cambia de nuevo el programa a como estaba en un principio. Ahora es el momento de incluir un "bate" mediante la **rutina** encargada de esa tarea. El bate se moverá hacia adelante y hacia atrás a lo ancho de la pantalla donde está por el momento el borde inferior de la cancha de juego.

Primeramente, suprime las líneas que trazan ese borde inferior de la cancha (las líneas 300 a 340). (La pelota continúa rebotando, desde luego). Ahora lo que queremos es un bate. Así que usaremos tres símbolos CHR$(131) para formar esta **figura.** La exposición en pantalla del bate debe ocurrir dentro del bucle principal del programa, así que la colocaremos después de que se haya expuesto en pantalla la pelota en la línea 100. Unicamente cambiará la posición horizontal del bate, así que sólo necesitamos una variable.

Para **exponer tres caracteres de código 131, basta teclear:**

```
PRINT CHR$(131);CHR$(131);CHR$(131)
```

Pero si recordamos que son caracteres repetidos, podemos aprovechar la función de formar **literal,** de clave STRING$, tal y como vimos en el Capítulo 7:

```
STRING$(3,131)
```

Ahora inserta las líneas:

```
4 bate$=" "+STRING$(3,131)+" "
109 LOCATE b,25: PRINT bate$;
```

Los espacios en blanco antes y después de la figura "bate" los hemos incluido para poder eliminar el **viejo** bate a medida que se mueve. Necesitamos ahora cambiar el valor de la variable 'b' para que caiga en la banda 1 (izquierda) a 36 (derecha). Tiene que ser 36, porque el bate, incluyendo los espacios en blanco que le rodean, tiene cinco caracteres de largo, y la columna 36 es por tanto lo máximo que puede ir el extremo anterior del bate sin que se salta de la pantalla.

Puedes indagar si se ha pulsado o no una tecla aprovechando la función INKEY$. Escojamos una tecla para la izquierda, digamos Q; y una para la derecha, digamos P. Tendremos que asegurarnos que está quitado el **enclavamiento de mayúsculas**, para conseguir 'q' y 'p', y no 'Q' y 'P'. Puedes teclear lo siguiente:

```
101 ky$=INKEY$
102 IF ky$="q" THEN b=b-1
103 IF ky$="p" THEN b=b+1
```

Claramente, 'b' aumentaría o disminuiría demasiado y haría que el programa se la pegara, así que añade:

```
104 IF b>36 THEN b=36
105 IF b<1 THEN b=1
```

Y dale a 'b' un valor inicial en la misma línea 20:

```
20 x=1: y=-1: b=18
```

Si ahora haces que el programa se ejecute, verás que la pelota rebota como es usual y que puedes mover el bate pulsando las teclas Q o P, pero no ambas a la vez. El bate no responde particularmente bien al haber usado la función INKEY$, así que busquemos otra manera.

INKEY

La palabra clave INKEY (indague) es muy similar a INKEY$ pero lee el teclado de manera diferente. Su sintaxis es:

```
INKEY (<variable entera>)
```

donde el argumento de la función ha de ser un **número de tecla** tal y como aparece en la página 16 del Apéndice 3 en la Guía de Usuario. Verás que la tecla Q es la que tiene como número 67 y la tecla P la 27.

Teclea esas líneas cambiadas:

```
101
```

(con eso se suprime la línea)

```
102 IF INKEY(67)=0 THEN b=b-1
103 IF INKEY(27)=0 THEN b=b+1
```

Encontrarás ahora que se golpea mucho mejor que antes. La siguiente línea puede reemplazar a las líneas 102, 103, 104 y 105, y es una manera mucho más pulcra de efectuar la misma cosa:

```
102 b=b+(INKEY(67)=0 AND b>1)-(INKEY(27)=0
                    AND b<36)
```

Funciona aprovechándose del valor lógico de la condición que aparece entre los paréntesis. Dicho valor es 0 si la condición es falsa y -1 si es cierta- (véase Capítulo 7).

Ahora que el bate se mueve más rápidamente podemos incluir una línea o dos para detectar cuándo golpea la pelota. La pelota estará tocando al bate cuando el siguiente valor de línea de la pelota sea 25 **y si** también el siguiente valor de columna es igual a b+1, b+2 o b+3 (recuerda que el bate tiene tres cuadratines de lago).

Podríamos pues teclear la línea:

```
IF l+y=25 AND (c=b+1 OR c=b+2 OR c=b+3)
   THEN.... un golpe
```

En lugar de eso, teclea estas otras líneas:

```
60 IF l+y<2 THEN y=-y
65 IF l+y>24 THEN y=-y:IF c>b AND c<b+4
   THEN s=s+1 ELSE s=s-5
```

La variable 's' se usa para mantener la puntuación (score). Si golpeas la pelota tu puntuación aumenta. Cada vez que fallas, tu puntuación disminuye en 5 puntos.

Inserta esta línea para exponer la puntuación:

```
106 LOCATE 16,1: PRINT "PUNTUACION =";s;
```

El programa se está haciendo bastante complejo y la pelota se mueve un poquito más lentamente. Eso es por lo que el uso de TAG no es práctico cuando se requiere velocidad.

Colisiones, colisiones

Hasta ahora, hemos usado nuestro conocimiento de dónde están los límites y el bate para detectar cuándo tropieza con ellos la pelota. De hecho, el límite y el bate no necesitan estar ahí en absoluto, y todavía la pelota rebotaría porque las condiciones dictan la colisión, y no realmente lo que haya en la pantalla.

¿Qué pasa si hubieramos introducido en la cancha algún aspecto aleatorio? ¿Cómo podríamos detectar una colisión sin saber de antemano dónde estaba?

Claramente debemos saber lo que estamos buscando, o como mínimo, algo sobre él; su forma o su color, por ejemplo. El CPC464 no tiene ninguna facilidad en forma de palabra clave para comprobar las condiciones dentro de un cuadratín particular ocupado por un carácter, i.e. comprobación del carácter o del color. En lugar de eso, hay una palabra clave para comprobar el color de una mota en una coordenada específica. Esa es apropiadamente la palabra TEST, que tiene como sintaxis:

TEST

```
TEST (<expresion entera>,<expresion entera.)
```

Donde el primer argumento representa la coordenada 'x' y el segundo la coordenad 'y'. Por tanto, PRINT TEST(100,100) expondrá el valor resultado de aplicar la función TEST y ha de ser el número entero que corresponde al color de la tinta INK en 100,100.

Por tanto, si mandaste PLOT 150,300,3 y luego mandas PRINT TEST(150,300) obtendrás expuesto en pantalla el resultado 3. Con el fin de detectar la presencia de un carácter dentro de un cuadratín cualquiera, uno tendría primero que convertir los valores de columna y fila en los valores de las coordenadas x e y correspondientes, y luego mirar el color de esa mota. Hacer la conversión exige pensar un poco sobre ello, especialmente en la dirección vertical, dado que las filas están numeradas de arriba hacia abajo mientras que las coordenadas 'y' se numeran de abajo a arriba. Los más orientados a las matemáticas lo encontrarán más fácil que los otros.

Una manera de salir del dilema es usar una tabla de dos dimensiones de acuerdo con los valores de columna y de fila. Por ejemplo, podríamos teclear DIM a(40,25). Cada variable elemental de esa tabla sería entonces un mero 'cliché' del cuadratín en pantalla correspondiente. Si por ejemplo, hiciste:

```
LOCATE 15,7: PRINT'*': LOCATE 34,13:
      PRINT 'O'
```

también hubieras rellenado los elementos de la tabla así:

```
a(15,7)=1: a(34,13)=2
```

donde los números 1 y 2 significan un carácter particular.

Obviamente podíamos usar el valor ASCII para esto, como:

```
a(15,7)=ASC('*'):a(34,13)=ASC('o')
```

Para detectar una colisión en pantalla comprobaríamos constantemente el contenido de esa tabla. Ahora añadiremos y cambiaremos algunas líneas más en la forma siguiente:

```
5 DIM a(40,25)
40 l=INT(RND*10+13)
107 IF a(c,l)=1 THEN a(c,l)=0:s=s+1:x=-x:y=-y
300 PEN 3
310 FOR l=8 TO 12
320 FOR c=2 TO 39
330 LOCATE c,l: PRINT CHR$(143);
340 a(c,l)=1
350 NEXT c
360 NEXT l
370 PEN 1
380 GOTO 20
```

Las líneas 300 a 360 prefijan las condiciones de la pared, y la tabla "cliché" cuyas dimensiones se establecen en la línbea 5. La línea 40 asegura que la bola siempre comienza por debajo de la pared. Si ocurre una colisión, la dirección de la bola se invierte y tu puntuación se incrementa en 1.

ENRIQUECIMIENTO

El juego como está no puede decirse que sea muy excitante, pero la intención no fue esa. La intención fue mostrar cómo un sencillo juego puede plantearse e irlo desarrollando. También se espera que tengas ahora una mayor comprensión de alguno de los procesos y problemas involucrados en la confección de un programa. Ahora es tu turno para hacer los ensayos. Aquí hay algunas sugerencias para mejoras:

- ☐ Haz que la pelota desaparezca cuando falla el golpe con el bate y sustitúyela con una nueva.
- ☐ Cambia las variables de dirección para que la pelota pueda moverse según ángulos distintos de 45 grados.
- ☐ Haz que la pelota rebote en el borde del bate de la manera que sería lógico esperar.
- ☐ Haz que la pelota rebote en la pared de acuerdo con el ángulo con el que golpea (y así recuerda las leyes de la reflexión).
- ☐ Añade algunos pitidos y un poco de follón sonoro.
- ☐ Añade algunos obstáculos en la cancha de juego.

TIEMPO DE JUGAR

El siguiente programa en la datocassette A es BOMBARDeo. Es un juego que te permitirá liberarte todos esos impulsos destructivos que puede que tengas reprimidos. Tu aeroplano está constantemente perdiendo altura sobre la ciudad y es probable que choques con cualquiera de los edificios. Tu única manera de librarte con seguridad es bombardear hasta que la ciudad quede arrasada.

Feliz aterrizaje.

CONTROL DEL ESTUDIO

Probablemente te hayas cansado de todas esas bombas, así que puede que sea una buena idea tomarte un breve descanso antes de comprobar tus avances con la autoevaluación TAE8.

Capítulo 9
SONIDO FM

Cuando exploramos los comandos para sonido en la Parte 1 de este curso, fue necesario restringir la información a los comandos en su forma más simple. Como ahora ya has adquirido un grado mayor de conocimiento y confianza, podemos pasar a explorar el tema con mucho más detalle y analizar ciertos aspectos de los que previamente prescindimos. Usando nuestra nueva notación, el comando SOUND se describe por:

SOUND

```
SOUND <estado canal>,<periodo tono>
  [,<duracion>[,<volumen>[,<envolvente volumen>
  [,<envolvente tono>[,<ruido>]]]]]
```

Todos los parámetros son expresiones enteras. No te confundas por tantos corchetes; lo que significan es que puedes prescindir de los parámetros comenzando a partir de la derecha, pero no puedes dejar fuera ninguno de los del medio.

CANALES

El parámetro **estado canal** es la parte más complicada del comando SOUND. En la Guía del Usuario verás que puede tener un valor entero entre 1 y 255. No te asustes, ya que por el momento sólo observaremos cómo se eligen los **tres canales** del CPC464.

Estos canales se llaman A, B y C, y pueden ser enteramente independientes unos de otros, cada uno con su propia envolvente de tono y de volumen. Sin embargo, más de un canal puede especificarse en un comando SOUND, tal y como se muestra en la siguiente tabla:

Estado Canal	Canal(es)
1	A únicamente
2	B únicamente
3	A y B
4	C únicamente
5	A y C
6	B y C
7	A, B y C

¡Puede que te parezca un poco extraño, pero funciona!

También puedes enchufar tu cadena de alta fidelidad Amstrad en la parte trasera de tu CPC464 para conseguir sonido estereofónico. El canal A sería el izquierdo, el canal C el derecho, y el canal B proporciona un poquito de ambos.

ENVOLVENTES

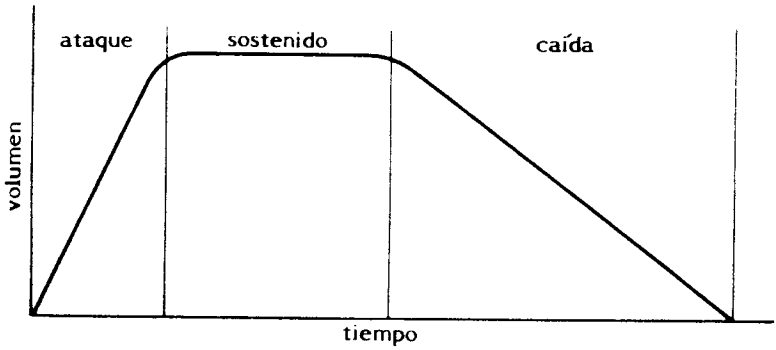
Las envolventes de volumen cambian el nivel de sonido especificado por el parámetro **volumen** en el comando SOUND, y las envolventes de tono cambian la nota, o sea la **frecuencia** especificada por su **período.** Estos cambios pueden ser "dinámicos" durante el tiempo que dura el sonido, tal y como se especifica por el parámetro **duración.** Ya sabes que puedes tener hasta 15 envolventes de volumen y hasta 15 envolventes de tono en un mismo programa. Lo que puede que no sepas es que cada envolvente a su vez puede tener hasta **cinco secciones.** Sigue leyendo.

Envolventes de volumen

Una envolvente con cinco secciones es una perspectiva amenazadora a primera vista, así que tendremos que examinar primeramente algunas de sólo tres secciones. Carga el programa "SONIDOS" en la datocassette A, y haz que funcione. Ignorando la envolvente de tono por el momento, puedes ver que se te da a elegir entre tres cosas llamadas "ataque", "sostenido" y "caída". El diagrama muestra lo que esas palabras significan.

Es un poquito como una historia que tiene su arranque, su parte media y su final. A diferencia, sin embargo, de las historias, es a menudo el comienzo y la terminación lo que es más interesante. Si continúas ensayando con el programa puedes comparar las diferencias en los sonidos producidos por los diferentes tipos de ataque y caída de la nota.

Un sonido típico con un "ataque brusco" es el de los instrumentos musicales tales como la guitarra. Otros ejemplos de sonido con bordes de ataque de las notas rápidos son los de percusión tales como el batido del tambor y los tronidos y explosiones. Un ataque suave o lento sin embargo, es más característico de los tubos largos de los órganos, o de un instrumento de viento tal como la tuba. Y a la inversa, una vez que dejas de soplar en la tuba, el sonido se detiene inmediatamente (caída rápida) mientras que el sonido de una guitarra muere gradualmente (caída lenta).



De esta manera puedes ver cómo una envolvente de volumen multisección nos permite remedar los sonidos naturales que oímos alrededor de nosotros. Ahora podemos echar de nuevo una mirada a la instrucción ENV:

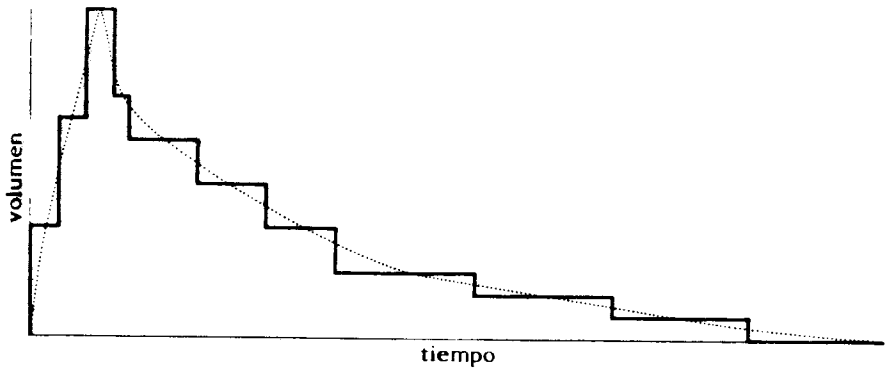
ENV

ENV <numero envolvente>[,<secciones envolvente>]

Y como ya sabes, cada **sección de envolvente** consta de:

<recuento pasos>,<altura paso>,<tiempo pausa>

y como habrás descubierto recientemente, puedes tener desde una hasta cinco de estas secciones. Aquí hay un ejemplo:



La codificación de ésta sería:

```
ENV 1, 3,5,2, 1,-4,1, 4,-2,5, 1,0,5, 3,-1,10
```

Si se usara esta envolvente con un comando SOUND cuyo volumen inicial fuera cero, el efecto sobre el volumen sería como sigue:

Número de sección	Efecto
1	Ataque medio hasta volumen máximo
2	Caída rápida a nivel 11
3	Caída media a nivel 3
4	Sostenido breve
5	Caída lenta a nivel 0

Puedes ver que el tramo en que el volumen se mantiene está producido especificando un único paso con un cambio de volumen de cero -el **período** es el que da la longitud de este tramo sostenido.

Ahora es hora de que comiences a experimentar con envolventes de volumen que tú imagines. Si a tu comando de sonido se le da una **duración** de cero, significa que la 'longitud' de la nota está controlada enteramente por la instrucción ENTER. Puedes ver eso a partir de los listados del programa 'SONIDOS'. Intenta añadir tus propias envolventes como ENV 13, ENV 14 y ENV 15.

```
100 ' Sonidos : demostracion de  Envolventes
110 '
120 ' por George Tappenden
130 ' amended by DA 25/9/84
140 '
150 CLEAR
160 DEF FNt(x$)=INSTR('FSLMH',x$)-1
170 FOR i=0 TO 4:READ n$(i):NEXT
180 READ e$,a$,s$,d$,t$
190 '
200 ' Pantalla
210 '
220 MODE 2   : BORDER 26
230 INK 0,26 : INK 1,0
240 PAPER 0  : PEN 1
250 '
260 ' Pantalla Descriptiva
270 '
280 PRINT STRING$(11,'*');' ENVOLVENTE DE VOLUMEN ';STRI
NG$(10,'*');
290 PRINT SPACE$(8);STRING$(2,'*');' ENVOLVENTE DE TONO
';STRING$(2,'*')
```

```
300 PRINT : PRINT
310 PRINT e$,a$,s$,d$,e$,t$
320 PRINT
330 FOR i=1 TO 12
340 READ a$,b$,c$
350 a=FNt(a$):b=FNt(b$):c=FNt(c$)
360 PRINT i,n$(a),n$(b),n$(c)
370 NEXT
380 FOR i=1 TO 4
390 READ tn$
400 LOCATE 53,4+i*2 : PRINT i
410 LOCATE 66,4+i*2 : PRINT tn$
420 NEXT
430 '
440 ' Envolventes de Volumen
450 '
460 ENV 1,1,15,1,1,0,40,1,-15,1
470 ENV 2,1,15,1,1,0,40,15,-1,4
480 ENV 3,1,15,1,1,0,5,15,-1,4
490 ENV 4,1,15,1,1,0,5,1,-15,1
500 ENV 5,5,3,1,1,0,40,1,-15,1
510 ENV 6,5,3,1,1,0,40,15,-1,4
520 ENV 7,5,3,1,1,0,5,1,-15,1
530 ENV 8,5,3,1,1,0,5,15,-1,4
540 ENV 9,15,1,2,1,0,40,1,-15,1
550 ENV 10,15,1,2,1,0,40,15,-1,4
560 ENV 11,15,1,2,1,0,5,1,-15,1
570 ENV 12,15,1,2,1,0,5,15,-1,4
580 '
590 ' Envolventes de Tono
600 '
605 ENT -1,1,0,1
610 ENT -2,2,1,2,2,-1,2
620 ENT 3,100,1,2
630 ENT 4,100,-1,2
650 '
660 ' Seleccion del usuario
670 '
680 WHILE ( n>=0 ) AND ( t>=0 )
690 '
700 LOCATE 10,22 : PRINT "  "
710 LOCATE 10,23 : PRINT "  "
720 LOCATE  1,20 : PRINT "Escriba el numero de envolvent
e"
730 PRINT
740 INPUT "volumen ";n
750 INPUT "tono    ";t
760 '
770 ' Tocar
```

```
780 '
790 FOR times=1 TO 3
800 LOCATE 6,(n+5)    : PRINT '*****'
810 LOCATE 58,(2*t+4): PRINT '*****'
820 SOUND 7,200,0,0,n,t
830 FOR delay = 1 TO 1500 : NEXT
840 LOCATE 6,(n+5)    : PRINT SPACE$(5)
850 LOCATE 58,(2*t+4): PRINT SPACE$(5)
860 FOR delay = 1 TO 500 : NEXT
870 NEXT
880 '
890 WEND
900 END ' del programa
910 '
920 ' Data para pantalla descriptiva
930 '
940 DATA Rapido,Lento,Largo,Medio,Corto
950 DATA Envolvente,Ataque,Sostenido,Caida,Tipo
960 DATA F,L,F,F,L,S,F,H,S,F,H,F
970 DATA M,L,F,M,L,S,M,H,F,M,H,S
980 DATA S,L,F,S,L,S,S,H,F,S,H,S
990 DATA constante,vibrato,decreciente,creciente
```

Envolventes de tono

Mucho de lo que se ha dicho sobre la estructura de las envolventes de volumen también se aplica a las envolventes de tono. Aunque en esta ocasión, es el tono grave o agudo de la nota la que puede cambiarse durante la ejecución de un comando SOUND. Su uso primordial en los efectos sonoros es de simular el ruido hecho por vehículos que se mueven rápidamente.

Cuando un caza de reacción pasa por encima de tu cabeza, el ruido que produce se hace más y más fuerte hasta que está directamente encima de ti, y luego se hace más suave a medida que se aleja. La **frecuencia de la nota** también cambia debido a eso que se conoce como el "efecto Doppler" -cae continuamente desde el momento que el avión está justamente encima de ti, hasta que está completamente fuera del alcance de tu oido. Puedes oir una versión acelerada de eso en el programa SONIDOS si usas ENV 5 y ENT 3. Esta clase de efecto es muy usado en los juegos de disparos para proveer mayor realismo y añadir excitación.

El efecto opuesto es necesario cuando simulas el despegue de un avión, cuando el piloto tiene que acelerar los motores para conseguir la máxima potencia. La **frecuencia de la nota** aumenta entonces continuamente hasta que las turbinas alcanzan su máxima velocidad. Puedes conseguir una idea de un tono creciente usando ENT 4 con cualquiera de las envolventes de volumen más largas dadas en el programa SONIDOS. Observa que la instrucción ENT no tiene ningún efecto sobre la 'longitud del sonido' (a diferencia de ENV que sí la tiene) y sólo sobre el tono de la nota emitida.

ENT

Como vimos para la envolvente de volumen, la sintaxis es:

```
ENT <numero envolvente>,<secciones envolvente>
```

donde el parámetro **número de envolvente** puede valer de 1 a 15, y puede haber hasta cinco **secciones de envolvente.** Cada una consta de:

```
<recuento pasos>,<altura paso>,<tiempo pausa>
```

Todos ellos son expresiones enteras. La **altura paso** es el cambio requerido en el **período** (no en la frecuencia) del tono de la nota que precisas, y es positivo si quieres que la frecuencia baje y negativo si quieres que suba. Si piensas sobre eso, "ésa es" la manera correcta.

Si colocas un signo menos (-) por delante de **número envolvente** hará que se mantenga repetidamente hasta el final de la duración especificada en el comando SOUND. Eso es útil para obtener efectos como el "vibrato".

TIEMPO DE JUEGO

Si te apeteciera oir algunas de las posibilidades para los efectos sonoros con el CPC464, carga el siguiente programa de la datocassette A, el de "DEMOSON" y ponlo a funcionar. Este programa no está listado aquí, pero puedes conseguir el listado en pantalla en cuanto lo desees.

Nuestra favorita es la imitación del tren.

CONTROL DEL ESTUDIO

Muchos aspectos de los comandos SOUND ya te serán familiares del trabajo que hiciste sobre ellos en la Parte 1. El aspecto realmente interesante de ellos, sin embargo, es la manera en que se pueden encajar conjuntamente los diferentes elementos para producir el sonido que deseas. Comprueba con la autoevaluación TAE9 para ver que has aprendido los principios en que se basan las envolventes multisección.

Capítulo 10
MUSICA

Antes de comenzar a estudiar este capítulo, merece la pena reflexionar sobre las cuatro clases de personas que hay en este mundo. Existen aquéllas que pueden programar ordenadores y también leer música; aquéllas que no pueden programar ordenadores pero sí pueden leer música; y aquéllos que sí pueden programar ordenador, pero no pueden leer música; y aquéllas que no pueden programar ordenadores ni tampoco leer música.

Sólo es justicia decirte que realmente si tú no puedes leer la notación musical de los pentagramas, va a serte un poco difícil que comprendas lo que sigue. Por tanto, será perfectamente comprensible que te saltaras este capítulo y pasaras al siguiente.

CORONEL BOGEY

En la Parte 1 te dimos una docena de líneas de programa que podías teclear para hacer que el CPC464 tocara una "tonada familiar". La tonada como habrás descubierto era la de **Coronel Bogey**. Dado que puede que hayas intentado algunas más por ti mismo, te recordamos simplemente la original ahora:

```
10 REM Una Tonada Familiar
20 SOUND 1,213
30 SOUND 1,253,60
40 SOUND 1,0,40
50 SOUND 1,253
60 SOUND 1,239
70 SOUND 1,213
80 SOUND 1,127,40
90 SOUND 1,0,1
100 SOUND 1,127,40
110 SOUND 1,159,60
120 END
```

Pero esta es una manera muy tediosa de programar música -especialmente cuando es una canción suficientemente larga. Ahora que ya sabes cómo usar los comandos DATA y READ puedes aprender métodos más elegantes y económicos para programar a tu CPC464 para que toque música.

En lugar de tener una serie de comandos SOUND, es mucho mejor usar **variables** como parámetros del comando. Por ejemplo:

```
SOUND cal,per,dur,vol,evo,eto,rdo
```

Ahora puedes incluir todos los valores pertinentes en instrucciones DATA y usar el comando READ para actualizar esas variables antes de ejecutar el comando de emisión de sonido.

Volvamos de nuevo a "Coronel Bogey". En virtud de la simplicidad, hemos dejado fuera los parámetros de volumen, envolvente y ruido, y usamos sólo el canal 1. Nuestro bucle principal sería entonces:

```
10 FOR conta =1 to 10
20 READ per,dur
30 SOUND 1,per,dur
40 NEXT conta
50 END
```

Como puedes ver, esta rutina tocaría cualquier tonada que tuviera 10 notas, dado que toda la música está reflejada en las **constantes** de las instrucciones DATA. Para finalizar con nuestro programa de tocar tonadas, debes por lo tanto, teclear la instrucción:

```
60 DATA 213,20,253,60,0,40,253,20,239,20,
213,20,127,40,0,1,127,40,159,60
```

Si ahora ejecutas este programa sonará (si no has cometido ninguna equivocación) exactamente igual que nuestra versión original. Pero te preguntarás ¿por qué todo este jaleo? Bien, hay muchas ventajas, una es que es mucho más fácil cambiar constantes individuales en las instrucciones DATA que buscar por todo el programa comandos SOUND. Otra ventaja es que puedes así cambiar fácilmente ciertos parámetros para toda o parte de la canción. Ensaya cambiando por ejemplo, el comando SOUND en la forma:

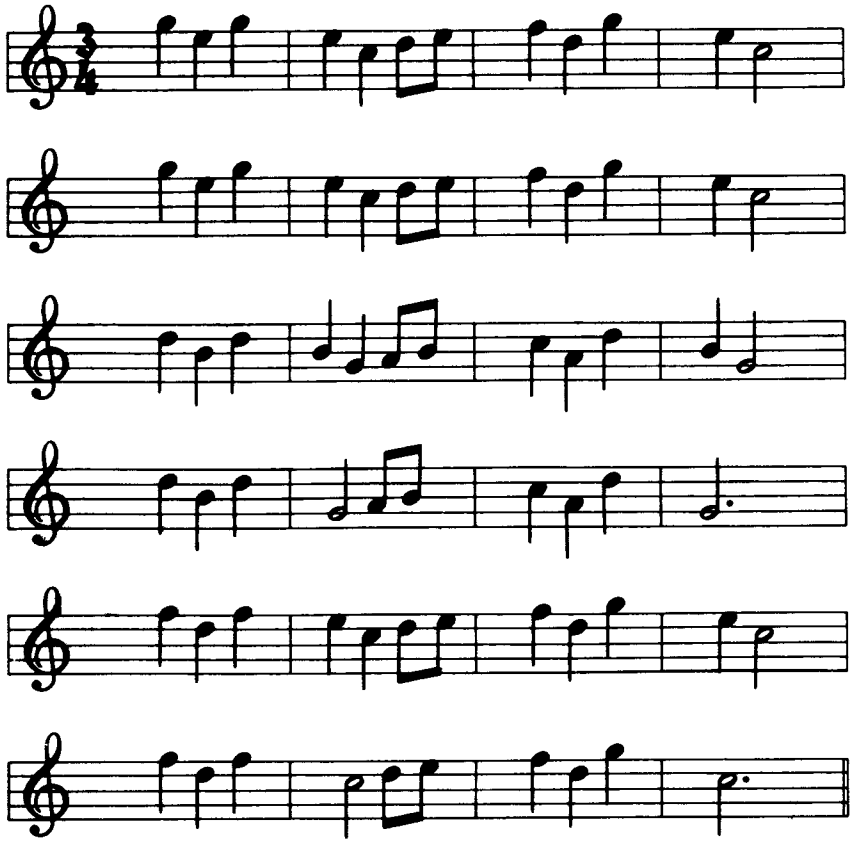
```
30 SOUND 1,per*2,dur
```

No hay una manera más simple de alterar todas las notas en una octava.

NARANJAS Y LIMONES

Volvamos de nuevo a la música y echa una mirada a esta nueva tonada -esta vez es un vals:

Naranjas y Limones **Tradicional**



Al igual que muchas canciones, está confeccionada como una serie de 'frases de cuatro barras', pero lo interesante es que sólo hay dos pautas básicas de notas en estas seis frases. Las primeras dos frases son, desde luego, idénticas. Pero la tercera frase también es la misma, sólo que está desplazada hacia abajo en 'tres semitonos'. La cuarta frase usa el segundo modelo aunque es sólo ligeramente diferente del primero, pero la quinta frase vuelve a ser el primer modelo. Finalmente, la sexta frase es la misma que la cuarta pero desplazada hacia arriba 'cinco semitonos'.

```
100 ' naranjas y limones
110 '
120 FOR frase=1 TO 6
130 '
140 IF frase=1 THEN RESTORE 360
150 IF frase=2 THEN RESTORE 360
160 IF frase=3 THEN RESTORE 360
170 IF frase=4 THEN RESTORE 410
180 IF frase=5 THEN RESTORE 360
190 IF frase=6 THEN RESTORE 410
200 '
210 FOR nota=1 TO 12
220 READ tono,duracion
230 '
240 IF frase=1 THEN periodo=tono
250 IF frase=2 THEN periodo=tono
260 IF frase=3 THEN periodo=tono*1.335
270 IF frase=4 THEN periodo=tono
280 IF frase=5 THEN periodo=tono
290 IF frase=6 THEN periodo=tono/1.335
300 '
310 SOUND 7,periodo,duracion
320 NEXT nota
330 NEXT frase
340 END
350 '
360 ' primera partitura
370 '
380 DATA 159,40,190,40,159,40,190,40,239,40,213,20
390 DATA 190,20,179,40,213,40,159,40,190,40,239,80
400 '
410 ' segunda partitura
420 '
430 DATA 213,40,253,40,213,40,319,40,319,40,284,20
440 DATA 253,20,239,40,284,40,213,40,319,80,319,40
```

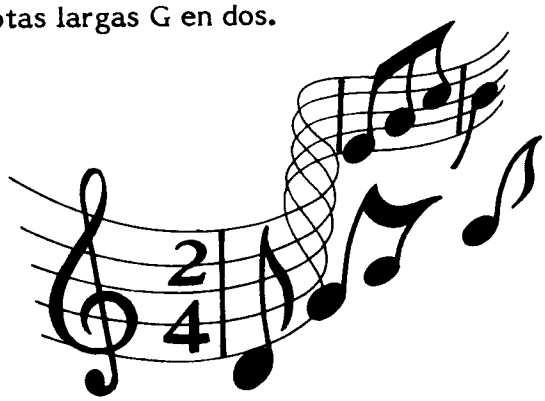
Como puedes ver, este programa consta de un bucle que toca las notas de una frase, anidado dentro de un segundo bucle que elige la frase musical a tocar. El selector de frase usa el comando RESTORE para que se **redireccione** el puntero hasta el bloque apropiado de instrucciones DATA, de manera que el comando READ en la línea 220 pueda tomar el dato apropiado y apuntarlo en las variables "nota" y "duración".

La correlación entre las notas musicales y las cifras dadas en las instrucciones DATA apuntadas como valores de "nota" se da en el Apéndice VII de la Guía del Usuario.

La variable intermedia "nota", se usa luego para generar la variable "período" de acuerdo con una segunda selección de frase. Una aproximación a los desplazamientos de cinco semitonos, se consigue multiplicando y dividiendo la "nota" por 1.335.

Llegamos ahora a nuestro solitario comando SOUND de la línea 320. Observarás que el valor del **canal** es 7, de manera que los tres canales tocan simultáneamente.

Si echas una mirada a la música de unas cuantas páginas atrás, observarás que el segundo modelo no tiene 12 notas en absoluto; sólo tiene 10. Así que ¿cómo hemos llegado a las 12 constantes de las instrucciones DATA? Si lo miras cuidadosamente verás que lo que se ha hecho es desglosar ambas notas largas G en dos.



Desarrollos adicionales

Nuestra melodía anterior se mantuvo deliberadamente sencilla, de manera que pudieras comprender los principios generales involucrados. No se ha hecho ningún intento de variar el volumen entre las frases, y se ignoraron completamente las envolventes de tono y de volumen. De manera que ¿por qué no ensayar a modificar el programa e incluir esas facetas? También pudieramos anidar el bucle principal dentro de uno adicional que repitiera la tonada un cierto número de veces. ¿Qué te parece una envolvente de volumen diferente para cada ronda del bucle?

Otra posibilidad sería la de enchufar tu cadena Hi-Fi en la conexión estéreo que hay en la parte posterior del CPC464 (el mini-zócalo marcado 'I/O'). Pudieras luego cambiar el valor del canal entre las frases de manera que consiguieras un efecto de 'llamada y respuesta' entre los dos canales estéreo.

De cualquier manera, sabes ahora lo suficiente como para programar cualquier melodía de **una sola voz** en el CPC464 y con toda probabilidad eres capaz de aprovecharte de eso para crear efectos sonoros con los comandos SOUND del BASIC Amstrad.

TEMAS SERIOS

El siguiente programa en la datocassette A es MINUETo de J.S. Bach. Es muy poco probable que cualquier otro compositor haya tenido su música ejecutada por tantos instrumentos diferentes. Incluso aunque él escribió originalmente esta pieza para un clavicordio, puede que desde entonces haya sido adaptada para piano, órgano, guitarra, orquesta, o una banda de instrumentos de viento. Así que aquí hemos continuado la tendencia -una versión para el Ordenador Personal a Color Amstrad CPC464.

Aparte de ser una buena interpretación de esta pieza musical, hay algunos detalles en este programa que merece la pena comentar -antes o después de que lo hayas tocado.

Nuestra intención es mostrarte lo bien que puede sonar en el CPC464 una pieza de música clásica. Sin embargo, fue necesario usar comandos y técnicas de las que todavía no encontrarás explicación en esta parte del curso. Pero no hay nada que te impida echar una mirada al listado en la pantalla si lo deseas.

La primera cosa que notarás es que las **tres voces** se usan para dar las líneas de la melodía, el acompañamiento y los bajos. Eso requiere una descripción bastante más detallada de los aspectos más sutiles de las operaciones con tres canales de sonido, y tendrás que esperar hasta la Parte 3 de este curso. También puede que te encuentres de vez en cuando con comandos que no te son familiares. De nuevo, y a no ser que quieras aprenderlos por ti mismo en la Guía del Usuario, también tendrás que esperar.

Habiendo dicho todo eso, merece la pena listar el programa en la pantalla para ver cómo se ensamblaron todos los efectos.

CONTROL DEL ESTUDIO

No te preocupes, no es un control sobre tus dotes musicales. Ejecuta TAE10 para comprobar si has comprendido las palabras clave y las técnicas usadas en este capítulo.

Capítulo 11
AVENTURA

Una de las clases más populares de juegos con ordenador es la aventura, en la que entras en un mundo de fantasía que sólo existe en la mente del programador. Al teclear los comandos y la información te conviertes en una parte integrante de una saga que el ordenador crea diciéndote (a través de la pantalla) dónde estás en el tiempo y en el espacio, cuál es tu entorno y el último suceso en la historia. La fascinación radica en el hecho de que puedes influenciar el curso de los eventos y que el juego transcurre de manera diferente cada vez.

La mayoría de los juegos de aventuras se basan en la idea de un laberinto, o de un edificio con montones de habitaciones, a través del cual tienes que encontrar tu camino en busca de diversos objetos valiosos o útiles. Hay también trampas y azar o incluso (en la historia) peligro mortal. La meta última usualmente es la consecución de un objeto particular y el escape con éxito.

Carga el siguiente programa de la datocassette A, que trata de la AVENTURA. Es un ejemplo que vamos usar para enseñarte cómo escribir tus juegos de aventuras.

ROLDAN EN LA CASA

AVENTURA es un ejemplo muy típico de esta clase de programas. La historia es que quieres pasar una tarde tranquila programando tu CPC464, pero alguien ha dejado todas las piezas que necesitas en diferentes habitaciones de la casa. El objeto es encontrarlas todas e instalar tu sistema en alguna otra parte sin interrupciones.

Hay tres clases de comandos:

- **Comandos de dirección:**
  - n = norte
  - e = este
  - s = sur
  - o = oeste
  - b = abajo
  - a = arriba
- **Comandos de una palabra:**
  - help ayuda
  - look mirar
  - list listar
  - quit abandonar
- **Comandos de dos palabras:**
  - get.... coger
  - drop.... dejar
  - open.... abrir
  - close.... cerrar
  - pull.... empujar
  - plug.... enchufar

Siempre que observes el signo de aviso (>), teclea uno de los comandos anteriores o similares. Siempre puedes teclear 'ayuda' si realmente necesitas socorro.

TIEMPO DE JUEGO

Aquí es donde puedes sacar algo de tiempo y simplemente disfrutar jugando a la aventura. Incluso una tan sencilla como ésta puede ser bastante atractiva, así que no te sorprendas si no vuelves al libro durante un rato.

SIMPLEMENTE COMO RECAPITULACION

Cuando hayas jugado unas cuantas veces a la aventura y hayas encontrado dónde está cada cosa y lo que necesitas para completar el juego, puedes tener diversas ideas sobre cómo te gustaría cambiarlos. Quizás te gustaría añadir algunas habitaciones y objetos más, o quizás te gustaría cambiar el nombre de todas las habitaciones para convertirlas en cavernas. Cualesquiera sean tus ideas, serás incapaz de efectuar ningún cambio hasta que no comprendas cómo se ha montado este programa.

Cuando estudies el listado que aparece más adelante, puede que te deje sorprendido por su complejidad aparente. Por favor, que no sea así; pronto verás lo simple que es realmente, y serás capaz de reconocer los comandos usados, los cuales se han explicado en capítulos anteriores.

Hay sin embargo, algunos aspectos que necesitas conocer antes de que puedas comenzar.

Algunas observaciones

Habrás notado algunas variables con el signo porcentaje (%) después del nombre. La primera de ellas aparece en la línea 500:

```
FOR i% = 0 TO 5
```

Este signo porcentaje señala al CPC464 que ha de tratar esa variable de una manera especial: como un **entero genuino.** Cuando el CPC464 está tratando con numerales, usa lo que se llama aritmética de coma flotante al efectuar los cálculos y guardar los números. Eso es bastante sensato dado que realmente te permite usar números que tienen una **parte fraccionaria,** i.e. con montones de posiciones decimales, como **pi** (3.1415926). Números tales como ése son los denominados **números reales** porque son los que existen en realidad.

Se explicó en la Parte 1 de este curso que a pesar de toda la 'inteligencia' del CPC464 es posible terminar con una respuesta que no sea exactamente lo que esperabas. Por ejemplo, supongamos que has efectuado un cálculo que tú has desarrollado y sabes que dará la respuesta 5. Lo puedes colocar en una línea de programa, tal como:

```
IF x=5 THEN GOTO 2000
```

Sin embargo, te encuentras con el que CPC464 rehusa ir a la línea 2000, por lo que expones el valor de "x" y obtienes como respuesta 4.9999999. Lo que ha sucedido es que en alguna parte del cálculo una pequeña inexactitud se vió suficientemente multiplicada como para poder ser apreciable. ¿No crees que eso sucede? Ensaya con esto:

```
10 FOR n=1 TO 10 STEP .15
20 PRINT n
30 NEXT n
```

Teniendo en cuenta estas cosas, sabes cómo puedes solventar tales problemas usando palabras clave para redondeo (ROUND) o para truncamiento a entero (INT).

El prevenir es mejor que el curar, y puedes prevenir los errores fraccionarios diciéndole al CPC464 que trate un valor siempre como un entero. Para eso es para lo que se usa el signo porcentaje. Verás incluso que al calcular 5+4, el CPC464 todavía usa aritmética de coma flotante en la forma:

```
5.0000000 + 4.0000000 = 9.0000000
```

Ni tú ni yo haríamos eso. Cuando vemos un número entero usamos la aritmética entera; cuando vemos un número fraccionario usamos la aritmética de coma flotante. Al CPC464 hay que contarle todo. En un juego de aventuras hay muy poca o ninguna necesidad de números fraccionarios, así que podemos trabajar con enteros, garantizando así que no ocurran errores y haciendo además que los cálculos sean más rápidos.

¿Verdadero o falso?

Cuando repasaste el listado, puede que hayas pensado que algunas de las instrucciones condicionales IF parecían incompletas, por ejemplo en la línea 1010:

```
IF INSTR(inv$,comando%) THEN GOSUB 1730
```

La función INSTR no está comparada con nada, y por tanto ¿cómo puede efectuarse la decisión? En el Capítulo 7 se hizo una breve mención de lo que significaba el valor **logical** de una condición. Tendrá un valor de -1 si es **cierta** y de 0 si es **falsa.** En la línea:

```
IF x=5 THEN...
```

La condición a corroborar es x=5. Puedes ver si es o no cierta, simplemente haciendo que se exponga su valor mediante:

```
PRINT x=5
```

Puedes hacer eso para cualquier condición porque la respuesta sólo puede ser cierta o falsa. Suficientemente sensato, podrías decir, pero ¿qué sucede con una línea como ésta?:

```
IF x THEN...
```

¿Si 'x' entonces qué...? El valor logical de la condición 'x' depende del valor numeral entero de 'x'. Tomada como condición, 'x' será cierta si 'x' es un valor distinto de cero, y será falsa sólo si es igual a cero. Tan sencillo como eso.

La función INSTR se aplica sobre dos literales para examinar si el segundo **pertenece** al primero; en cuyo caso entrega como resultado un número que indica la posición del primer carácter donde se produce la coincidencia. Por tanto, si el primer literal **contiene** al segundo, el valor resultante será distinto de cero y considerado como cierto; si no lo contiene será igual a cero y considerado como falso.

Ahora serás capaz de comprender cómo trabaja ante líneas como la 1010 y similares. Es un concepto muy provechoso.

DISEÑO DE "AVENTURA"

Imagínate que tuvieras que diseñar un juego de aventuras a partir de la nada total (no te preocupes, eso no lo tendrás que hacer aquí). En la Parte 1 de esta guía tutorial (Capítulo 9) se presentó una técnica de programación denominada Lenguaje para Desarrollo de Programas, o PDL para abreviar, como método de entrenarte en pensar de una manera lógica y estructurada, dimos un ejemplo de un cartero mecanotrónico que entregaba el correo a las casas de una calle.

El resto de este capítulo te mostrará ahora:

- ☐ Cómo se traduce PDL a un programa 'trabajador'
- ☐ Cómo se modifica el programa para cambiar la historia.

Otro poco de PDL

Hay muchas maneras de escribir en PDL, y nadie osaría decir que su manera es mejor que la de los demás. Aparte del número limitado de palabras clave empleadas, el objetivo principal es asegurarte que las rutinas y las subrutinas están ensambladas en estructuras lógicas -y de ahí el término (que puede que hayas oido ya) de **programación estructurada.**

A continuación, está nuestro juego de aventuras expresado usando un **lenguaje para el desarrollo de programas,** y puedes ver que aunque el BASIC ocupa 10K de memoria, el esquema en este lenguaje es muy compacto.

Obviamente, éste es un simple esbozo y no incluye el detalle de las subrutinas necesarias para llevar a cabo cada una de las tareas. Pero da una panorámica clara y concisa del programa y puede usarse como un esquema de las diferentes tareas a realizar.

```
IF      Si  se ha introducido comando
THEN    Entonces  interprete comando
                  En caso  de ser:
                     Un comando de movimiento-
        IF                Si  la direccion no esta permitida
        THEN              Entonces  presente mensaje
        ELSE              'Endemas'  cambie posicion
                     Comando de accion-
        IF                Si  la accion no es posible
        THEN              Entonces  presente mensaje
        ELSE              'Endemas'  realice la accion
                     Instruccion (ayuda, listar, salir, mirar)
                         Ejecute la instruccion
     IF             Si  se corresponde final del juego
     THEN           Entonces  presente felicitacion
ELSE  'Endemas'  espere que le introduzcan comando
```

```
100 ' Aventuras de Arnold
110 '
120 ' DA  30/9/84
130 '
140 ' Inicializar
150 '
160 CLEAR
170 scrwidth=40 ' o cambiar a 80 y usar MODE 2
180 '
190 MODE scrwidth/40-0.25
200 BORDER 24
210 INK 0,3: INK 1,24
220 PAPER 0: PEN 1
230 LOCATE 11,5
240 PRINT"Aventuras de Arnold"
250 LOCATE 1,8
260 PRINT"    En esta aventura estas intentando"
270 PRINT"    pasar una tarde tranquila jugando"
280 PRINT" con tu ordenador CPC464 pero alguien"
290 PRINT" ha dejado lo que necesitas tirado por"
300 PRINT"              toda la casa"
310 PRINT:PRINT"     El objetivo es encontrar todo y"
320 PRINT"    montar tu sistema en algun sitio"
330 PRINT"          sin interrupciones."
340 PRINT:PRINT"         /Cuidado con el gato!"
'
360 WHILE INKEY$="" : WEND
370 CLS
380 BORDER 23:INK 0,23
390 INK 1,1:INK 2,5:INK 3,8
400 '
410 ' Leer mapa
420 '
430 rooms=11 : items=11  'numero de habit.y art.(-1)
440 fixed=7 ' los primeros (fixed-1) art. son moviles, l
os restantes no
450 held=0:heldmax=4  ' max num  objetos que puede tener
 en las manos
460 goes=0 ' numero de goes
470 '
480 '
490 DIM ex$(5) ' opcional
500 FOR i%=0 TO 5
510 READ ex$(i%) : NEXT
520 '
530 ' leer habit.
540 '
550 DIM loc$(rooms),dir$(rooms),dest$(rooms)
```

```
560 FOR i%=1 TO rooms
570 READ loc$(i%),dir$(i%),dest$(i%)
580 NEXT
590 '
600 ' leer objetos
610 '
620 DIM object$(items),objloc(items)
630 FOR i%=0 TO items
640 READ object$(i%),objloc(i%)
650 NEXT
660 '
670 closed=-1:open=0:onn=-1:off=0
680 backdoor=closed:frontdoor=closed
690 switch=off:plugged=off
700 DIM sw$(1) : sw$(0)="off" : sw$(1)="on"
710 position=3 'posicion inicial
720 direction$="NSEOBA"
730 get$="COGERRECOGERTOMARLEVANTAR"
740 put$="TIRARPONERSALIR"
750 pul$="EMPUJARTIRARPRESIONARTORCER"
760 inv$="LISTAINVENTARIO"
770 clo$="CERRADACERRAR"
780 ope$="ABRIR"
790 loo$="MIRARDONDEEXAMINAR"
800 qui$="TERMINARFINALIZARPARARCONCLUIR"
810 plu$="ENCHUFARCONECTAR"
820 hel$="AYUDA"
830 nul$=CHR$(0) ' poner a cero
840 '
850 GOSUB 2140 : REM buscar (comienzos)
860 '
870 ' Bucle principal
880 '
890 PRINT
900 INPUT ">",command$
910 IF command$="" THEN 890
920 IF LEFT$(command$,1)=" " THEN command$=MID$(command$
,2) : GOTO 910
930 command$=UPPER$(command$)
940 goes=goes+1
950 '
960 ' Ahora analizar cadena
970 '
980 ' una palabra
990 '
1000 IF LEN(command$)=1 THEN GOSUB 1270 : GOTO 1190 :REM
  move
1010 IF INSTR(inv$,command$) THEN GOSUB 1730 : GOTO 1190
 : REM inventario
```

```
1020 IF INSTR(loo$,command$) THEN GOSUB 2120 : GOTO 1190
 : REM mirar
1030 IF INSTR(qui$,command$) THEN GOSUB 2070 : GOTO 1190
 : REM terminar
1040 IF INSTR(hel$,command$) THEN GOSUB 2380 : GOTO 1190
 : REM ayuda
1050 '
1060 ' dos palabras
1070 '
1080 d%=INSTR(command$,' ')
1090 IF d%=0 THEN PRINT' No entiendo eso':GOTO 1210
1100 noun$=MID$(command$,d%+1) : command$=LEFT$(command$
,d%-1)
1110 IF INSTR(noun$,' ') THEN P$=' Dos palabras cada vez
, por favor':GOSUB 2680: GOTO 890
1120 IF INSTR(get$,command$) THEN GOSUB 1380 : GOTO 1190
 : REM coger
1130 IF INSTR(put$,command$) THEN GOSUB 1520 : GOTO 1190
 : REM poner
1140 IF INSTR(pul$,command$) THEN GOSUB 1650 : GOTO 1190
 : REM tirar
1150 IF INSTR(clo$,command$) THEN GOSUB 1840 : GOTO 1190
 : REM cerrar
1160 IF INSTR(ope$,command$) THEN GOSUB 1950 : GOTO 1190
 : REM abrir
1170 IF INSTR(plu$,command$) THEN GOSUB 2270 : GOTO 1190
 : REM enchufar
1180 PRINT ' No se como hacer eso';
1190 '
1200 GOSUB 2470 ' esta terminada aventura?
1210 '
1220 GOTO 890
1230 '
1240 '
1250 ' Mover
1260 '
1270 d%=INSTR(direction$,command$)
1280 IF d%=0 THEN P$=' No se que quieres decir':GOSUB 26
80: GOTO 1340
1290 d%=INSTR(dir$(position),command$)
1300 IF d%=0 THEN P$='No puedes ir en esa direccion':GOS
UB 2680: GOTO 1340
1310 IF (position=4 OR (position=3 AND d%=1)) AND backdo
or=closed THEN P$='La puerta trasera esta cerrada':GOSUB
 2680: GOTO 1340
1320 IF (position=1 OR (position=2 AND d%=2)) AND frontd
oor=closed THEN P$='La puerta delantera esta cerrada':GO
SUB 2680: GOTO 1340
```

```
1330 position=ASC( MID$( dest$(position),d%,1 ) )-64 : G
OSUB 2140
1340 command$=nul$ : RETURN
1350 '
1360 'coger
1370 '
1380 o%=0
1390 FOR i%=0 TO items
1400 IF INSTR(UPPER$(object$(i%)),noun$) THEN o%=o%+1 :
GOSUB 1440 : GOTO 1430
1410 NEXT
1420 IF o%=0 THEN P$='No se lo que es eso':GOSUB 2680
1430 command$=nul$ : RETURN
1440 IF i%>=fixed THEN P$='No puedes coger el '+object$(
i%):GOSUB 2680: GOTO 1480
1450 IF held=heldmax THEN p$='Tienes las manos ocupadas'
:GOSUB 2680:GOTO 1480
1460 IF objloc(i%)=position THEN P$= 'Ya lo tengo':GOSUB
 2680:held=held+1:objloc(i%)=0 : GOTO 1480
1470 IF objloc(i%)=0 THEN P$='Ya lo tienes':GOSUB 2680 E
LSE P$='No esta aqui':GOSUB 2680
1480 RETURN
1490 '
1500 ' poner
1510 '
1520 o%=0
1530 FOR i%=0 TO items
1540 ff%=0
1550 IF INSTR(UPPER$(object$(i%)),noun$) THEN o%=o%+1 :
GOSUB 1600
1560 IF o%>0 THEN i%=items
1570 NEXT
1580 IF o%=0 THEN P$='No se que es eso':GOSUB 2680
1590 command$=nul$ : RETURN
1600 IF objloc(i%)=0 THEN P$='Has dejado caer '+object$(
i%):GOSUB 2680:held=held-1:objloc(i%)=position ELSE P$='
No tienes '+object$(i%):GOSUB 2680
1610 RETURN
1620 '
1630 ' tirar
1640 '
1650 d%=INSTR('HANDLEDOOR',noun$)
1660 IF d%=0 THEN P$='Sin punto':GOSUB 2680: GOTO 1690
1670 IF d%=7 THEN ' ir a rutina puerta abierta
1680 IF position=2 THEN switch=-(1+switch): P$='La manil
la indica que esta'+sw$(ABS(switch))+'y en posicion':GOS
UB 2680 ELSE P$='No veo ninguna manilla':GOSUB 2680
1690 command$=nul$ : RETURN
1700 '
```

```
1710 ' inventario
1720 '
1730 temp=0
1740 P$="Llevas ":GOSUB 2680
1750 o%=0
1760 FOR i%=0 TO items
1770 IF objloc(i%)=temp THEN P$=""+object$(i%)+" ":GOSUB
 2680:o%=o%+1
1780 NEXT
1790 IF o%=0 THEN P$="...nada ":GOSUB 2680
1800 command$=nul$ : RETURN
1810 '
1820 ' cerrar
1830 '
1840 d%=INSTR("DOOR",noun$)
1850 IF d%<>1 THEN P$="No puedo cerrarla":GOSUB 2680: GO
TO 1910
1860 IF position>4 THEN P$="Que puerta?":GOSUB 2680: GOT
O 1910
1870 IF (position=1 OR position=2) AND frontdoor=closed
THEN P$="La puerta delantera ya esta cerrada":GOSUB 2680
: GOTO 1910
1880 IF (position=3 OR position=4) AND backdoor=closed T
HEN P$="La puerta trasera ya esta cerrada":GOSUB 2680: G
OTO 1910
1890 IF (position=1 OR position=2) THEN P$="Has cerrado
la puerta delantera":GOSUB 2680: frontdoor=closed
1900 IF (position=3 OR position=4) THEN P$="Has cerrado
la puerta trasera":GOSUB 2680: backdoor=closed
1910 command$=nul$ : RETURN
1920 '
1930 ' abrir
1940 '
1950 d%=INSTR("DOOR",noun$)
1960 IF d%<>1 THEN P$="No puedo abrirla":GOSUB 2680: GOT
O 2030
1970 IF position>4 THEN P$=".Que puerta?":GOSUB 2680: GO
TO 2030
1980 IF (position=1 OR position=2) AND frontdoor=open TH
EN P$="La puerta delantera ya esta abierta":GOSUB 2680:
GOTO 2030
1990 IF (position=3 OR position=4) AND backdoor=open THE
N P$="La puerta trasera ya esta abierta":GOSUB 2680: GOT
O 2030
2000 IF (position=1 AND frontdoor=closed) OR (position=4
 AND backdoor=closed) THEN P$="Mala Suerte. Te has queda
do fuera":GOSUB 2680:GOTO 2580
2010 IF position=2 THEN P$="Has abierto la puerta delant
era":GOSUB 2680: frontdoor=open
```

```
2020 IF position=3 THEN P$="Has abierto la puerta traser
a" :GOSUB 2680: backdoor=open
2030 command$=nul$:RETURN
2040 '
2050 ' terminar
2060 '
2070 P$=".De verdad quieres terminar ? (S/N) ":GOSUB 268
0
2080 INPUT "", command$
2090 IF UPPER$(command$)="S" THEN END
2100 command$=nul$ : RETURN
2110 '
2120 ' mirar
2130 '
2140 P$=" Estas en la "+loc$(position):GOSUB 2680
2150 P$=". La salida conduce al ":GOSUB 2680
2160 FOR i%=1 TO LEN(dir$(position))
2170 temp$=MID$(dir$(position),i%,1)
2180 P$=ex$(INSTR(direction$,temp$)-1)+", ":GOSUB 2680
2190 NEXT
2200 P$="y puedes ver ":GOSUB 2680
2210 temp=position
2220 GOSUB 1750
2230 command$=nul$ : RETURN
2240 '
2250 ' enchufar
2260 '
2270 IF position=5 THEN P$="Tu familia esta mirando la t
ele y te detienen.":GOSUB 2680:GOTO 2340
2280 IF objloc(1)<>10 THEN p$="El monitor no esta aqui":
GOSUB 2680:GOTO 2340
2290 IF objloc(3)<>10 THEN p$="El ordenador no esta aqui
":GOSUB 2680:GOTO 2340
2300 IF position<>10 THEN P$="Aqui no hay enchufe.":GOSU
B 2680: GOTO 2340
2310 IF INSTR("MONITORCTM640ORDENADORAMSTRADCPC464",noun
$)=0 THEN P$="No puedo enchufarlo.":GOSUB 2680: GOTO 234
0
2320 plugged=onn
2330 IF switch=off THEN P$="No funciona.":GOSUB 2680 ELS
E P$="Tu ordenador al fin funciona.":GOSUB 2680
2340 command$=nul$ : RETURN
2350 '
2360 ' ayuda
2370 '
2380 RESTORE 3190
2390 p$="Los comandos son :" : GOSUB 2680
2400 FOR h1=0 TO 25
2410 READ p$:p$=" "+p$:GOSUB 2680
```

```
2420 NEXT
2430 command$=nul$ : RETURN
2440 '
2450 ' 'terminado?'
2460 '
2470 fr=10 'finish room
2480 IF position<>fr THEN RETURN
2490 ready=(objloc(1)=fr) AND (objloc(3)=fr) AND (objloc
(4)=fr) AND (objloc(5)=fr) AND (objloc(6)=fr)
2500 ready=ready AND switch AND plugged
2510 catout=((objloc(0)=4) OR (objloc(0)=1)) AND backdoo
r=closed AND frontdoor=closed
2520 IF NOT catout AND objloc(3)=fr AND objloc(0)<>0 THE
N P$=' El gato sigue saltando sobre el teclado':GOSUB 26
80: objloc(0)=fr
2530 ready=ready AND catout
2540 IF NOT ready THEN RETURN
2550 PRINT
2560 p$='Bien hecho. Has terminado la aventura en':GOSUB
 2680
2570 p$=STR$(goes)+' intentos. Ahora puedes disfrutar de
 una tarde tranquila':GOSUB 2680
2580 PRINT:PRINT
2590 P$='.Quieres jugar otra vez? (S/N) ':GOSUB 2680
2600 INPUT '', command$
2610 IF UPPER$(command$)='S' THEN RUN
2620 END
2630 '
2640 ' rutina impresion
2650 ' Entrar con P$=text para imprimir
2660 ' Regresar con cursor en la siguiente pos.
2670 ' por lo que CRLF nec. antes >
2680 p$=p$+' '
2690 ln=LEN(p$):xp=POS(#0)
2700 IF INSTR(p$,' ')=0 AND (ln+xp)>(scrwidth-2) THEN PR
INT:xp=0
2710 FOR pr%=1 TO ln-1
2720 sp=INSTR(pr%,p$,' ')
2730 IF sp+xp >(scrwidth-2) THEN PRINT :xp=xp-scrwidth
2740 PRINT MID$(p$,pr%,1);
2750 NEXT
2760 RETURN
2770 '
2780 '
2790 ' Direcciones
2800 '
2810 DATA Norte,Sur,Este,Oeste,Bajo,Altos
2820 '
2830 ' No. lugares:(habit.)
```

```
2840 ' nombrar direcciones,resp.hab. num. (A=1 B=2...)
2850 '
2860 DATA el jardin delantero,N,B
2870 DATA la sala,NSEO,CAEH
2880 DATA la cocina,NS,DB
2890 DATA el jardin trasero,S,C
2900 DATA la sala de estar,O,B
2910 DATA el bano,E,H
2920 DATA el dormitorio delantero,N,H
2930 DATA el dormitorio,NSEOAB,IGJFKB
2940 DATA el dormitorio trasero,S,H
2950 DATA el cuarto trastero,O,H
2960 DATA el desvan,D,H
2970 '
2980 ' Objetos  No:(items+1)
2990 ' nombre,pos inic ( <=hab.)
3000 '
3010 DATA el gato negro,2
3020 DATA el monitor CTM640,3
3030 DATA el Televisor,5
3040 DATA el ordenador Amstrad CPC464,7
3050 DATA la mesa de trabajo,11
3060 DATA una silla de cocina,3
3070 DATA el manual del ordenador,9
3080 '
3090 ' objetos fijos desde no.7
3100 '
3110 DATA la manilla roja,2
3120 DATA el enchufe,10
3130 DATA la rosaleda,1
3140 DATA la papelera,4
3150 DATA la pintura valiosa,11
3160 '
3170 ' comando de ayuda
3180 '
3190 DATA CERRAR,CONECTAR,TIRAR,FINALIZAR,EXAMINAR
3200 DATA CONCLUIR,COGER,AYUDA,INVENTARIO,SALIR,LEVANTAR
3210 DATA LISTAR,MIRAR,ABRIR,RECOGER,ENCHUFAR,PRESIONAR,
TIRAR
3220 DATA EMPUJAR,PONER,TERMINAR,CERRADA,PARAR,TOMAR,TOR
CER,DONDE
```

Pasándolo a BASIC

Si ya has estudiado y comprendido el listado de este programa, puedes saltarte esta sección; en caso contrario continúa leyendo. El programa al comenzar expone los titulares. Luego te espera en la línea 360. Pulsando cualquier tecla haces que comience propiamente a funcionar.

Las líneas 430-460 fijan algunas variables cuyo propósito será más claro posteriormente. Las líneas 490 a 510 tomar las primeras seis constantes del censo de datos (línea 2810) y las asisgnan a la tabla literal 'ex$'. Las líneas 550 a 580 toman las siguientes 33 constantes del censo de datos (3*11, líneas 2860 a 2960) y lo apuntan como valores de las variables múltiples 'loc$', 'dir$' y 'dest$', respectivamente. Las líneas 620 a 650 tomar las siguientes 24 constantes del censo de datos y las asignan a las tablas 'object$' y 'objloc'. A partir de la línea 670 se preparan más variables, incluyendo 11 variables literales muy curiosas. Cada una de ellas está formada con unas cuantas palabras que todas pueden ser sinónimos significando la misma cosa, y serán usadas en el bucle principal del programa para ayudar al CPC464 a determinar lo que tus comandos significan.

Hagamos una pausa aquí para recapitular sobre las tablas (arrays), previamente comentadas en el Capítulo 2. Las tablas te permiten accesar a tu propia **discreción** y directamente 'colecciones de objetos' mucho más fácilmente. Para ilustrar eso, ejecuta el programa "aventura" y pulsa ESCape cuando te presente el aviso para teclear comandos.

Ahora teclea las siguientes líneas:

```
5000 FOR n=0 TO 11
5010 PRINT n;" ";loc$(n)
5020 NEXT n
```

Pero **no teclees RUN** para esto, o todos los valores de las variables se perderán. En lugar de eso, teclea:

```
CLS:GOTO 5000
```

Obtendrás una lista de todos los elementos de la tabla literal denominada 'loc$'. Ensaya con las otras tablas. Todas estas son tablas monodimensionales (y propiamente son las 'ristras') y son la base de todo el programa. Usando estas tablas, el CPC464 puede seguir la pista de dónde está cada objeto y en qué condiciones.

Volvamos ahora al listado. Los mensajes se exponen en pantalla por la subrutina que comienza en la línea 2680. Lo que se desea exponer queda asignado a la variable 'p$'. En la línea 2140 encontramos que 'p$' está formado a partir de dos literales que se empalman por el extremo. El primer literal es un estándar, "Estás en la ", y el segundo dependerá de tu posición. Inicialmente, tu posición es la cocina porque la variable de posición fue prefijada al valor 3 en la línea 710 y el elemento 3 de la tabla 'loc$' es "cocina".

Las subrutinas en la 2150 y 2200 combinan 'p$' de la misma manera para exponer las salidas y el contenido de la posición, respectivamente. Una vez que está hecho, es tu decisión teclear tus comandos.

Cuando haces eso, cualquier cosa que teclees es dado como valor impuesto a la variable de nombre 'command$'. Eso tiene luego que ser analizado y comprobado antes de que pueda realizarse lo que has mandado. Este análisis de la frase es lo que se denomina en inglés "parsing" (y procede del latín, de analizar las partes de una oración), de ahí el comentario del programador en la línea 960. Aquí la función INSTR demuestra sus méritos. Si se sobrepasa la línea 1000, entonces las siguientes líneas investigan lo que has introducido para ver si hay alguna palabra reconocible en eso. Cualquiera que sea reconocida (y son aquéllas contenidas en las variables literales curiosas de las líneas 720 a 820) es obedecida; en los demás casos se te presenta un comentario apropiado. Cada acción es efectuada en su propia subrutina y cada una de ellas está claramente marcada en el listado.

Unicamente cuando consigues todos los objetos correctamente situados en las habitaciones apropiadas y algunas otras condiciones más están satisfechas, es cuando termina el juego. Estudiando la rutina que comienza en la línea 2470 puedes hallar lo que necesitas hacer con el objetivo de terminar el juego. (Si deseas hallarlo jugando y no haciendo trampas, entonces sáltate los siguientes tres párrafos).

Comienza en la línea 2470. La habitación final es la habitación 10, i.e. la habitación que es el elemento décimo de la tabla 'loc$'. Ese es el "cuarto trastero". En la línea 2480 se efectúa una comparación para ver si tu corriente posición es la habitación 10, i.e. variable posición=10. Si no es así, entonces el juego continúa.

Si sí es la habitación correcta, pasamos a la línea 2490 que parece compleja pero realmente no lo es. El valor de la variable llamada 'ready' dependerá de los valores que tengan las cinco condiciones expresas en la línea. Si 'ready' va a ser cierta, las cinco condiciones deben ser ciertas. Estas cinco condiciones son simplemente los elementos de la tabla 'objloc', que son los objetos localizables y puestos en la habitación. Si repasas el censo de DATA (líneas 3010 a 3070) verás que los elementos 1, 3, 4, 5 y 6 son respectivamente el monitor CTM640, el ordenador Amstrad CPC464, una pequeña mesa de trabajo, la silla de la cocina y el manual del ordenador. El número después de cada uno de estos elementos es el número que identifica la habitación en la que están al comenzar el juego. Es ese número el que debe ser igual a 10 al final del juego.

La variable 'ready' no depende sólo de que los objetos sean correctos. Puedes ver en la línea 2500 que 'ready' también depende de las variables llamadas 'switch' (conmutador) y 'plugged' (enchufado), que han de ser asimismo, ciertas. Estas están prefijadas a partir de las líneas 1650 y 2270 respectivamente. Y finalmente, en la línea 2530 apreciarás que 'ready' también depende de la variable 'catout' (el gato fuera) cuyo valor se asigna en la línea 2510. Con el fin de que sea cierto que "el gato está fuera", el elemento 0 de la tabla 'loc$' (el gato negro) debe estar o bien en la posición 1 o en la 4 (jardín delantero o trasero) y ambas puertas deben estar cerradas. Si no es así, y tú no tienes agarrado al gato (línea 2520) entonces saltará sobre el teclado.

Para resumir, cuando tienes el monitor, el CPC464, el manual, la silla y la mesa en el cuarto trastero, has girado la manilla roja grande, enchufado, sacado el gato al jardín y cerrado las puertas delantera y trasera, es cuando has llegado a la terminación del juego.

Esta es la esencia de este juego de aventuras. A medida que te mueves por la casa, recogiendo cosas y cambiando el estado de cosas tales como las puertas, entonces las tablas y otras variables alteran su valor y quedan actualizadas. Si repasas unas cuantas páginas atrás, verás que exactamente eso es como nuestro esquema original en lenguaje para desarrollo de programas, fue **estructurada.**

TU CASA PUDIERA SER UN CASTILLO

Como ayuda adicional en tus esfuerzos para comprender cómo funciona esta aventura, haremos ahora algunas modificaciones al programa. Añadiremos otra habitación, también más objetos y otro objeto para incluir en el cuarto trastero. ¿Interesado? Si estás interesado, entonces prosigue...

¿Qué tal si ponemos un garaje al este del jardín delantero? La primera cosa a hacer es cambiar la variable 'rooms' (habitaciones) en la línea 430, pasándola de 11 a 12. Eso altera posteriormente la DIMensión de las tablas. Luego, añade una duodécima instrucción DATA a la lista que comienza en la línea 2860. Cada línea tiene tres constantes.

Primero el nombre de la ubicación -garaje. La segunda es la dirección o direcciones para salir de esa ubicación. Eso sería al oeste para pasar al jardín delantero -O de oeste. Finalmente, una clave de una letra para las ubicaciones adjuntas. Esas ubicaciones están codificadas de la A a la K para los elementos 1 a 11 de la tabla 'loc$'. El jardín delantero es el elemento 1, así que su letra clave es A. Teclea pues esta línea:

```
2970 DATA garaje,W,A
```

Ahora debemos alterar la información concerniente al jardín delantero. Revisa mediante EDIT o bien vuelve a teclear la línea 2860 para que sea ahora:

```
2860 DATA jardin delantero,NE,BL
```

La "E" se añade porque ahora ya puedes salir del jardín por el Este, y la "L" se añade porque siendo el elemento duodécimo de la tabla, el garaje tiene como letra clave L (duodécima letra del alfabeto).

Si ahora ejecutas el programa serás capaz de entrar y salir del garaje, pero desde luego no hay nada en él.

Así que ahora, vamos a poner algo dentro del garaje. ¡Un coche parece una idea conveniente! Primero necesitas cambiar la variable 'items' en la línea 430 de 11 a 12, luego añadir una duodécima línea DATA para el censo de objetos que comienza en la línea 3010. Observa que algunos objetos son movibles pero otros no lo son. El número de objetos movibles está determinado por la variable 'fixed' (inmóvil, fijado) en la línea 440. Por el momento, éste tiene el valor 7 que significa que los primeros 7 artículos pueden ser movidos, el resto no. No queremos que seas capaz de mover el coche, así que no cambies este valor. Y debes asegurarte que nuestra nueva línea DATA no pasa a ser una de las primeras 7. Incluye por tanto la siguiente línea:

```
3160 DATA coche de segunda mano,12
```

El 12 es el número que identifica la posición de esa habitación, el garaje -cuando ejecutes el programa habrá un coche ahí.

Ahora vamos a poner algo en el garaje que sí puedas sacar - una bicicleta. Otra vez habrá que cambiar la variable 'items' (línea 430), esta vez para que valga 13. Ahora cambia la variable 'fixed' de 7 a 8.

Nuestra línea DATA debe ser ahora una de las primeras ocho, así que incluye:

```
3080 DATA nueva bicicleta de marca,12
```

Ahora ejecútalo. Debes ser capaz de coger la bici, pero no el coche.

Finalmente, puedes alterar las condiciones de finalización del juego de manera que necesites una lecto-grabadora de datos en el cuarto trastero antes de que el juego pueda terminar.

Primeramente, incrementa las variables 'items' y 'fixed' otra vez en una unidad, y luego añade otra nueva línea:

```
3075 DATA lecto-grabadora,9
```

El 9 es el número del dormitorio trasero, el noveno elemento de la tabla 'loc$'. Para hacer que la lectograbadora sea uno de los artículos que debes recolectar, debes añadirlo a la línea que comprueba si la variable 'ready' (preparado) es cierta. Así que añade al final de la línea 2490 la condición:

```
AND (objloc(7)=fr)
```

La lecto-grabadora es el séptimo elemento en su tabla, y ahora debe estar en la habitación de terminación. Ejecuta el programa para asegurarte que funciona.

A estas alturas, debieras tener una buena idea de cómo funciona el programa. Así que ¿por qué no añades más habitaciones? una bodega o un cobertizo; o mejor todavía, ¿por qué no conviertes tu casa en un castillo? con los salones para los banquetes, sus torres, sus mazmorras, las doncellas que hay que rescatar y los dragones que hay que derrotar. Es bastante factible. Sólo exige tiempo, un pensamiento lógico y proceder paso a paso. Incluso aunque no estés seguro, haz un intento.

La mejor manera de aprender es experimentando. ¡El **aprendizaje** es también una aventura!

CONTROL DEL ESTUDIO

Habiendo comprobado tus facultades detectivescas jugando con AVENTURA, es ahora tiempo de comprobar tu capacidad retentiva de memoria mediante la autoevaluación TAE11.

Capítulo 12

Y LUEGO ¿QUE?

EL AMSTRAD AVANZADO

Hace tiempo, mucho tiempo (de hecho, antes en la Parte 1) hubo una sugerencia sobre el teclado numérico separado, situado a la derecha del teclado principal del CPC464, sobre cómo podía persuadírsele para producir efectos distintos que meramente hacer la vida más fácil al teclear datos numéricos. Probablemente ya hayas hecho uso de la forma abreviada mediante CTRL y ENTER para cargar y ejecutar programas. Lo que vas a aprender ahora es cómo puedes preparar tus propias funciones monotecla para ahorrar tiempo en el CPC464, y así acomodarlas a la clase de programas que escribes más a menudo o a uno particular que estás desarrollando en un momento dado.

Uno de los parámetros preceptuados que operan al encender inicialmente el CPC464 es la definición de los caracteres que pueden ser generados cuando se pulsan las teclas de este teclado numérico separado. Puedes modificar eso en cualquier momento. Primero, pulsa la tecla 0 de ese teclado. Consigues en pantalla un cero ¿o no? Ahora ensaya tecleando el siguiente comando:

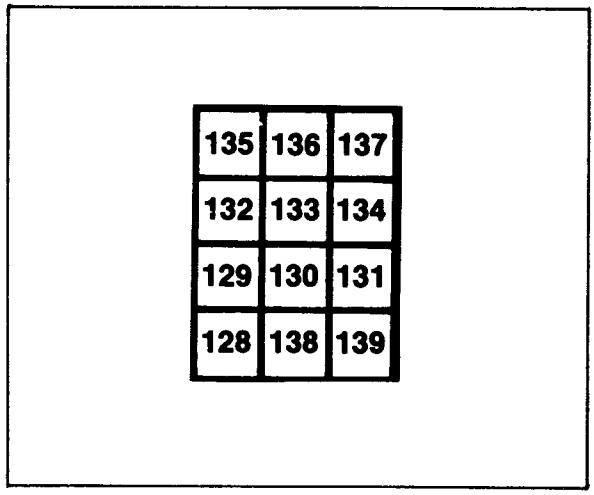
```
KEY 128,"hola"
```

Sorpresa, sorpresa, cuando pulsas otra vez la tecla cero consigues 'hola', en lugar del cero. Esta nueva palabra clave KEY (que generalmente vale como tecla, llave y clave) te permite adscribir cualquier literal que desees a cualquiera de las 12 teclas de este teclado numérico separado. La estructura del comando es:

KEY

```
KEY<numero tecla>,<expresion literal>
```

El parámetro **número de tecla** es un entero entre 128 y 140, que hace referencia a la posición real de la tecla tal y como aparece en el siguiente diagrama.



Observarás que los números sólo llegan al 139 y no al 140. La decimotercera tecla es nuestro viejo amigo ENTER que juntamente con CTRL, sabes muy bien que emite el comando:

```
run"
```

De manera que el parámetro **expresión literal** puede por tanto representar un comando para ejecución directa durante el tecleado de un programa. Teclea el siguiente comando:

```
KEY 129,"mode 2"
```

Si ahora pulsas la tecla marcada '1' observarás que:

```
mode 2
```

aparece en el monitor igual que si lo hubieras deletreado completamente. Si ahora pulsas la tecla RETURN ese comando será ejecutado exactamente igual que si lo hubieras tecleado.

¡Y eso no es todo! También puedes dar el comando en la forma:

```
KEY 129,"mode 2"+chr$(13)
```

Recuerda ese número. Es el código ASCII para la marca CR que por tradición corresponde a retorno de carro (el carro se refiere a aquellas máquinas de escribir electromecánicas que acostumbraban a ser la única manera de hablar con los ordenadores). Como puede que te hayas dado cuenta, eso es exactamente lo que consigues cuando pulsas la tecla ENTER. Así que, pulsando la tecla '1' en el teclado numérico separado, el comando se ejecuta inmediatamente y como consecuencia el ordenador se coloca en el MODO 2. Ensaya por ti mismo.

Aquí hay un ejemplo de un corto programa que puedes desear incluir en un cassette especial para desarrollo de programas, junto con la prefijación de tus valores favoritos para el reborde (BORDER), la pluma (PEN), el papel (PAPER) y la tinta (INK).

```
10 key 128,"mode 0"+chr$(13)
20 key 129,"mode 1"+chr$(13)
30 key 130,"mode 2"+chr$(13)
40 key 131,"list"
50 key 132,"list"+chr$(13)
60 key 133,"save"
70 key 134,"renum"
80 key 135,"auto"
```

Cuando estás escribiendo y depurando un programa largo, la posibilidad de tener estos comandos adscritos a las teclas, y que puedes aprovechar con sólo la pulsación de esa tecla, te ahorra un montón de trabajo. Puede que te encuentres con que clases diferentes de programas exigen adscribir expresiones literales diferentes. Si hay un montón de mensajes iguales usados en un programa por ejemplo, puede que te guste adscribir el mensaje (no sería comando) a una de esas teclas y no tener que tecleario completo cada vez.

CARGA RAPIDA

Otro de los parámetros cuyo valor está preceptuado al ponerse en marcha el CPC464 es la rapidez o velocidad con que se transfieren los programas desde o hasta la lecto-grabadora. Para una fiabilidad máxima, las dos datocassettes que acompañan este curso están escritas en esa velocidad (1000 baudios para los que tienen ese conocimiento técnico). Es posible doblar la velocidad de escritura en la cinta mediante el comando:

```
SPEED WRITE 1
```

Y ya sabes que speed es rapidez. Cualquier comando SAVE dado a continuación guardaría el programa en el cassette empleando esa velocidad de transferencia (2000 baudios).

Cuando un programa guardado en cassette a 2000 baudios es traído por el CPC464 del cassette y cargado en memoria, automáticamente se ajusta el CPC464 a esa velocidad de lectura. De hecho, puedes mezclar programas grabados a diferentes velocidades en la misma cinta.

Si usas cintas de alta calidad y mantienes el cabezal de lectura limpio, hay poca probabilidad de que tengas problemas al efectuar la lectura. Para volver a la velocidad primitiva puedes teclear el comando:

SPEED WRITE

```
SPEED WRITE 0
```

COPIA 'FIRME' EN PAPEL

El término equivalente 'hard copy' es la jerga de los anglófilos para designar la salida de información o el listado de los programas que se imprimen en papel, dado que el papel es 'duro' según ellos. Esta sección está pensada para cualquiera que haya comprado una impresora matricial de puntos Amstrad DMP-1 de manera que pueda obtener salida impresa con su CPC464.

Si enchufas la impresora en el zócalo de conexión situado en el panel trasero de tu CPC464, enciendes el ordenador y lanzas los comandos PRINT o LIST nada sucederá. Eso es porque todos los datos de entrada y salida del ordenador se manejan a través de cauces, y el cauce asignado a la impresora no ha sido especificado. Hay disponibles 10 de esos cauces, numerados del 0 al 9, pero no los analizaremos en detalle hasta la Parte 3 de este curso. Por el momento, nos limitaremos a los cauces identificados por los números 0 y 8.

El parámetro **número de cauce** es opcional en los comandos PRINT y LIST, y si se omite está preceptuado que se adopte el valor 0. Por ejemplo:

```
LIST 2000-
```

hará que el listado se exponga en la pantalla, a partir de la línea 2000 en adelante, dado que 0 es el número identificativo del cauce de exposición de datos por pantalla. Sin embargo, si das el comando en la forma:

LIST #8

```
LIST 2000-,#8
```

Todas las líneas a partir de la 2000 en adelante serán "expuestas" ('puestas fuera') a través del cauce identificado con ese número, que es el de envío hacia la impresora en lugar de hacia la pantalla.

Igualmente, puedes dar el comando:

PRINT #8

```
PRINT #8,'Esto es un ejemplo de salida por impresora'
```

y el literal será recibido por la impresora pertinente.

Si repasas el listado del programa de PRESUPuestos del Capítulo 5, la subrutina para imprimir el presupuesto parece que tiene una variable sin usar denominada '#str' (que es abreviatura de string=cauce, reguero de un arroyo). Si observas las instrucciones para exponer datos en esta subrutina, verás que es realmente esa variable 'str' la que está colocada como parámetro del comando PRINT, y dado que nunca tuvo asignado ningún valor, permanece con cero: el cauce hacia la pantalla. Así que si haces que 'str=8' obtendrás tu presupuesto por impresora.

BASE DE DATOS

Nuestro último programa en la datocassette A es posiblemente el mejor del lote.

Si quieres preparar un catálogo de tu colección de disco, de tus recetas favoritas, o de los nombres y direcciones de tus amigos, éste es un programa para eso. Es igual que un archivo electrónico con sus carpetas, excepto que usa montones de diferentes **rótulos** o etiquetas al mismo tiempo. Una vez que has preparado tu fichero database puedes examinarlo bajo montones de diferentes conceptos o argumentos de búsqueda. Sopas, por ejemplo, o rock, o lo que sea.

Hay un ejemplo de un fichero database grabado después del programa BASEDAT en el cassette. Cuando en el menú principal se te pide la opción que deseas, teclea "R"; luego se te pedirá el nombrefichero. Teclea

```
Paises
```

y aprenderás un poquito sobre geografía.

CONTROL DEL ESTUDIO

La prueba final de evaluación TAE12 repasará todos los temas tratados en esta parte del curso. Como dijimos antes, no es exactamente un exámen; es meramente una ayuda para que puedas comprobar que no hay nada que debes repasar para asegurarte que lo has comprendido antes de pasar a la siguiente etapa.

La siguiente etapa a partir de ahora es la Parte 3 de este curso. De ahora en adelante comenzarás a tener que aprender sobre el extraño mundo de la aritmética binaria, los números en hexadecimal y las interrupciones. ¡Buena suerte!

LISTA DE PROGRAMAS

La datocassette A contiene los siguientes programas en el mismo orden en que son mencionados en este libro. La datocassette B contiene las pruebas de autoevaluación (TAEs) que el lector deberá completar al final de cada capítulo, excepto el Capítulo 5.

Capítulo 1

PASTOR

Capítulo 2

CASA
CIUDAD

Capítulo 3

SENCOS
PASTEL

Capítulo 4

RELOJDIG
DESPERTA

Capítulo 5

PRESUP

Capítulo 6

PLANVUEL

Capítulo 7

SOPALE

Capítulo 8

BOMBARD

Capítulo 9

SONIDOS
DEMOSON

Capítulo 10

MINUET

Capítulo 11

AVENTURA

Capítulo 12

BASEDAT
PAISES

LISTA DE CLAVES

La siguiente es una lista, capítulo por capítulo, de todas la palabras clave del BASIC Amstrad comentadas en este libro. No se han tratado todas las variaciones y ampliaciones dado que, aunque este no es un libro para principiantes, el conocimiento de los procedimientos internos del CPC464 es necesario para la comprensión correcta.

Las palabras clave marcadas con un asterisco (*) se presentaron en la Parte 1, pero aquí se les ha dado un tratamiento más profundo.

Capítulo 2

DIM
READ
DATA
AUTO
RENUM
RESTORE
DELETE
LIST*

Capítulo 3

PEN*
PAPER*
INK*
PLOT*
DRAW*
ORIGIN*
CLG
TAG
TAGOFF

Capítulo 4

FOR-NEXT*
IF-THEN-ELSE*
WHIL-WEND

Capítulo 5

ZONE
PRINT*
PRINT USING
SPACE$
STRING$
SPC
TAB
INPUT*

Capítulo 6

SIN
COS
TAN
DEG
PI
RAD
ATN
SQR
ABS

Capítulo 7

INSTR
ASC
VAL
LEN
LEFT$
RIGHT$
MID$
STR$
TIME*
CHR$
AND
OR
XOR
UPPER$

Capítulo 8

SPEED INK
RND*
INT
LOCATE*
TEST
INKEY

Capítulo 9

SOUND*
ENV*
ENT*

Capítulo 12

KEY
PRINT #8
LIST #8
SPEED WRITE

INDICE

AMSTRAD
ESPAÑA
Avda. del Mediterráneo, 9 28007 MADRID