AMSTRAD CPC-464

PROGRAMACION AVANZADA

APROBADO POR AMSOFT PARA USUARIOS DEL CPC-464

MARK HARRISON

ra-ma

PROGRAMACION AVANZADA DEL AMSTRAD CPC-464

•

GUIA DEL USUARIO

•

MARK HARRISON

ta-ma
CHIQUINQUIRA, 28 (COCUY) - 28033 MADRID.
TELEF.: 764 50 95

Titulo de la obra original:
THE AMSTRAD CPC-464 ADVANCED USER GUIDE
por Mark Harrison

Traducido por:
M.C.Dopazo

Edición en inglés:
© SIGMA PRESS,
5 Alton Road,
Wilmslow,
Cheshire,
U.K.

Edición en español:
© RA-MA
c/ Chiquinquirá, 28 (Cocuy)
28033 Madrid
España

Reconocimiento

CPC-64 es una marca registrada por Amstrad Consumer Electronics plc. Nuestro agradecimiento a Amsoft, la división de productos de informática de Amstrad, por el permiso concedido para la inclusión del rótulo "Aprobado para los usuarios del CPC-64 por Amsoft".

La figura 1.2 se reproduce con permiso de Spectron Artist Ltd.

IMPRIME: SIGNO IMPRESORES, S.A.
D.LEGAL M-12496-1985
I.S.B.N. 84-86381-01-0

PROLOGO

Cualquier nuevo microprocesador atrae a un gran número de autores y editores; una máquina realmente interesante y completa es un terreno fértil para un autor con imaginación, por ello, nosotros en AMSOFT, estamos particularmente agradecidos a Mark Harrison por llenar la mayoría de los huecos que inevitablemente ocurren en el manual original del usuario, así como por hacer llegar al usuario normal, la mayoría de las utilidades generales de la máquina.

Es verdaderamente relevante encontrar un autor que entienda la máquina tan bien y tan rápidamente, ya que la mayoría de los libros que se escriben cuando acaban de aparecer las máquinas, son virtuales copias de libros similares para otros ordenadores. Es también sumamente gratificante encontrar un editor que desée continuar el estilo que nosotros hemos establecido para los libros de referencia de los sistemas - reduciendo al mínimo los problemas que se crean por una mala interpretación.

Hay muchísimas cosas que decir sobre el CPC464 - es realmente una máquina tanto para los principiantes como para los más experimentados. Esperamos que Mr. Harrison encuentre tiempo para continuar con su exploración, ya que libros como éste solo pueden beneficiar al producto y a sus usuarios.

William Poel

AMSOFT

CONTENIDO

LISTA DE PROGRAMAS

Estos programas ilustran tècnicas de programaciòn del CPC-464 y son de suma utilidad por sí mismos.

CAPITULO UNO

EL AMSTRAD

INTRODUCCION

El microprocesador Amstrad CPC 464 salió al mercado a mediados del año 1984. Tiene un diseño verdaderamente elegante y, a diferencia de otros micros de su mismo precio, es el único que se vende con su propio monitor y con un cassette incorporado. Esto le da una cierta ventaja sobre sus competidores, ya que el sistema completo consta solamente de dos unidades y un sencillo cable para la toma de corriente, dejando atràs la gran cantidad de cables que necesitan la mayoría de los microprocesadores.

En muchos aspectos el Amstrad se asemeja a un iceberg, con los dos tercios de su potencia fuera de la vista. El propòsito de este libro es progresar màs allà del simple entendimiento de la funciones bàsicas para llegar a conocer todas las posibilidades del Amstrad. En principio asumimos que el usuario ha leído ya el manual del Amstrad y conoce bien tèrminos como programa, flujo del programa, variables, BASIC, proceso de datos, 'software', 'hardware' etc. No le vamos a hacer perder el tiempo con este tipo de cosas, asi que pasemos a una visiòn general del sistema Amstrad.

LOS PRINCIPALES COMPONENTES DEL AMSTRAD CPC 464

El Amstrad CPC 464, como la mayoría de los ordenadores, tiene varios componentes 'hardware' en comùn, los màs importantes son la memoria, los dispositivos de E/S y la unidad central de proceso.

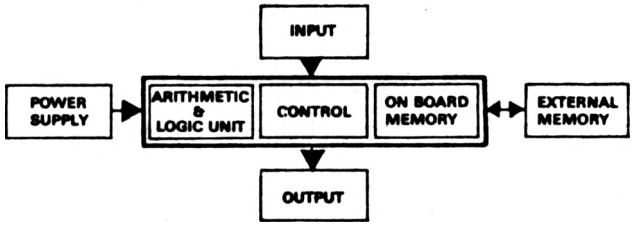


Fig 1.1 Los principales componentes del Amstrad CPC 464

Los componentes son:

PROCESADOR Z80

La unidad central de proceso(CPU) del Amstrad es una Z80A y es la responsable de controlar todas las operaciones del

ordenador. Es un procesador de bajo costo, adecuado para los sistemas pequeños y usado en muchas de las màquinas màs populares como el Lynx, Sord o Sinclair por nombrar algunas. Otra funciòn de la CPU es la de evaluar cualquier expresiòn matemàtica que implique suma, resta, multiplicaciòn o divisiòn(aritmètica) y tambièn verificar si los nùmeros son positivos, negativos o cero(lògica). La necesidad de las funciones aritmèticas es bastante clara. las funciones lògicas son necesarias para que el ordenador pueda tomar decisiones y conocer las acciones que debe realizar posteriormente.

DISPOSITIVOS DE ENTRADA/SALIDA

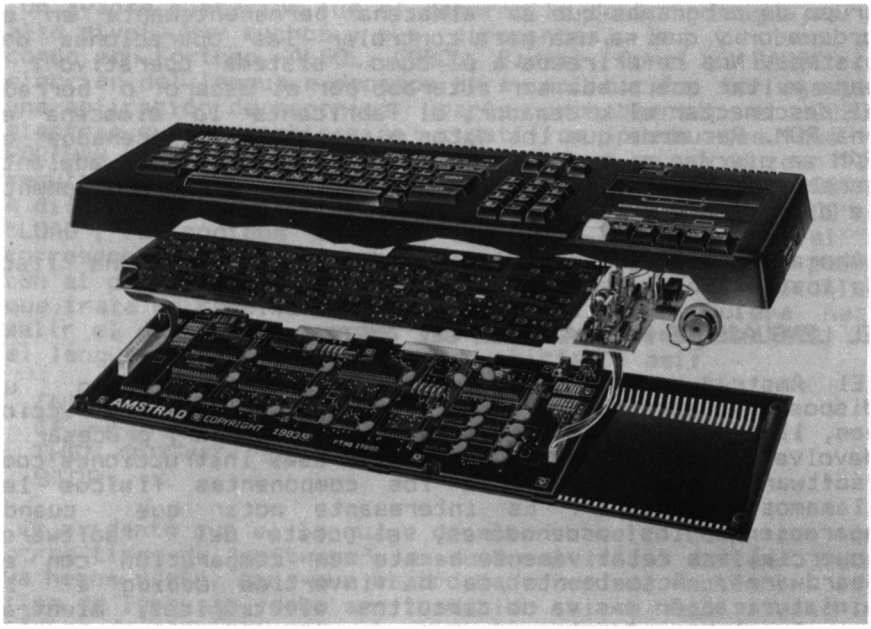
Los datos son introducidos y accedidos desde el Amstrad por medio de los dispositivos de entrada y salida (E/S), tales como el teclado y los 'joystick' para entrada, la pantalla y el altavoz para salida, y el cassette o la unidad de discos para ambos, entrada y salida. El Amstrad se vende con monitor de media resoluciòn en color o con monitor monocromo verde de alta resoluciòn. El teclado, de teclas mòviles, està construido de forma estàndar con disposiciòn 'qwerty' y con un teclado numèrico adicional. El cassette que lleva incorporado usa un sistema de grabaciòn avanzado para transferencia de datos y no es compatible con otros sistemas. En la parte posterior de la màquina hay varias puertas de expansiòn para interconectarse con impresoras, discos y otros perifèricos. Actualmente se està desarrollando un sistema de ficheros en disco que serà compatible con CP/M.

MEMORIA

La funciòn de la memoria es la de almacenar todos los datos que tiene el sistema. El Amstrad contiene dos tipos diferentes, memoria de solo lectura (ROM) y memoria de acceso aleatorio (RAM), la diferencia entre ellas es:

ROM	RAM
Esta memoria ha sido pre-programada por el fabricante con datos permanentes. Su contenido no puede ser alterado.	Es una memoria de uso general en la que se almacenan los datos y se sacan cuando se necesitan. la RAM pierde su contenido cuando se apaga el ordenador.

Una secciòn de la memoria RAM del Amstrad està reservada para el 'software' del sistema, que està compuesto por un



Algunas características de interès de la placa inferior son:

La última fila de chips de izquierda a derecha:
El generador de sonido, el controlador periférico, procesador Z80A, ULA de Ferranti.
Los de la derecha, en frente del ULA, son los 64K de RAM.

La ROM está en el extremo izquierdo, frente al generador de sonido.

En la parte trasera se ven: la puerta de E/S, puertas de usuario, interfase de impresora, interfase de discos, entrada de 5V (del monitor) y el conector del monitor.

El pequeño circuito de la derecha es el controlador del cassette.

El circuito largo debajo del teclado, es su matriz de contactos.

Fig 1.2 El sistema del microordenador Amstrad CPC 464

grupo de programas que se almacena permanentemente en el ordenador y que se usa para controlar las operaciones del sistema. Nos referiremos a èl como 'sistema operativo'. Y para evitar que pueda ser alterado por el usuaro o borrado al desconectar el ordenador, el fabricante lo almacena en una ROM. Recuerde que los datos o programas almacenados en RAM se pierden si se desconecta el aparato. Màs adelante veremos un mapa de la memoria del Amstrad, pero de momento le diremos que tiene 64K de RAM y 32K de ROM.

Ahora vamos a ver còmo el Amstrad puede ejecutar una lista de instrucciones.

EL LENGUAJE DE PROGRAMACION DEL AMSTRAD

El Amstrad puede ser descrito brevemente como un dispositivo electrònico que utiliza una lista de instrucciones, llamadas programa, para recibir, memorizar, procesar y devolver datos. Solemos referirnos a esas instrucciones como "software" mientras que a los componentes fìsicos les llamamos "hardware". Es interesante notar que cuando aparecieron los ordenadores, el costo del "software" comercial era relativamente barato en comparaciòn con el "hardware". Actualmente se ha invertido debido a la miniaturizaciòn masiva de circuitos electrònicos, mientras que la labor de los programadores que desarrollan el "software" se ha incrementado.

Desafortunadamente, un ordenador es relativamente estùpido y solo puede reconocer instrucciones en forma de còdigos numèricos simples, tales como 50, 27 o 114. Por ejemplo, la instrucciòn para sumar dos nùmeros puede ser 35. Ya que el programa puede contener màs de una instruciòn para que sea prâctico, ese programa podrìa ser asì:

35,27,133,211,23,39,5,221,10 ..... etc.

Obviamente, este programa no tiene ningùn significado para el que no conoce las instrucciones que representa cada còdigo. Este tipo de programa se conoce como còdigo de màquina y puede ser bastante difìcil de usar. Los primeros programadores no tuvieron otra alternativa, pero enseguida se dieron cuenta de que era màs fâcil para ellos usar còdigos que se parecieran màs a las instrucciones reales (como "ADD","LOAD"). En este punto es preciso recordar al lector que el ordenador sòlo puede entender un programa escrito en còdigo de màquina, de modo que si el programa es escrito en còdigo diferente, debe ser traducido al còdigo de màquina equivalente. Este proceso de traducciòn es realizado por otro programa del ordenador.

Un avance posterior fuè el desarrollo de los lenguajes de alto nivel. Hay muchos y muy diferentes, los de uso màs comùn son Fortran, ALGOL, COBOL, Pascal, Coral y BASIC. La elecciòn del lenguaje depende de su aplicaciòn. Por ejemplo, una aplicaciòn de negocios usarà, probablente, el COBOL, mientras que una aplicaciòn cientìfica puede que use el FORTRAN o una aplicaciòn que requiera que se produzcan los resultados en unos lìmites de tiempo estrictos, usarà CORAL. A diferencia de los còdigos de bajo nivel, como "ADD" y "LOAD", los còdigos de los lenguajes de alto nivel no corresponden uno a uno con los de màquina , sino que permiten al programador acoplarse a la naturaleza del problema que trata de resolver. Por ejemplo, si se requiere hacer salir el mensaje "Correcto" cuando la respuesta sea 99, en el lenguaje de alto nivel lo escribirìamos asì:

```
INPUT respuesta
IF respuesta=99
THEN OUTPUT "Correcto";
STOP;
```

Es evidente que el lenguaje de alto nivel, comparado con otros tipos de "software", es mucho màs fàcil de leer. Como ya hemos visto, para que el ordenador pueda entender este tipo de lenguaje, debe ser traducido a su equivalente en còdigo de màquina; una vez que la versiòn en còdigo de màquina està funcionando, no importa de donde ha venido. Hay dos mètodos para traducir los lenguajes de alto nivel. El primero usa un programa llamado "compilador" que convierte todo el programa de alto nivel en còdigo de màquina de una sola vez, dejando la versiòn de màquina lista para ser usada tantas veces como sea necesario. El segundo mètodo, que es el que usa el Amstrad y otros micros que funcionan con BASIC, es lo que se llama un "intèrprete" que convierte pequeñas partes de lenguaje de alto nivel en còdigo de màquina, a medida que se va ejecutando. Interpretar programas es mucho màs lento que ejecutar una versiòn compilada pero, generalmente es màs fàcil de usar en un ordenador personal.

El lenguaje de programaciòn que usa el Amstrad es una de las mùltiples versiones del BASIC (Beginners All-purpose Symbolic Instruction Code). Este lenguaje simple fuè diseñado originalmente para los principiantes pero se ha hecho el màs popular de los lenguajes de programaciòn. A pesar de que se hicieron grandes esfuerzos para normalizar las sentencias del BASIC, ha habido una tendencia en los fabricantes de ordenadores a producir sus propias versiones con sentencias propias de sus màquinas. Como resultado tenemos algunas sentencias disponibles en otros ordenadores que no lo estàn en el Amstrad, aunque este pequeño inconveniente se puede

salvar usando una o varias sentencias del Amstrad. Del mismo modo, el Amstrad, tiene varias sentencias que no se encuentran en otros ordenadores.

Aunque hemos incluido una lista de las sentencias de BASIC disponibles en el Amstrad, no está en el ánimo de este libro el enseñar los fundamentos de la programación BASIC, ya que existen numerosos libros en este campo. Le recomiendo que use el resto de este capítulo como una simple lista de referencia. Echela una rápida ojeada en este momento y úsela posteriormente como referencia, si lo necesita.

Para cada palabra clave que se menciona, la sintaxis requerida aparece en el siguiente formato:

COMANDOS DE BASIC DEL AMSTRAD

Palabras clave BASIC, en mayúsculas.
datos variables, entre "<>"
datos opcionales, entre corchetes "[]"
datos repetitivos van seguidos por ...

COMANDOS DEL SISTEMA

Comando	Propósito	Sintaxis
AUTO	Genera números de línea automáticamente	AUTO [<línea>][,<increm>]
CAT	Lista ficheros en cinta	CAT
CLEAR	Limpia las variables	CLEAR
DELETE	Borra líneas de programa	DELETE[<num>][,][<num>]
EDIT	Entra en modo edición	EDIT<num lin>
LIST	Lista el programa	LIST[<num lin>][-][<num lin>][,#<fuente>]
LOAD	Carga programas	LOAD <fichero>[,<direcc>]
MEMORY	Restaura los apuntadores del BASIC	MEMORY <dirección>
MERGE	Mezcla un programa del cassette con el de la memoria	MERGE <fichero>
NEW	Borra la memoria	NEW

RENUM	Renumera el programa o parte de èl	RENUM[<num nuevo>][,[<num viejo>][,<incremento>]]
RUN	Ejecuta el programa	RUN[<fichero> o <num lin>]
SAVE	Salva programas o memoria en el cassette	SAVE<fichero>[,<tipo>][[,<paràmetros>]]
TROFF	Quita la facilidad trace	TROFF
TRON	Pone la facilidad trace	TRON

SENTENCIAS DE ASIGNACION

Sirven para asignar valores numèricos o secuencias de caractères a secciones específicas de la memoria del ordenador.

Comando	Propósito	Sintàxis
DATA	Especifica una lista de valores para el READ.	DATA <dato>[,<dato>]....
LET	Asigna un valor o una secuencia de caracteres a una variable.	[LET]<variable>=<expresión>
MID$	Reemplaza parte de una cadena. También se puede usar como función.	MID$(<cadena>,<princ>[,<long>])=<cadena>
POKE	Asigna un valor a una posición de memoria.	POKE <dirección>,<expresión>
READ	Toma datos de una sentencia DATA y los asigna a las variables	READ <variable>[,<variable>] ....
RESTORE	Posiciona el apuntador del READ a la línea especificada. A la primera, si se omite.	RESTORE [<num linea>]

SENTENCIAS DE CONTROL

Estas sentencias son de las màs vitales del repertorio, y nos permiten controlar el orden en que se ejecutarán las instrucciones del programa.

Comando	Propósito	Sintaxis
AFTER	Invoca una subrutina en un intérvalo dado.	AFTER <entero>[,<entero] GOSUB<num lin>
CALL	Llama a una subrutina externa.	CALL<direcc>[,<param]....
CHAIN	Carga y ejecuta un programa.	CHAIN[MERGE]<fichero>[,<num lin>]
CONT	Continúa desde la parada anterior.	CONT
END	Fin del programa	END
ERROR	Toma una acción de error	ERROR<entero>
EVERY	Invoca a una subrutina a intérvalos regulares	EVERY<entero>[,<entero>] GOSUB<num lin>
FOR	Controla un bucle de una sección del programa por el valor de una variable que es incrementada	FOR<var>=<valor>TO<valor> STEP<incremento>
GOSUB	Llama a una subrutina	GOSUB<num lin>
GOTO	Salta a una línea dada	GOTO<num lin>
IF... THEN... ELSE	Si la condición es cierta, ejecuta una acción o salta a una línea específica.	IF<cond>THEN[<sentencia> o <num lin>][ELSE<sentencia>]
NEXT	Fin del bucle FOR	NEXT[<variable>]
ON{GOSUB GOTO}	Pasa control a la línea o subrutina cuya posición corresponda al entero de la expresión.	ON<expres>{GOSUB o GOTO} <num lin>[,<num lin>]....
ON BREAK GOTO	Permite tomar la rutina de interrupción de BREAK	ON BREAK GOTO <num lin>
ON BREAK STOP	Desactiva la rutina de interrupción de BREAK	ON BREAK STOP
ON ERROR GOTO	Pone una trampa de error	ON ERROR GOTO <num lin>

ON SQ GOSUB	Permite la interrupción de la cola de sonido	ON SQ<canal>GOSUB<num lin>
RESUME	Reanuda la ejecución después de una trampa	RESUME[{NEXT o <num lin>}]
RETURN	Retorna de un GOSUB	RETURN
WEND	Fin de un bucle WHILE	WEND
WHILE	Controla un bucle de programa mientras se cumple una condición	WHILE<condición>

SENTENCIAS DE ENTRADA/SALIDA

Permiten que se introduzca o se obtenga información:

Comando	Propósito	Sintaxis
CLOSEIN	Cierra fichero entrada	CLOSEIN
CLOSEOUT	Cierra fichero salida	CLOSEOUT
ENT	Define envolvente tono	ENT<num envolv>[,<param>]...
ENV	Define envolvente volum.	ENV<num envolv>[,<param>]...
INPUT	Introduce datos	INPUT[#<cauce>,][;]["<mensaje>";]<var>[,<var>]..
LINE INPUT	Introduce línea	LINE INPUT[#,<cauce>,][;]["<mensaje>";]<var cadena>
OPENIN	Abre fichero de entrada	OPENIN <fichero>
OPENOUT	Abre fichero de salida	OPENOUT <fichero>
OUT	Saca un valor por una puerta de E/S	OUT<puerta>,<expresión>
PRINT	Saca datos	PRINT[#<cauce>,]<dato>[<separador><dato>]....
RELEASE	Libera canal de sonido	RELEASE<canales>
SOUND	Pone un sonido en cola	SOUND<estado>,<periodo tono> [,<duración>[,<vol>[,<envol vol> [,<envol tono>[,<period o ruido>]]]]]

SPEED KEY	Velocidad repeticiòn de teclas	SPEED KEY<demora comienzo>,<demora repeticiòn>
SPEED WRITE	Velocidad escritura del cassette	SPEED WRITE<expresiòn>
WAIT	Espera en puerta E/S	WAIT<puerta>,<màscara>[,<inversiòn>]
WIDTH	Ancho linea impresora	WIDTH <expresiòn>
WRITE	Saca datos	WRITE[#<cauce>,]<dato>[<separador><dato>]....

SENTENCIAS DE GRAFICOS

Se usan para controlar las facilidades de pantalla disponibles en el Amstrad.

Comando	Propòsito	Sintàxis
BORDER	Cambia color del borde	BORDER<color1>[,<color2>]
CLG	Limpia pantalla gràfica	CLG[<màscara tinta>]
CLS	Limpia ventana	CLS[#<cauce>]
DRAW	Dibuja una linea desde la posiciòn actual de PLOT a una posiciòn absoluta.	DRAW<x>,<y>[,<màscara tinta>]
DRAR	Dibuja una linea desde la posiciòn actual de PLOT a una posiciòn relativa.	DRAWR<desp x>,<desp y>[,màscara tinta>]
INK	Pone color de tinta	INK<tinta>,<color1>[,<color2>]
LOCATE	Posiciona el cursor	LOCATE[#<cauce>,]<x>,<y>
MODE	Pone modo de pantalla	MODE<expresiòn>
MOVE	Mueve posiciòn de PLOT a posiciòn absoluta	MOVE<x>,<y>
MOVER	Mueve posiciòn de PLOT a posiciòn relativa	MOVER<desp x>,<desp y>

ORIGIN	Recoloca el origen de la pantalla (0,0)	ORIGIN<x>,<y>[,<izq>,<der>,<arriba>,<abajo>]
PAPER	Pone color de fondo	PAPER[#<cauce>,]<màsc tinta>
PEN	Pone color de tinta	PEN[#<cauce>,]<màsc tinta>
PLOT	Dibuja un punto en una posición absoluta	PLOT<x>,<y>[,màscara tinta>]
PLOTR	Dibuja un punto en una posición relativa	PLOTR<desp x>,<desp y>[,<màscara tinta>]
SPEED INK	Pone periodo parpadeo	SPEED INK<periodo>,<periodo>
SYMBOL	Define un carácter con una matriz de 'bits'	SYMBOL<num car>[,<datos fila>]....
TAG	Permite poner texto en la posición de PLOT	TAG[#<cauce>]
TAGOFF	Cancela el TAG	TAGOFF[#<cauce>]
WINDOW	Crea una ventana	WINDOW[#<cauce>,]<izq>,<der>,<arriba>,<abajo>
ZONE	Pone zona impresión	ZONE<expresión entera>

Y finalmente,

OTRAS SENTENCIAS

Comando	Propósito	Sintáxis
DEF FN	Define funciones de usuario	DEF FN<nombre>(<paràmetro>[,<paràmetro]...)=<expresión>
DEFINT DEFREAL DEFSTR	Pone tipos por defecto (entero,real o cadena)	DEFINT<rango de letras> DEFREAL<rango de letras> DEFSTR<rango de letras>
DEG	Pone modo grados	DEG
DI	Anula interrupciones	DI
DIM	Prepara matrices	DIM<variable>(<subind>[,<subind>]...)[,<variable>(<subind>[,<subind>]...]...
EI	Permite interrupciones	EI

ERASE	Borra matrices	ERASE<variable>[,<variable>]
KEY	Pone teclas de funciòn	KEY<num funciòn>,<cadena>
KEY DEF	Redefine teclas	KEY DEF<num tecla>,<rep>[,<normal>[,<mayus>[,<ctrl>]]]
RAD	Pone modo radianes	RAD
RANDOMIZE	Inicia nùmero aleatorio	RANDOMIZE[<expresiòn num>]
REM	Comentario	REM<comentario>

FUNCIONES DE BASIC DEL AMSTRAD

Una funciòn es una regla que relaciona un grupo de valores con otros. El valor del primer grupo es conocido por argumento y el del segundo es el resultado.

FUNCIONES SIMPLES, NUMERICAS Y MATEMATICAS

Son funciones especializadas que tienen numerosas aplicaciones en matemàticas, ciencias e ingenierìa. Se ha incluido tambièn un generador de nùmeros aleatorios.

Funciòn	Propòsito(resultado)	Sintàxis
ABS	Valor absoluto	ABS(<expresiòn num>)
ATN	Arco tangente	ATN(<expresiòn num>)
CINT	Convierte a entero	CINT(<expresiòn num>)
COS	coseno	COS(<expresiòn num>)
CREAL	Convierte a real	CREAL(<expresiòn num>)
EXP	Valor exponencial	EXP(<expresiòn num>)
FIX	Trunca a entero	FIX(<expresiòn num>)
INT	Redondea a entero	INT(<expresiòn num>)
LOG	Valor logaritmo natural	LOG(<expresiòn num>)
LOG10	Valor logaritmo comùn	LOG10(<expresiòn num>)
MAX	Determina valor màximo	MAX(<expresiòn num>[,<expresiòn num>]...)

MIN	Determina valor mínimo	MIN(<expresión num>[, <expresión num>]...)
PI	Constante 3.141592653	PI
RND	Genera número aleatorio	RND[(<expresión num>)]
ROUND	Redondea el valor a un número de decimales dado	ROUND(<expresión num>,[<posiciones>])
SGN	Valor del signo	SGN(<expresión num>)
SIN	Valor del seno	SIN(<expresión num>)
SQR	Raiz cuadrada	SQR(<expresión num>)
TAN	Tangente	TAN(<expresión num>)
UNT	Convierte valor sin signo a entero	UNT(<dirección>)

FUNCIONES DE CADENA

Estas son las funciones para manipular caracteres y texto.

Función	Propósito(resultado)	Sintáxis
ASC	Código ASCII	ASC(<cadena>)
BIN$	Convierte un número en una cadena binaria	BIN$(<entero sin signo>[, <ancho del campo>])
CHR$	Carácter cuyo código ASCII es especificado	CHR$(<expresión num>)
HEX$	Convierte un número en una cadena hexadecimal	HEX$(<entero sin signo>[, <ancho del campo>])
INSTR	Busca una subcadena	INSTR([<posición>,]<cadena buscada>,<cadena>)
LEFT$	Saca la parte izquierda de una cadena	LEFT$(<cadena>,<longitud>)
LEN	Longitud de una cadena	LEN(<cadena>)
LOWER$	Convierte a minúsculas	LOWER$(<cadena>)
MID$	Saca una subcadena	MID$(<cadena>,<posición>, <longitud>)

RIGHT$	Saca la parte derecha de una cadena	RIGHT$(<cadena>,<longitud>)
SPACE$	Cadena de espacios	SPACE$(<longitud>)
STR$	Convierte un número en una cadena de dígitos	STR$(<expresión num>)
STRING$	Cadena del carácter especificado	STRING$(<long>,<carácter>)
UPPER$	Convierte a mayúsculas	UPPER$(<cadena>)
VAL	Convierte una cadena de dígitos en números	VAL(<cadena>)

FUNCIONES DE E/S

Estas se usan con las operaciones de E/S.

Función	Propósito(resultado)	Sintáxis
INKEY	Estado de una tecla	INKEY(<num tecla>)
INKEY$	Carácter de una tecla	INKEY$
INP	Entrada de una puerta	INP(<num puerta>)
JOY	Estado de "joystick"	JOY(<num joystick>)
POS	Posición de un cauce	POS(#<cauce>)
REMAIN	Resto que queda en el contador de demora	REMAIN(<num contador>)
SQ	Estado de cola de sonido	SQ(<canal>)
VPOS	Posición cauce vertical	VPOS(#<cauce>)

FUNCIONES DE SISTEMA

Se usan para informarse del estado del sistema.

Función	Propósito(resultado)	Sintáxis
EOF	Prueba fin de fichero	EOF
ERR	Número de error	ERR
ERL	Número de línea donde ocurrió el error	ERL

FRE	Memoria libre o libera espacio sin usar	FRE(0) FRE(" ")
HIMEM	Direcciòn màs alta de memoria para BASIC	HIMEM
PEEK	Contenido de una direcciòn de memoria	PEEK(<direcciòn>)
TEST	Color de tinta en la posiciòn especificada	TEST(<x>,<y>)
TESTR	Color de tinta en la posiciòn relativa desde la actual de PLOT	TESTR(<desp x>,<desp y>)
TIME	Tiempo(1/300 seg) desde la puesta en marcha	TIME
XPOS	Coordenada horizontal de la posiciòn de PLOT	XPOS
YPOS	Coordenada vertical de la posiciòn de PLOT	YPOS

Para una descripciòn màs detallada de las sentencias disponibles, refièrase al manual del usuario del Amstrad.

A continuaciòn vamos a dar un curso de programaciòn para explorar algunas de las tècnicas màs avanzadas del Amstrad. Todas las facilidades que se traten irán ilustradas con varios programas de ejemplo que pueden ser usados y alterados para ajustarlos a sus propias necesidades, asi que sièntase libre para cambiar en ellos todo lo que quiera.

CAPITULO DOS

MANIPULACION DE CADENAS Y CARACTERES

Se conoce por "cadena" a una secuencia de caracteres alfanumèricos o gràficos que no necesitan ser de naturaleza numèrica. El Amstrad tiene varias facilidades que permiten usar cadenas de caracteres, ademàs de los nùmeros puros, para aumentar la versatilidad de sus programas. Enseguida veremos que las funciones de cadenas del Amstrad son fundamentales para los programas que requieren manipulaciòn de caracteres.

EL JUEGO DE CARACTERES DEL AMSTRAD

El Amstrad tiene un juego de 256 caracteres. Estos incluyen caracteres numèricos y alfabèticos, letras griegas, sìmbolos gràficos, signos de puntuaciòn, sìmbolos matemàticos, caracteres de control, etc. El juego de caracteres del Amstrad està basado en el ASCII (American Standard Code of Information Interchange), sin embargo, como la mayoría de los micros que se fabrican, tiene algunas alteraciones para cubrir sus propias necesidades, de modo que no es del todo estàndar. Cada caràcter se distingue por su propio còdigo que es un nùmero entre 0 y 255. Los caracteres y sus còdigos aparecen listados en el apèndice A, y se pueden convertir mediante las funciones ASC y CHR$.

Por ejemplo:
Estas dos expresiones
ASC("A") y ASC("AMSTRAD") examinan el primer caràcter y devuelven 65, el còdigo ASCII de "A"

De modo similar
CHR$(65)
nos devuelve el carácter "A"

Intente confirmar los còdigos ASCII de otros caracteres en su Amstrad; por supuesto debe usar el comando PRINT para visualizar los resultados, ej. PRINT ASC("464")

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

Ademàs de los caracteres esperados(como a,b,c... o 1,2,3) hay varios caracteres que representan funciones de control y causan algunas irregularidades en la pantalla. Estos son los caracteres ASCII estàndar y estàn localizados en las 32

primeras posiciones. El siguiente programa reduce el juego de caracteres visualizado a aquellos que son imprimibles.

```
10 FOR j = 32 TO 255
20 PRINT CHR$(j);
30 NEXT j
```

OPERADORES DE CADENA

Podemos comparar dos cadenas entre sí usando el signo '=', obteniendo un resultado "verdadero", si son iguales, ej. todos los caracteres son iguales y los espacios están en la misma posición (incluidos los del final). El uso más obvio es su incorporación en una sentencia IF...THEN.

ej.

```
100 INPUT "CONTINUE";q$
110 IF q$ = "NO" OR q$ = "no" THEN STOP
```

Algunas veces los espacios del final de una cadena pueden confundirnos en estas comparaciones. Estos espacios pueden ser descubiertos con la función LEN que nos devuelve el número de caracteres de la cadena.

```
PRINT LEN(a$)
```

Las cadenas también se pueden comparar con operadores ">" y "<"; una cadena es menor que otra si la precede alfabéticamente, como en un directorio telefónico. Lo usaremos más adelante al escribir un programa de clasificación alfabética.

Finalmente, las cadenas pueden ser unidas(concatenadas) mediante el operador "+"

ej.

```
10 a$ = "AMSTRAD "
20 b$ = "CPC 464"
30 c$ = a$ + b$
40 PRINT c$
```

Observe, sin embargo, que las cadenas no pueden ser restadas, multiplicadas, divididas ni elevadas a una potencia.

FUNCIONES DE MANIPULACION DE CADENAS

El Amstrad tiene tres funciones de cadena que son de un valor inapreciable para manejar texto y caracteres. Esta son LEFT$, RIGHT$ y MID$ y se usan para extraer una subcadena de

otra cadena. LEFT$(a$,n) y RIGHT$(a$,n) extraen n caracteres de la izquierda y de la derecha de la cadena a$, respectivamente.

ej.
Si a$ = "ABCDEFGHIJ"
entonces LEFT(a$,3) nos devolverà "ABC"
y RIGHT(a$,4) nos devolverà "GHIJ"

MID$ es una sentencia màs potente, se puede usar para obtener trozos de una cadena, a diferencia de LEFT$ y RIGHT$ que estàn limitados al principio y al final de la cadena.

MID(a$,n,m) extraerà m caracteres a partir del nûmero n de la cadena. Si omitimos el argumento m, se obtienen todos los caracteres que siguen al n.

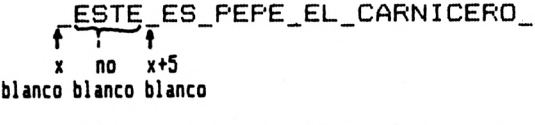
La versiòn del Amstrad tiene una ventaja màs que otras versiones del MID$, y es que se puede usar para asignar a una cadena una secciòn de otra. El trozo que especifica MID$ serà el destino de la cadena que se va a asignar.

ej. MID$(a$,n,m) = b$ asignarà los primeros m caracteres de b$ a a$, empezando por el carâcter nûmero n.

El programa 2 ilustra el uso del troceado de cadenas con la funciòn MID$. Se introduce una sentencia en español y la muestra despuès con todas las palabras de cuatro letras emborronadas.

Se introduce la cadena a$ de forma que los caracteres primero y último sean blancos. Prepara un apuntador x para moverse a lo largo de cada carâcter. Primero mira si el carâcter x es un espacio, despuès mira si los cuatro caracteres siguientes no son espacios y, finalmente, que el siguiente es un espacio. Si se cumplen todas estas condiciones quiere decir que la palabra tiene cuatro letras y las reemplaza por cuatro estrellas. Si no se cumplen todas estas condiciones, x es incrementado en 1 y pasa al siguiente grupo de caracteres.

ejemplo



PROGRAMA 2: MARIA CASABLANCA

```
10 CLS
```

```
20 PRINT "EL CENSOR DE SILICONA ATACA DE NUEVO"
30 PRINT : PRINT "Introduzca una frase"
40 PRINT : INPUT a$ : a$ = " "+a$+" "
50 FOR X = 1 TO LEN(a$)-5
60 IF MID$(a$,x,1)<>" " THEN 120
70 FOR y = 1 TO 4
80 IF MID$(a$,x+y,1) = " " THEN 120
90 NEXT y
100 IF MID$(a$,x+5,1)<>" " THEN 120
110 MID$(a$,x+1,4) = "****"
120 NEXT x
130 PRINT : PRINT a$
140 PRINT : INPUT "Otra vez?";q$
150 IF LEFT$(q$,1) = "S" OR LEFT$(q$,1) = "s"
THEN 10
160 END
```

MAS FUNCIONES DE CADENA

En algunas aplicaciones es recomendable almacenar números en forma de cadenas de dígitos. Por ejemplo, puede querer colocar el punto decimal a un número; esto es mucho más fácil si convierte el número en una cadena y examina después cada carácter. También es recomendable practicar con la introducción de cadenas y convertirlas en valores numéricos. Esto evita el problema de que falle un programa que pide un valor numérico, si le introducimos inadvertidamente otro tipo de carácter. En el capítulo 3 veremos la forma de evitar estos problemas, cuando estudiemos la validación de los datos de entrada.Los números se convierten a cadenas y viceversa con STR$ y VAL.

STR$ aplicado a un valor numérico lo convierte en una cadena de caracteres, en la forma en que se imprime normalmente.

```
STR$(-12.01) nos dará "-12.01"
STR$(1000000000) nos dará "1E+09"
```

De forma similar, VAL aplicado a una cadena, devuelve el valor numérico de los dígitos de la cadena.

```
INPUT v$ : v = VAL(v$)
```

Si se introducen caracteres extra, éstos son ignorados y devuelve el número. Si el primer carácter no es reconocido como un número, devuelve el valor 0.

También se necesita algunas veces convertir cadenas de mayúsculas a minúsculas y viceversa; esto se puede hacer con las funciones UPPER$ y LOWER$. Cuando se usan, cualquier

caràcter no alfabètico permanece sin cambios.

UPPER$("amstrad") nos devolverà "AMSTRAD"
LOWER$("ORDENADOR") nos devolverà "ordenador"

Finalmente, otras dos funciones: SPACE$ y STRING$ que nos devuelven una cadena de espacios o caracteres de la longitud requerida.

SPACE$(5) nos devolverà 5 espacios
STRING$(10,"*") nos devolverà "**********"

BUSQUEDA DE PALABRAS

La funciòn INSTR nos permite buscar una palabra especifica dentro de una frase; el programa 3 ilustra esta funciòn. Nos permite introducir una palabra y una frase para buscar la primera dentro de la segunda; la sentencia es mostrada posteriormente con todas las ocurrencias de la palabra resaltadas.

PROGRAMA 3:BUSQUEDA DE PALABRAS

```
10 CLS : PEN 1
20 PRINT "BUSQUEDA DE PALABRAS"
30 PRINT : PRINT "Introduzca una palabra"
40 INPUT p$
50 PRINT : PRINT "Introduzca una frase"
60 INPUT f$
70 CLS
80 k=1 : v=1
90 WHILE v<>0
100 v = INSTR(k,f$,p$)
110 FOR j=k TO LEN(f$)
120 IF v=0 OR j<v THEN PEN 1 ELSE PEN 2
130 PRINT MID$(f$,j,1);
140 IF v<>0 AND j>v+LEN(p$)-2 THEN
k=v+LEN(p$):GOTO 160
150 NEXT j
160 WEND
170 PRINT : PRINT : INPUT "Pulse ENTER para
continuar";q$
180 IF q$ = "" THEN 10
190 END
```

En este programa se usan PEN 1 y PEN 2 para alterar el color con que se imprime el texro - veremos èsto con màs detalle cuando estudiemos el color en el capítulo 3.

El siguiente ejemplo nos muestra la tècnica de bùsqueda aplicada a tantas series de datos como se requiera, cada una

de ellas está contenida en una sentencia DATA. El programa 4 nos devuelve también una palabra que está relaccionada a la clave.

El programa ha sido diseñado para una firma hipotética que tiene gran cantidad de empleados, cada uno con un nivel de destreza y experiencia en diferentes lenguajes de máquina. El jefe de personal necesita una forma de registrar los conocimientos de cada empleado y un método de obtener los nombres de aquellos que están familiarizados con unos conocimientos específicos. Así que se necesita un clasificador de la base de datos.

Los nombres de los empleados se almacenan en sentencias DATA seguidas por sus conocimientos. Para distinguir entre el nombre y los conocimientos, el primero va precedido por el signo "#". El programa hace una búsqueda por cada grupo de datos; si encuentra una palabra precedida por el "#", entonces almacena temporalmente el nombre del empleado mientras examina sus conocimientos. Si encuentra los conocimientos requeridos, el nombre del empleado aparece en la pantalla. El programa continúa su búsqueda hasta que localiza el símbolo de terminación "%".

Esta es, sin duda, una de las formas más simples que existen de programas de bases de datos, pero puede ser bastante útil y está limitado solamente por el tamaño de la memoria del ordenador. Tiene la ventaja de que el jefe de personal puede añadir registros fácilmente a medida que se expanden los conocimientos de sus empleados, añadiendo las sentencias DATA apropiadas. Usted puede cambiar las sentencias DATA por otras que le sean más útiles, por ejemplo, registros de libros.

PROGRAMA 4: BASE DE DATOS

```
10 CLS
20 PRINT : PRINT "SOFTWARE CORTES Y CIA"
30 RESTORE : n$ = ""
40 PRINT : PRINT "Introduzca dato requerido"
50 INPUT s$ : s$ = UPPER(s$)
60 PRINT
70 READ w$
80 IF w$ = "%" THEN 120
90 IF LEFT$(w$,1) = "#" THEN n$ = MID$(w$,2)
100 IF w$ = s$ THEN PRINT n$,s$
110 GOTO 70
120 PRINT : INPUT "Pulse ENTER para continuar"
;q$
130 IF q$ = "" THEN 10
140 END
```

```
1000 'DATOS
1010 DATA #GARRIDO A, ADA, FORTRAN, VAX
1020 DATA #ANTON F, BASIC, COBOL, DEC, TAL
1030 DATA #DEL CAMPO A, CORAL 66, COBOL, ARGUS,
TANDEM
1040 DATA #MARTINEZ L, BASIC, COBOL, PDP, ARMY
SYSTEMS
1050 DATA #CUADRADO M, ALGOL, COBOL, BURROUGHS
1060 DATA #ESTEBAN J, CAD/CAM, UNIX, PROLOG,
MICROCOBOL
1070 DATA #JIMENEZ A, PASCAL, POLICE SYSTEMS
1080 DATA #LOPEZ J, BASIC, Z80, 8088, CP/M,
MICROS
1090 DATA %
```

El último ejemplo de este capítulo muestra cómo manipular textos. Se elije aleatoriamente una palabra de la lista contenida en las sentencias DATA y se mezclan las letras para producir un anagrama. Después se muestra este anagrama al jugador para que adivine la palabra original. La lista de palabras puede ser expandida según se quiera añadiendo sentencias DATA.

PROGRAMA 5: MANARAGA!

```
10 CLS
20 PRINT "ANAGRAMA"
30 PRINT : PRINT "Encuentre la palabra oculta"
40 a$ = "" : g$ = "" : RESTORE
50 n = 80 : 'numero de palabras
60 FOR j = 1 TO RND*n+1
70 READ w$
80 NEXT j
90 x = LEN(w$) : ww$ = w$
100 FOR j = 1 TO x
110 n = INT(1+RND*x)
120 x$ = MID$(w$,n,1) : IF x$ = "*" THEN 110
130 MID$(w$,n,1) = "*"
140 a$ = a$ + x$
150 NEXT j
160 IF a$ = ww$ THEN 60
170 PRINT : PRINT,a$
180 PRINT : INPUT "Introduzca su respuesta";g$
: g$ = UPPER(g$)
190 IF g$ = "" THEN 180
200 IF LEFT(g$,1) = "?" THEN PRINT : PRINT
"Respuesta",ww$ : GOTO 230
210 IF g$<>ww$ THEN PRINT "pruebe otra vez" :
GOTO 180
220 PRINT : PRINT "MUY BIEN"
```

```
230 PRINT : INPUT "Pulse ENTER para continuar"
; q$
240 IF q$ = "" THEN 10
250 END

1000 ' LISTA DE PALABRAS
1010 DATA HIDROGENO, MONOPOLIO, BUCLES, CHIMENEA,
PAQUETE, CORAL, CRONICO, DIETA, ELIPSE
1020 DATA FASCISMO, DINERO, GRUESO, MALARIA,
RACIONAL, RADIO, ESCUDO, LLORAR, CHAQUETA,
LIBELULA, PRUEBA
1030 DATA EGOISTA, SANTO, BAUTIZO, ABACO, NEON,
ESTERA, TRIPODE, ESPERA, CIEGO, NUCLEO
1040 DATA VERA, SOPLO, MEMORIA, SERVICIO, OMEGA,
SOCIABLE, NEGLIGENTE, AFICION, RAQUETA, HOLA
1050 DATA RASCAR, GARANTIA, COLUMNA, CABALLERO,
GRUESO, RECIBIR, TRACTOR, AGUA, NADAR, ZAPATO
1060 DATA REJILLA, ACIDO, POTENCIA, ARROJAR,
INTERNO, JOYA, GRUTA, CARNAVAL, GIGANTE, BIKINI
1070 DATA JUNGLA, ZORRO, CAQUI, ACUATICO, JUBILO,
ENIGMA, LLAVE, POLTRONA, ORACULO, PORTERO
1080 DATA ORTODOXO, INTERINO, ESCOCER, LACRE,
ACERO, FLEXIBLE, HORRIBLE, ORDENAR, JOTA, FLUIDO
1090 ......etc
```

Si se encuentra perdido al ejecutar el programa, introduzca "?" y el ordenador le darà la respuesta. Debo mencionar algo acerca de RND, es un generador de nùmeros aleatorios que devuelve un valor entre 0 y 1. En el capìtulo 4 veremos los paràmetros que se le pueden pasar al RND y còmo genera los nùmeros.

CAPITULO TRES

TECNICAS SIMPLES DE E/S

Probablemente no le sorprenderà al lector saber que un ordenador, incluido el Amstrad, no sirve para nada a menos que reciba informaciòn que èl pueda entender. La transferencia de esa informaciòn se hace mediante los dispositivos de entrada, como vimos en el capítulo uno. Tampoco sería muy útil la màquina si no pudieramos acceder a los resultados una vez procesados por el ordenador; esta informaciòn puede ser transferida a travès de los dispositivos de salida, mencionados tambièn en el primer capítulo. Los dispositivos que sirven para entrada y salida se llaman dispositivos de Entrada/Salida (E/S). En este capítulo hablaremos de las tècnicas de E/S del Amstrad y veremos los mètodos para mejorar la relacciòn entre el hombre y la màquina, llamados algunas veces "el interfase".

Es frecuente el caso del programador que invierte gran parte de sus esfuerzos en mejorar la eficiencia de un programa, cuando sería màs beneficioso que lo invirtiera en facilitar la entrada de datos al usuario, eliminando las posibilidades de introducir datos invàlidos, manteniendo al usuario lo màs ajeno posible al estado interno del programa y consiguiendo que la presentaciòn de los datos sea mejor. La informaciòn puede ser dirigida al usuario por tres mètodos distintos; Formas, Colores y Sonidos, cada uno de ellos es importante a su manera.

Las Formas pueden presentarse como símbolos o dibujos, que se pueden reconocer inmediatamente, o texto, que aunque tenga que ser descifrado no presenta mayores problemas. Aunque el texto es menos directo que los símbolos, puede decir muchas màs cosas de una forma eficiente. La animaciòn, en la que la pantalla es modificada ràpidamente para producir una imàgen, es una extensiòn de las figuras. Este tipo de imàgenes son captadas ràpidamente por el usuario; por ejemplo, un mensaje de error parpadeante es màs efectivo que uno estàtico. Las imàgenes dinàmicas proporcionan movimientos muy reales, como en los juegos de tipo "arcade".

El color es menos potente que las formas, pero se puede usar para ayudar a la identificaciòn de las formas dificiles de distinguir. Por ejemplo, una imàgen con la parte superior azul y la inferior verde nos aparece como cielo e hierba mucho mejor que una imàgen monocromàtica. El color se puede usar tambièn para resaltar ciertas àreas de la pantalla; por ejemplo, un mensaje de error mostrado en color rojo vivo

llama antes la atenciòn del usuario. Finalmente, una imàgen en color es màs agradable de mirar si se escojen cuidadosamente ls colores, que una de dos tonos.

Mientras que con imàgenes y colores el usuario debe estar màs atento a la pantalla para recibir la informaciòn, con el sonido puede llegarle aunque no estè prestando atenciòn. El sonido puede ser ùtil para atraer la atenciòn del usuario si, por ejemplo, se completa una operaciòn con errores. Estas notas deben ser altas y agradables para las operaciones correctas y bajas para los errores. El sonido puede añadir realidad adicional a las imàgenes. En un juego del tipo de invasores del espacio, los sonidos de explosiones y rayos dan realismo y señalan al jugador el resultado de su operaciòn.

Una de las principales causas de insatisfacciòn son las largas esperas que ocurren en un programa con cálculos numèricos largos y detallados. Para que un programa sea màs eficiente, los largos periodos de inactividad de la pantalla sin mostrar señales de vida, deben ser reducidas mediante el uso de mensajes como "Programa iniciado","Secciòn uno completada" etc. Esto no solo tranquiliza al usuario, sino que le mantiene informado de la posiciòn del programa. Al final de un proceso largo, es recomendable atraer la atenciòn del usuario con un sonido del tipo "beep".

El usuario puede introducir datos dentro del ordenador mediante el teclado o el controlador de juegos. El teclado es, con mucho, el sistema màs flexible y potente para introducir texto, valores numèricos o respuestas simples. El Amstrad tiene 75 teclas, pero cada una tiene significados alternativos cuando se pulsa simultàneamente con las teclas [SHIFT] o [CONTROL]. Desgraciadamente es muy comùn en el programador que usa el teclado, olvidar que los usuarios de sus programas pueden no considerarlo tan sencillo. En este capítulo veremos como preparar la introducciòn de datos mediante el teclado tan simple como sea posible. Cuando un usuario tiene que introducir datos, se le deben dar mensajes ràpidos y, cualquier dato invàlido, debe ser rechazado con un mensaje de error explicando el problema y dando la oportunidad de volver a introducir los datos. Ya que las teclas tienen etiquetas predefinidas, intente usar estas etiquetas siempre que le sea posible. Por ejemplo, La mejor forma de terminar un programa puede ser [CONTROL S] (Salir) o [CONTROL F] (Final) mientras que el uso de algo como [CONTROL 4] puede no ser significativo.

Como alternativa al teclado se puede usar un controlador de juegos, como un "joystick" o un botòn, aunque tiene grandes limitaciones, puede ser preferible para algunas aplicacio-

nes. Un "joystick" tiene un juego de cinco pulsadores, cada uno se llama cuando el mango hace un movimiento determinado. El botòn es similar al mando de volùmen de una radio e introduce dentro del ordenador un valor simple dentro de un rango limitado. Sin embargo, la ventaja real de los controladores de juego es que el interfaz con la màquina pasa a segundo tèrmino; unos pocos minutos moviendo el "joystick" a derecha e izquierda para mover algo en la pantalla en distintas direcciones y podremos comenzar a hacerlo sin tener que pensar demasiado.

Vamos a proceder a estudiar algunas tècnicas de E/S pero recuerde los puntos que hemos cubierto en esta introducciòn:

1)entrada fàcil y validada,
2)mantener al usuario totalmente atento, y
3)presentaciòn de los resultados

ENTRADA EN PANTALLA

El mètodo màs comùnmente usado para introducir datos es con la sentencia INPUT, que ya hemos usado varias veces. Hay un par de puntos que debemos resaltar. Es posible introducir varias variables con una sola sentencia INPUT; simplemente deben ponerse seguidas por comas, y cuando se estàn ejecutando, se deben introducir los datos separados tambièn por comas. Si el nùmero de datos introducidos no coincide con los requeridos, se hace una peticiòn para que se vuelvan a introducir los datos. Si està introduciendo una cadena que requiere una coma, serà interpretada como un separador de entrada entre datos distintos. Esto puede evitarse introduciendo la cadena flanqueada por comillas.

Como ya hemos visto, es posible incluir un mensaje dentro de la sentencia INPUT, que aparecerà antes de la peticiòn de datos. Si separamos el mensaje de las variables mediante un punto y coma, aparecerà una interrogaciòn, mientras que si los separamos con dos puntos, evitamos que aparezca. Si ponemos punto y coma entre la sentencia INPUT y el mensaje, suprimimos el caràcter de salto de lìnea al final de la entrada de datos de forma que la impresiòn continuarà al final del ùltimo dato introducido.

Se puede leer una lìnea entera, sin tener en cuenta la coma, mediante el comando LINE INPUT con una sola variable de cadena.

```
100 LINE INPUT "mensaje";a$
```

Màs adelante veremos el uso de estas dos sentencias con otros dispositivos, mediante el uso de un nùmero de cauce.

INPUT SIN PARADA

Una excelente facilidad del Amstrad es la posibilidad de detectar què tecla ha sido pulsada durante la ejecuciòn del programa. La funciòn INKEY$ se usa sin argumentos. Si pasa el control por INKEY$ cuando se està ejecutando el programa, INKEY$ devuelve el caràcter leido del teclado; se puede pulsar [SHIFT] o [CTRL] junto con cualquier otra tecla. Si no se pulsa ninguna tecla, nos devuelve una cadena vacìa; de este modo podemos crear una parada temporal hasta que se haya pulsado una tecla.

```
100 PRINT "Pulse una tecla para continuar"
110 IF INKEY$ = "" THEN 110
120 'continuaciòn del programa
```

La introducciòn de informaciòn mediante la funciòn INKEY$ tiene la ventaja sobre la INPUT que no tiene que ser seguida por [ENTER], pero recuerde que, a diferencia de INPUT, no esperarà a que pulse una tecla. Para introducir informaciòn usando INKEY$, se prepara un bucle de forma que si no se pulsa una tecla, reconoce la cadena vacìa y no sigue adelante. Una cosa importante a tener en cuenta es que INKEY$ puede cambiar entre lìneas, por lo que es recomendable que tan pronto como se lea el INKEY$, se asigne a una variable y, en adelante, se use la variable en su lugar.

```
100 q$ = INKEY$
110 IF q$ = "" THEN 100
120 'continuaciòn del programa
```

El siguiente ejemplo usa la concatenaciòn con la funciòn INKEY$ para forzar al usuario a introducir una cadena de datos en formato correcto y es una buena alternativa a la sentencia INPUT. La cadena completa puede ser comprobada para ver su validez.

```
100 d$ = ""
110 PRINT "Enter : DD/MM/AA ";
120 FOR j = 1 TO 8
130 IF j = 3 OR j = 6 THEN c$ ="/" : GOTO 160
140 c$ = INKEY$ : IF c$ = "" THEN 140
160 PRINT c$;
170 d$ = d$ + c$
180 NEXT j
190 PRINT
200 'validaciòn y continuaciòn
```

INKEY es una pequeña extensiòn de INKEY$, y analiza el teclado y devuelve un valor que indica el estado de una tecla especìfica, analizando si està pulsada y si lo està,

comprueba si està pulsada tambièn la tecla [SHIFT] o la [CTRL]. Los valores devueltos aparecen listados en la figura 3.1. El argumento de INKEY es un nûmero entero que identifica a una sola tecla; en el apèndice B se proporciona una lista completa de los còdigos.

Esta funciòn es ûtil para detectar respuestas con S/N (si o no) cuando el estado de la tecla SHIFT no es importante. En la figura 3.1 se puede ver que el valor retornado no es -1 si la tecla està pulsada.

```
100 PRINT "Quieres continuar (S/N)?"
110 IF INKEY (60)<>-1 THEN GOTO 140
120 IF INKEY (46)<>-1 THEN STOP
130 GOTO 110
140 'continuaciòn
```

TECLAS			*VALOR RETORNADO*
TECLA	*SHIFT*	*CTRL*	
ARRIBA	?	?	-1
ABAJO	ARRIBA	ARRIBA	0
ABAJO	ABAJO	ARRIBA	32
ABAJO	ARRIBA	ABAJO	128
ABAJO	ABAJO	ABAJO	160

Figura 3.1 VALORES DE INKEY

REDEFINICION DE TECLAS

Algunas veces es necesario introducir frecuentemente una cadena completa. En el Amstrad tenemos la posibilidad de programar las teclas de modo que cuando presionamos una, nos devuelva una secuencia de caracteres; este tipo de teclas se llaman teclas de funciòn. Hay 32 teclas, los valores ASCII 128 a 159, que pueden ser expandidos a un màximo de 32 caracteres, sin embargo, el nûmero total de caracteres expandidos no puede pasar de 120.

Supongamos, por ejemplo, que queremos programar una tecla de forma que cuando la pulsemos nos devuelva

RUN "CUENTAS" seguido del caràcter ['ENTER'].

Primero debemos decidir la tecla que queremos usar. Por el momento estamos restringidos a las teclas que tienen valor ASCII entre 128 y 159. Y como las teclas del teclado numèrico nos devuelven valores entre 128 y 140, y [CTRL]

pulsado simultaneamente con [ENTER] en el teclado numèrico nos devuelve 140; podemos programar esta tecla con el comando que damos a continuaciòn

```
KEY 140,"RUN" + CHR$(34) + "CUENTAS" + CHR$(34)
+ CHR$(13)
```

Intèntelo ahora.

Usando KEY DEF podemos modificar los valores ASCII que devuelven ciertas teclas; el uso màs comùn es modificar aquellas teclas que devuelven valores ASCII en un rango que permita programarlas como teclas de funciòn. Otro uso puede ser el inutilizar el [ESC](tecla Break). Debe codificar los siguientes paràmetros en el orden dado:

nùmero de tecla	,vea el apèndice B
repeticiòn	,1 permite repeticiòn, 0 la impide
normal	,caràcter ASCII cuando se pulsa sola
con shift	,caràcter ASCII cuando se pulsa con [SHIFT] (no con [CTRL])
control	,caràcter ASCII si se pulsa con [CTRL]

Cuando no se especifica un nuevo valor, permanece el valor anterior. Por ejemplo, para programar la tecla [CTRL E] para que devuelva EDIT:

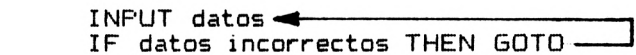
```
KEY DEF 58, 0, 101, 69, 128
KEY 128 , "EDIT"
```

Acabamos de mencionar la posibilidad de poner o quitar la repeticiòn, hay que decir que el periodo de repeticiòn puede ser alterado usando SPEED KEY para especificar la demora de arranque, el periodo de repeticiòn se da en unidades de 0.02 segundos.

```
SPEED KEY 10, 10
```

VALIDEZ DE DATOS

Un mètodo para prevenir que fallen los programas es comprobar los datos que se estàn introduciendo. Si los datos no son correctos, el control debe volver hacia atràs a la misma sentencia INPUT. El programa solo deberà continuar si los datos son introducido de forma correcta. Es de gran ayuda sacar un mensaje que indique al usuario el problema que se ha encontrado en los datos introducidos.



A diferencia de otros micros, el Amstrad comprueba si los datos introducidos son del tipo correcto, de acuerdo a las variables especificadas en la sentencia INPUT y, en vez de fallar, pide que se le introduzcan los datos de nuevo.

Ya hemos visto, al estudiar el INKEY$, que podemos forzar al usuario a introducir los datos en el formato correcto pero èsto no nos garantiza que los valores introducidos sean vàlidos, ej. un mes 13. Sin embargo no es muy dificil comprobar si el mes està en el rango de 1 a 12 y referirse despuès a una lista de los dias de cada mes para ver si la fecha es vàlida - por supuesto, debe tener en cuenta los años bisiestos. Los valores numèricos se extraen de la cadena de la fecha usando las funciones VAL y MID$ que vimos en el capítulo 2.

Otros mètpdos de comprobar la validez de los datos son, inspeccionar la longitud de las cadenas usando LEN, comprobar si los valores estàn dentro de rango con los operadores '>' y '<', y comprobar si estàn presentes ciertos caracteres inspeccionando los còdigos ASCI.

Otro problema que puede ocurrir es que el usuario introduzca valores vàlidos pero incorrectos. Si èsto crea resultados difíciles de rectificar, es interesante mostrarlos de nuevo y pedir al usuario que los confirme pulsando S o N antes de que se tomen otras acciones.

La falta de cuidado en comprobar los datos de entrada puede llevar a pèrdidas de tiempo innecesarias y causar frustración al usuario, por lo que debe intentar hacer siempre sus programas lo màs robustos posibles.

SALIDA A PANTALLA

Ya se habrà dado cuenta que para sacar datos a pantalla se usa el comando PRINT, màs adelante veremos còmo especificando un nùmero de cauce, se pueden sacar datos a otros dispositivos. La sentencia va seguida por los datos, que pueden ser variables, expresiones aritmèticas, cadenas, funciones de control, caracteres de control y màscaras de formatø. Es posible incluir varios grupos de datos en una sola sentencia PRINT mediante el uso de la coma o el punto y coma.

Punto y coma;
Un punto y coma separando datos de impresión causa que el

segundo dato aparezca inmediatamente detras del primero.

Coma,
La pantalla de televisiòn puede considerarse como dividida en una secuencia de zonas de impresiòn de 13 caracteres cada una. Una coma separando dos datos a imprimir causa que el segundo dato aparezca en la primera posiciòn de la siguiente zona.

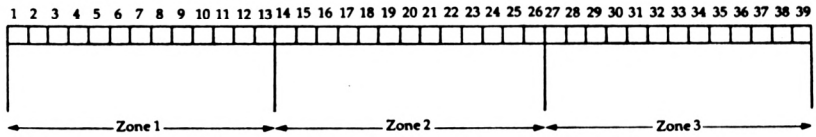


Figura 3.2

Es posible alterar la zona de 13 para acoplarla a sus propias necesidades mediante el comando ZONE

```
ZONE 10
```

PRINT USING

El Amstrad posee el comando PRINT USING, que permite imprimir los resultados en el formato requerido. El USING incluye una especie de plantilla formada por los caracteres "# + - . * $ ↑ , ! \ &". Cada caràcter de la plantilla es inspeccionado uno por uno y controlando la imàgen a imprimir. Tambièn se pueden imprimir caracteres de control que no estèn incluidos entre los anteriores u otros caracteres de control fuera del contexto o precedidos por el caràcter "_". En la figura 3.3 de la siguiente pàgina damos una lista completa de las definiciones - los valores exponenciales se explican en el capítulo 4.

El rebasamiento de un campo se indica por el símbolo '%'.

```
PRINT USING "###";a
PRINT USING "COSTO+$##,##.##";c
PRINT USING "\   \";a$
```

Fig 3.3:Formato PRINT USING

ESPECIFICADOR		*NUMERO DIGITOS POSIBLES*	*NUMERO DE CARACT.*	*DEFINICION*
NUMERICO (màx. 20 caracte-res ex-cluidos signos y dìgitos exponen-ciales)	#	1	1	Posiciòn del dìgito
	.	0	1	Punto decimal
	,	1	1	Posiciòn del dìgito, pone una coma cada 3 dìgitos a la izquierda del punto.
	$$	1	2	Signo dòlar flotante, pre cede los dìgitos no ocupa dos
	**	2	2	Rellena con asteriscos
	**$	2	$	Dòlar flotante y asteris-cos
	+	0	1	Pone '+' o '-'. Puede apa recer al principio o al final de la plantilla
	-	0	1	Pone '-' si es negativo, solo puede aparecer al principio de la plantilla
	↑↑↑↑	0	4	Formato exponencial
CADENAS	!			Solo primer caràcter
	\<n espacios>\			Primeros n caracteres
	&			Cadena completa

Figura 3.3 especificadores de formato de PRINT USING

CARACTERES DE CONTROL

En modo comando es posible mover el cursor sobre la pantala usando las teclas de control del cursor. El BASIC del Amstrad le permite controlar este tipo de funciones dentro de las sentencias PRINT, permitiendo controlar el cursor en modo programa. Cuando se programa dentro de una cadena, cada caràcter de control se representa por un simbolo gràfico que aparece en el listado del programa - otra alternativa es acceder al caràcter por medio de la funciòn CHR$.

← Mueve el cursor un espacio a la izquierda CHR$(8)[CTRL H]
→ Mueve el cursor un espacio a la derecha CHR$(9) [CTRL I]
↓ Mueve el cursor un espacio hacia abajo CHR$(10) [CTRL J]
↑ Mueve el cursor un espacio hacia arriba CHR$(11) [CTRL K]

El siguiente ejemplo produce cinco 'V' en formaciòn

```
10 CLS
20 PRINT "→↓V→↓V→↓V→↑V→↑V↓↓↓↓↓"
```

Si experimenta descubrirá que hay otras teclas que pueden ser programadas en sentencias PRINT. Tres muy útiles son:

CHR$ (7) [CONTROL G] - sonido del 'beep'
CHR$ (24) [CONTROL X] - intercambia tinta de PEN y PAPER
CHR$ (30) [CONTROL ↑] - reposiciona el cursor en la esquina superior izquierda de la pantalla

En algunos casos puede ser necesario imprimir un símbolo asociado con uno de estos caracteres de control en lugar de usarlo en su acción de control. Esto se puede hacer imprimiendo primero CHR$ (1)

```
PRINT CHR$(1)+CHR$(8)
```

EFECTOS DE ANIMACION

Con la ayuda de los caracteres de control y usando los bucles FOR...NEXT podemos mostrar una secuencia de caracteres que produzcan una animación simple. La idea es construir una cadena que incluya los caracteres que se van a mostrar junto con algunos caracteres de control del cursor para mover los caracteres desde sus posiciones originales y algunos blancos para borrar los anteriores. Controlando la ejecución del programa con un bucle FOR...NEXT, podemos imprimir la cadena varias veces en sucesión para producir algunos efectos interesantes.

```
10 'movimiento horizontal
20 CLS
30 PRINT : PRINT : PRINT
40 FOR j = 1 TO 35
50 PRINT CHR$(32) + CHR$(62) + CHR$(8);
60 FOR k = 1 TO 20 : NEXT k
70 NEXT j
80 FOR j = 1 TO 35
90 PRINT CHR$(32) + CHR$(8) + CHR$(8) + CHR$(60)
+ CHR$(8);
100 FOR K = 1 TO 20 : NEXT k
110 NEXT j
120 GOTO 40
```

Podemos usar una técnica similar para conseguir un movimiento vertical. El siguiente ejemplo combina ambos movimientos para producir un movimiento diagonal.

```
10 'movimiento diagonal
20 CLS
30 FOR j = 1 TO 23
```

```
40 PRINT CHR$(32) + CHR$(10) + CHR$(214) +
CHR$(11) + CHR$(11) + CHR$(32) + CHR$(10);
50 PRINT CHR$(214) + CHR$(8) + CHR$(8) + CHR$(8)
+ CHR$(10);
60 NEXT j
70 GOTO 20
```

CONTROL DE IMPRESION

El modo estàndar de imprimir informaciòn en una pantalla es a lo largo de una lînea o hacia abajo en una columna. Esto puede ser aceptable para pequeñas cantidades de datos, pero cuando la impresiòn pasa de la fila 25, la pantalla se mueve hacia arriba. Desgraciadamente èsto significa que la lînea superior se desvanece y no da tiempo a leer los resultados.

Una forma de evitar esto es contar las lîneas que se escriben y parar la salida cuando se llene la pantalla. Se puede continuar, pulsando cualquier tecla. Ademàs los tîtulos deben aparecer en lo alto de cada pàgina.

Un segundo problema que suele ocurrir es que una variable numèrica aumenta (comparada con el valor anterior) y cambia el nûmero de dîgitos haciendo que se desplacen todos los datos que salen a continuaciòn. Esto puede restar legibilidad a los resultados que aparecen en la pantalla, especialmente en una tabla de resultados.

Con el BASIC del Amstrad podemos eliminar estos problemas, ya que disponemos de las funciones TAB y SPC que se usan en conjunciòn con el comando PRINT para dirigir la salida a una determinada columna de la pantalla.

SPC se usa para sacar un nûmero determinado de espacios a la pantalla o, como veremos màs adelante, a otro dispositivo perifèrico, mientras que la funciòn TAB se usa para mover el cursor un nûmero fijo de columnas relativas al principio de de la lînea actual de impresiòn, empezando en una nueva lînea si ya se ha pasado la columna. Un punto y coma termina ambas funciones.

Podemos ver un ejemplo de estas dos funciones en el programa 6, CALENDARIO.

Este programa nos muestra un calendario de un mes o año en particular. El calendario original fuè creado por Julio Cesar, pero fijò el año con once minutos màs de lo que debîa. Esto se corrigiò con la introducciòn del calendario Gregoriano, en Italia en 1582, aunque en Inglaterra no se introdujo hasta 1752. El programa no da respuestas correctas antes de esta fecha.

El programa usa una fórmula en la subrutina 1000 que calcula el dia de la semana en que comienza un mes determinado. Para usar el programa, introduzca el mes como un número del 1 al 12, seguido del año con los cuatro dígitos.

PROGRAMA 6;CALENDARIO

```
10 CLS
20 PRINT "CALENDARIO"
30 PRINT "INTRODUZCA MES MM/AAAA: ";
40 d$ = ""
50 FOR j = 1 TO 7
60 IF j = 3 THEN c$ = "/" : GOTO 80
70 c$ = INKEY$ : IF c$ = "" THEN 70
80 PRINT c$;
90 d$ = d$ + c$
100 NEXT j
110 m = VAL(LEFT$(d$,2)) : y = VAL(RIGHT$(d$,4))
120 IF m<1 OR m>12 THEN 10
130 CLS
140 PRINT " CALENDARIO "
150 PRINT : PRINT "MES ";m;SPC(5);" Año ";y
160 PRINT : PRINT "DOM LUN MAR MIE JUE VIE SAB"
170 GOSUB 1000
180 d = dd: m = m+1
190 IF m>12 THEN m = 1 : y = y+1
200 GOSUB 1000
210 IF dd<d THEN dd =dd+7
220 n = 28+dd-d
230 p = 4*d+1
240 FOR x = 1 TO n
250 PRINT TAB(p);x;
260 p = p+4 : IF p>=26 THEN p = 1
270 NEXT x
280 PRINT : PRINT : INPUT "Pulse ENTER para
continuar",q$
290 IF q$ = "" THEN 10
300 END
100 mm = m-2 : yy = y
1010 IF mm>0 THEN 1030
1020 mm = mm+12 : yy = yy-1
1030 c = INT(y/100) : yy = yy-100*c
1040 dd = 1 + INT(2.6*mm-0.19)+yy+INT(yy/
4)+INT(c/4)-2*c
1050 dd = dd MOD 7
1060 RETURN
```

El BASIC del Amstrad tiene otra forma más de modificar la posición actual de impresión; usando el comando LOCATE con las nuevas coordenadas x e y.

LOCATE 10,1

Nos encontraremos este comando en numerosas ocasiones a lo largo de este libro.

MODOS DE PANTALLA

Hasta ahora hemos trabajado con una pantalla de 25 líneas de 40 caracteres. Cada carácter está compuesto por una matriz de 8x8 puntos o "pixels" que pueden estar iluminados u oscuros. Esto nos puede describir una pantalla compuesta de 320x200 puntos. Más adelante veremos cómo se pueden producir imágenes de alta resolución, mediante el control individual de estos puntos. Cuando la pantalla se encuentra en este modo, se dice que está en MODE 1. Hay otros dos modos llamados MODE 0 y MODE 2 que toman los atributos de la figura 3.4.

	MODE 0	MODE 1	MODE 2
CARACTERES	25 líneas de 20 caracteres	25 líneas de 40 caracteres	25 líneas de 80 caracteres
PUNTOS	200 alto por 160 ancho	200 alto por 320 ancho	200 alto por 640 ancho

Figura 3.4:Modos de pantalla

Cada uno de los modos de pantalla se pueden seleccionar con el comando MODE.

COLOR

Por fin hemos llegado al capítulo del color, un tema cuya importancia se ha mencionado en la introducción de este capítulo. El Amstrad es capaz de mostrar información en 27 colores aunque el número de los que se pueden mostrar simultáneamente está restringido, de acuerdo con el modo de pantalla, a los valores que aparecen en la figura 3.5.

	MODE 0	MODE 1	MODE 2
Máximo número de colores	16	4	2

Figura 3.5:Máximo número de colores

Cada uno de los 27 colores disponibles está identificado por un código de color único que aparece listado en el apéndice C. Cada posición de carácter tiene dos colores

asociados. El color del fondo se elige con el comando PAPER y el color del carácter con el comando PEN. El número asociado a ambos comandos no es el código del color, sino un número de tinta que tiene un código de color asociado, que a su vez es definido con el comand INK.

PEN 1 pone el primer plano de escritura del color asignado a INK 1
PAPER 2 pone el fondo de los caracteres a imprimir del color asignado a INK 2

mientras que

INK 1,0 asigna el color 0 a INK número 1

El color del borde se controla independientemente con el comando BORDER y el número especifica el código de color.

Introduzca el siguiente ejemplo para ver pasar los colores cuando pulse una tecla cualquiera (excepto [ESC])

```
10 PEN 2 : PAPER 1
20 FOR j = 0 TO 26
30 CLS
40 BORDER j
50 INK 1,j : INK 2,(j+12)MOD 26
60 LOCATE 15,10 : PRINT "COLOR";j
70 IF INKEY$ = "" THEN 70
80 NEXT j
```

SELECION POR MENU

Es una práctica muy común en los programadores, empezar con una idea inicial y continuar añadiendo refinamientos extras hasta que la lógica del programa se asemeja a un plato de spagetis, con una estructura con numerosos saltos mandando el control en todas direcciones; es mejor usar la programación estructurada. Esta es una metodología en la que el programa se rompe en bloques o módulos, cada uno de los cuales tiene un propósito específico. La programación estructurada intenta evitar la sentencia GOTO en un esfuerzo por hacer comprensible la lógica del programa. El romper el programa de esta forma permite probar por separado las rutinas para poder localizar fácilmente los fallos.

Un método ideal para producir un programa que pueda tomar varias funciones es escribir cada una en un bloque separado y permitir al usuario seleccionar la sección mediante un "menú" con las opciones disponibles.

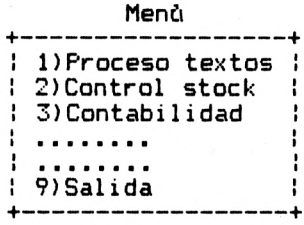
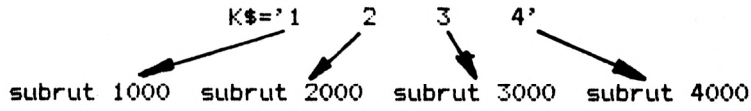


Figura 3.6:Selecìon por menù

Por supuesto, no hay ninguna razòn que impida que una vez hecha una selecciòn aparezca otro menù mostrandonos una lista de opciones relacionadas con la elegida anteriormente de forma que el usuario pueda realizar su seleciòn o retornar al menù principal.

Hay numerosas formas en las que se puede escribir un menù. La màs comùn es el mètodo 'Chino' donde el usuario seleciona una letra o un nùmero de la lista. Veamos el siguiente ejemplo; en este caso el usuario tiene cuatro opciones para elegir pulsandò 1, 2, 3 o 4. Cuando se seleciona una opciòn, el programa comprueba que la opciòn seleccionada aparece en la cadena K$ y despuès, si està presente, ejecuta la opciòn requerida como una subrutina, o si no està, vuelve a mostrar el menù principal. Construyendo el programa de esta forma, es fàcil añadir funciones extras al programa con un mìnimo de cambios en el còdigo existente.



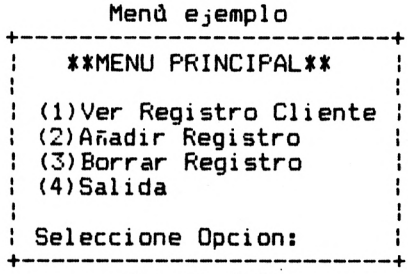


Figura 3.7

Una pequeña variaciòn de este mètodo es mover el cursor de arriba hacia abajo usando las teclas de control del cursor hasta que estè en la opciòn requerida. Despuès, pulsando la

tecla adecuada, se ejecuta la acción elegida. Esto se hace manteniendo un contador que se incrementa o decrementa a medida que se mueve el cursor; la selección del contador corresponde a la opción requerida.

Programa 7: SELECCION POR CURSOR

```
10 CLS
20 LOCATE 9,6 : PRINT "*** MENU PRINCIPAL ***"
30 RESTORE : READ n
40 FOR j = 1 TO n
50 READ a$
60 LOCATE 10,7+j : print j;a$
70 NEXT J
80 p = 1
90 IF p>n THEN p = 1
100 IF p<1 THEN p = n
110 pp = p
120 LOCATE 9,7+p : PRINT ">"
130 FOR j = 1 TO 200 : NEXT j
140 IF INKEY(18) <>-1 THEN 190
150 IF INKEY(0) <>-1 THEN p = p-1
160 IF INKEY(2) <>-1 THEN p = p+1
170 IF p<>pp THEN LOCATE 9,7+pp : PRINT " " :
GOTO 90
180 GOTO 140
190 ON p GOSUB 1000,2000,3000,4000,5000
200 GOTO 10
210 DATA 5
220 DATA "Salida"
230 DATA "Ver Registro"
240 DATA "Borrar Registro"
250 DATA "Añadir Registro"
260 DATA "Ayuda"
1000 END
2000 RETURN
3000 RETURN
4000 RETURN
5000 RETURN
```

El programa ha sido escrito para que le resulte fàcil de acoplar a sus propios requerimientos. Para hacerlo, àñada sus propias opciones a las sentencias DATA, la línea 210 debe tener el número de opciones, y después añada sus propias subrutinas, (asegúrese de que las llama en la línea 190). Las teclas de control del cursor, arriba y abajo, moverán el cursor, y la opción será seleccionada pulsando la tecla [ENTER].

PETICIONES POR PANTALLA

Es muy normal encontrar programas en los que la informaciòn se pide línea por línea. Por ejemplo:

```
Introduzca Dispositivo de Entrada ? $TERM
Introduzca Dispositivo de Salida ? $LP
Introduzca Nombre del Fichero ? PAGOS
Introduzca Modo de Acceso ? SOLO LECTURA
Introduzca Modo de Exclusiòn ? COMPARTIDO
```

La introducciòn de todos estos datos podría mejorarse si se pudieran mostrar todos los mensajes simultàneamente y el usuario pudiera corregir previamente los datos introducidos.

Vamos a ver un ejemplo de petición por pantalla que maneja la entrada de un nombre, una dirección y un número de telèfono, para su uso en una aplicaciòn como una lista de correo. Ha sido escrito de forma que pueda ser fàcilmente ampliado para manejar sus propios requerimientos. La lista de preguntas junto con la longitud màxima de los campos de entrada se epecifican en sentencias DATA, al final del programa.

Los datos de entrada se introducen en la posición del cursor. Este se puede mover por la pantalla usando las teclas de control del cursor. Cuando se han introducido todos los datos, se pueden aceptar pulsando la tecla [COPY].

Se usan dos contadores, LP y CP. LP se modifica con las teclas de cursor arriba y abajo, CP se modifica con las de izquierda y derecha; el cursor apunta al caràcter en la línea que se està introduciendo. Para registrar los datos de entrada se usa la matriz F$. A medida que se pulsan teclas alfabèticas y numèricas, èstas aparecen en la pantalla, añadiendolas a la matriz F$ en la posición apropiada.

```
Nombre          Juan Hernandez
Domicilio       Gran Via, 32
Ciudad          Galapagar
Provincia       Madrid
Còdigo postal   28078
Telèfono        6304152
```

Figura 3.8:Ejemplo de Pantalla

Programa 8:PETICION POR PANTALLA

```
10 PAPER 0 : CLS
20 READ a$
```

```
30 LOCATE 14,4 : PRINT a$
40 READ n
50 DIM f$(n),x(n)
60 cp = 1 : lp = 1 : v = 0
70 FOR j = 1 TO n
80 READ a$,x(j)
90 LOCATE 2,5+j : PRINT a$
100 f$(j) = SPACES(x(j))
110 NEXT j
120 IF lp>n THEN cp = 1 : lp = 1
130 IF cp<1 THEN cp = 1
140 IF cp>x(lp) THEN cp = 1 : IF v<>2 THEN lp =
lp+1
150 IF lp>n THEN cp = 1 : lp = 1
160 IF lp<1 THEN cp = 1 : lp = n
170 FOR j = 1 TO n
180 PAPER 3
190 LOCATE 14,5+j : PRINT f$(j)
200 NEXT j
210 PAPER 0
220 LOCATE cp+13,lp+5 : PRINT CHR$(143)
230 v = 0
240 c$ = INKEY$
250 IF c$ = "" THEN 240
260 c = ASC(c$)
270 IF c = 224 THEN 400
280 IF c = 13 THEN cp = 1 : lp = lp+1 : v = 2
290 IF c = 244 OR c = 240 THEN lp = lp-1 : v = 2
300 IF c = 241 OR c = 245 THEN lp = lp+1 : v = 2
310 IF c = 242 OR c = 246 THEN cp = cp-1 : v = 1
320 IF c = 247 OR c = 243 THEN cp = cp+1 : v = 1
330 IF v = 0 THEN MID$(f$(lp),cp,1) = c$ : cp =
cp+1
340 GOTO 120
400 CLS
410 FOR j = 1 TO n
420 PRINT f$(j)
430 NEXT j
440 END
1000 DATA "AGENDA DE DIRECCIONES"
1010 DATA 6
1020 DATA "Nombre",20
1030 DATA "Dirección",25
1040 DATA "Ciudad",15
1050 DATA "ProVincia",12
1060 DATA "Código Postal",5
1070 DATA "Teléfono",11
```

COLORES PARPADEANTES

Para hacer que el texto, el fondo o el borde parapdeen,

podemos especificarle un color secundario en las sentencias INK o BORDER.

INK 1,10,20 cualquier color asociado con INK 1 alternarà entre los colores 10 y 20
BORDER 10,20 de forma similar, el borde alternarà entre los colores 10 y 20.

La frecuencia de la alternancia de colores puede ser especificada con el comando SPEED INK, el primer valor especifica el periodo del color primario y el segundo valor especifica el periodo del color secundario(los periodos se dan en unidades de 0.02 segundos).

CAUCES

Cuando 'hablamos' con los dispositivos de E/S lo hacemos por medio de los cauces. El concepto de cauce es muy comùn en los ordenadores y no es difícil de comprender. En el Amstrad hay diez cauces de entrada y otros tantos de salida y cada uno està asociado a uno de los dispositivos de E/S que aparecen en la figura 3.9. Para comunicar con un dispositivo determinado, debemos especificar el nùmero de cauce despuès del comando de E/S. Hasta ahora hemos omitido el nùmero de cauce en nuestros programas ya que el sistema operativo del Amstrad toma como defecto el cauce 0.

Nùmero de Cauce		Dispositivo
0		0
1		1
2		2
3		3
4	ventana	4
5		5
6		6
7		7
8		Impresora
9		Casette

Figura 3.9

Por ejemplo, para sacar un listado a la impresora, debemos usar LIST,#8. Hablaremos de las E/S del casette en el capìtulo 8. Lo màs interesante de la figura 3.9 es que hay ocho cauces dedicados a pantallas y ya veremos còmo nos permite dividir la pantalla en àreas separadas, cada una de las cuales tiene diferentes atributos o se usan para propòsitos diferentes. Este tipo de àreas de la pantalla se llaman ventanas.

VENTANAS

Para preparar una ventana en la pantalla, debemos definir su posición usando el comando WINDOW que especifica el cauce seguido de las coordenadas de la ventana. Por ejemplo, considere la ventana de la figura 3.10.

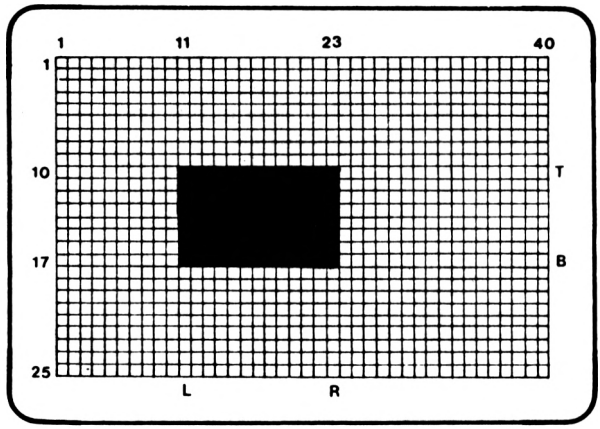


Figura 3.10

Para definir esta ventana al cauce 1, usaremos:

```
WINDOW #1, 11, 23, 10, 17
```

Y a partir de ahora, todos los comandos de E/S que especifiquen el cauce 1, irán a parar a esta ventana, Cuando se use el interior de la ventana se debe tener en cuenta que las coordenadas empiezan con 1,1 en la esquina superior izquierda de la ventana.

Veamos el siguiente ejemplo que puede modificarse fácilmente para incluirlo en cualquier programa. La idea es que si ocurre un error, la subrutina de la línea 1000 es llamada y muestra un mensaje de error parpadeante en la línea inferior. Da tambien un pequeño 'beep' para que el usuario lo reconozca pulsando una tecla. Observe que, como la ventana 0 está sobre el mensaje de error, limpiando esta ventana limpiamos automáticamente el mensaje de error. En el capítulo 6 veremos como poner trampas de error que llamen a la subrutina correspondiente.

```
10 e$ = "mensaje del usuario"
20 WINDOW #1,1,40,25,25
30 INK 3,3,26
```

```
40 PAPER #1,3 : PEN #1,0
50 CLS
60 PRINT "Pulse 'e'"
70 IF INKEY$ <>"e" then 70
80 GOSUB 1000
90 GOTO 50
1000 LOCATE #1,1,1
1010 PRINT #1,"Error : ";e$ : PRINT CHR$(7)
1020 IF INKEY$ = "" THEN 1020
1030 RETURN
```

Con èsto concluimos el capítulo . Las funciones que hemos visto hasta ahora, como color, modos de pantalla, control de impresiòn, cauces, ventanas, etc, las volveremos a encontrar de nuevo a medida que nos introduzcamos en E/S màs avanzados, incluyendo gràficos y sintetizador de sonido.

CAPITULO CUATRO

ORDENADORES, NUMEROS Y MATEMATICAS

La mayoría de las personas piensa que un ordenador puede manejar expresiones aritméticas muy complejas, es un error pensar que ésta tarea le resulta sencilla ya que en realidad es una de las más duras. Hay dos razones principales para afirmar esto; primero que se espera que que el ordenador sea capaz de tratar un amplio rango de números y segundo que las unidades que componen la memoria del ordenador están limitadas a números enteros dentro de un rango específico; el Amstrad puede tratar del 0 al 255. Sin embargo, si usamos un lenguaje de alto nivel no hace falta conocer los métodos complicados que se usan para evaluar expresiones y funciones aritméticas y matemáticas, ya que lo hace automáticamente el sistema operativo. Cuando se programa en código máquina es posible acceder a estas rutinas del sistema operativo mediante llamadas a las subrutinas que las componen, pero hay que tener mucho más cuidado al pasar operandos y operadores de forma que las rutinas puedan comprenderlos, comprobando que no existan rebasamientos, acediendo a los resultados desde posiciones correctas de memoria e interpretando los resultados mediante la conversión de la forma en que son devueltos a la requerida.

Las matemáticas son un campo muy amplio y su uso es muy común en ordenadores. El control financiero por ordenador, navegación, defensa, diseño y manufactura asistidos por ordenador, automatización de oficinas, control de existencias, juegos, etc, requieren multitud de evaluaciones numéricas. Debido a la compleja tarea que puede representar el seguimiento y vuelo de un aeroplano, que involucra factores como velocidad, condiciones meteorologicas, la forma esférica de la Tierra, o mantener simplemente la puntuación en un juego. Hay varios niveles de matemáticas y no es necesario preocuparse si no se comprenden algunas de las funciones complejas que es capaz de desarrollar el Amstrad. Las matemáticas del tipo de suma y multiplicación no suelen despertar gran interés, pero no se alarme si en la última sección de este capítulo tratamos algunas específicas para ingenieros, científicos o técnicos. Aunque el lector tenga dificultad, es recomendable intentar realizar los ejemplos que ilustran lo que el Amstrad es capaz de hacer. Las matemáticas pueden ser divertidas y con la ayuda de un ordenador pueden hacerse aún más entretenidas. Los lectores que estén interesados en estos temas pueden leer el libro:

"MATHS + COMPUTERS = FUN" por G T Childs
(publicado por Sigma Technical Press)

Este libro es de utilidad para los niños y para muchos adultos, ya que demuestra que las matemàticas no son nada difícil. Trata muchos temas desde matemàticas elementales hasta las avanzadas y contiene numerosas ayudas, divertidos rompecabezas y unos 50 programas en BASIC.

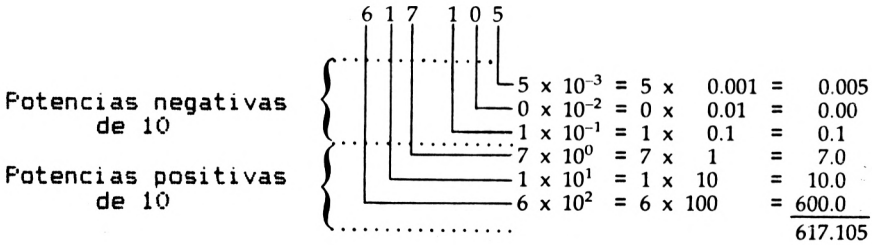
LOS NUMEROS EN EL AMSTRAD

Aquellos lectores que tienen poca experiencia en matemàticas, probablemente nunca se han parado a pensar la cantidad de formas en que se pueden representar los nùmeros. Por ejemplo, los tres tipos de nùmeros que siempre nos han enseñado son los enteros, fracciones y decimales. De modo similar, hay tres notaciones particulares que debemos tener en cuenta y todas ellas se usan comùnmente en los ordenadores. Las dos primeras le serán familiares aunque se hayan cambiado los nombres.

ENTERO — Los enteros son aquellos nùmeros que estàn formados por una secuencia de dígitos y que no tienen punto decimal ni componente factorial. Pueden ser positivos o negativos. ej. -156, -22, -1, 0, 3, 69.

REAL — Los nùmeros reales son una secuencia de dígitos con punto decimal al principio, al final o entre los dígitos. Los que se encuentran a la izquierda del punto son potencias positivas de 10 y forman el componente entero del nùmero. Los de la derecha del punto son potencias negativas de 10 y forman el componente fraccionario. Los nùmeros reales pueden ser positivos o negativos. ej. -2.3, 0.4, 31.0234, .125, -0.275.

Para ilustrar esta idea:



EXPONENCIAL — Los nùmeros exponenciales son los escritos en una notaciòn capaz de representar tanto los nùmeros muy grandes como los muy pequeños. Se

escriben usando un número real seguido por el símbolo 'E' y otro número entero. 'E' significa "veces 10 a la potendia". Donde yEx significa "y veces 10 a la potencia x".

ej.

$1.234E{-}2 = 1.234 \times 10^{-2} = 0.01234$
$1.234E0 = 1.234 \times 10^{0} = 1.234$
$1.234E2 = 1.234 \times 10^{2} = 123.4$

El efecto de un entero seguido de 'E' es indicar cuantos lugares ha sido corrido el punto decimal. Un valor positivo significa que ha sido corrido hacia la derecha y un valor negativo hacia la izquierda.

1E10 significa 1 seguido por 10 ceros
10,000,000,000

Usando esta notación científica, el mayor número (el más lejano de cero) que es capaz de almacenar el Amstrad es aproximadamente 1.7×10^{38}; y el más pequeño (el más cercano al cero) 2.9×10^{-39}. Los números son almacenados con una exactitud entre 9 y 10 dígitos.

En el capítulo 5 trataremos los sistemas numéricos hexadecimal y binario, que son formas diferentes de expresar los números mucho, más aproximadas a la forma en que los almacenanan los ordenadores; hasta entonces, tendremos bastante con las notaciones que acabamos de ver.

Variables

Todos los lectores saben que a los datos almacenados por el usuario en la memoria del Amstrad se le dan nombres simbólicos, llamados variables. Las variables se pueden usar para contener datos numéricos enteros, reales o cadenas de datos, y se distinguen por los respectivos símbolos %, ! y $.

```
a% entero
a! real
a$ cadena
```

En efecto, se puede usar el mismo nombre para los tres, ya que constituyen variables separadas. Si se omite el tipo de variable, se asume el tipo real para la variable. Es posible redefinir el tipo por defecto mediante los comandos DEFINT, DEFREAL Y DEFSTR; con ellos se especifica un rango de letras, dentro del cual toma el valor por defecto.

DEFINT I-N las variables oue comien en po I, J, L, M, N se considerarán variables enteras si se omite el tipo de variable.

Y ahora veamos las funciones del Amstrad.

FUNCIONES NUMERICAS SIMPLES

INT

La función INT nos devuelve el componente entero del argumento redondeándolo por abajo.

```
PRINT INT(-5.6),INT(0),INT(5.6)
    imprimirà -6, 0 y 5.

PRINT (X+0.5) redondea X al entero màs cercano
```

El programa 9 nos demuestra el uso de la función INT buscando el divisor mayor de dos números positivos, encuentra el mayor número por el que ambos números puedan ser divididos en partes enteras. Usa un algoritmo muy conocido y que resulta fàcil de entender siguiendo el programa sobre el papel. Escribe tres columnas etiquetadas como A, B y C y guarda el valor de cada variable en su columna correspondiente a medida que cambia.

Programa 9:MAYOR FACTOR COMUN

```
10 CLS
20 PRINT "MAYOR FACTOR COMUN"
30 PRINT
40 INPUT "Introduzca el primer número";a
50 input "Introduzca el segundo número";b
60 IF a<1 OR b<1 THEN 40
70 c = b : b = a-b*INT(a/b) : a = c
80 IF b<>0 THEN 70
90 PRINT "M.F.C. es";c
100 IF INKEY$ = "" THEN GOTO 100
110 GOTO 10
```

CINT

La función CINT devuelve el componente entero del argumento redondeandolo al entero màs cercano en el rango -32768 a 32767.

```
PRINT CINT(n)
```

CREAL

La función CREAL devuelve el argumento de un número real.

```
PRINT CREAL(n)
```

FIX
La función FIX devuelve el componente entero del argumento truncando el componente fraccionario.
PRINT FIX(n)

ROUND
La función ROUND devuelve el argumento redondeandolo a un número especificado de decimales o potencias de diez.
PRINT ROUND (341982,-4) dará 342000
PRINT ROUND (3.1415926,4) dará 3.1416

MIN
La función MIN aplicada a una lista de argumentos nos devuelve el valor mínimo.
PRINT MIN(10,5,15,25,10) dará 5

MAX
La función MAX es similar a la MIN pero devuelve el valor mayor
PRINT MAX (-1,-2,-1,-4) dará -1

RND
La función RND es un generador de números aleatorios que puede se sumamente útil en los juegos que requieren un elemento de suerte. El generador de números no es realmente aleatorio ya que funciona con los valores secuenciales de una larga lista de números. La lista es muy larga por lo que el generador de números aleatorios, empezando en una posición desconocida, cumple con los requerimientos de la mayoría de los programas.

LET a = RND asignará un valor a 'a' que será mayor o igual que 0 y menor que 1.

RND se ilustra en el programa 10 que simula un dado que gira cien veces; se consigue obteniendo un entero aleatorio entre 1 y 6 con la siguiente línea:

```
INT(RND*6+1)
```

Ya que se ha puesto en duda lo aleatorio que puede ser RND, el programa evalúa continuamente dos valores matemáticos, promedio y variación. Los promedios son muy comunes y usted probablemente sabrá que es la suma del valor de lo datos dividido por el número de ellos; cuando tiramos un dado verdadero, el promedio esperado es de 3.5. La variación es menos común para los no matemáticos e indica la extensión a la que tienden los valores del promedio, o una medida de la anchura de distribución. La fórmula matemática para evaluar los promedios y variaciones puede no ser interesante para algunos lectores, pero debe ser conocida para comprender el

funcionamiento del programa. Debemos saber que el símbolo Σ significa "la suma de" por lo que:

$\sum_{i=1}^{n} x_i$ significa la suma de $x_1, x_2, x_3 \ldots$ hasta x_n .

Promedio: $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ Variación: $\sigma = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}$

$= \frac{\sum_{i=1}^{n} x_i^2 - \bar{x}^2}{n}$ (despejando matemáticamente)

Para algunos lectores puede resultar difícil comprender cómo funcionan las variaciones; se ha omitido una explicación al respecto porque involucra unas matemáticas que no son importantes para el usuario del Amstrad. Sin embargo son de gran uso en programación comercial.

Volviendo a los números aleatorios, hay un comando RANDOMIZE que coloca la posición del número aleatorio en un punto determinado. El parámetro de este comando es un valor numérico conocido generalmente como gérmen. Iniciando el gérmen a un valor prefijado garantizamos que realizará la misma secuencia. Para generar un número aleatorio real debemos usar RANDOMIZE TIME que pone el gérmen a un valor basado en el reloj interno del Amstrad de forma que el resultado difícilmente se repetirá. Si no se le pone el valor del gérmen, el Amstrad preguntará por él. Y ahora vamos a nuestro programa:

Programa 10: DISTRIBUCION DE DADOS

```
10 RANDOMIZE TIME
20 CLS
30 PRINT "DISTRIBUCION DE DADOS" : PRINT
40 s = 0 : ss = 0
50 FOR x = 1 TO 100
60 d = INT(RND*6+1)
70 s = s+d : ss = ss+d*d
80 PRINT d;
90 NEXT x
100 a = s/100
110 PRINT : PRINT : PRINT "PROMEDIO";a
120 v = ss/100-a*a
130 PRINT : PRINT "VARIACION";v
140 PRINT : PRINT "Pulse una tecla"
150 IF INKEY$ = "" THEN 150
160 GOTO 20
```

Puede cambiar la entrada del promedio y de la variación a otra cosa que sea más útil para usted. Estoy seguro que el director de su banco se impresionará si le enseña el promedio y la variación de su cuenta para persuadirle de que le descuente las comisiones bancarias.

La función RND tiene una forma más, darle un argumento numérico, con lo que conseguimos los siguientes resultados:

Si es positivo	-devuelve un número aleatorio.
Si es cero	-devuelve el número aleatorio anterior
Si es negativo	-recoloca la posición en la secuencia y devuelve el primer número de la nueva posición.

El próximo programa demuestra cómo, poniendo el gérmen con RANDOMIZE a un valor, la secuencia de números generada es siempre la misma. El programa 11 se usa para codificar y decodificar mensajes secretos y funciona de modo similar al dispositivo alemán "Enigma" usado en la segunda guerra mundial. Esta máquina dependía de un código secreto para cifrar los mensajes, reemplazando unos caracteres por otros. Para hacerlo más difícil, si una letra aparecía varias veces en un mensaje, nunca era reemplazada por el mismo carácter.

Después de introducir el programa y el código, pulse:
'c' para codificar el mensaje.
'd' para decodificarlo
'r' para iniciar el código
'n' para introducir un nuevo código
't' para volver a modo comando

El programa usa un generador de números aleatorios para reemplazar los caracteres. La idea básica es la de ejecutar RANDOMIZE n para comenzar siempre en la misma posición en la lista de secuencias. El gérmen que se usa depende de los caracteres del código. Pulsando 'r' volvemos la secuencia de números aleatorios a su posición inicial para el código elegido. Cuando se mande un mensaje es importante que el que lo recibe conozca el código, para que puede colocar el generador de números aleatorios en la posición correcta de comienzo. Recuerde cambiar el código cuando sea necesario y hacerle conocer a su aliado la posición de los espacios(si hay un espacio en la primera posición, debe introducir el mensaje entre comillas.

Programa 11:ENIGMA

```
10 b$ = SPACE$(1) +
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 b$ = b$ + b$
30 CLS : PRINT "CODIFICADOR·ENIGMA"
```

```
40 PRINT : INPUT "Introduzca el Còdigo";c$ :
c$ = UPPER$(c$)
50 IF LEN(c$)<2 THEN 40
60 RANDOMIZE 10*ASC(LEFT$(c$,1)) +
ASC(RIGHT$(c$,1))
70 CLS : PRINT "CODIFICADOR ENIGMA" : PRINT
80 PRINT "Pulse:" : PRINT
90 PRINT "t)TERMINAR" : PRINT "c)CODIFICAR" : PRINT
"d)DECODIFICAR"
100 PRINT "r)REINICIAR CODIGO" : PRINT "n)NUEVO
CODIGO"
110 PRINT : INPUT a$ : a$=UPPER$(a$)
120 IF a$ = "T" THEN CLS:END
130 IF a$ = "R" THEN 60
140 IF a$ = "N" THEN 30
150 IF a$ = "C" THEN 250
160 IF a$<>"D" THEN PRINT CHR$(7) : GOTO 70
170 PRINT : PRINT "Introduzca el mensaje a
decodificar"
180 PRINT : INPUT m$ : m$ = UPPER$(m$) : PRINT
190 GOSUB 360
200 p = INT(RND*26+1)
210 PRINT MID$(b$,v+p,1);
220 m$ = MID$(m$,2)
230 IF m$ = "" THEN 330
240 GOTO 190
250 PRINT : PRINT "Introduzca mensaje a codificar"
260 PRINT : INPUT m$ : m$=UPPER$(m$) : PRINT
270 GOSUB 360
280 p = INT(RND*26+1)
290 PRINT MID$(b$,37+v-p,1);
300 m$ = MID$ m$,2)
310 IF m$ = "" THEN 330
320 GOTO 270
330 PRINT : PRINT : PRINT "Pulse una tecla para
continuar"
340 IF INKEY$ = "" THEN 340
350 GOTO 70
360 v = ASC(m$)
370 IF v<48 OR v>90 THEN v=47
380 IF v>64 AND v<91 THEN v=v-7
390 v = v-46
400 RETURN
```

FUNCIONES MATEMATICAS

Ahora vamos a hechar un vistazo a las funciones màs comunes del Amstrad, para aplicaciones matemàticas y cientificas; aunque, como ya hemos dicho anteriormente, algunos lectores puedan encontrarlo muy alejado de sus propias necesidades.

EXP

EXP(x) es igual a la constante 'e' elevada a la potencia x donde 'e' se define como 2.718281828. Para explicar el origen de este curioso número, considere lo que ocurre a la expresión $(1 + 1/n)^n$, como n se hace bastante largo, los matemáticos dicen que "n tiende a infinito" y lo escriben como $n \to \infty$). Ejecutando el programa de abajo, vemos que nuestra expresión tiende al número e a medida que n tiende a infinito.

escrito matemáticamente $\lim_{n \to \infty} (1 + 1/n)^n = e$

Se obtiene un resultado más general de la expresión $(1 + x/n)^n$. Encontramos que tiende a a medida que n tiende a infinito.

$$\lim_{n \to \infty} (1 + x/n)^n = e^x$$

De modo que podemos escribir un programa simple para calcular el número 'e':

```
10 PRINT "exponencial e"
20 FOR x = 0 TO 5
30 n = 10↑x
40 e = (1 + 1/n)↑n
50 PRINT n;TAB(10);e
60 NEXT x
```

Observe que no podemos demostrar completamente n tendiendo a infinito ya que tendríamos una gran inexactitud debida a las restricciones contenidas en las rutinas numéricas del sistema operativo del Amstrad.

Se puede obtener un gráfico de e^x en función de x que se vería así, a este tipo de curva se le llama 'curva exponencial'.

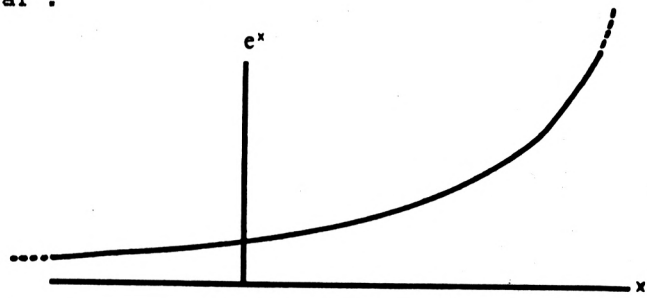


Figura 4.1:Curva exponencial

LOG
La función LOG produce el "logaritmo natural" de un número. El logaritmo de un número es la potencia a la que hay que elevar la base para producir dicho número. Los logaritmos naturales usan la base e.

$$LOG(25)=3.21887583 \text{ ya que } e^{3.21887583}=25$$

Usando la siguiente relación se pueden obtener logaritmos en cualquier base a partir de los logaritmos naturales.

$$\text{Log}_s(x)=\frac{LOG(x)}{LOG(s)}$$

LOG10
Si el valor de s de la ecuación anterior fuera 10, el logaritmo resultante se llamaría "logaritmo común". LOG10 produce el logaritmo común de un número.

SQR
La función SQR obtiene la raíz cuadrada de un número. Se debe tener cuidado ya que la raíz cuadrada de un número negativo producirá un error.

$$SQR(x)=\begin{cases}\sqrt{x} & \text{si } x \text{ es mayor que } 0\\ \text{error} & \text{si } x \text{ es menor que } 0\end{cases}$$

ABS
La función ABS devuelve el valor absoluto de un número, éste tiene la misma magnitud que el argumento pero con signo positivo.

$$ABS(x)=\begin{cases}x & \text{si es mayor o igual a } 0\\ -x & \text{si es menor que } 0\end{cases}$$

SGN
La función SGN devuelve el signo de un número, +1 si el argumento es positivo y -1 si el argumento es negativo o 0 si el argumento es cero.

$$SGN(x)=\begin{cases}+1 & \text{si } x \text{ es mayor que } 0\\ 0 & \text{si } x \text{ es igual a } 0\\ -1 & \text{si } x \text{ es menor que } 0\end{cases}$$

Ahora veamos algunos ejemplos del uso de este tipo de funciones.

Ecuaciones de segundo grado

Este programa usa la función de la raíz cuadrada, SQR. Una ecuación de segundo grado es la que viene en la forma:

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

donde a, b y c se conocen como constantes y x se desea encontrar el valor de x. La siguiente fórmula sirve para solucionar la ecuación pero debemos tomar la regla del programador comercial y considerar que el método descrito es correcto. Así que la solución es:

$$x = \frac{-b \pm \sqrt{(b^2-4*a*c)}}{2*a}$$

Si $b^2-4*a*c$ es mayor que cero, la expresión dentro de la raíz cuadrada puede ser evaluada y tendremos dos soluciones. Se las conoce como soluciones reales.

Si $b^2-4*a*c$ es igual a cero, la parte de la raíz cuadrada no existe y solo hay una solución para x. Esta es también una solución real.

Si $b^2-4*a*c$ es menor que cero, la raíz cuadrada de la expresión no es válida. La mayoría de los lectores se contentarían con decir que no tiene solución pero, es eso cierto? Los matemáticos han rodeado el problema usando una notación llamada números complejos. Estos números se representan por x+iy, donde x es el componente real, y es el imaginario e i representa $\sqrt{(-1)}$. Volviendo a nuestro problema, la solución a nuestra ecuación es un complejo cuya parte real es $-b/(2*a)$ y la parte imaginaria es igual a $\sqrt{(4*a*c-b^2)}/(2*a)$ y ambas expresiones pueden ser evaluadas. Observe que las soluciones complejas no pueden ser evaluadas directamente por el BASIC del Amstrad y debe tenerse cuidado de no usar SQR con un número negativo ya que nos produciría un error que interrumpiría la ejecución. Por fortuna para la mayoría de nosotros, los números complejos no se usan todos los días.

Programa 12: ECUACIONES DE SEGUNDO GRADO

```
10 CLS : PRINT "ECUACIONES DE SEGUNDO GRADO" :
PRINT
20 INPUT "Introduzca el coeficiente a";a
30 INPUT "Introduzca el coeficiente b";b
40 INPUT introduzca el coeficiente c";c
50 PRINT
60 d = b*b-4*a*c
70 IF d<0 THEN 120
```

```
80 PRINT "Raices reales"
90 PRINT "1. x = ";(-b+SQR(d))/(2*a)
100 PRINT "2. x = ";(-b-SQR(d))/(2*a)
110 GOTO 150
120 PRINT "Raices complejas"
130 PRINT "Parte real = ";-b/(2*a)
140 PRINT "Parte imaginaria = ";SQR(-d)/(2*a)
150 PRINT : PRINT "Pulse una tecla para
continuar"
160 IF INKEY$ = "" THEN 160
170 GOTO 10
```

Números primos

Un número primo es un entero que solamente es divisible por sí mismo o por 1. El programa 13 imprime los 100 primeros números primos, asumiendo que los dos primeros son el 2 y el 3 y que todos los demás números primos son impares.

El programa comprueba si el número impar x es primo dividiéndolo por todos los números primos menores que que se han encontrado; Si el resto no es cero, el número x es primo también. Hay un punto que necesita una explicación, por qué dividir solamente por los números primos menores que ?. No se usan los números no-primos porque pueden descomponerse en, por lo menos, dos números primos. Los números mayores que no se usan porque, si fueran divisores, existiría por lo menos uno menor que

A medida que se obtienen los números primos, se almacenan en la matriz P(100) de forma que pueda ser usado para comprobar los siguientes números. Ya que la aritmética del Amstrad sólo puede calcular con exactitud hasta figuras significativas, asumimos que un número es divisor si el resto es menor que 1E-10.

Programa 13:NUMEROS PRIMOS

```
10 CLS : PRINT "NUMEROS PRIMOS"
20 DIM p(100)
30 p(1)=2 : p(2)=3 : x=5 : n=3
40 PRINT "El primo ";1;TAB(18); "es" ; p(1)
50 PRINT "El primo ";2;TAB(18); "es" ; p(2)
60 FOR j = 2 TO n
70 IF x/p(j) - INT(x/p(j)) <1E-10 THEN 130
80 IF SQR (x) < p(j) THEN 100
90 NEXT j
100 p(n) = x
110 "El primo ";n;TAB(18); "es" ; p(n)
120 n = n+1
```

```
130 x = x+2
140 IF n<101 THEN 60
```

FUNCIONES TRIGONOMETRICAS

El Amstrad dispone de cuatro funciones trigonomètricas, SIN (seno), COS (coseno), TAN (tangente) y ATN (arco tangente).

Las funciones trigonomètricas funcionan con àngulos, y aquí es donde encontramos el primer problema. La mayoría de las personas miden los ángulos en grados; donde el círculo està dividido en 360 grados, un àngulo recto tiene 90 grados y una línea recta tiene 180 grados. Sin embargo, los matemàticos prefieren medirlos con una unidad mayor llamada radian que es el àngulo delimitado por un arco cuya longitud es igual a la del radio.

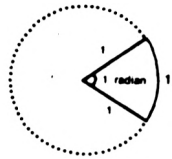


Figura 4.2

Por definición hay dos Pi radianes en un círculo, siendo Pi aproximadamente 3.14159265358 y se escribe π. Por lo tanto un radian tiene $360/2\pi$ radianes que vienen a ser 57.3 grados.

Para convertir radianes a grados, multiplicamos por $180/\pi$

radianes = $\pi/3 * 180/\pi$ = 60 grados

Para convertir grados a radianes se multiplica por $\pi/180$

5 grados = $45 * \pi/180 = \pi/4$ radianes.

En el Amstrad tenemos la función PI. También tenemos la opción de acceder a radianes o grados en las operaciones trigonomètricas mediante los comandos DEG y RAD; los radianes se usan como defecto. Mientras que la definición de las funciones trigonomètricas es bastante simple, la explicación de su uso no lo es. La lista de las aplicaciones sería infinita, por lo que será suficiente que digamos que son de un valor inapreciable; su importancia para la ciencia y la tecnología es equivalente a la del oxígeno para la vida.

Veamos ahora las cuatro funciones. Consideremos el siguiente triàngulo rectàngulo con un àngulo de X radianes y sus lados etiquetados como opuesto, adyacente e hipotenusa.

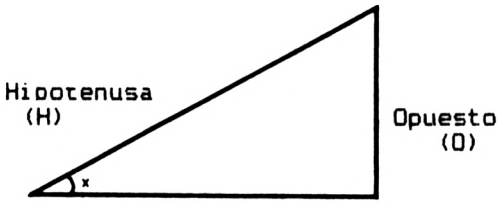


Figura 4.3

Seno, coseno y tangente se pueden definir como la razòn de las longitudes de unos lados respecto a otros.

SIN
El seno del àngulo X se define como la razòn de la longitud del lado opuesto con la longitud de la hipotenusa, en cualquier triàngulo rectàngulo con un àngulo de X radianes.

$$SIN(X) = \frac{O}{H}$$

Un gràfico de SIN(X) dibujado se veria asi:

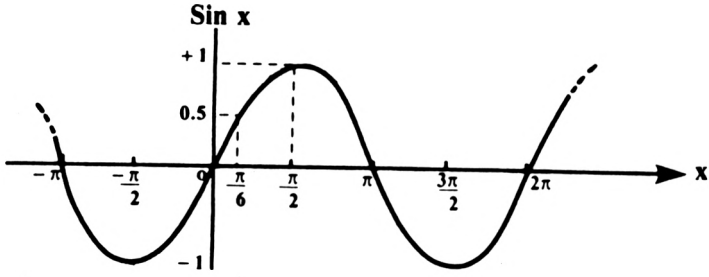


Figura 4.4:La Onda del Seno

COS
El coseno del àngulo x se define como la razòn de la longitud del lado adyacente con la hipotenusa.

$$COS(X) = \frac{A}{H}$$

Un gràfico de COS(X) se veria asi:

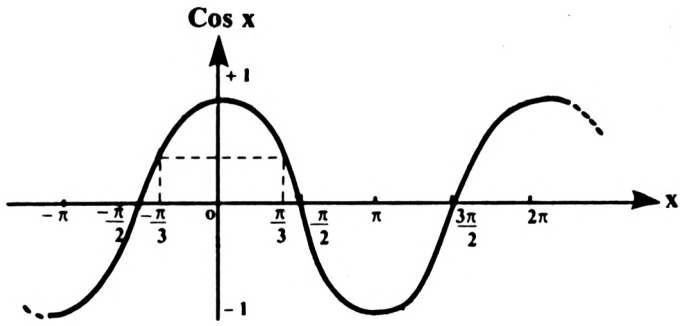


Figura 4.5:La Onda del Coseno

TAN
La tangente del àngulo X se define como la razòn de la longitud del lado opuesto con la del adyacente.

$$TAN(X) = \frac{O}{A}$$

Un gràfico de TAN(X) se verìa asì:

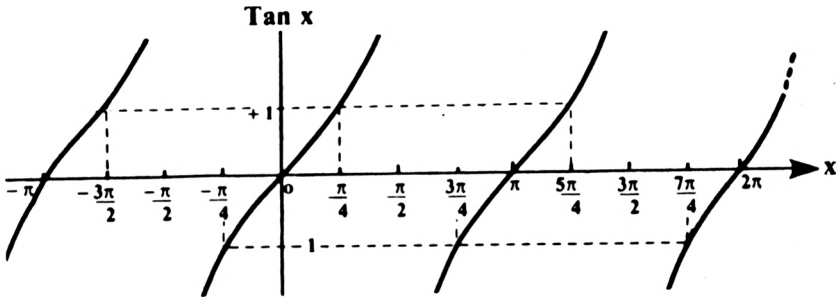


Figura 4.6:Las Lìneas de la Tangente

La funciòn trigonomètrica que queda es:

ATN
El arco tangente de y, devuelve el arco cuya tangente es y.

Los valores del arco seno y arco coseno, que son los inversos del seno y el coseno, pueden calcularse mediante la siguiente fòrmula

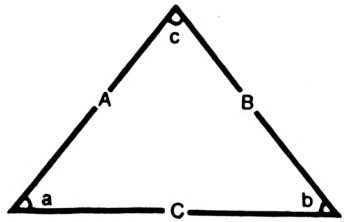
$$\text{Arco seno}(y) = \text{ATN}\left[\frac{y}{\text{SQR}(1-y^2)}\right]$$

$$\text{Arco coseno}(y) = \text{ATN}\left[\frac{y}{\text{SQR}(1-y^2)}\right] + \frac{\pi}{2}$$

En ambos casos y se encuentra en el rango -1<=y<=+1

Aunque desafortunadamente el Amstrad carece de las funciones para obtener los valores del arco seno y arco coseno, el siguiente programa nos demuestra que es relativamente simple su càlculo usando la funciòn ATN.

El programa 14 calcula los ângulos de un triàngulo del que se conocen las longitudes de sus tres lados. Consideremos el triàngulo de la siguiente figura del que se conocen las longitudes de A, B y C.



Sabemos que existe la siguiente relaciòn:

$$C^2 = A^2 + B^2 - 2*A*B*\text{Coseno}(c)$$

Despejando la ecuaciòn tendremos: $c = \text{arco cos}\,\frac{(A^2 + B^2 - C^2)}{(2*A*B)}$

Existen relaciones similares para los dos ângulos restantes. El programa 14 permite introducir las longitudes de los tres lados y calcula los ângulos en grados. Recuerde que la suma de los tres ângulos de un triàngulo es 180 grados, de este modo tendremos una pequeña prueba de la exactitud del Amstrad.

Programa 14: TRIANGU' ?

```
10 CLS : PRINT "TRIANGULO" : PRINT
20 DEF FN t(x,y,z) = (y*y+z*z-x*x)/(2*y*z)
```

```
30 DEF FN a(x) = 90-ATN(x/SQR(1-x*x))
40 INPUT "Introduzca lado uno";a
50 INPUT "Introduzca lado dos";b
60 INPUT "Introduzca lado tres";c
70 PRINT : PRINT
80 DEG
90 x = FN t(a,b,c)
100 IF ABS(x)>1 THEN PRINT "Invalido!!" : GOTO 230
110 aa = FN a(x)
120 x = FN t(b,a,c)
130 IF ABS(x)>1 THEN PRINT "Invalido!!" : GOTO 230
140 bb = FN a(x)
150 x = FN t(c,a,b)
160 IF ABS(x)>1 THEN PRINT "Invalido!!" : GOTO 230
170 cc = FN a(x)
180 PRINT "Angulo opuesto a lado uno" ; ROUND
(aa,2)
190 PRINT "Angulo opuesto a lado dos" ; ROUND
(bb,2)
200 PRINT "Angulo opuesto al lado tres" ; ROUND
(cc,2)
210 PRINT
220 PRINT "La suma de los àngulos es" ; aa + bb + cc
230 PRINT
240 PRINT "Pulse una tecla para continuar"
250 IF INKEY$ = "" THEN 250
260 GOTO 10
```

PROGRAMACION RECURRENTE

Vamos a hechar una breve ojeada a la tècnica de "software" llamada recurrente que puede ser ùtil algunas veces, especialmente para evaluar funciones matemàticas que contienen bucles de proceso repetitivo. En BASIC, una rutina recurrente es aquella que se llama a sí misma y que suele producir soluciones claras y elegantes.

Por ejemplo, consideremos la expresiòn matemàtica factorial n (o n!) que se expande a:

n!=n*(n-1)*(n-2)*.....*3*2*1

donde n es un entero positivo (ej. 6!=6*5*4*3*2*1=720)

que equivale a:

$$n! = \begin{cases} n*(n-1)! & n>1 \\ 1 & n=1 \end{cases}$$

Se puede escribir una subrutina para evaluar n! con n multiplicado por el resultado de llamarse a sí misma para evaluar (n-1)!. Debemos tener mucho cuidado de para en 1! para evitar las llamadas recurrentes.

En este momento es necesario ver como controla el sistema operativo la llamada anidada a la subrutina usando lo que se llama la pila GOSUB. Veremos con más detalle las pilas en le capítulo 7; hasta entonces es suficiente saber que, cuando se llama a una subrutina, el número de la línea de la sentencia GOSUB que llama se almacena hasta que encuentra una sentencia RETURN; entonces retorna el control a la línea siguiente a la última almacenada, que es borrada de la pila. Si se ejecutan más RETURN que sentencias GOSUB, se vacía la pila y ocurre un error.

Programa 15:RECURRENTES

```
10 CLS
20 INPUT "Introduzca un valor";x
30 GOSUB 1000
40 PRINT : PRINT "Valor factorial";f
50 END
1000 f = 1
1010 IF x>1 THEN f = f*x : x = x-1 : GOSUB 1010
1020 RETURN
```

SIMULACION

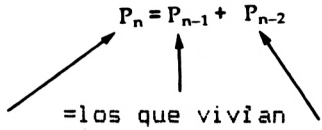
Muchos procesos o sistemas, como por ejemplo las reacciones químicas, modelos de población, colas, flujo del tráfico, batallas, circuitos electrónicos, etc. pueden ser examinados en un ordenador mediante la simulación de datos que son propensos a cambiar. Se requiere construir un modelo matemático que cubra las relacciones entre las variables y tenga en cuenta el efecto del tiempo y de sucesos externos. Algunos procesos tardan bastante tiempo en completarse, una simulación puede presentar el resultado en segundos . Es también más barato que un piloto en entrenamiento se estrelle con un simulador de vuelo que con un avión real.

Los modelos matemáticos usados en simulación pueden implicar modelos extremadamente complicados que están muy alejados de los objetivos de este libro. Sin embargo, vamos a examinar un modelo simple que simula el crecimiento de la población.

En el siglo trece, el matemático italiano Leonardo Fibonacci, estudió la explosión demográfica de los conejos. Consideró el crecimiento de la población, empezando con solo

un par de conejos y asumió que tendrían un par de hijos justo un mes antes de que volvieran a ser fértiles y se pudieran volver a reproducir. Así pues, habría una nueva generación cada mes. Se asume que no hay muertes ni migraciones en el periodo en consideración.

Analizando el problema, podemos ver que la población en el mes n sería , por lo que



ej. nueva población

=los que vivían el mes anterior y que todavía viven.

+un par más por cada uno de los fértiles. ej. estos vivían hace un mes

También sabemos que $P_1 = 1$ y $P_2 = 1$

Los valores de se encuentran con el programa 16, la secuencia de números que se obtiene se conoce como la secuencia de Fibonacci.

Programa 16:CONEJOS

```
10 CLS : PRINT "SECUENCIA DE FIBONACI" : PRINT
20 p1 = 1 : p2 = 1 : m = 2
30 PRINT "Mes 1 :";p1;"Conejo"
40 PRINT "Mes 2 :";p2;"Conejo"
50 p = p1 + p2 : m = m + 1
60 PRINT "Mes";m;":";p;"Conejos"
70 p1 = p2 : p2 = p
80 IF m<36 THEN 50
```

CAPITULO CINCO
MAPA DE LA MEMORIA DEL AMSTRAD

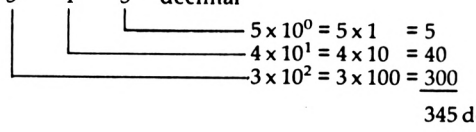
Hasta ahora hemos tratado de la memoria del Amstrad como de un componente "hardware" cuya función es la de almacenar todos los datos que hay en la ROM y en la RAM del sistema, descritos en el capítulo uno. La mejor forma de describir la estructura interna de la memoria del ordenador es imaginarla como una larga secuencia de cajas o celdas identificadas cada una con una etiqueta numérica única que comienza con cero y se va incrementando de uno en uno, como si fueran las taquillas de una estación. La etiqueta de cada celda de memoria se llama dirección. En cada caja o celda, se puede almacenar un número, que en el caso del Amstrad y de la mayoría de los ordenadores personales, está en el rango de 0 a 255. Además de números, es posible almacenar caracteres e instruciones de programa representados por sus valores numéricos; estos valores se conocen como código de carácter y códigos de instrucción. Cuando nos referimos al tamaño de la memoria, se elige la abreviatura K que representa 1024(o 2 elevado a la potencia 10) celdas de memoria.

NUMEROS BINARIOS

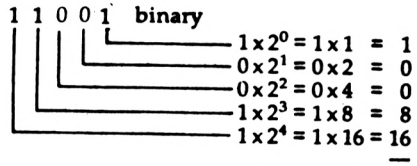
Veamos ahora la forma simple en que el Amstrad almacena los datos. La naturaleza de la electrónica restringe a los circuitos a estar conectados o desconectados y esto significa que el ordenador solo puede reconocer dos estados; estos estados se suelen escribir como '0' o '1'. Así como las palabras y sentencias se construyen usando más de una letra y los números constan de varios dígitos, las expresiones de ordenador se representan por secuencias de '0' y '1'. A este tipo de modelo de '0' y '1' se le llama 'números binarios'.

Cuando se escriben los números en forma decimal o de base 10, el dígito de más a la derecha representa a las unidades, el de su izquierda las decenas (la base de nuestro sistema), el siguiente a su izquierda representa las centenas (el cuadrado de la base), y así sucesivamente. Los números binarios usan como base el 2; el dígito de más a la derecha da el número de unidades, el de su izquierda da el número de doses (la base), el siguiente da el número de cuatros (el cuadrado de la base), el siguiente da el número de ochos (el cubo de la base) y así sucesivamente.

Convencionalmente, el número 345 en sistema decimal es:



Del mismo modo el número binario 11001 equivale a:



Cuando contamos en binario, pasa un '1' a la siguiente columna siempre el resultado pase de '1' ej. después del 0 viene el 1, después del 1 viene el 10, después del 10 viene el 11, después del 11 viene el 100 y así sucesivamente. Esta secuencia se puede ilustrar con la siguiente rutina, que usa la función BIN$ que convierte un argumento numérico dentro del rango -32768 a 65535 a una cadena conteniendo su equivalente en forma binaria.

```
10 CLS
20 FOR x = 0 TO 255
30 PRINT "Decimal";x;TAB(13);"Binario";BIN$(x)
40 NEXT x
```

Se puede ver que el sistema numérico binario trabaja con el mismo principio que el decimal, pero se requieren más dígitos para representar un número en binario que en decimal. Si queremos usar números binarios en el Amstrad, debemos precederlos por &X para distinguirlos de los decimales.

Ahora es posible explicar porqué un número almacenado en una dirección particular de la memoria del Amstrad debe estar en el rango 0 a 255. Esto es debido a que una celda de memoria contiene exactamente ocho dígitos binarios; por lo que el rango desde 0000 0000(0 decimal) a 1111 1111(255 decimal). Cuando nos referimos a números binarios, cada dígito individual se llama bit y el grupo de ocho bits se llama octeto o byte. Los bits se suelen numerar del 0 al 7 con el bit 0 en la posición menos significativa y el 7 en la más significativa (el bit 0 es el de más a la derecha y el 7 el de más a la izquierda dentro del octeto).

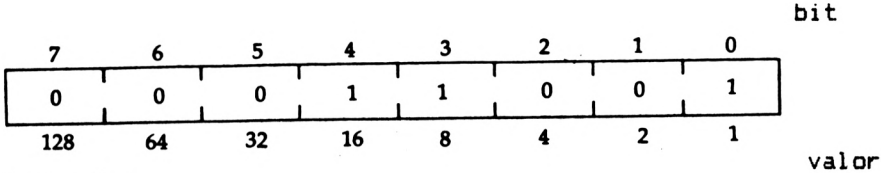
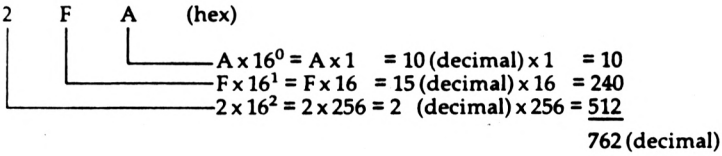


Figura 5.1

Los números entre 0 y 255 que no sean enteros, se representan combinando varios octetos.

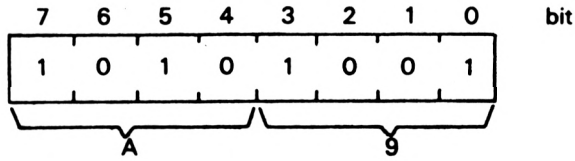
Debido a que las cadenas largas de '0' y '1' son difíciles de memorizar, resulta bastante complicado trabajar con ellas, se han desarrollado algunas notaciones más simples. Una de éstas usa un sistema numérico de base diez y seis y es conocido como sistema hexadecimal. Para usarlo son necesarios 16 dígitos pero, en vez de diseñar seis nuevos símbolos, se han usado las cinco primeras letras del alfabeto. Los 16 dígitos son 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, F. Es importante no confundir los nuevos símbolos con las letras. Como ejemplo veamos el número hexadecimal 2FA:



La función HEX$ devuelve el argumento en forma hexadecimal.

```
10 CLS
20 FOR x = 0 TO 255
30 PRINT "Decimal";x;TAB(13);"Hex.";HEX$(x)
```

Ya que un octeto contiene ocho bits, su valor puede ser expresado con dos números hexadecimales; con algunas excepciones, el sistema hexadecimal es el que suele usarse en los ordenadores.



Al Amstrad se le pueden introducir directamente los números

hexadecimales precediendolos por &. ej. &AB.

MAPA DE LA MEMORIA

La memoria del Amstrad està subdividida en varias secciones distintas, cada una de las cuales tiene un propòsito especìfico. La estructura visual de la memoria se da en un diagrama llamado mapa de memoria y se puede ver en la figura 5.2.

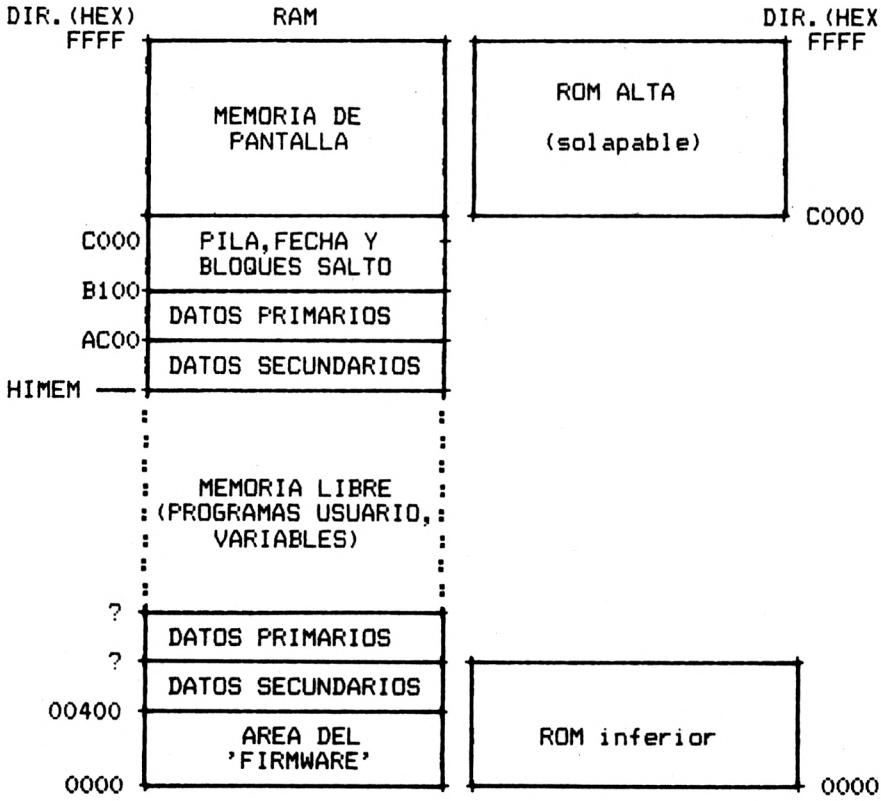


FIGURA 5.2:MAPA DE MEMORIA DEL AMSTRAD

El mapa de memoria es bastante complicado debido a que, aunque solo tiene 64K direccionables, el Amstrad tiene 64K de RAM y 32K de ROM, ademàs tiene previsto expansiòn de ROM hasta 252*16K (cerca de 4 Mega octetos.'

Los 32K de ROM estàndar se dividen en dos partes de 16K; los 16K de superiores, direcciones C000-FFFF contienen el intèrprete de BASIC, y los 16K inferiores ,direcciones 0000-3FFF contienen el 'firmware' del Amstrad que son las rutinas que controlan el 'hardware' y otras que pueden ser llamadas por el còdigo màquina del usuario. En la figura 5.2 podemos ver que el intèrprete de BASIC se solapa con una secciòn de RAM que contiene la memoria de pantalla. Un componente electrònico llamado matriz de puertas se encarga de bascular entre la ROM y la RAM dependiendo de los requerimientos de acceso del 'firmware'. El controlador de pantalla y la CPU estàn sincronizados de forma que nunca necesitan acceder a estas direcciones simultàneamente.

Como ya hemos dicho, se pueden solapar hasta 252 ROMs de 16K en las 16K superiores y se definen como primarios y secundarios (màximo 7 ROMS). Una ROM secundaria contiene programas simples (por ejemplo otros leguajes, sistemas operativos, aplicaciones de negocios, juegos) que se ejecutan uno de cada vez, mientras que las ROM primarias contienen rutinas (por ejemplo rutinas perifericas de expansiòn) que pueden ser llamadas por otros programas.

Cuando conectamos el Amstrad, el 'firmware' busca estas ROM de una en una (pàgina por pàgina) y ejecuta el arranque de la primera que encuentra. En un Amstrad sin expansiones, la primera (y ùnica) ROM presente es la de BASIC que se obtiene al encender. El BASIC està en la pàgina 1; si se tiene que obtener otros programas, como cartuchos de juegos, deben estar en la pàgina 0. Cualquier ROM presente se puede seleccionar mediante la barra vertical '|' seguida del nombre de la ROM. De este modo podemos pasar a la ROM del BASIC mediente el comando |BASIC.

Por fortuna, el Amstrad tiene un BASIC excelente que raramente profundiza en la memoria. Sin embargo los programadores de còdigo màquina necesitan conocer en detalle el mapa de memoria y la posiciòn de las rutinas del 'firmware'. El manual del Amstrad proporciona informaciòn detallada sobre este particular. Ya que puede variar la posiciòn de estas rutinas si se cambia de ROM, al llamarlas desde un programa se supone que se quieren ejecutar en la ROM original. Para evitar este problema hay un bloque de bifurcacion en las posiciones BB00 a BD39; el programa llama a la rutina a una direcciòn especìfica del bloque de bifurcaciòn que, a su vez, llama a la rutina requerida. Los cambios en la ROM deben ir acompañadas de los correspondientes en el bloque de bifurcaciòn para que no afecte al programa del usuario. Tambièn es posible modificar el bloque de bifurcaciòn para apuntar a sus propias rutinas alterando el efecto del 'firmware'- tenga en cuenta que los valores

devueltos en todos los registros deben ser compatibles con los del sistema original.

El programa en BASIC junto con sus variables son almacenados en la memoria libre que se ve en la figura 5.2. La dirección más baja la determina el 'firmware' cuando se inicia el BASIC. La dirección más alta se loacaliza mediante el apuntador HIMEM al que se puede acceder directamente desde el BASIC mediante el comando HIMEM. Podemos engañar al Amstrad para que piense que que tiene menos memoria, cambiando este apuntador con el comando MEMORY, para que apunte a una direción inferior y dejar libre una sección de memoria para tener un sitio seguro para nuestros programas en código máquina, gráficos etc.

Para liberar 10 octetos del BASIC teclear el comando:
MEMORY-10

La función FRE(0) devuelve el número de octetos libres; intente ejecutar PRINT FRE(0) antes y después del comando anterior para comprobar que se han liberado los 10 octetos.

Podemos leer y escribir directamente en la RAM del Amstrad mediante los comandos PEEK y POKE.

```
  POKE A,V
```
escribirá el valor V en la direción A
(V está restringido a enteros en el rango 0 a 255)

```
  V=PEEK(A)
```
asigna a la variable V el contenido de la dirección A.

CAPITULO SEIS
TIEMPO, RELOJ
E INTERRUPCIONES

LA FACILIDAD DE TIEMPO

El Amstrad tiene un contador interno que comienza a contar desde el momento en que el Amstrad es conectado. El contador se incrementa cada 1/300 de segundo y es sûmamente preciso. Tenemos al posibilidad de añadir tiempos a nuestros programas mediante el uso de este contador, por ejemplo como reloj, despertador o movimientos de juegos dependientes de tiempo. Tambièn podemos medir la longitud de los procesos para comparar la eficiencia de los diferentes mètodos de programaciòn.

Para acceder a este contador usamos el comando TIME y para conocer la longitud del tiempo en segundos necesitamos conocer los valores inicial y final de TIME y dividir la diferencia por 300. Por ejemplo, el programa 17 nos da una idea de la funciòn TIME calculando la velocidad de sus reacciones. Tenga en cuenta que debemos restar 1/100 de segundo del tiempo calculado para quitar el tiempo de proceso - pruebe el programa eliminando la lînea 120:

PROGRAMA 17:REACCION

```
10 CLS
20 LOCATE 10,10 : PRINT "Pulse ENTER para
empezar"
30 IF INKEY(18) = -1 THEN 30
40 CLS
50 FOR x = 0 TO 3000*RND
60 IF INKEY(47)<>-1 THEN CLS : PRINT "CHASCO!!"
: END
70 NEXT x
80 LOCATE 10,10 : PRINT "Pulse ESPACIO!!"
90 BORDER 7,16
100 t1 = TIME
110 PRINT CHR$(7)
120 IF INKEY(47) = -1 THEN 120
130 t2 = TIME
140 BORDER 1
150 LOCATE 8,13
160 PRINT "Tiempo " ; ROUND ((t2-t1-3)/
300,2);"segundos"
170 FOR x = 0 TO 3000 : NEXT x
180 GOTO 10
```

Es dificil de describir con precisiòn què es lo que hace adictivo a un programa de ordenador, pero incluir una facilidad de 'màxima puntuaciòn obtenida' hace incrementar la adicciòn!. Esto hace que los jugadores tiendan a conseguir 'un punto màs' para colocarse en lo alto de las listas.

El programa 38 demuestra esta facilidad con un simple juego de tipo 'aracade'. El jugador tiene que evadir un ataque de misiles moviendo a un hombrecito de izquierda a derecha bajo el control de la teclas del cursor. El objeto del juego es sobrevivir tanto tiempo como sea posible registrando el mejor tiempo conseguido.

PROGRAMA 18:ATAQUE DE MISILES

```
10 DEFINT p : DIM m(25) : RANDOMIZE TIME
20 CLS : BORDER 2+RND*25
30 n = 20 : t = 100 : z1 = TIME
40 p = 40*RND+1
50 LOCATE p,25 : PEN 1
60 x = x+1 : IF x>25 THEN x=1
70 m(x) = p : v = 0
80 PRINT CHR$(239) : PRINT
90 IF m((x+3)MOD 25) = n+1 THEN 170
100 IF INKEY(8) <>-1 AND n>1 THEN n = n-1 : v = 3
110 IF INKEY(1) <>-1 AND n<38 THEN n = n+1 : v =
2
120 LOCATE n,1 : PEN 2
130 PRINT SPACE$(1) + CHR$(248+V) + SPACE$(1)
140 FOR j = 1 TO t : NEXT j
150 IF t = 1 THEN t = t-1
160 IF m((x+2) MOD 25) <> n+1 THEN 40
170 z2 = ROUND ((TIME-z1)/300,2)
180 PRINT CHR$(7)
190 LOCATE n+1,1 : PRINT CHR$(238)
200 FOR j = 1 TO 300 : NEXT j
210 IF z2>h OR h = 0 THEN h= z2
220 CLS : PEN 2
230 LOCATE 14,10 : PRINT "Tiempo";z2
240 LOCATE 14,12 : PRINT "Mayor";h
250 LOCATE 14,15 : PRINT "Otra vez? S/N"
260 IF INKEY(60)<>-1 THEN ERASE m : GOTO 10
270 IF INKEY(46)<>-1 THEN CLS : BORDER 1 : END
280 GOTO 260
```

INTERRUPCIONES

En ordenadores es comùn el uso de interrupciones. Una interrupciòn ocure cuando, bajo ciertas condiciones, el ordenador para lo que està haciendo y ejècuta otro proceso y cuando se ha completado, vuelve para completar la tarea

anterior. El manejo de interrupciones se escribe normalmente en código máquina; sin embargo el Amstrad tiene algunos comandos de BASIC, únicos, que pueden manejar las interrupciones de tiempo, esto es, se ejecuta una rutina cuando pasa un periodo de tiempo determinado y, cuando se ha completado, vuelve el control a la posición del programa donde ocurrió la interrupción. Se pueden usar dos comandos, AFTER y EVERY; el primero prepara una interrupción para que ocurra después de un tiempo especificado mientras que el segundo prepara interrupciones para que ocurran regularmente con la frecuencia especificada. Hay cuatro temporizadores disponibles para estas interupciones y por lo tanto, se pueden usar hasta cuatro rutinas de interrupción. Cada temporizador tiene una prioridad - el temporizador 3 tiene la prioridad más alta y el 0 la más baja. Si ocurre una interrupción durante la ejecución de otra rutina de interrupción, se ejecutará instantáneamente si es de más alta prioridad o será encolada hasta la complexión del primero, si es de menor prioridad. Cuando se prepara una interrupción, el periodo debe especificarse en 1/50 de segundo seguido por el número del temporizador usado.

```
AFTER 100,3 GOSUB 1000 - ejecuta la subrutina
                         de la línea 100 una
                         vez cada dos segundo.

EVERY 50 GOSUB 1000 - ejecuta la subrutina de
                      la línea 1000 cada segundo.
                      Tenga en cuenta que, si
                      no especifica número de
                      temporizador, asume el 0.
```

Debe tener cuidado cuando use estos comandos ya que las subrutinas se pueden ejecutar en cualquier punto del programa.

El siguiente ejemplo de interrupciones muestra un reloj en la parte superior izquierda de la pantalla que suena cada hora. Ya que el reloj se procesa en una subrutina de interrupciones, puede incorporarse en sus propios programas. Se prepara una ventana #1 para mostrar el reloj por lo que su programa debe evitar este cauce.

PROGRAMA 19:ESQUELETO DE RELOJ

```
10 WINDOW #1,1,9,1,1 : WINDOW #0,1,40,2,25
20 CLG : PRINT "Introduzca tiempo:" : PRINT
30 INPUT "Introduzca horas";h
40 IF h<0 OR h>23 THEN 20
50 INPUT "Introduzca minutos";m
60 IF m<0 OR m>59 THEN 20
```

```
70 INPUT "Pulse ENTER a la señal de tiempo";q$
80 CLS
90 EVERY 50 GOSUB 1000
100 'continuación del programa
110 GOTO 110


1000 s = s+1
1010 IF s>59 THEN s = s MOD 60 : m = m+1
1020 IF m>59 THEN m = m MOD 60 : h = h+1 : PRINT
CHR$(7)
1030 IF h>23 THEN h = h MOD 24
1040 IF s MOD 2 = 0 THEN c$ = ":" ELSE c$ =
SPACE$(1)
1050 LOCATE #1,1,1
1060 PRINT #1, USING "##!";h;c$;
1070 PRINT #1, USING "##!";m;c$;
1080 PRINT #1, USING "##";s
1090 RETURN
```

Puede haber casos en los que sea importante que no ocurran interrupciones - se puede hacer inhibiendo las interrupciones con el comando DI y, cuando se quiera que se ejecuten de nuevo, se pueden habilitar con el comando EI. Por último, es posible conocer el tiempo que le queda a un temporizador determinado mediante la función REMAIN.

Interrupción de BREAK

La interrupción de BREAK tiene la prioridad más alta. [BREAK] supone pulsar la tecla [ESC] dos veces - pulsando [ESC] una vez, se para la ejecución temporalmente hasta que se pulse otra tecla. Sin embargo, [BREAK] devuelve control al modo comando, podemos dirigir la ejecución a una subrutina de [BREAK] usando ON BREAK GOSUB.

```
10 CLS
20 ON BREAK GOSUB 1000
30 'continuación del programa

1000 CLS
1010 PRINT "Quiere salvar los resultados? (s/n)"
1020 IF INKEY(46)<>-1 THEN END
1030 IF INKEY(60) = -1 THEN 1020
1040 'salva resultados
```

Un truco peligroso pero útil es poner una sentencia RETURN en la subrutina de BREAK que significa que el usuario no podrá hacer [BREAK] del programa (excepto en una sentencia

INPUT).

Las interrupciones de BREAK se pueden inhibir con:

```
ON BREAK STOP
```

Los comandos DI y EI no afectan a las interrupciones de BREAK.

TRAMPAS DE ERRORES

Normalmente, cuando ocurre un error en nuestros programas BASIC, la ejecución se para y devuelve control al modo comando señalando que ha ocurrido un error. Sin embargo, el Amstrad tiene una potente facilidad para poner trampas a los errores. Esto significa que cuando ocurre el error, en vez de devolver el control al modo comando, toma control una rutina que puede rectificar el problema o dar al usuario más información sobre lo que ha sucedido.

Para preparar una trampa de error usamos la sentencia ON ERROR GOTO seguido por el número de la línea a la que se tiene que transferir el control.

```
ON ERROR GOTO 1000
```

La trampa de error se puede inhibir especificando un número de línea 0.

```
ON ERROR GOTO 0
```

Hay dos funciones que nos dan información adicional de los errores, ERL que devuelve el número de la línea de la sentencia donde ha ocurrido el error y ERR que es el número del error (ver el apéndice D).

```
PRINT "Error";ERR;" ocurrido en línea";ERL
```

En este punto remitimos al lector al capítulo 3 donde se crea una ventana para mensajes de error en la que se muestra una línea al pie de la pantalla y suena un beep para que el usuario pulse una tecla para continuar - una trampa ideal para errores.

Una vez que se ha entrado en una trampa de error, podemos usar el comando RESUME para continuar la ejecución del programa en la línea en que ocurrió el error, la línea siguiente o una línea especifica.

```
RESUME        continúa en la línea ERL
RESUME NEXT   continúa el la línea siguiente a ERL
RESUME 500    continúa en la línea 500
```

Por último, es posible programar errores de usuario con el comando ERROR seguido de un número de error que no debe estar en el rango de los del BASIC. Así como los errores de programa son interceptados por el intérprete BASIC, los datos inválidos pueden serlo por el programa de forma similar.

```
100 INPUT " Introduzca número de cuenta";aN$
110 IF LEN(an$)<>7 THEN ERROR 100
```

CAPITULO SIETE

ESTRUCTURAS DE DATOS

Vamos a hechar una ojeada al tema de las estructuras de datos que nos va a suponer manejar eficientemente grupos de datos similares. Para empezar refrescaremos nuestra memoria con las matrices.

MATRICES - ESTRUCTRAS ESTATICAS DE DATOS

Frecuentemente necesitamos reservar un bloque de celdas de memoria del Amstrad para que puedan ser referenciadas con un nombre simple (referido al bloque) y un número(referido a una celda particular dentro del bloque); este tipo de estructura se llama matriz. Para reservar bloques de una longitud determinada se usa la sentencia DIMension.

DIM x(15) instruye al ordenador para que reserve una matriz llamada 'x' de diez y seis elementos.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig 7.1

Cada celda puede almacenar un número. Cuando se declara un bloque, cada celda se inicia con cero. Podemos referirnos a cualquier celda mediante el nombre de la variable seguido del número de la celda entre parèntesis. Por ejemplo, la celda 5 es referida como x(5). Al término entre parèntesis se le conoce como subíndice. Se puede referenciar la misma celda con una variable en el lugar del subíndice, por ejemplo x(a), a la variable a se le ha dado el valor 5. Los subíndices pueden ser también expresiones variables, pero sus resultados deben estar en el rango especificado por la sentencia DIM. Si hacemos referencia a una cadena que no ha sido DIMensionada, asume que tiene once elemetos de longitud.

También es posible preparar matrices con màs de un subíndice:
DIM y(3,4)

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)

y

Fig 7.2

A este tipo de estructura se le llama matriz bidimensional; recuerde que las celdas se almacenan físicamente en forma secuencial dentro de la memoria. La misma idea es válida para matrices de n dimensiones. Si la variable de la matriz va seguida por los signos % o $, las variables son de enteros o cadena respectivamente.

El programa 20 demuestra las matrices simulando una competición de fútbol consistente en seis encuentros con 64 clubs en competición. Cada club es almacenado en una sentencia DATA junto con el valor que representa la calidad del equipo. Si no aparece en la lista su club favorito, cambielo por cualquier otro. Usaremos una matriz r(7,64) para almacenar los números de los equipos que queden a cada vuelta. Los contendientes son elegidos de forma aleatoria; cuando un club ha sido seleccionado, su número de club en la matriz es negado para que no sea seleccionado de nuevo. El número de goles depende del valor del equipo y de la ventaja de jugar en casa. Si se empata un partido, la repetición se efectúa en el campo del visitante - las semifinales y la final se juegan en campos neutrales. A divertirse!!.

PROGRAMA 20:COPA DEL REY

```
10 RANDOMIZE TIME
20 CLS : PEN 1
30 PRINT "COPA DEL REY" : PRINT "==========="
40 PRINT "Ganará su equipo la Copa del Rey?"
50 PRINT "Compruebelo viendo la competición"
60 PRINT : PRINT "Las semifinales y la final se
jugarán en campos neutrales" : PRINT
70 PRINT "Pulse ENTER para continuar"
80 IF INKEY(18) = -1 THEN 80
90 CLS : PEN 2
100 CLEAR
110 DIM t$(64),r(7,64),g(64)
120 FOR x = 1 TO 64
130 READ t$(x),g(x)
140 r(1,x) = x
150 NEXT x
160 FOR y = 1 TO 6
170 PEN 1 : PRINT "Vuelta";y
180 IF y = 5 THEN PRINT "Semi-"; : IF y>5 THEN
PRINT "Final"
190 PEN 2
200 n = 2^(7-y)
210 FOR x = 1 TO n/2
220 d1 = INT(1+RND*n)
230 IF r(y,d1)<0 THEN 220
240 r(y,d1) = -r(y,d1)
250 d2 = INT(1+RND*n)
```

```
260 IF r(y,d2)<0 THEN 250
270 r(y,d2) = -r(y,d2)
280 d1h = INT(RND*g(d1)*RND*(3+(y>4)))
290 d2a = INT(RND*g(d2)*RND*2)
300 PRINT t$(-r(y,d1));TAB(13);d1h:TAB(18);t$(-
    r(y,d2)); TAB(30);d2a
310 IF d1h<>d2a THEN 350
320 PEN 3 : PRINT "Repetir" : PEN 2
330 t = d1 : d1 = d2 : d2 = t
340 GOTO 290
350 IF d1h>d2a THEN r(y+1,x) = -r(y,d1)
360 IF d1h<d2a THEN r(y+1,x) = -r(y,d2)
370 PRINT
380 FOR z = 1 TO 500 : NEXT z
390 NEXT x
400 NEXT y
410 PEN 1
420 PRINT "CAMPEONES";t$(r(7,1)) : PRINT
430 PRINT "Pulse SPACE para jugar de nuevo"
440 IF INKEY(47) = -1 THEN 440
450 GOTO 10
460 DATA "R Madrid",6,"At Madrid",6,"Barcelona"
    ,6,"R Sociedad",6
470 DATA "At Bilbao",5,"Español",4,"Betis",3,
    "Sevilla",3
480 DATA "R Murcia",2,"Ràcing",2,"Elche",1,
    "Hèrcules",1
490 DATA "Màlaga",2,"Osasuna",2,"Spòrting",1,
    "Valencia",3
500 DATA "Valladolid",1,"Zaragoza",2,"A Madrile
    ño",1,"Barcelona A",2
510 DATA "Bilbao A",2,"Càdiz",1,"Calvo Sotelo"
    1,"Cartagena",2
520 DATA "Castellòn",3,"Castilla",3,"Celta",3,
    "Granada",2
530 DATA "Las Palmas",3,"Logroñès",2,"Lorca",2,
    "Mallorca",4
540 DATA "Oviedo",3,"Recreativo H",2,"Sabadell"
    ,3,"Salamanca",3
550 DATA "Tenerife",3,"Deportivo",2,"Rayo V",4,
    "Alcalà",3
560 DATA "Talavera",2,"Torrejòn",2,"Pegaso",3,
    "Manchego",3
570 DATA "Carabanchel",4,"Aranjuez",3,"Tarancòn
    ",3,"Daimiel",3
580 DATA "A Pinto",2,"Valdepeñas",3,"Alcobendas
    ",3,"S Fernando",2
590 DATA "Segoviana",3,"Getafe",4,"Ciempozuelos
    ",2,"Leganès",2
600 DATA "Alcorcòn",4,"Conquense",2,"Moscardò",
    3,"R Avila",3
```

```
DATA "Melilla",4,"Cebreros",3,"El Tiemblo",
2,"C D Hoyo",2
```

y ocasiones, sin embargo, en las que se requiere u
tema màs flexible de ordenar datos. Vamos a ve
erentes mètodos que se pueden titular en general "*estruc*
as dinàmicas de datos". Aunque muchas de estas tècnicas
den ser manejadas màs eficientemente por otros lenguajes
ejemplo el ALGOL o el Pascal, se pueden aplicar al BASI
Amstrad.

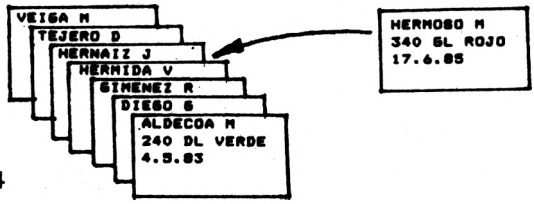
RUCTURAS DINAMICAS DE DATOS

mo ejemplo de lo que son las estructuras dinàmicas d
os, consideremos la siguiente tabla que contiene una
ta de pedidos de clientes de un vendedor de coches. Todos
registros estàn ordenados por el nombre del cliente com
la figura 7.3 que permite referirse
ilmente a la informaciòn
un cliente en particular. Supongamos que se ha de
dir el cliente HERMOSO a la lista.

BRE	INICIAL	MODELO	COLOR	FECHA PEDIDO
ECOA	M	240DL	Verde	04.05.85
GO	G	060GLE	Plata	30.06.85
ENEZ	R	360GLT	Rojo	25.05.85
MIDA	V	240GL	Blanco	25.05.85
NAIZ	J	360GLS	Azul	17.06.85
ERO	D	260GLE	Negro	19.06.85
GA	M	340DL	Rojo	17.06.85
RMOSO	M	340GL	Plata	30.06.85

. 7.3

ra que la tabla permanezca en orden, el nuevo registro
e ser colocado entre HERMIDA y HERNAIZ. Si tuviéramos el
hivo en tarjetas de oficina, cualquier nuevo registro se
ría añadir intercalando la nueva ficha en la posición
opiada. Del mismo modo, cuando el pedido del cliente ha
o servido, el registro correspondiente debe ser borrado
sistema retirando la ficha.



. 7.4

Nuestro problema consiste en simular este sistema donde podemos añadir o borrar registros y, reusando el espacio vacante, mantener un mínimo de memoria ocupada. La estructura que se debe usar para este tipo de aplicación se llama lista encadenada. Cuando se usan listas encadenadas para pequeñas cantidades de datos, la ventaja de la flexibilidad queda anulada por la desventaja de la complejidad. Sin embargo, es útil aprender las técnicas para poder manejar luego estructuras más complicadas.

LISTAS ENCADENADAS HACIA ADELANTE

A diferencia de las estructuras de datos estáticas, el orden de la lista encadenada se mantiene junto con los datos. Cada elemento en la estructura contiene dos tipos de información; el primero son los datos y el segundo es un apuntador que lo relacciona con otro elemento. Los datos pueden contener varios campos, ej. nombre, dirección, número de teléfono, etc; pero para simplificar nuestras explicaciones consideraremos que contienen un campo de datos simple.

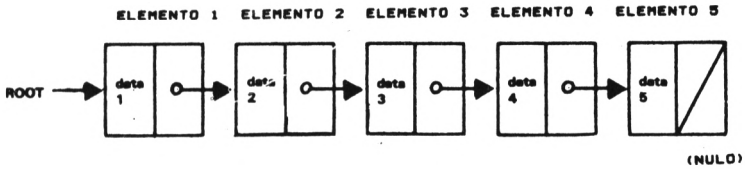


Fig.7.5

Ya que el elemento 1 está encadenado al elemento 2, el apuntador del primero tendrá el valor 2; de modo similar, los apuntadores 2, 3 y 4 tendrán los valores 3, 4 y 5 respectivamente. Como el elemento 5 es el último de la lista, su apuntador es 'nulo' y puede ser indicado mediante un valor negativo, digamos -1.

Otro método podría ser colocar como apuntador la dirección de la memoria donde se encuentra el siguiente elemento. Esta es una forma de 'direccionamiento indirecto' que no necesitamos revisar.

La estructura que se muestra en la figura 7.5 puede ser almacenada en el Amstrad mediante una matriz bidimensional, donde x$(e,1) contendrá los datos del elemento e y x$(e,2) contendrá un apuntador al siguiente elemento. Se puede usar una variable ROOT para apuntar al primer elemento ej.

```
ROOT = 1
x$(1,1) = "data 1"      x$(1,2) = "2"
x$(2,1) = "data 2"      x$(2,2) = "3"
x$(3,1) = "data 3"      x$(3,2) = "4"
x$(4,1) = "data 4"      x$(4,2) = "5"
x$(5,1) = "data 5"      x$(5,2) = "-1"
```

Para obtener los datos accedemos a todos los elementos de la lista; esto se hace empezando por el que apunta la variable ROOT y moviéndonos a lo largo de la estructura, accediendo a cada dato, hasta que legamos al apuntador nulo.

Un dato puede ser borrado fàcilmente de la lista encadenada cambieando el apuntador del elemento anterior para que apunte al siguiente. El apuntador del elemento que ha sido borrado debe ser cambiado a un valor que nos indique que està vacío - esto se hace para poder reusar el elemento de la matriz.

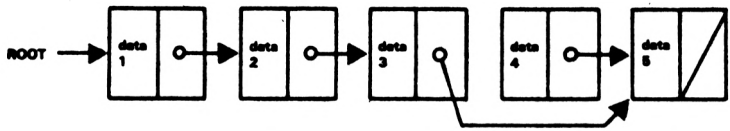


Fig 7.6

Si queremos insertar un elemento dentro de nuestra lista, hay dos etapas que cubrir; primero, debemos localizar el lugar de la lista donde debe ser insertado el nuevo elemento, y segundo, reconstruir los apuntadores para incluir los nuevos datos. La posición donde se coloca el nuevo elemento depende del usuario; puede ser colocado inmediatamente detràs del último o, si la lista es ordenada, en la posición correcta como en nuestro ejemplo:

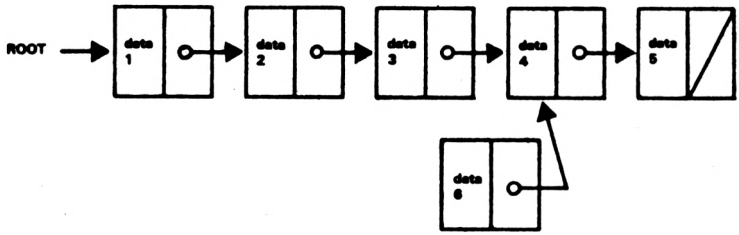


Fig.7.7

Primero debemos cambiar el apuntador del elemento 6 para que apunte al elemento 4 y después el apuntador del elemento 3 para que apunte al elemento 6:

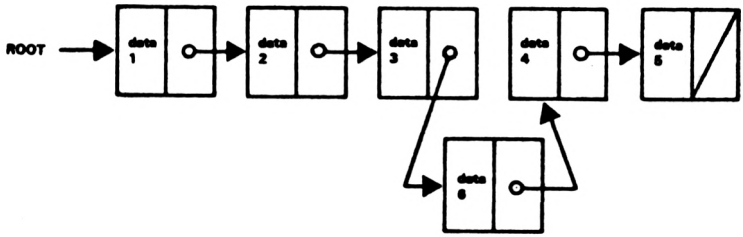


Fig.7.8

```
x$(6,1) = "dato 6"
x$(6,2) = "4"
x$(3,2) = "6"
```

Veamos ahora algunas rutinas para manejar las listas encadenadas; tenga en cuenta que se debe hacer un proceso diferente para tratar el principio y el final de la lista.

PROGRAMA 21:LISTA ENCADENADA

```
10 CLS
20 nulo = -1
30 n = 100 : DIM x$(n,2) : GOSUB 1000
40 CLS
50 LOCATE 10,10 : PRINT "LISTA ENCADENADA"
60 LOCATE 10,12 : PRINT "1...Iniciar"
70 LOCATE 10,13 : PRINT "2...Listar"
80 LOCATE 10,14 : PRINT "3...Borrar dato"
90 LOCATE 10,15 : PRINT "4...Añadir dato"
100 k$ = INKEY$ : IF k$ = "" THEN 100
110 k = ASC(k$)-48
120 IF k<1 OR k>4 THEN 100
130 ON k GOSUB 1000, 2000, 3000, 4000
140 GOTO 40

1000 root = nulo
1010 FOR x = 1 TO n
1020 x$(x,2) = ""
1030 NEXT x
1040 LOCATE 10,17 : PRINT "Iniciación
Completa"
1050 FOR j = 1 TO 500 : NEXT j
1060 RETURN
```

```
2000 CLS
2010 p = root
2020 WHILE p<>nulo
2030 PRINT x$(p,1)
2040 p = VAL(x$(p,2))
2050 WEND
2060 PRINT : PRINT "Fin de la lista"
2070 IF INKEY$ = "" THEN 2070
2080 RETURN

3000 CLS
3010 LOCATE 1,10
3020 PRINT "Introduzca nûmero dato a borrar";
INPUT d$
3030 IF d$ = "" THEN 3000
3040 IF root = nulo THEN 3120
3050 p = root
3060 IF x$(p,1) = d$ THEN root = VAL(x$(p,2)) :
x$(p,2) = "" : GOTO 3100
3070 pp = p : p = VAL(x$(p,2)) : IF p = nulo THEN
3120
3080 IF x$(p,1)<>d$ THEN 3070
3090 x$(pp,2) = x$(p,2) : x$(p,2) = ""
3100 PRINT "Datos borrados"
3110 GOTO 3130
3120 PRINT "Datos inexistenetes"
3130 FOR j = 1 TO 500 : NEXT j
3140 RETURN

4000 CLS
4010 LOCATE 1,10
4020 PRINT "Introduzca datos a insertar" :
INPUT d$
4030 FOR x = 1 TO n
4040 IF x$(x,2) = "" THEN p = x :  GOTO 4070
4050 NEXT x
4060 STOP
4070 x$(p,1) = d$
4080 IF root = nulo THEN x$(p,2) = STR$(root) :
root = p : GOTO 4150
4090 IF d$<x$(root,1) THEN x$(p,2) = STR$(root)
: root = p : GOTO 4150
4100 q = root
4110 qq = q : q = VAL(x$(q,2))
4120 IF q = nulo THEN x$(qq,2) = STR$(p) :
x$(p,2) = STR$(nulo) : GOTO 4150
4130 IF x$(p,1)>x$(q,1) THEN 4110
4140 x$(qq,2) = STR$(p) : x$(p,2) = STR$(q)
4150 PRINT "Insertado"
4160 FOR j = 1 TO 500 : NEXT j
4170 RETURN
```

LISTAS MAS AVANZADAS

Aunque la mayoría de las aplicaciones pueden manejarse con listas encadenadas, hay algunas mejoras que aumentan la potencia de las estructuras que acabamos de ver.

LISTA CIRCULAR

En una lista circular o anillo, el último elemento apunta al primero. La principal ventaja de este tipo de estructura es que se puede acceder a un elemento que precede al identificado sin tener que rearrancar desde ROOT.

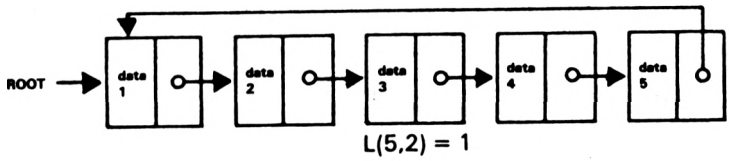


Fig.7.9

LISTAS CON DOBLE ENCADENAMIENTO

Se le puede añadir aún más potencia si incluimos un apuntador hacia atrás que encadene cada elemento a su predecesor.

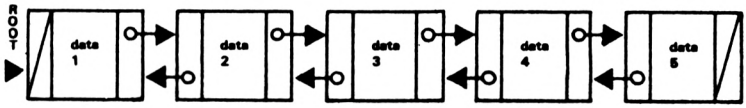


Fig.7.10

Esto permite la búsqueda en cualquier dirección. Usando más de un apuntador es posible ordenar la lista en más de un tipo de orden.

En pequeñas aplicaciones la ventaja de este tipo de estructura queda compensada por el aumento de la memoria requerida para los apuntadores adicionales.

PILAS Y COLAS

Hay dos estructuras lineales de suma utilidad en ordenadores, se llaman *pilas* y *colas*.

Una pila es un método para almacenar y recoger datos que se basa en el principio "último en llegar primero en salir". El almacenar información se llama *pushing* y recogerla es *poping*

En nuestro ejemplo, el elemento en lo alto de la estructura es apuntado por la variable TOP.

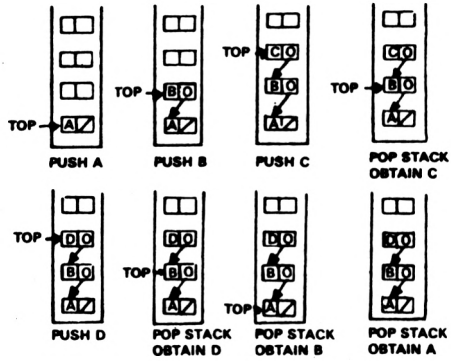


Fig.7.11

El sistema operativo usa este tipo de estructuras cuando usa las subrutinas. Cuando se ejecuta un GOSUB, los números de posición de la línea y la sentencia son colocados en la pila y se pasa control a la línea especificada por la sentencia. Cuando se ejecuta el RETURN, el proceso continúa desde la posición en que se encontraba gracias a los datos almacenados en la pila. Esta técnica permite anidar subrutinas en varios niveles. Si se ejecuta un RETURN sin su correspondiente sentencia GOSUB se encuentra con la pila vacía y ocurre un error.

Una estructura de colas se basa en el concepto de que el primero almacenado es el primero en salir. Se requieren dos apuntadores, HEAD y TAIL que apuntan al primero y al último elemento de la estructura.

Una estructura de colas es la que se usa en la memoria intermedia del teclado donde se almacenan las teclas que se han pulsado. Como es una cola, se accede antes a la primera tecla pulsada. Cualquier espacio que se quede libre queda disponible para otra tecla.

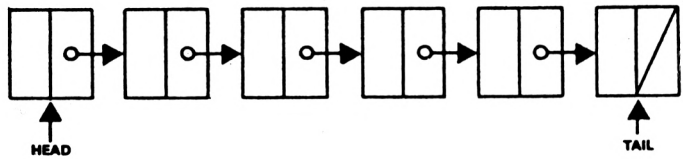


Fig.7.12

Cuando se recoge un dato, HEAD debe apuntar al siguiente

elemento de la estructura. ej. el elemento al que apunta el que se està obteniendo.

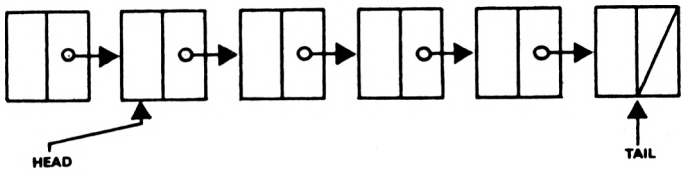


Fig.7.13

Cuando se añade un dato, se hace que apunten a èl TAIL y el último elemento.

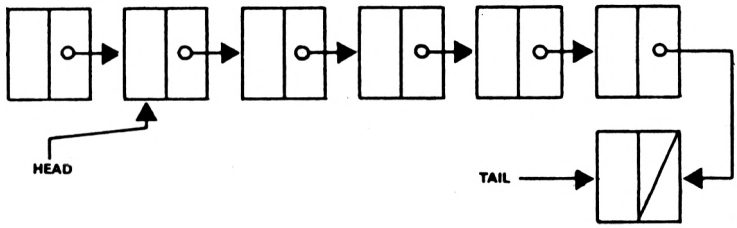


Fig.7.14

El problema es que la cola va llenando la memoria gradualmente a medida que se introducen y se recogen datos. Una solución es usar la lista circular de forma que, si se llena la memoria con la estructura, los datos puedan ser añadidos al principio. Si TAIL llega a HEAD, entonces nos hemos quedado sin espacio.

GRAFICOS

Aunque los conceptos de las listas encadenadas son muy útiles, su uso està limitado seriamente, ya que solo pueden funcionar en una dimensiòn, ya sea hacia adelante o hacia atràs. Para usar estas tècnicas con ideas pràcticas necesitamos poder manejar estructuras en màs de una dimensiòn; este tipo de estrucuras se conoce como *gràfico*.

Consideremos el siguiente ejemplo que representa las rutas aèreas de cierta compañia de aviaciòn.

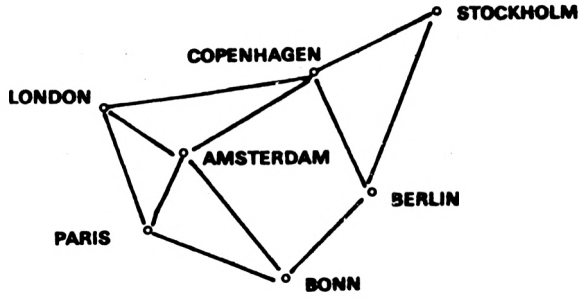


Fig.7.15

Esto se puede representar por la siguiente estructura.

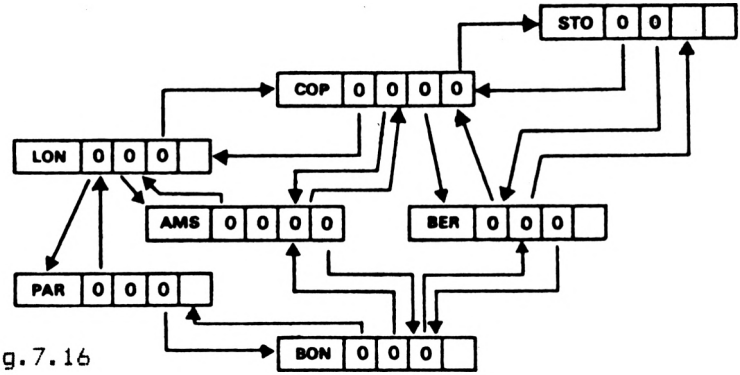


Fig.7.16

Se pueden incluir datos adicionales que, por ejemplo, pueden darnos el costo, las distancias, etc. entre dos ciudades.

Cuando referenciamos gràficos; los puntos de intersecciòn se llaman ramales los encadenamientos que unen los ramales se llaman bordes. Los bordes pueden ser directos o indirectos y pueden contener o no un valor (por ejemplo, en el nuestro, distancia, costo etc.) Los gràficos tienen multitud de aplicaciones - como ejemplo podemos intentar localizar el camino mas corto entre dos nodos, ej. refiriendonos a la figura 7.15, cual es el camino màs corto entre Lòndres y Berlìn?.

El algoritmo que vamos a usar localiza el camino màs corto entre un par de nodos y puede imprimir la ruta òptima.

El gràfico de n nodos se puede representar e el ordenador por una matriz de n x n elementos, donde el elemento x(s,d)

será la distancia desde la ciudad de partida s hasta la de destino d.

	1	2	3	4	5	6
1	0	8	∞	∞	7	∞
2	8	0	15	∞	10	10
3	∞	15	0	10	∞	5
4	∞	∞	10	0	10	7
5	7	10	∞	10	0	∞
6	∞	10	5	7	∞	0

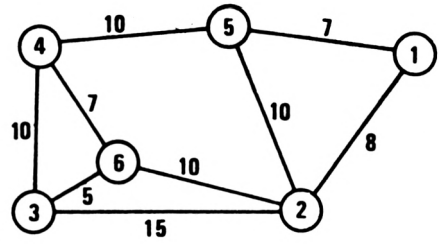


Fig.7.17

Cualquier par de ciudades que no tengan conexiòn directa toman un valor de distancia de infinito - en el Amstrad lo tenemos que hacer con un valor muy largo, como 1E+20.

El procedimiento para calcular la ruta màs corte es:
La distancia de cada ruta se considera separadamente con la de la misma ruta pero llendo a travès de otra ciudad(si es posible). ej. la distancia 1 - 6 se compara con las distancias 1-2-5, 1-3-6, 1-4-6 y 1-5-6, y el valor màs corto se coloca dentro de nuestra matriz en el elemento x(1,6).

Se conserva una matriz separada para recordar la ruta màs corta.

El programa 22 nos permite introducir un gràfico, ramal a borde mediante el nûmero de la ciudad de origen, un nûmero de ciudad de destino y una distancia. Cuando haya introducido toda la informaciòn, introduzca un grupo de datos invàlidos como 0,0,0. El programa empezarà a calcular las distancias màs cortas entre cada par de ciudades. Luego, especificando dos nûmeros de ciudad, el programa nos mostrarà la ruta màs corta y su distancia. Pulse la barra de espaciado para introducir otro par de claves para comenzar de nuevo.

PROGRAMA 22:RUTAS MAS CORTAS

```
10 CLS : CLEAR
20 ON ERROR GOTO 10
30 INPUT "Introduzca el nûmero màximo de
ciudades":n
40 DIM x(n,n),y(n,n)
50 CLS
60 FOR p = 1 TO n : FOR q = 1 TO n
70 IF p<>q THEN x(p,q) = 1E+20
80 NEXT q : NEXT p
90 PRINT "Introduzca ciudad s, ciudad d, distancia"
```

```
100 INPUT s,d,v
110 IF s = d OR s<1 OR s>n OR d<1 OR d>n THEN 130
120 x(s,d) = v : x(d,s) = v : y(d,s) = s : y(s,d)
= d : GOTO 100
130 print "Espere un momento por favor"
140 FOR p = 1 TO n
150 FOR q = 1 TO n
160 FOR r = 1 TO n
170 d = x(q,p) + x(p,r)
180 IF x(q,r) <= d THEN 200
190 x(q,r) = d : y(q,r) = y(q,p)
200 NEXT r : NEXT q : NEXT p
210 CLS
220 PRINT "Costo menor entre dos ciudades"
230 PRINT : PRINT "Introduzca dos ciudades"
240 INPUT s,d
250 IF s<1 OR s>n OR d<1 OR d>n THEN 240
260 IF x(s,d) = 1E+20 OR x(s,d) = 0 THEN
PRINT "No hay conexiòn" : GOTO 380
270 PRINT
280 PRINT "El coste desde";s; "a";d; "es";
x(s,d)
290 PRINT : PRINT "Via" : PRINT
300 IF y(s,d)<>d THEN 340
310 PRINT "Directo"
320 GOTO 380
330 PRINT "Pasando por";
340 IF y(s,d) = d THEN 380
350 PRINT y(s,d);
360 s = y(s,d)
370 GOTO 340
380 PRINT : PRINT : PRINT "Pulse SPACE para
continuar"
390 k$ = INKEY$
400 IF k$ = "s" OR k$ = "S" THEN 10
410 IF k$ = SPACE$(1) THEN 210
420 GOTO 390
```

ARBOLES

Hay un tipo especial de gràfico que se usa comunmente en ordenadores llamado àrbol. Un àrbol es un gràfico que no contiene nodos ni ciclos, esto es, solo hay una ruta para ir de un nodo a otro. Normalmente conocemos àrboles en los que los nodos salen directamente de un nodo especifico llamado ROOT(raiz); cada nodo, excepto la raiz, tiene un solo ramal entrando en èl.

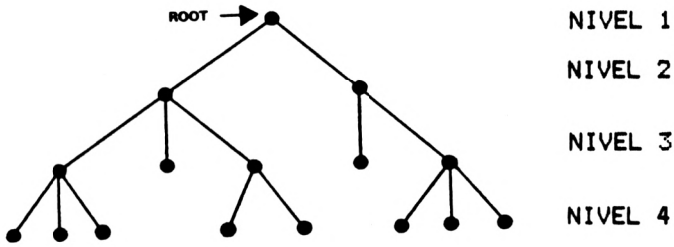


Fig.7.18

Los nodos que salen de la raíz forman el nivel 1. El nivel máximo lo da la longitud del árbol. Normalmente, al nodo que precede a otro se le llama nodo padre y al descendente, hijo.

Para no complicar demasiado las cosas vamos a restringirnos a árboles binarios: los que tienen un máximo de dos ramales saliendo de cada nodo.

Los árboles binarios pueden almacenarse en el Amstrad usando matrices de forma similar a las listas encadenadas, excepto que en el árbol se requieren tres campos; uno para los datos y.los otros dos para los apuntadores.

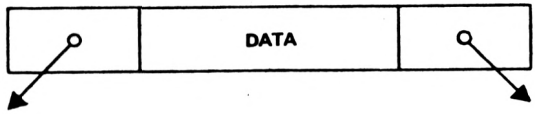


Fig.7.19

Como en los anteriores, necesitamos que se preparen los datos en algún orden lógico. Por ejemplo, veamos como se podrían almacenar una secuencia de marcas de coches en forma alfabética.

MERCEDES,FERRARI,PORSCHE,LOTUS,VOLVO,BMW,SAAB

Comenzamos con el primer nombre, MERCEDES, como raíz de nuestro árbol, y en este momento el nodo no tiene descendientes.

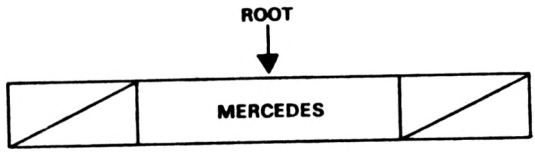


Fig.7.20

A continuación introducimos FERRARI y como precede a MERCEDES alfabéticamente lo conectaremos a la izquierda.

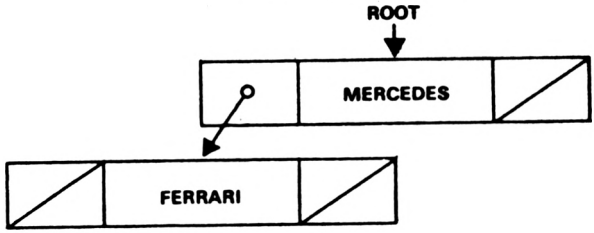


Fig.7.21

El siguiente es PORSCHE que sigue a mercedes y por lo tanto lo pondremos como descendiente a la derecha.

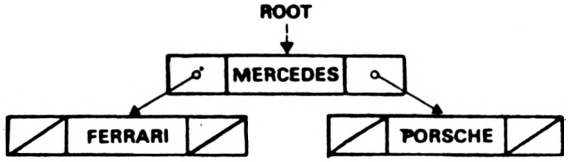


Fig.7.22

Después de analizar toda la información y colocarla alfabéticamente, el arbol se verá así.

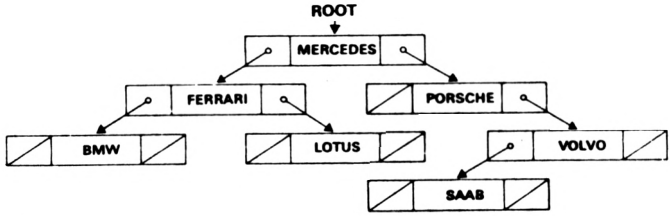


Fig.7.23

Con el BASIC del Amstrad podemos usar la matriz n$ para almacenar los datos, e 'i' y 'd' para almacenar los apuntadores de la izquierda y de la derecha. Otro datos como los detalles técnicos sobre los coches, pueden ser añadidos a la estructura almacenandolos en matrices y usando los subíndices correspondientes.

```
1  n$(1)="MERCEDES"   l(1)= 2      r(1)= 3
2  n$(2)="FERRARI"    l(2)= 6      r(2)= 4
3  n$(3)="PORSCHE"    l(3)=-1      r(3)= 5
4  n$(4)="LOTUS"      l(4)=-1      r(4)=-1
5  n$(5)="VOLVO"      l(5)= 7      r(5)=-1
6  n$(6)="BMW"        l(6)=-1      r(6)=-1
7  n$(7)="SAAB"       l(7)=-1      r(7)=-1
```

La siguiente tarea es conseguir seleccionar un nombre y pedir todos los datos asociados a èl. Una vez introducido el nombre, el programa busca por el àrbol empezando en ROOT y siguiendo a derecha o izquierda dependiendo del orden alfabètico del nombre introducido y del que aparece en cada nodo.Si se llega a un apuntador nulo antes de haber localizado el nombre, quiere decir que no existe en el àrbol. Este mètodo de bùsqueda es una forma de lo que se llama separaciòn binaria y puede ser muy ràpido, aùn con grandes cantidades de datos. Por ejemplo, mediante siete comparaciones podemos llegar al nivel 8 de un àrbol que significa que hemos buscado a travès de 255 elementos.

Para sacar todos los elementos del àrbol en el orden establecido, requerimos un mètodo sistemàtico para recorrer cada nodo siguiendo primero las ramas de su izquierda y despuès las de su derecha.

Si consideramos un nodo cualquiera del àrbol, suponiendo que sus apuntadores no son nulos, està encadenado a dos subàrboles. Del mismo modo, todos los nodos de cada uno de estos subàrboles està unido a otro subàrbol. De esta forma, cada subàrbol binario se puede abreviar a:

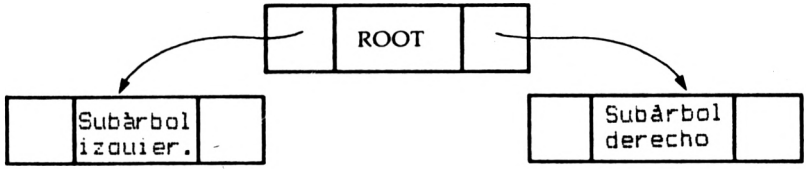


Fig.7.24

El desplazarse a cada uno de los componentes del àrbol en un orden fijo se llama *transversal*.

Hay tres transversales que se pueden recorrer.

1)Subàrbol Izq.	1) Subàrbol Izq.	1) Nodo
2)Nodo	ò 2) Subàrbol Der.	ò 2) Subàrbol Izq.
3)Subàrbol Der.	3) Nodo	3) Subàrbol Der.
(IND)	(IDN)	(NID)

Los transversales que usen posteriormente el mismo orden deben hacerse en cada subàrbol. Si usamos el orden IND, podemos obtener nuestros datos en el orden en que se ha preparado la estructura.

Consideremos el siguiente àrbol con el transversal IND:

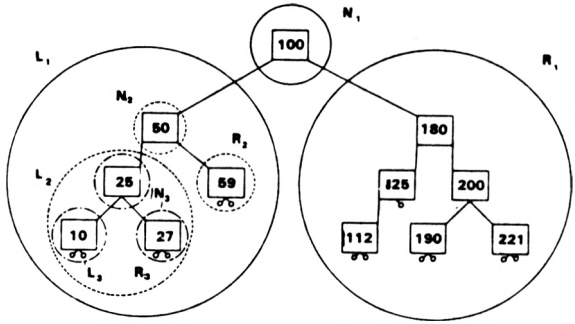


Fig.7.25

Nuestro orden serà L1,100,R1
Subàrbol L1 puede usar el transversal IND para tener L2,50,59
y L2 puede usar el IND para coger 10,25,27.
Por lo tanto nuestro orden serà 10,25,27,50,59,10,R1.
Cuando usemos el transversal IND en R1 habremos accedido a todo el àrbol por orden.

En el capítulo 8, cuando estudiemos los mètodos de clasificar datos, veremos como clasificar una secuencia de nùmeros creando un àrbol binario y usando luego el transversal IND. Puede echar una ojeada el capítulo 8 ahora que tiene el tema fresco en la mente.

PROGRAMACION HEURISTICA

Por el momento ya debe tener una idea de la flexibilidad que pueden proporcionar las estructuras dinàmicas de datos. Los àrboles son la forma màs comùn que se encuentra en los ordenadores y se usan en una gran variedad de programas. Muchos juegos se pueden estudiar mediante el uso de àrboles, donde las diferentes ramas proporcionan las opciones disponibles; por medio de algunos algoritmos optimizados, el ordenador puede examinar todos los estados posibles que pueden ocurrir para cualquier posible movimiento y los siguientes.

Una idea similar se usa cuando un programa aprende de las operaciones anteriores y recuerda què resultados ocurren para ciertos movimientos, por medio de un àrbol. El programa comienza con un àrbol consistente en un simple nodo, el ROOT, y en ese momento no sabe nada; con el tiempo el àrbol se irà expandiendo e irà haciendose "inteligente". Los programas que aprenden con la experiencia se llaman heurísticos.

En el programa 23 hacemos una demostraciòn de programaciòn heurìstica, que empieza con un nùmero pequeño de animales en su memoria. Le dirà que piense en un animal e intentarà adivinarlo mediante una serie de preguntas a las que tiene que responder 'si' o 'no'. Si el ordenador no es capaz de adivinar el animal, le harà preguntas cuyas respuestas le puedan servir en el futuro.

El programa construye un àrbol binario con los animales almacenados en los nodos inferiores y las preguntas en otros nodos. Mediante las preguntas y analizando las respuetas, el programa puede decidir en què sentido se debe desplazar por el àrbol. Cuando se encuentra un animal, el ordenador pregunta si es correcto; si no lo es, hace una pregunta para distinguir entre el animal del àrbol y el elegido por nosotros.

PROGRAMA 23:ANIMALES

```
10 GOSUB 380
20 CLS : PRINT "ANIMALES" : PRINT
30 PRINT "Piensa un animal" : PRINT
40 FOR j = 1 TO 2500 : NEXT j
50 PRINT "Responde S/N a las preguntas"
60 p = 1
70 IF l(p) = 0 AND r(p) = 0 THEN 140
80 FOR j = 1 TO 1000 : NEXT j
90 PRINT q$(p);" ? ";
100 k$ = INKEY$ : IF k$ = "" THEN 100 ELSE k$ =
LOWER$(k$)
110 IF k$ = "n" THEN PRINT "n" : p = r(p) :
GOTO 70
120 IF k$ = "s" THEN PRINT "s" : p = l(p) :
GOTO 70
130 GOTO 100
140 FOR j = 1 TO 1000 : NEXT j
150 PRINT "El animal es un";q$(p);" ? "
160 k$ = INKEY$ : IF k$ = "" THEN 160 ELSE k$ =
LOWER$(k$)
170 IF k$ = "s" THEN PRINT "Ya lo sabia yo!" :
GOTO 330
180 IF k$<>"n" THEN 160
190 nf = nf+2 : IF nf>n THEN PRINT "Aumente el
tamaño de las matrices" : STOP
200 PRINT
210 INPUT "Que animal habìa pensado?;q$(nf+1) :
q$(nf+1) = UPPER$(q$(nf+1))
220 q$(nf) = q$(p)
230 PRINT
240 PRINT "Introduzca la pregunta que distinga"
250 PRINT "un";q$(nf); "de un";q$(nf+1)
```

```
260 PRINT : INPUT q$(p) : q$(p) = UPPER$(q$(p))
270 PRINT : PRINT "La respuesta es S o N para"
280 PRINT "un";q$(nf);" ? "
290 k$ = INKEY$ : IF k$ = "" THEN 290 ELSE k$ =
LOWER$(k$)
300 IF k$ = "s" THEN l(p) = nf : r(p) = nf+1 :
GOTO 330
310 IF k$ = "n" THEN r(p) = nf : l(p) = nf+1 :
GOTO 330
320 GOTO 290
330 PRINT : PRINT "Quiere jugar de nuevo?"
340 k$ = INKEY$ : IF k$ = "" THEN 340 ELSE k$ =
LOWER$(k$)
350 IF k$ = "n" THEN END
360 IF k$ = "s" THEN GOTO 20
370 goto 340
380 REM iniciación
390 n = 150 : nf = 12
400 DIM q$(n),l(n),r(n)
410 FOR j = 1 TO 13
420 READ q$(j),l(j),r(j)
430 NEXT j
440 RETURN
450 DATA "ES UN AVE",3,2
460 DATA "ES UN MAMIFERO",4,5
470 DATA "VUELA",6,7
480 DATA "TIENE UNA TROMPA",8,9
490 DATA "a RANA",0,0
500 DATA " PETIRROJO",0,0
510 DATA " AVESTRUZ",0,0
520 DATA " ELEFANTE",0,0
530 DATA "HIBERNA",10,11
540 DATA "a ARDILLA",0,0
550 DATA "NADA",12,13
560 DATA "a FOCA",0,0
570 DATA " PERRO",0,0
```

Un pequeño problema: cuando apague su 464, olvidarà todo lo que ha aprendido sobre ranas, elefantes, etc. Puede rectificarlo reemplazando la estructura DATA por un fichero que se grabarà antes de apagarlo.

Ya que el tema de las estructuras de datos es largo y complicado, cualquiera que quiera estudiarlo màs en profundidad puede conseguir uno de los numerosos libros especializados que existen en el mercado. Yo le recomiendo "Successful Software for Small Computers" por G Bech, publicado por Sigma Technical Press. Contiene numerosos programas en BASIC para las diferentes estructuras.

CAPITULO OCHO
PROCESO DE DATOS

TECNICAS PARA CLASIFICAR

Una de las pequeñas ventajas que tiene un ordenador sobre el cerebro humano es su habilidad para acometer trabajos largos y laboriosos, pero realativamente simples, en un pequeño periodo de tiempo. Una de las tareas que vamos a examinar es el trabajo de clasificaciòn de datos numèricos y alfabèticos en cualquier orden - usualmente en orden alfabètico ascendente. Hay varios algoritmos desarrollados para hacer este tipo de operaciones aunque su eficiencia es normalmente inversamente proporcional a su complejidad.

Nos vamos a limitar a ver los cuatro algoritmos siguientes.

METODO	COMPLEJIDAD	EFICIENCIA	
		Pequeñas cantidades de datos	Grandes cantidades de datos
Burbuja	Muy sencillo	Excelente	Pobre
Inserciòn	Muy sencillo	Excelente	Pobre
Concha	Normal	Buena	Buena
Rèpido	Complejo	Buena	Muy Buena

Fig.8.1

Los datos se suelen almacenar en matrices, cuyo tamaño viene limitado por la disponibilidad de RAM. Una de las cosas que debemos considerar siempre cuando nos decidimos por un algoritmo, es el requerimiento de memoria; algunos mètodos requieren una gran cantidad de memoria ademàs de la matriz que contiene los datos.

Ahora explicaremos en detalle cada uno de los algoritmos mencionados anteriormente, con un programa que los ilustre.

El còdigo principal que damos a continuaciòn puede usarse para preparar datos aleatorios para llamar a cada programa de clasificaciòn. Para poder comparar los diferentes mètodos, el programa debe darnos tambièn el tiempo de proceso.

CODIGO PRINCIPAL

```
10 CLS : PRINT "TECNICAS DE CLASIFICACION"
20 PRINT : ZONE 8
30 INPUT "Introduzca el nûmero de datos":n
40 IF n<5 OR n>1000 THEN 10
50 DIM d(n),s(40,2)
60 FOR j = 1 TO n
70 d(j) = INT(RND*2500)
80 NEXT j
90 CLS : PRINT "TECNICAS DE CLASIFICACION"
100 PRINT : PRINT "INTRODUZCA" : PRINT
110 PRINT "1...de Burbuja"
120 PRINT "2...de Inserciòn"
130 PRINT "3...de Concha"
140 PRINT "4...Ràpido"
150 k$ = INKEY$ : IF k$ = "" THEN 150 ELSE k =
ASC(k$)-48
160 IF k<1 OR k>4 THEN 150
170 t1 = TIME
180 ON k GOSUB 1000,2000,3000,4000
190 t = ROUND ((TIME-t1)/300,2)
200 PRINT : PRINT "Datos clasificados"
210 PRINT : PRINT "Tiempo ";t;"segundos"
220 PRINT "Quiere ver los datos? (s/n/p)"
230 k$ = INKEY$ : IF k$ = "" THEN 230 ELSE k$ =
LOWER$(k$)
240 IF k$ = "n" THEN 60
250 IF k$ = "p" THEN END
260 IF k$<>"s" THEN 230
270 CLS PRINT "Datos clasificados" : PRINT
280 FOR j = 1 TO n
290 PRINT d(j),
300 NEXT j
310 FOR j = 1 TO 5000 : NEXT j
320 GOTO 60
```

CLASIFICADION DE BURBUJA

Al primero de los mètodos de clasificaciòn se le llama de 'burbuja'(bubble sort) porque los valores màs bajos parece que flotan hacia el extremo superior de la matriz, y los valores màs altos se hunden hacia el final. La matriz es recorrida continuamente y los datos contiguos son intercambiados cuando el primero es mayor que el segundo; si no se realizan intercambios, la lista està en orden.
Ejemplo:

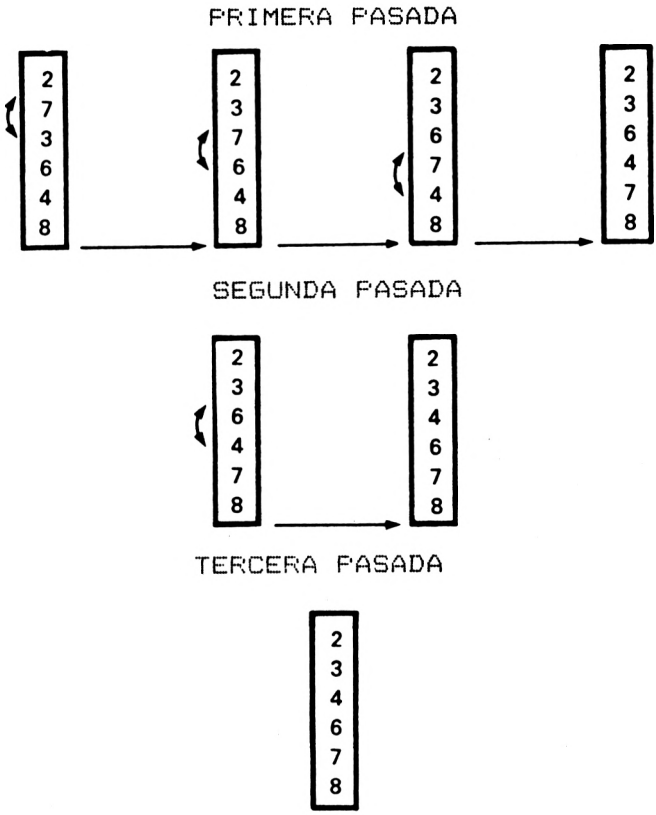


Fig.8.2

En la tercera pasada no se han hecho cambios, lo que indica que la lista ya está ordenada.

PROGRAMA 24:CLASIFICACION DE BURBUJA

```
1000 REM clasificaciòn de burbuja
1010 FOR j = 1 TO n-1
1020 f = 0
1030 FOR i = 1 TO n-j
1040 IF d(i)<= d(i+1) THEN 1070
1050 f = 1
1060 t = d(i+1) : d(i+1) = d(i) : d(i) = t
1070 NEXT i
1080 IF f = 0 THEN 1100
1090 NEXT j
1100 RETURN
```

CLASIFICACION POR INSERCION

El mètodo de clasificaciòn por inserciòn (insertion sort) permite ordenar una lista de datos en una simpla pasada de la matriz. Cuando se encuentra un dato fuera de orden, se busca su posiciòn correcta y se mueven los otros datos un lugar a lo largo de la matriz para posicionarlo correctamente.
Ejemplo:

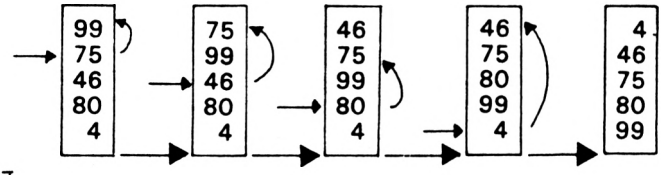


Fig.8.3

PROGRAMA 25:CLASIFICACION POR INSERCION

```
2000 REM clasificaciòn por inserciòn
2010 FOR j = 2 TO n
2020 t = d(j)
2030 FOR i = j-1 TO 1 STEP -1
2040 IF d(i) <= t THEN 2070
2050 d(i+1) = d(i)
2060 NEXT i
2070 d(i+1) = t
2080 NEXT j
2090 RETURN
```

CLASIFICACION DE CONCHA

Mientras que el sistema de burbuja compara solamente elementos adyacentes en la matriz, el de concha hace comparaciones iniciales entre elementos que estàn alejados, asumiendo que si dos elementos muy separados han de ser intercambiados, es màs eficiente hacerlo lo antes posible. La separaciòn entre elementos de datos se llama 'intèrvalo de clasificaciòn'. Inicialmente, el intèrvalo se inicia con el nùmero de elementos y en cada pasada se va reduciendo a la mitad hasta que la pasada final equivale a una de burbuja.

PROGRAMA 26:CLASIFICACION DE CONCHA

```
3000 REM clasificaciòn de concha
3010 si = n
3020 IF si<1 THEN 3120
```

```
3030 si = INT(si/2)
3040 f = 0
3050 FOR i = 1 TO n-si
3060 IF d(i) <= d(i+si) THEN 3090
3070 f = 1
3080 t = d(i+si) : d(i+si) = d(i) : d(i) = t
3090 NEXT i
3100 IF f = 0 THEN 3020
3110 GOTO 3040
3120 RETURN
```

CLASIFICACION RAPIDA

Y por último llegamos al algoritmo de clasificación rápida (quick sort). Es uno de los algoritmos de clasificación más complejos pero que suele dar los resultados más rápidos. La idea general es dividir los elementos en dos subgrupos (particionado) que, por orden, son particionados hasta que los subgrupos son suficientemente pequeños para que se pueda hacer una especie de clasificación de burbuja de modo eficiente. El programa debe recordar la posición de los elementos de datos no clasificados mientras se clasifican los subgrupos. Esto se hace almacenando la posicion inicial y final en la mtriz, de los elementos no clasificados. Estas posiciones se almacenan en una segunda matriz de forma que la última posición almacenada es la primera recuperada; a este tipo de estructura se le llama pila. La clasificación se completa cuando la pila se queda vacía.

Se usan dos apuntadores, uno indica el primer elemento de datos en la matriz y el otro el último. Uno de los dos apuntadores estará apuntando siempre al elemento que se encontraba inicialmente en la primera posición de la matriz; a este apuntador se le llama 'pivot'. Los dos elementos que están siendo apuntados se intercambian si el más cercano al principio es mayor que el otro. El segundo apuntador se mueve entonces una posición a lo largo de la matriz hacia el 'pivot' y se repite el proceso. Esto continúa hasta que los dos apuntadores se encuentran, en cuyo caso es particionado sobre el 'pivot' en dos subgrupos y se repite el proceso en cada uno de ellos. Las posiciones se almacenan en la matriz s(n,2). Si, debido a la gran cantidad de datos, se produce un desbordamiento de la pila, aparece un mensaje para aumentarla de tamaño.

Ejemplo:

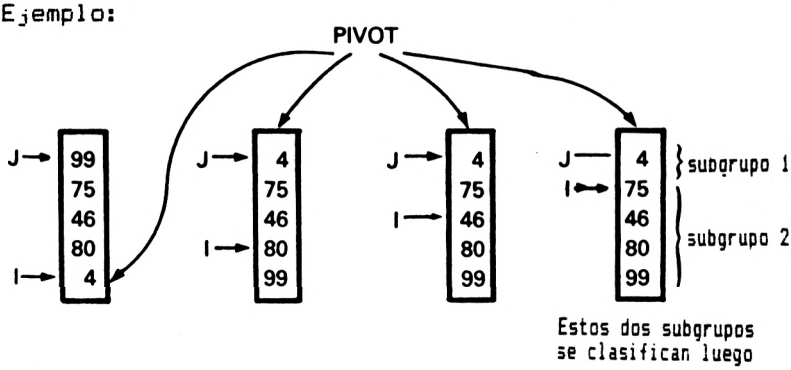


Fig.8.4

PROGRAMA 27:CLASIFICACION RAPIDA

```
4000 REM clasificación ràpida
4010 ps = 1
4020 s(1,1) = 1
4030 s(1,2) = n
4040 IF ps = 0 THEN 4200
4050 ii = s(ps,1) : jj = s(ps,2) : ps = ps-1
4060 p = 0 : i = ii : j = jj
4070 IF d(i) <= d(j) THEN 4100
4080 p = 1-p
4090 t = d(i) : d(i) = d(j) : d(j) = t
4100 IF p = 0 THEN i = i+1
4110 IF p = 1 THEN j = j-1
4120 IF i<j THEN 4070
4130 IF i >= jj THEN 4170
4140 ps = ps+1
4150 IF ps>40 THEN PRINT "PILA DESBORDADA" : STOP
4160 s(ps,1) = i+1 : s(ps,2) = jj
4170 jj = i-1
4180 IF jj>ii THEN 4060
4190 GOTO 4040
4200 RETURN
```

CLASIFICACION ALFABETICA

Un último programa para demostrar como clasificar una secuencia de cadenas alfabéticas. Se hace creando un àrbol binario y usando un transversal IND (ver capítulo 7). El transversal se hace empezando en el ROOT y dirigiendose continuamente hacia la izquierda, metiendo los nodos en la pila. Cuando se llega a un apuntador nulo, sacamos el nodo de la pila e imprimimos sus datos. Cuando lleguemos al final

de esa rama, volveremos hacia atrás al nodo padre tomándolo de la pila e imprimiendo su valor. Después, si hay rama derecha, repetimos el mismo proceso en el subárbol. Este proceso se repite para todo el árbol. Recuerde que, cuando comparamos cadenas, se considera menor la que está primera en orden alfabético (como en un diccionario).

PROGRAMA 28:CLASIFICACION ALFABETICA

```
10 CLS
20 nulo = -1 : m = 150
30 DIM n$(m),r(m),l(m),s(m)
40 GOSUB 250
50 CLS : PRINT "CLASIFICACION ALFABETICA" : PRINT
60 PRINT "Introduzca" : PRINT
70 PRINT "1...Iniciar"
80 PRINT "2...Añadir al árbol"
90 PRINT "3...Listar el árbol"
100 k$ = INKEY$ : IF k$ = "" THEN 100 ELSE k =
ASC(k$)-48
110 IF k<1 OR k>3 THEN 100
120 ON k GOSUB 200,400,600
130 GOTO 50

200 REM iniciar
210 PRINT : PRINT "Iniciar"
220 PRINT : PRINT "Está seguro?"
230 k$ = INKEY$ : IF k$ = "" THEN 230
240 IF k$ = "n" THEN 310
250 FOR j = 1 TO m
260 n$(j) = "" : r(j) = 0 : l(j) = 0
270 NEXT j
280 root = nulo : v = 0
290 PRINT : PRINT "Iniciación completa"
300 FOR j = 1 TO 1000 : NEXT j
310 RETURN

400 REM añadir
410 CLS : PRINT "Añadir al árbol" : PRINT
420 INPUT "Introduzca datos";d$
430 v = v+1 : IF v>m THEN PRINT "Aumentar
el tamaño de la matriz" : STOP
440 IF d$ = "" THEN 420
450 IF root = nulo THEN n$(v) = d$ : r(v) =
nulo : l(v) = nulo : root = v : GOTO 530
460 vv = root
470 IF n$(vv)<d$ THEN 500
480 IF l(vv) = nulo THEN l(vv) = v : GOTO 520
490 vv = l(vv) : GOTO 470
500 IF r(vv) = nulo THEN r(vv) = v : GOTO 520
510 vv = R(vv) : GOTO 470
```

```
520 n$(v) = d$ : r(v) = nulo : l(v) = nulo
530 PRINT : PRINT "Elemento añadido"
540 PRINT : PRINT "Salir s/n"
550 k$ = INKEY$ : IF k$ = "" THEN 550
560 IF k$<>"s" THEN 400
570 RETURN

600 REM listar
610 CLS : PRINT "Pulse SPACE para listar" : PRINT
620 IF INKEY$ <> SPACE$(1) THEN 620
630 IF root = nulo THEN 730
640 vv = root : p = 1
650 IF vv = nulo THEN 690
660 s(p) = vv : p = p+1
670 vv = l(vv)
680 GOTO 650
690 p = p-1 : vv = s(p)
700 PRINT n$(vv)
710 vv = r(vv)
720 IF p>0 THEN 650
730 PRINT : PRINT "Fin del listado"
740 IF INKEY$ = "" THEN 740
750 RETURN
```

FICHEROS EN CASSETTE

Ya debe haberse dado cuenta de que, cuando se desconecta el Amstrad, el programa almacenado y sus datos se pierden para siempre. Los programas se pueden almacenar fácilmente en un cinta de cassette normal para poder recuperarlos en cualquier momento. La información se puede almacenar de la misma manera comunicandose con el cauce número 9 - aunque esta técnica tiene severas limitaciones. El sistema estándar tiene una sola unidad de cassette, de forma que el manejo de la entrada y la salida del cassette ha de hacerse cambiando las cintas cuando se requiera. También, se pierde mucho tiempo si la sección de cinta que contiene la información requerida está muy lejos de su posición actual. Debido a que todos los datos se almacenan y acceden un elemento de cada vez, se dice que son ficheros sequenciales. Los datos entran y salen del cassette en bloques de 2K - estos son suficientemente pequeños para almacenarlos en la memoria, y se graban en la cinta con pequeños intervalos entre ellos, que permiten a la unidad de cassette parar y arrancar, ver figura 8.5. Cuando cargamos datos desde cinta, se graba un bloque en la memoria intermedia y se pueden leer los datos. Del mismo modo, cuando se almacenan datos en cinta se escriben primero en la memoria intermedia que es transferida a la cinta cuando se llena o se llega al final del fichero. Esta técnica evita tener que estar arrancando y parando la

por cada elemento de datos, y por lo tanto permite empaquetar los datos más unidos dentro de la cinta.

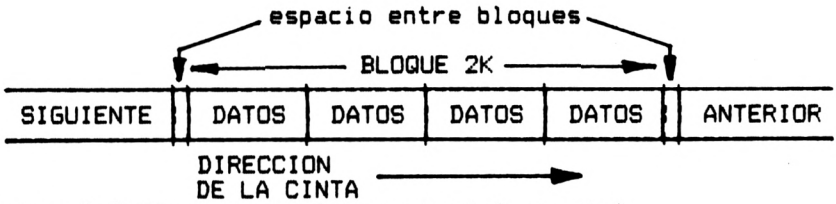


Figura 8.5:Almacenamiento secuencial en cinta

En las grandes instalaciones comerciales, la lentitud de los ficheros secuenciales en cinta magnètica hace que la mayoría de las modificaciones se hagan usando 'ficheros de transacciones'. Consideremos, por ejemplo, un banco con todas las cuentas corrientes de sus clientes almacenadas en cinta magnètica. Durante el dia se altera el valor de algunas cuentas a medida que se reciben cheques o dinero en efectivo. Sería poco eficiente, si no imposible, cambiar los valores de las cuentas a medida que van ocurriendo las alteraciones. En su lugar, se crea durante el dia un fichero de transacciones que recoje cada alteración de las cuentas en el momento en que ocurren. Al final del dia, se clasifica el fichero de transacciones usando una clave única, probamente el número de cuenta, hasta que los registros están en el mismo orden que los del archivo maestro. Entonces se comparan las dos cintas para ir modificando las cuentas y producir un nuevo fichero maestro.

La única diferencia entre comunicarse con el cassete y con otros dispositivos de E/S es que en el primero hay que abrir el cauce con el comando OPENIN y OPENOUT para entrada y salida respectivamente. En ambos casos se debe especificar el nombre del fichero - tenga en cuenta que si el primer carácter es un signo ! el cassete no lanzara los mensajes.

```
OPENIN"!CUENTAS"
```

A partir de entonces, se puede pasar la información desde y hacia el cassete usando cualquiera de las sentencias de E/S y especificando el cauce número 9.

Cuando se ha completado la operación de E/S, los ficheros deben ser cerrados usando CLOSEIN Y CLOSEOUT. Tenga en cuenta que es vital cerrar los ficheros de salida, ya que al hacerlo escribe el contenido de la memoria intermedia en la cinta y añade el marcador de fin de fichero (EOF). Este marcador puede ser verificado durante el INPUT para

comprobar que se han leido todos los datos. mediante la funciòn EOF.

El siguiente programa nos demuestra el uso de ficheros en cassette manteniendo un fichero de nombres y direcciones. El fichero se lee en memoria y despuès el usuario puede elegir entre añadir, borrar o aceder a los nombres y direcciones. Si se modifica el fichero, el usuario debe reescribirlo en la cinta. En la practica, se recomienda usar tres cintas; una para el programa, otra para el fichero maestro y la otra como copia de seguridad para mantener la segunda versiòn màs reciente.

El màximo nùmero de nombres en este programa es 75 pero puede ser incrementado con la variable de la línea 70.

PROGRAMA 29:AGENDA

```
10 CLS
20 ON BREAK GOSUB 200
30 WINDOW #1,1,40,25,25 : PEN #1,3 : INK 3,3,24
40 e$ = CHR$(164)+ " Mark R Harrison "
50 GOSUB 7000
60 nombre=1 : calle=2 : ciudad=3 : provincia=4 :
codpost=5 : telefono=6
70 m = 75 : DIM d$(m,6) : ptr = 1 : f$ = "DATOS"
80 CLS : PRINT "AGENDA" : PRINT
90 PRINT "Selecione:" : PRINT
100 PRINT "1...Cargar fichero"
110 PRINT "2...Salvar fichero"
120 PRINT "3...Añadir registro"
130 PRINT "4...Borrar registro"
140 PRINT "5...Ver registro"
150 PRINT "6...Ver nombres"
160 k$ = INKEY$ : IF k$ = "" THEN 160 ELSE K =
ASC(K$)-48
170 IF k<1 OR k>6 THEN e$ = "Opciòn Invàlida" :
GOSUB 7000 : GOTO 160
180 ON k GOSUB 1000,2000,3000,4000,5000,6000
190 GOTO 80
200 CLS : PRINT "BREAK"
210 PRINT "Quiere salvar los datos?"
220 k$ = INKEY$ : IF k$ = "" THEN 220 ELSE k$ =
LOWER$(k$)
230 IF k$ <> "n" THEN GOSUB 2000
240 END

1000 CLS : PRINT "Cargar fichero"
1010 PRINT
1020 OPENIN f$
1030 FOR ptr = 1 TO m
```

```
1040 FOR j = 1 TO 6
1050 INPUT #9,D$(ptr,j)
1060 NEXT j
1070 IF EOF THEN 1100
1080 NEXT ptr
1090 e$ = "Estructuras llenas" : GOSUB 7000
1100 CLOSEIN
1110 RETURN

2000 CLS : PRINT "Salvar fichero"
2010 PRINT : PRINT "Elija velocidad cassette"
2020 PRINT "0...Super seguro"
2030 PRINT "1...Super veloz"
2040 k$ = INKEY$ : IF k$ = "" THEN 2040
2050 IF k$ = "0" THEN SPEED WRITE 0 ELSE SPEED
WRITE 1
2060 PRINT
2070 OPENOUT f$
2080 FOR prt = 1 TO m
2090 IF d$(ptr,nombre) = "" THEN 2130
2100 FOR j = 1 TO 6
2110 WRITE #9,d$(prt,j)
2120 NEXT j
2130 NEXT ptr
2140 CLOSEOUT
2150 RETURN

3000 CLS : ptr = 0
3010 PRINT "Añadir registro" : PRINT
3020 PRINT "Introduzca:" : PRINT
3030 INPUT "NOMBRE";n$
3040 IF n$ = "" THEN e$ = "No ha especificado
nombre" : GOSUB 7000 : GOTO 3000
3050 FOR j = m TO 1 STEP -1
3060 IF d$(j,nombre) = "" THEN ptr = j
3070 IF d$(j,nombre) = n$ THEN e$ = "Nombre
duplicado" : GOSUB 7000 : GOTO 3190
3080 NEXT j
3090 IF ptr = 0 THEN e$ = "Fichero lleno" :
GOSUB 7000 : GOTO 3190
3100 d$(ptr,nombre) = n$
3110 INPUT "CALLE";d$(ptr,calle)
3120 INPUT "CIUDAD";d$(ptr,ciudad)
3130 INPUT "PROVINCIA";d$(ptr,provincia)
3140 INPUT "CODIGO POSTAL";d$(ptr,codpost)
3150 INPUT "TELEFONO";d$(ptr,telefono)
3160 PRINT " : PRINT "Correcto?"
3170 k$ = INKEY$ : IF k$ = "" THEN 3170 ELSE
k$ = LOWER$(k$)
3180 IF k$ = "n" THEN PRINT : GOTO 3110
3190 RETURN
```

```
4000 CLS : PRINT "Borrar registro" : PRINT
4010 INPUT "Introduzca nombre";n$ : PRINT
4020 FOR ptr = 1 TO m
4030 IF d$(ptr,nombre) = n$ THEN d$(ptr,nombre)
= "" : GOTO 4060
4040 NEXT ptr
4050 e$ = "No està en la lista" : GOSUB 7000 :
GOTO 4080
4060 PRINT : PRINT "Nombre borrado"
4070 FOR j = 1 TO 2500 : NEXT j
4080 RETURN

5000 CLS : PRINT "Ver registro" : PRINT
5010 INPUT "Introduzca nombre";n$ : PRINT
5020 FOR ptr = 1 TO m
5030 IF d$(ptr,nombre) = n$ THEN 5060
5040 NEXT ptr
5050 e$ = "No existe ese registro" :
GOSUB 7000 : GOTO 5120
5060 PRINT "CALLE",d$(ptr,calle)
5070 PRINT "CIUDAD",d$(ptr,ciudad)
5080 PRINT "PROVINCIA",d$(ptr,provincia)
5090 PRINT "CODIGO POSTAL",d$(ptr,codpost)
5100 PRINT "TELEFONO",d$(ptr,telefono)
5110 IF INKEY$ = "" THEN 5110
5120 RETURN

6000 CLS
6010 FOR ptr = 1 TO m
6020 IF d$(ptr,nombre)<> "" THEN WRITE
d$(ptr,nombre)
6030 NEXT ptr
6040 PRINT : PRINT "EOF"
6050 FOR j = 1 TO 2500 : NEXT j
6060 RETURN

7000 PRINT CHR$(7)
7010 PRINT #1,e$
7020 IF INKEY$ = "" THEN 7020
7030 PRINT #1,SPACE$(20)
7040 RETURN
```

Es muy importante asegurarse que las cabezas del cassette estèn siempre limpias. Cuando la cinta pasa sobre la cabeza, va dejando pequeños depòsitos; con el tiempo estos depòsitos son tantos que las grabaciones pierden calidad y el Amstrad no es capaz de descifrar las señales. Es recomendable tambièn usar cintas de buena calidad ya que las malas pierden la grabaciòn con el tiempo. Otro problema con las cintas de baja calidad es que se les desprende la capa

magnètica fàcilmente. Esto causa que la acumulaciòn de òxido sobre las cabezas se acelere y, peor aùn, puede que le haga perder alguna informaciòn valiosa.

FICHEROS EN DISCO

En el momento de escribir este libro se està anunciando que CP/M, un sistema operativo de discos, estarà disponible en breve para el Amstrad. El disco està empezando a ser el producto màs comùn usado por los microprocesadores para almacenar informaciòn. Sus ventajas incluyen gran seguridad y tiempos ràpidos de acceso. Por desgracia su precio los ha hecho objetos de lujo para los aficionados.

CP/M se trata en profundidad en **"CP/M - The Software Bus"** (publicado por Sigma Press) y es un compañero ideal tanto para los experimentados como para los novatos.

CAPITULO NUEVE

GRAFICOS EN EL AMSTRAD

Todos los lectores de este libro tienen probablemente un amigo que, cuando vea la màquina, le pedirà que la encienda y esperarà ver gràficos como los que se ven en las màquinas de los bares. Vamos a dar los primeros pasos para consegui es etipo de gràficos.

Hay dos mètodos para añadir informaciòn a la pantalla; estos son:

1) Limpiar la pantalla con el comando CLS y reconstruirla; esto se hace lînea por lînea, cada una de las cuales se debe almacenar usando gran cantidad de espacio. Otra desventaja es que el reconstruir la pantalla puede tomar mucho tiempo.

2) Añadir la secciòn de pantalla requerida reposicionando el cursor, ya sea con los caracteres de control del cursor o con el comando LOCATE, y despuès imprimir sobre la secciòn original.

Algunos lectores estaràn familiarizados con una tècnica llamada gràficos PEEK/POKE; donde la informaciòn se muestra en la pantalla con un POKE de la secciòn de memoria que contiene la informaciòn de la pantalla y haciendo PEEK de la misma secciòn se puede loacalizar la posiciòn de los caracteres. Desafortunadamente, debido a la disposiciòn de la memoria de pantalla del Amstrad, la tècnica de PEEK/POKE no es fàcil. Sin embargo, podemos solventar el problema usando el comando LOCATE y almacenando la posiciòn de los caracteres en variables.

CARACTERES DEFINIDOS POR EL USUARIO

Aunque el juego de caracteres estàndar tiene una gran variedad de sîmbolos, siempre hay algùn momento en que el usuario requiere un sîmbolo que no està disponible. Vamos a describir còmo definir nuestros propios caracteres y crear sîmbolos gràficos como los de matemàticas, animales, cohetes, explosiones, invasores del espacio, etc.

Todos los caracteres se definen en ocho octetos de memoria que representan la rejilla de 8 x 8 puntos que forman el caràcter. Cada bit se pone a '1' para representar el punto, que tomarà el valor del INK en uso, ver figura 9.1.

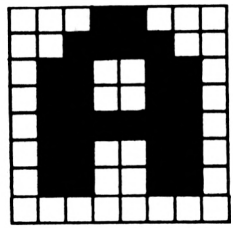
DECIMAL	BINARIO	CARACTER
24	0 0 0 1 1 0 0 0	
60	0 0 1 1 1 1 0 0	
102	0 1 1 0 0 1 1 0	
102	0 1 1 0 0 1 1 0	
126	0 1 1 1 1 1 1 0	
102	0 1 1 0 0 1 1 0	
102	0 1 1 0 0 1 1 0	
0	0 0 0 0 0 0 0 0	

Fig.9.1

Todos los datos de formaciòn de caracteres se origina en la ROM del Amstrad, en una secciòn llamada generador de caracteres. Para que el usuario pueda añadir nuevos caracteres, se deben colocar los datos en una determinada posiciòn de la RAM; sin embargo, si almacenaramos los 256 caracteres en RAM, requeririamos 256x8 o 2K octetos. De forma que, a tiempo de conexiòn, el Amstrad toma de la ROM los datos de formaciòn de los 240 primeros caracteres y los 16 restantes de una copia de la ROM localizada en RAM. Estos se pueden alterar con SYMBOL AFTER que especifica el còdigo ASCII del primer caràcter que se ha de colocar en RAM. ej. SYMBOL AFTER 128 - los caracteres del 128 al 255 se pueden redefinir. Tenga en cuenta que cuando ejecuta SYMBOL AFTER todos los caracteres definidos previamente vuelven a sus valores originales.

Los caracteres se redefinen con el comando SYMBOL que especifica el còdigo del caracter seguido de ocho valores que definen los datos de los puntos. Si se especifican menos de ocho valores, se asume cero para los ùltimos.

Como ejemplo vamos a definir CHR$(255) para que represente un amigable invasor del espacio.

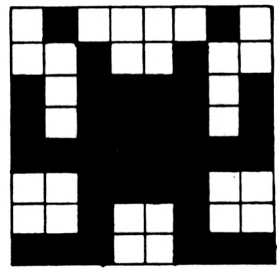
DECIMAL	BINARIO	CARACTER
66	0 1 0 0 0 0 1 0	
36	0 0 1 0 0 1 0 0	
189	1 0 1 1 1 1 0 1	
189	1 0 1 1 1 1 0 1	
255	1 1 1 1 1 1 1 1	
60	0 0 1 1 1 1 0 0	
36	0 0 1 0 0 1 0 0	
231	1 1 1 0 0 1 1 1	

Fig.9.2

De modo que lo ùnico que tenemos que decirle al Amstrad es:

```
SYMBOL 255,66,36,189,255,60,36,231
```

Ahora, cuando imprimamos el caracter CHR$(255) aparecerà nuestro invasor del espacio.

```
10 CLS
20 SYMBOL 255,66,36,189,255,60,36,231
30 INK 1,21,17 : INK 0,0
40 PEN 1 : PAPER 0 : BORDER 0
50 x = 20 : y = 12
60 x = x + INT(RND*3-1) : y = y + INT(RND*3-1)
70 IF x<1 OR x>40 OR y<1 OR y>25 THEN END
80 LOCATE x,y : PRINT CHR$(255)
90 FOR j = 1 TO 1000 : NEXT j
100 LOCATE x,y : PRINT SPACE$(1)
110 GOTO 60
```

Observe como, haciendo parpadear ligeramente al invasor se aumenta el efecto.

Se pueden conseguir caracteres màs grandes imprimiendo varios caracteres unos junto a otros.

Los efectos de animaciòn se pueden conseguir diseñando varios caracteres para crear series de formas, e imprimiendolos ràpidamente uno despuès del otro. El siguiente ejemplo nos muestra a un pequeño hombrecillo bailando:

```
10 CLS
20 v = INT(RND*4+248)
30 LOCATE 20,12
40 PRINT CHR$(v)
50 for k = 1 TO 200 : NEXT k
60 GOTO 20
```

IMPRESION TRANSPARENTE

Hay un còdigo de caràcter, que no hemos visto aùn, que es el CHR$(22) y se usa para poner o quitar la opciòn de impresiòn transparente. Se le añade un segundo caràcter que actùa de conmutador.

```
PRINT CHR$(22) + CHR$(0) quita la opciòn
PRINT CHR$(22) + CHR$(1) pone la opciòn
```

Normalmente, cuando se imprime un caràcter, el que estaba en la posiciòn donde se imprime, desaparece completamente. Si està activado el modo de escritura transparente, el caracter que se imprime se superpone encima del primer caràcter, y los puntos del fondo toman los colores que

estaban presentes anteriormente. Esto permite que se muestren varios colores en una posición de carácter. Para ilustrarlo, ejecute el siguiente ejemplo que usa el logotipo de Sigma Press.

```
10 CLS
20 PRINT CHR$(22) + CHR$(1)
30 LOCATE 25,1
40 INK 3,20 : PEN 3
50 PRINT STRING$(13,CHR$(143))
60 LOCATE 25,1
70 INK 2,0 : PEN 2
80 PRINT CHR$(190) + CHR$(8) +
STRING$(13,CHR$(210))
90 LOCATE 26,1
100 INK 1,15 : PEN 1
110 PRINT "Sigma Press"
120 PRINT CHR$(22) + CHR$(0)
```

CONTROL DE ALTA RESOLUCION

Inicialmente consideramos la pantalla del Amstrad como dividida, en MODE 1, en 25 líneas de 40 caracteres; después vimos como cada posición estaba dividida en una rejilla de 8x8 pequeños puntos que pueden estar iluminaods o no. Ahora vamos a ver como controlar los puntos individualmente. Como con las posiciones de caracteres, el número de puntos presentes depende del modo en que está la pantalla, como se puede ver en la figura 9.3.

MODO	No. de puntos
0	160x200
1	320x200
2	640x200

Figura 9.3

Un punto específico se define con dos números de coordenada (x,y) donde x es la posición horizontal e y la posición vertical donde cae el punto.

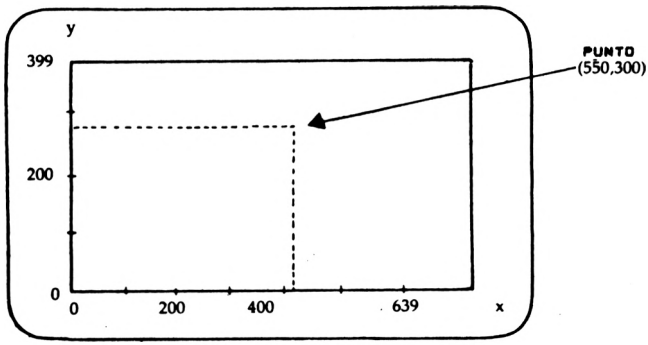


Figura 9.4:Distribución de puntos del Amstrad

Es importante recordar que las coordenadas de la figura 9.4 son las mismas para los tres modos; de forma que en algunos modos ciertas coordenadas se refieren al mismo punto. La relación del mapa vertical/horizontal está preparado para tener una relación de 1;(de modo que si dibujamos un círculo no aparecerá deformado). Es posible dar color a un punto en particular mediante la sentencia PLOT x,y o PLOT x,y,i donde i es el número de INK usado. La sentencia PLOTR es una extensión de la anterior, donde los valores x e y especificados son desplazamientos desde una posición previa.

De modo similar, DRAW dibuja una línea desde la posición actual hasta el nuevo punto (x,y) relativo al origen (0,0); DRAWR dibuja ls línea desde la posición actual hasta un nuevo punto especificado por desplazamientos desde una posición anterior. Podemos preguntar por la posición actual de PLOT con las funcione XPOS e YPOS.

Veamos algunos ejemplos:

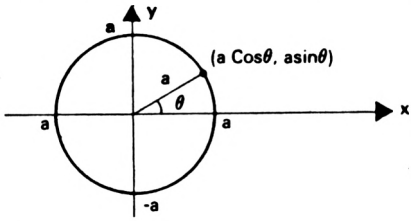
PROGRAMA 30:BURBUJAS

```
10 CLS : PRINT "BURBUJAS"
20 DEG : RANDOMIZE TIME
30 FOR j = 1 TO 25
40 x = INT(RND*640)
50 y = INT(RND*350)
60 r = INT(RND*75+25)
70 i = INT(RND*3+1)
80 PLOT x+r,y
```

```
90 FOR q = 0 TO 360 STEP 10
100 DRAW x+r*COS(q),y+r*SIN(q),i
110 NEXT q
120 NEXT j
```

Intente remplazar la linea 90 por:

```
90 FOR q =0 TO 7200 STEP 123
```



Esto ilustra cómo podemos dibujar circulos haciendo PLOT en los puntos dados por ***(a cos θ ,a sin θ)*** *donde 'a' es el radio*

Fig.9.5 Dibujo de un circulo

PROGRAMA 31:EPICICLOS

```
10 CLS : PRINT "EPICICLOS"
20 v = 4/3
30 move 400,200
40 FOR j = 0 TO 6*PI STEP PI/40
50 x = 300+100*COS(j*v)*COS(j)
60 y = 200+100*COS(j*v)*SIN(j)
70 DRAW x,y
80 NEXT j
```

Intente reemplazar el valor de v por otros como 2/3; en algunos casos puede ser necesario aumentar el valor de terminaciòn de control del bucle en la linea 40.

PROGRAMA 32:ONDAS DEL SENO/COSENO

```
10 CLS
20 p = 0
30 PRINT CHR$(23) + CHR$(p)
40 FOR j = 0 TO 10*PI STEP PI/25
50 x = j*50/PI
60 y1 = 100*SIN(j)
70 y2 = 100*COS(j)
80 PLOT x,150 : DRAWR 0,y1,2
90 PLOT x,150 : DRAWR 0,y2,1
100 NEXT j
```

Pruebe a cambiar a 3 el valor de p en la linea 20 - que pasarà cuando ejecute el programa?. Las àreas del coseno que cubren a las del seno aparecerán en rojo. Mientras que el

carácter de control CHR$(23) seguido de CHR$(0) pone la tinta normal, CHR$(23) seguido de CHR$(3) causa que la tinta tome un valor dependiente de la tinta que esté presente y la especificada. El nuevo valor de tinta lo toma ejecutando un O lógico de las dos tintas; esto es, inspecciona los dos valores de la tinta en forma binaria y obtiene el nuevo valor con los bits tomando los que estén puestos en cualquiera de las otras dos.

Ej. las dos tintas son 1 y 2 -	DECIMAL	BINARIO
	1	01
	2	10
	3	11

de modo que la tinta de las áreas que se sobreponen será 3.

Aquí tenemos un sumario de los valores que pueden seguir a CHR$(23).

CHR$(23)+		*El nuevo valor de INK toma los bits puestos solamente si:-*
CHR$(0)	Normal	
CHR$(1)	XOR	los bits correspondientes son diferentes.
CHR$(2)	AND	los bits correspondientes están puestos.
CHR$(3)	OR	cualquiera de los bits correspondientes está puesto.

Fig 9.6: MODOS DE INK

El programa 3 demuestra que es posible dibujar una función de dos variables en tres dimensiones con superficies sólida. Para ver como funciona el programa, considere la figura 9.7. La figura está hecha trazando líneas curvas en posiciones de la pantalla que son relativas a las posiciones ya trazadas. Si observa la figura 9.7, verá que vamos generando una 'rebanada' cada vez, desde el frente al fondo de la superficie 'sólida'.

Para producir el efecto de tres dimensiones, el programa considera la línea ST y todas sus paralelas, y calcula el valor de la función en los puntos en que las líneas rectas cortan a las curvas. El punto es marcado en la pantalla en una posición que tiene en cuenta el valor de la función y también la línea curva que se está tratando. Sin embrago, para dar la impresión de una superficie sólida, el punto debe ser marcado si no tiene una superficie en frente de él, esto es, el punto solo debe ser marcado si su coordenada y es mayor que cualquiera de las que hemos trazado anterior-

mente en la línea que estamos tratando.

Pruebe a remplazar la línea de la función con una que se le ocura, tenga en cuenta que la escala debe ser tal que no se salga de la pantalla.

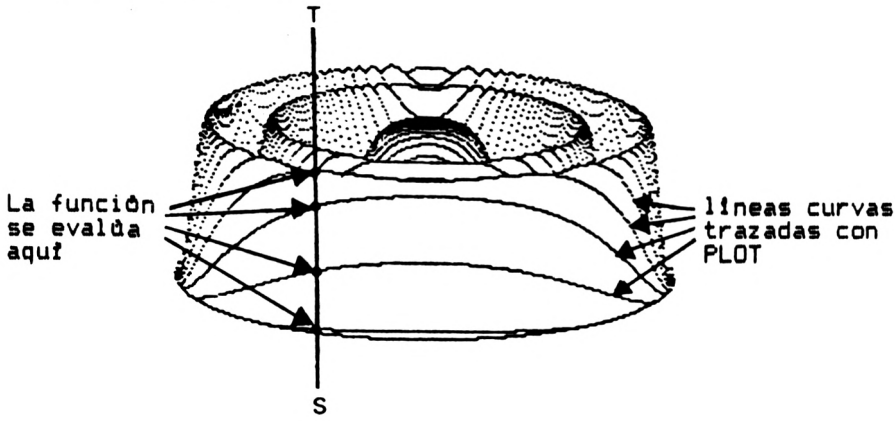


Figura 9.7:TRAZADO TRIDIMENSIONAL

PROGRAMA 33:TRAZADO TRIDIMENSIONAL

```
10 DEF FN a(j,k) = 50*(SIN((j*j+k*k)/1000))
20 CLS
30 q = 175
40 FOR x = 0 TO q STEP 2
50 y = SQR(q*q-x*x)
60 m = -1E+09
70 FOR z = -y TO y STEP 4
80 v = FN a(x,z)
90 p = v + z/3 + 100
100 IF p <= m THEN 120 ELSE m = p
110 PLOT 320+x,p : PLOT 320-x,p
120 NEXT z
130 NEXT x
```

Pruebe a experimentar con las dos variables de la función de la línea 10, pero recuerde que deben tener una escala adecuada para que no se salgan de la pantalla.

El programa 34 traza un histograma tridimensional de hasta 18 valores. El trazado principal lo efectúa la subrutina situada en la línea 20 y está escrita de forma que pueda ser incorporada fácilmente en sus propios programas. Es preciso preparar dos valores antes de llamarla; q es un número entre

1 y 18 que corresponde a la posición de la pantalla donde va a aparecer el bloque e y es la longitud vertical del bloque. Vertical del bloque. El efecto tridimensional se consigue coloreando las tres caras visibles en diferentes tonos de malva.

Presentamos una nueva sentencia, ORIGIN, que permite reposicionar los gràficos sobre las coordenadas de la pantalla. La coordenada (0,0) se mueve a la posición especificada. Se pueden especificar otros cuatro valores que preparan una nueva ventana de gràficos, como se ve a continuación:

```
ORIGIN x,y,izquierda,derecha,arriba,abajo
```

En nuestro programa, la coordenada y ha sido corrida verticalmente hacia arriba, para hacer más simple la inclusión de una línea de texto bajo el histograma.

El principal problema para el programa es elegir una escala adecuada para los datos. En este caso, inspecionando la línea 120, puede ver un bloque de 300 unidades para el elemento mayor de datos y los demás bloques se dibujan en proporción.

PROGRAMA 34:HISTOGRAMA TRIDIMENSIONAL

```
10 CLS
20 ORIGIN 0,25
30 DIM d(20)
40 READ v$
50 LOCATE 3,25 : PRINT v$
60 FOR j = 1 TO 18
70 READ v : IF v = -1 THEN 110
80 n = j : d(j) = v
90 IF v>m THEN m = v
100 NEXT j
110 FOR q = 1 TO n
120 y = d(q)/m*300
130 GOSUB 200
140 NEXT q
150 PEN 2
160 LOCATE 1,1 : END

200 FOR p = 0 TO y
210  PLOT 32*q,p
220 INK 2,8 : DRAWR 30,0,2
230 INK 3,7 : DRAWR 10,10,3
240 NEXT p
250 FOR p = 1 TO 10
260 PLOT 32*q+p,y+p
```

```
270 INK 1,4 : DRAWR 32,0,1
280 NEXT p
290 RETURN

500 DATA "E F M A M J J A S O N D VENTAS 1984"
510 DATA 100,77,155,77,23,56,24,100,78,145
520 DATA 166,6,-1
```

Podemos comprobar el color de una posición concreta mediante las funciones TEST y TESTR; la primera requiere coordenadas absolutas y la segunda desplazamientos relativos de la posición de PLOT actual. Esto se demuestra en el programa 35 en el que usted se debe encargar del control de una nave espacial con el ordenador de a bordo.- un Amstrad y sus teclas de control del cursor!!. El objetivo es navegar evitando las estrellas, su propia estela y los bordes del universo. El mejor tiempo conseguido se guarda y se muestra continuamente - buena suerte!!.

PROGRAMA 35:EXPLORADOR GALACTICO

```
10 MODE 1 : RANDOMIZE TIME
20 CLS : PRINT EXPLORADOR GALACTICO" : PRINT
30 INPUT "Introduzca nivel 1 - 99";sf
40 IF sf<1 OR sf>99 THEN 30
50 sf = sf/100
60 h = 0
70 INK 0,0 : INK 1,24 : INK 2,23 : INK 3,26
80 BORDER 0 : PAPER 0 : CLS
90 PRINT "Mejor tiempo";h,
100 PLOT 0,0
110 DRAW 639,0,1 : DRAW 639,380,1 :
DRAW 0,380,1 : DRAW 0,0,1
120 x = 300 : y = 190 : k = INT(RND*4)
130 kk = 3-k : t1 = TIME
140 k$ = INKEY$ : IF k$ <>"" THEN
kk = ASC(k$)-240
150 IF kk >= 0 AND k<4 THEN k = kk
160 IF k = 0 THEN y = y+2
170 IF k = 1 THEN y = y-2
180 IF k = 2 THEN x = x-2
190 IF k = 3 THEN x = x+2
200 IF TEST(x,y)<>0 THEN PRINT "CRASH!!!" :
GOTO 250
210 PLOT x,y,2
220 p = INT(RND*638+1) : q = INT(RND*378+1)
230 IF TEST(p,q) = 0 AND RND<sf THEN PLOT p,q,3
240 GOTO 140
250 t = ROUND((TIME-t1)/300,2)
260 IF t>h THEN h = t
270 IF INKEY$ <> "" THEN 270
```

```
280 IF INKEY$ = "" THEN 280
290 GOTO 80
```

Hasta ahora hemos considerado la pantalla con dos tipos de ventanas - caracteres y puntos gràficos. Sin embargo, hay veces que queremos tener las dos. Con el comando TAG podemos redirigir la salida de un cauce de modo que se pueda escribir en la posiciòn de PLOT actual, permitiendo combinar sìmbolos y texto con gràficos. Ya que la resoluciòn de la pantalla gràfica es mayor, el uso de TAG permite obtener un movimiento de la pantalla màs preciso.

TAG#0-dirige la parte superior izquierda del caracter que se saque al cauce 0, a la posiciòn de PLOT actual.

Este modo se puede inhabilitar con TAGOFF

TAGOFF #0

Cuando imprimimos caracteres en modo TAG, deben ir seguidos de un punto y coma para suprimir los caracteres de control, como el salto de lìnea o el retorno de carro. El siguiente ejemplo ilustra los efectos de animaciòn de precisiòn que se pueden realizar usando TAG.

PROGRAMA 36:ANIMACION

```
10 CLS
20 p = 0 : s = 10
30 TAG #0
40 PLOT 0,183 : DRAWR 639,0,1
50 RESTORE
60 FOR j = 1 TO 4
70 READ v
80 p = p+1 : IF p = 285 THEN 160
90 MOVE p,200
100 PRINT SPACE$(1);CHR$(v);
110 MOVE 600-p,200
120 PRINT SPACE$(1);CHR$(v);
130 FOR q = 1 TO s : NEXT q
140 NEXT j
150 GOTO 50
160 PLOT 315,211,3 : PRINT CHR$(228);
170 DATA 248,250,249,251
```

Se puede conseguir un excelente efecto de movimiento dibujamdo un objeto con puntos de varios colores y haciendo girar los colores. Esto se puede ver en el programa 37 que muestra una esfera en rotaciòn.

PROGRAMA 37:ROTACION

```
10 DEG
20 INK 0,1 : INK 1,1 : INK 2,1 : INK 3,1
30 PAPER 3 : BORDER 1 : CLS
40 i(0) = 13 : i(1) = 24 : i(2) = 0
50 b = 100
60 i = 0
70 ORIGIN 320,200
80 FOR a = -b TO b STEP 5
90 MOVE 0,b
100 FOR t = 0 TO 180 STEP 18
110 DRAW a*SIN(t),b*COS(t),i
120 NEXT t
130 i = (i+1)MOD 3
140 NEXT a
150 FOR a = 0 TO 2
160 INK 0,i(a)
170 INK 1,i((a+1)MOD 3)
180 INK 2,i((a+2)MOD 3)
190 FOR b = 0 TO 200 : NEXT b
200 NEXT a
210 GOTO 150
```

Con esto terminamos nuestra visiòn de los gràficos en el Amstrad. Le he dado una muestra del vasto rango de control de gràficos que se puede obtener.

CAPITULO DIEZ

SONIDO Y SINTESIS

Los usuarios del Amstrad pueden estar gullosos de las excelentes facilidades de sonido de su màquina. En nuestro siguiente tema trataremos de esta caracterìstica y veremos què tipo de sonidos se pueden producir. Cualquiera que haya aprendido alguna vez a tocar un instrumento musical sabrà que se requieren años de prâctica para conseguir perfeccionarse, y lo mismo es aplicable a tocar mûsica con el Amstrad. Una vez que haya leido este capìtulo, la ûnica forma de dominar esta tècnica es la de experimentar y aprender de los resultados obtenidos.

Antes de entrar en los dominios de la sìntesis del sonido en el Amstrad, hechemos una ojeada a lo que es el sonido.

CARACTERISTICAS Y ONDAS DE SONIDO

Todos nosotros tenemos gran cantidad de experiencias con el sonido durante nustra vida y sabemos que se pueden distinguir diferentes sonidos unos de otros - pero, que hace a un sonido diferente de otro?. Cada sonido està hecho por una transferencia de un lugar a otro de la energìa causada por la vibraciòn del aire. La diferencia entre los sonidos estàn afectadas por tres factores: I) tono II)calidad y III) volûmen.

Si la fuente de sonido oscila f veces por segundo, el sonido emitido se dice que tiene una frecuencia de f Hz (Hercios). Si la energìa viaja una distancia λ desde la fuente, en un segundo, el sonido producido tiene una longitud de onda de λ . Esto se puede representar por el diagrama de la figura 10.1.

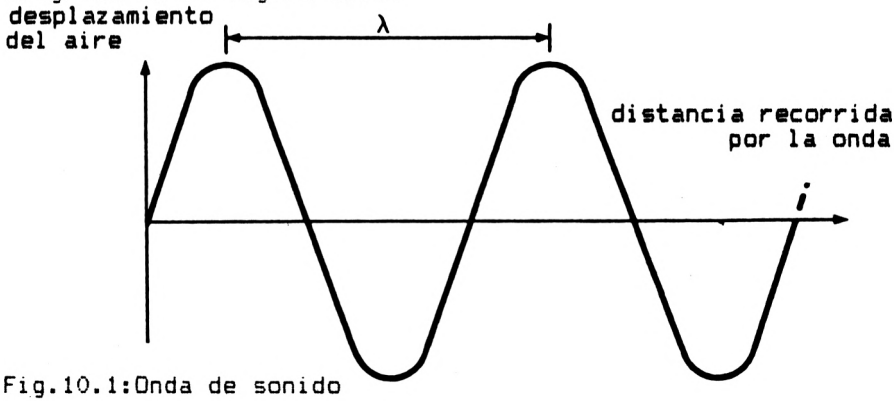


Fig.10.1:Onda de sonido

El tono de una nota depende de la frecuencia de la onda de sonido producida; una frecuencia alta produce un tono màs agudo y una frecuencia baja produce una nota grave. El intervalo musical entre dos notas es proporcional a la relaciòn de las frecuencias y no a la diferencia. Esto es, la diferencia de tono para el oido, es la misma si una nota crece de 250 Hz a 500 Hz que si lo hace de 1000 Hz a 2000 Hz. Si una nota tiene el doble de frecuencia que otra, se dice que las dos notas estàn separadas una octava.

Un mismo tono tocado por diferentes instrumentos musicales pueden ser diferenciados aunque las notas tocadas sean exactamente las mismas. Esto se debe a que la calidad de las formas de onda de cada instrumento son diferentes. Por ejemplo, cuando se hace sonar un diapasòn producirà una forma de onda como la de la figura 10.1, un instrumento musical producirà una forma de onda semejante a la de la figura 10.2.

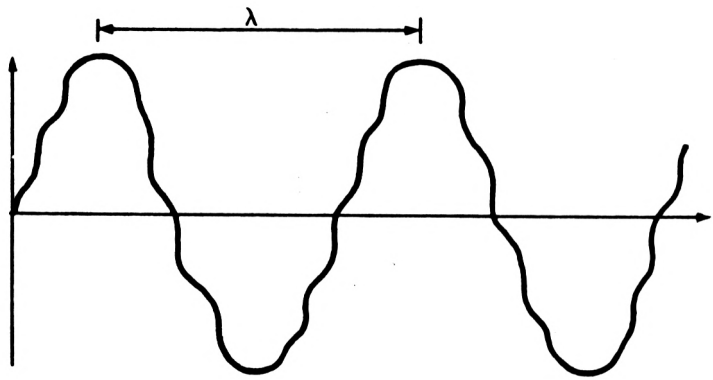


Fig.10.2:Forma de onda con "Ruido" añadido

EL SONIDO EN EL AMSTRAD

En el Amstrad se obtienen los sonidos con el comando SOUND que tiene una lista de parámetros; todos excepto los dos primeros, son opcionales. Cuando no se pone algùn parámetro se toma su valor por defecto.

```
SOUND ec,pt,d,v,ev,et,pr

     donde ec - estado del canal
           pt - periodo del tono
            d - duraciòn
            v - volùmen
           ev - envolvente de volùmen
           et - envolvente de tono
           pr - periodo del ruido
```

Ahora nos introduciremos en profundidad en la investigaciòn de cada uno de los paràmetros.

ESTADO DEL CANAL

El Amstrad tiene tres osciladores de sonido independientes (algunas veces son referidos como voces, en otros ordenadores) y por lo tanto en la sentencia SOUND, debemos especificar el canal de voz al que queremos encolar nuestro sonido. Podemos combinar juntas dos o màs voces , pero debemos tener cuidado de que las notas estèn sincronizadas correctamente. Nos referimos a los tres canales de sonido como A, B y C; cada uno de los cuales puede tener hasta cinco sonidos encolados. El paràmetro de estado de canal es un patròn de bits que dirige la salida del sonido a los canales requeridos como se muestra en la figura 10.3.

	7	6	5	4	3	2	1	0	bit
com.	FLUSH	HOLD	Acor- de C	Acor- de B	Acor- de A	manda a C	manda a B	manda a A	
	128	64	32	16	8	4	1	1	valor

Fig.10.3 Estado de Canal

En la siguiente secciòn, recuerde que los nùmeros binarios se sobreentienden que van precedidos por &X en el Amstrad.

Para emitir sonido por el canal A el estado de canal necesita tener puesto el primer bit, ej 00000001 (en binario) o 1 (decimal). De modo similar, la salida a los canales B o C necesitan tener el estado de canal puesto a 00000010 (binario),2 (decimal) o 00000100 (binario),4(decimal) respectivamente.

Para emitir sonido por varios canales, el estado de canal necesita ser puesto a la suma de los estados de los canales

correspondientes. Por lo tanto, para emitir sonido por los tres canales debemos tener puestos los tres bits, ej 00000111(binario) o 7(decimal).

Como puede ver en la figura 10.3 hay otros bits que pueden ponerse. Dos canales pueden ser sincronizados(acorde) de forma que sean accionados simultàneamente poniendo en cada estado de canal los bits de acorde del canal de voz opuesto.

El bit 6 del estado de canal se llama de retenciòn(HOLD) y cuando se pone se congelan las colas de sonido, ej. no se emiten màs sonidos del canal. Las colas pueden ser descongeladas con el comando RELEASE que permite que el sonido sea procesado en los canales especificados, para continuar. El valor pasado es un nûmero entre 1 y 7(3 bits) y especifica los siguientes canales:

Bit 0:Canal A
Bit 1:Canal B
Bit 2:Canal C

De forma similar, al bit 7 se le llama de evacuaciòn(FLUSH) y borra todas las colas en los canales en que se pone y activa la sentencia SOUND actual inmediatamente. El bit de evacuaciòn se puede usar tambièn para descongelar un canal de sonido, pero recuerde que se pierden las colas de sonido. Las colas de sonido se pierden tambièn cuando se imprime el carâcter de control CHR$(7).

PERIODO DE TONO

El periodo de una nota da las caracterìsticas del tono y es inversamente proporcional a la frecuencia que hemos visto al principio de este capìtulo. La relacciòn exacta entre los dos se calcula con la siguiente fòrmula:

frecuencia=125000/periodo

En la figura 10.4 se dan los valores que se usan en la escala musical. Recuerde que las notas que estân separadas una octava tienen una relaciòn de frecuencia de 2; como resultado, se puede calcular fâcilmente el periodo de una nota.

Poniendo el periodo a 0 se consigue el 'ruido blanco' donde la forma de onda es generada aleatoriamente; sin embargo se debe pasar un parâmetro de periodo de ruido(1 a 15) como sèptimo parâmetro de SOUND.

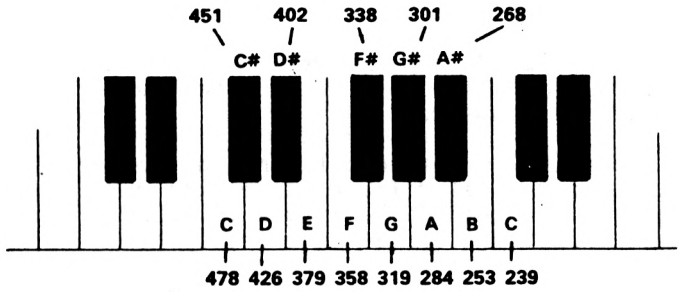


Fig.10.4 La escala musical-Valor de los periodos

Aunque nuestro control de sonido en este momento es muy elemental, el programa 38 nos muestra un instrumento operado con las teclas del 0 al 9.

PROGRAMA 38: INSTRUMENTO SIMPLE DE TECLADO

```
10 DIM n(10)
20 ON BREAK GOSOB 80
30 SPEED KEY 5,1
40 FOR j = 1 TO 10 : READ n(j) : NEXT j
50 k$ = INKEY$ : IF k$="" THEN 50 ELSE k =
ASC(k$)-47
60 IF k>0 AND k<11 THEN SOUND 1,n(k) ELSE 50
70 GOTO 50
80 SPEED KEY 10,3
90 END
100 DATA 284,478,451,426,402,379,358,338,319,301
```

DURACION

En este momento, la longitud de cada nota de sonido es de 1/5 segundos pero se puede alterar especificando el tercer paràmetro que da la longitud en centèsimas de segundo. El rango de valores es desde -32767 a 32767;los valores negativos y el cero tienen sus propios usos, que estudiaremos al tratar de la sìntesis de sonido.

VOLUMEN

La potencia del sonido viene dada por el cuarto paràmetro y debe estar en el rango de 0(sin sonido) al 7(màximo); los valores 8 a 15 se usan con la sìntesis de sonido.

SINTESIS DE SONIDO

Antes de explorar la capacidades de síntesis del Amstrad, hechemos una ojeada a lo que hace diferente un sintetizador de un instrumento musical.

En la figura 10.2 vimos como la onda de sonido emitida por un instrumento musical difiere de la nota pura, que representa la onda del seno de la figura 10.1. Es el modo en que son modificadas estas formas de onda, lo que da a un sonido su su carácter específico, y a esta modificación se le llama 'control de la envolvente'. La envolvente de volumen se divide en tres secciones: Ataque, Sostenido y Caida(ASC), como se ve en la figura 10.5.

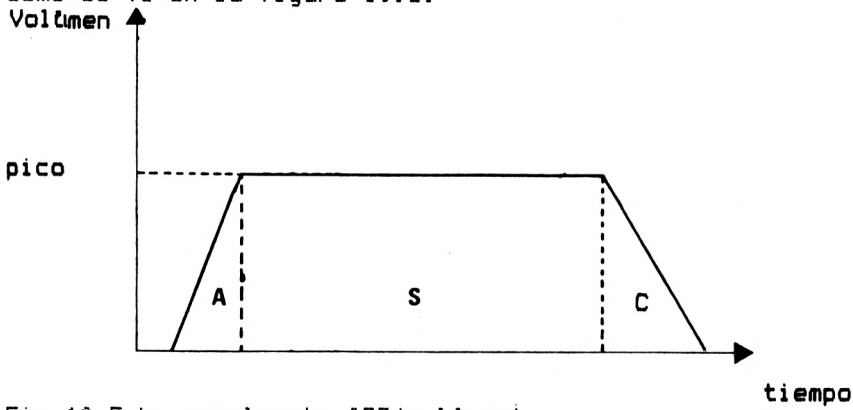


Fig.10.5 La envolvente ASC(volúmen)

ATAQUE:
Es el tiempo que tarda en llegar el pico de la onda. Esto se ve mejor examinando dos instrumentos musicales, el piano y el violín. El piano usa un mecanismo de martillos para crear el sonido; el martillo golpea la cuerda, la cuerda vibra y se crea el sonido. Ya que el sonido llega al pico de la onda virtualmente al mismo tiempo que se golpea la cuerda, el ataque es corto y potente. El violín, sin embargo, requiere un arco que roze las cuerdas y la vibración de èstas se construye a medida que el arco las cruza. Este, por lo tanto, tiene un ataque largo.

CAIDA:
Es el tiempo que tarda en ir desde lo alto de la onda hasta su final. Cuando, en el piano, se retira el martillo, la cuerda contiúa vibrando hasta que la energía creada por el golpe se disipa; por lo tanto, la caida es larga. Sin embargo el sonido de una flauta cesa casi inmediatamente que el músico deja de soplar; por lo tanto, tiene una caida corta.

SOSTENIDO:
Es el tiempo que la onda permanece en su pico.

Se pueden obtener cambios ràpidos de frecuencia, consiguiendo el efecto del vibrato, contolando la envolvente de tono, como en el ejemplo de la figura 10.6.

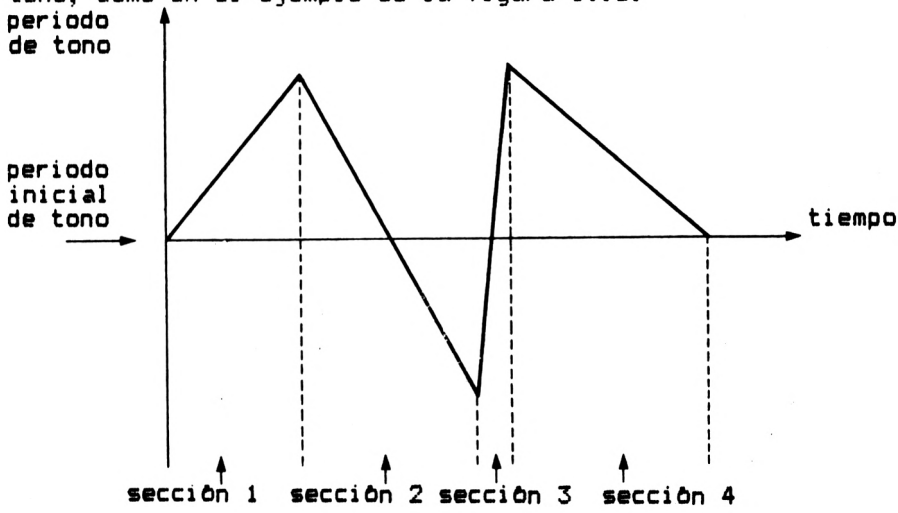


Figura 10.6:La envolvente de tono

De modo que para producir sonidos característicos, y no simples 'beeps', usamos la envolvente para controlar la forma de las ondas sintetizadas. Còmo podemos programar el control de la envolvente en el Amstrad?.

CONTROL DE ENVOLVENTE DE VOLUMEN

Para controlar la envolvente de volùmen, debemos definirla primero usando la sentencia ENV, y despuès hacerla referencia en la sentencia SOUND. En efecto, se pueden definir hasta 15 envolventes(etiquetadas del 1 al 15), cada una de las cuales puede tener hasta 5 secciones(periodos de ataque, sostenido y caida).

Para definir una envolvente usamos:

ENV nùmero,P1,Q1,R1,P2,Q2,R2,.....,P5,Q5,R5

donde Pi,Qi,Ri definen las caracterìsticas de cada secciòn

de la envolvente:

Pi=cuenta de pasos(0 a 127)
Qi=Tamaño de paso(-128 a 127)
Ri=Tiempo de pausa(0 a 251)

Consideremos una envolvente de solo tres secciones:

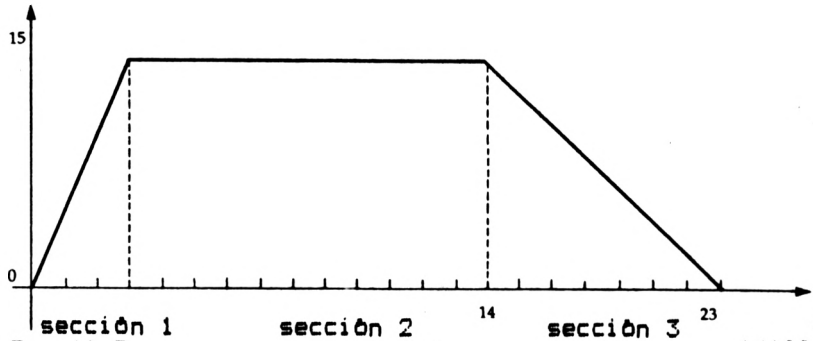


Fig.10.7

Cada sección la dividimos en pasos, cada uno múltiplo de 1/100 segundos. Por ejemplo, podemos dividir la figura 10.7 como sigue:

sección 1=3*1/100
sección 2=1*11/100
sección 3=3*3/100

pasos tiempos de pausa

por lo tanto, se requieren los siguientes parámetros:

P1=3,R1=1;P2=1,R2=11;P3=3,R3=3

El tamaño de paso se evalúa calculando el número de unidades de volúmen que se aumentan dentro de él.

Q1=15/3=5;Q2=0;Q3=-15/3=-5

Finalmente, necesitamos una etiqueta dentro del rango 1 a 15 para podernos referir a esta envolvente en nuetras sentencias SOUND; por ejemplo 1.

De modo que nuestra envolvente será:

ENV 1,3,5,1,1,0,11,3,-5,3

La etiqueta de ENV se pasa en el quinto parámetro de la sentencia SOUND. El volúmen inicial se controla por el cuarto parámetro pero, cuando se crea una envolvente, el rango es de 8(mínimo) a 15(máximo). El número de veces que se debe repetir la envolvente de volúmen es un valor negativo en el tercer parámetro.

En este punto probablemente se habrá dado cuente que, a menos que sea un experto en acústica, la única forma de conseguir un sonido exacto es mediate experimentos.

CONTROL DE LA ENVOLVENTE DE TONO

La envolvente de tono se controla de forma similar a la de volúmen; se pueden definir también cinco secciones usando el comand ENT:

ENT número,P1,Q1,R1,P2,Q2,R2,....,P5,Q5,R5

donde Pi,Qi,Ri son como en el anterior, pero sus rangos son:

Pi=Cuenta de pasos(0 a 239)
Qi=Tamaño del paso(-128 a 127)
R1=Tiempo de pausa(0 a 255)

La envolvente de tono se repetirá en toda la longitud de la envolvente de sonido, si se usa un número negativo(aunque se especifique como positivo en la sentencia SOUND).

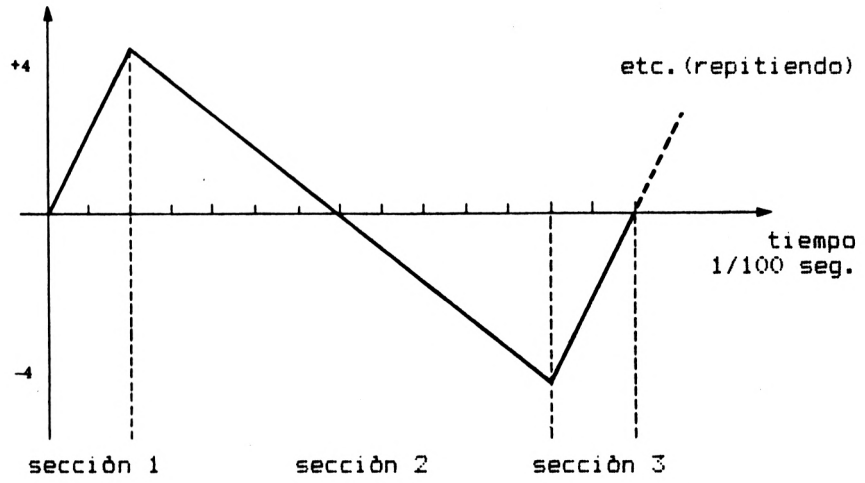


Fig.10.8

En la figura 10.8 definimos las secciones como:

```
sección 1=1*2/100
sección 2=2*5/100
sección 3=1*2/100
        pasos    tiempo de pausa
```

de modo que:

P1=1	Q1=4	R1=2
P2=2	Q2=-4	R2=5
P3=1	Q3=4	R3=2

Así que, para la envolvente número 1 debemos usar:

```
ENV -4,1,4,2,2,-4,5,1,4,2
```

El número de la envolvente de tono es el sexto parámetro de SOUND.

El siguiente programa está escrito con disculpas a los amantes de la música. Su idea es tocar la canción 'Ogonek' pero se ha escrito en forma de esqueleto para que usted pueda experimentar con el control de la envolvente. En la matriz n(39), se han preparado periodos de tono para 40 notas. Los subíndices de las notas requeridas se almacenan en sentencías DATA y, una vez leidas, se tocan por los canales A y B que se sincronizan poniendo los bits de acorde. El canal B se ha organizado para tocar dos octavas por debajo del A dividiendo el periodo del tono por cuatro.

PROGRAMA 39:OGONEK

```
10 GOSUB 200 : REM iniciación
20 ENV 1,3,20,1,1,0,20,1,-1,1
30 ENT -2,1,1,20,2,-1,50,1,1,20
40 READ v : IF v = -1 THEN END
50 SOUND 16+1,n(v),-1,15,1,2
60 SOUND 8+2,n(v)/4,-1,15,1,2
70 GOTO 40

200 DIM n(39)
210 FOR k = 0 TO 3
220 RESTORE
230 FOR j = 0 TO 9
240 READ v
250 n(k*10+j) = v/(k+1)
260 NEXT j
270 NEXT k
280 RETURN
290 DATA 478,451,426,402,379,358,338,319,301,284
```

```
1000 REM tonada de ejemplo
1005 DATA 0,4,9,12,16,4,14,2,6,4,14,4,8,1,14,4,
     11,12,14,4,11,12,9
1010 DATA 0,4,9,12,9,4,0,4,16,17,19,8,17,16,19,
     8,11,17,17,9,14,9,17,9,17,19,21
1015 DATA 9,19,21,16,4,8,11,16,14,11,8,0,12,24,
     24,24,12,23,21,24,12,16,23,17
1020 DATA 9,14,9,17,9,17,9,1,7,19,21,9,19,21,16,
     8,12,16,8,12,16,0,8,11,16
1025 DATA 0,4,9,12,16,4,14,2,6,4,14,4,8,1,14,4,
     11,12,14,4,11,12,9
1030 DATA 12,16,12,9,21
1035 DATA -1
```

SQ

Con la funciòn SQ podemos averiguar el estado de uno de los canales de sonido. El argumento es 1(canal A), 2(canal B) o 4(canal C) y devuelve un valor compuesto por los siguientes bits que especifican el estado del canal:

7	6	5 4 3	2 1 0
Puesto activo	Puesto retenido	Estado del acorde (como en Fig 10.3)	num. entradas libres en la cola(0-4)
128	64	32 16 8	4 2 1

Fig.10.9:Funciòn de estado de canal

INTERRUPCIONES DE SONIDO

Habrà algunas ocasiones en las que quiera un sonido de fondo para un programa. Intente colocar algunos comandos de sonido dentro de su programa, aunque las colas de sonido tienen prioridad es bastante difícil asegurar un sonido continuo. Podemos preparar interrupciones de sonido para que se ejecute una subrutina de sonido cuando exista un espacio dentro de la cola de sonido. Como en otras subrutinas, al terminar, devuelve control a la posiciòn en que se tomò la interrupciòn. Controlando la salida de sonido dentro de esta subrutina conseguimos que la facilidad de sonido funcione como una tarea secundaria del programa principal. Es posible disponer de interrupciones separadas con:

```
ON SQ (x) GOSUB línea
```

donde x toma los valores 1,2 o 3 dependiendo del canal.

Sin embargo, cuando ocurren interrupciones de sonido, o se ejecutan las sentencia SOUND o SQ, se inhabilitan estas interrupciones; de forma que, si deben continuar las interrupciones de sonido, deben rehabilitarse en las subrutinas de sonido.

Nuestro último programa nos demustra el uso de las interrupciones de sonido. El programa muestra una pantalla simple del tipo 'arcade', en la que un grupo de audaces alienígenas recorren continuamente la pantalla y se aproximan a la superficie de la tierra. Usted debe controlar el laser, usando la teclas de control del cursor a derecha e izquierda y [COPY] para disparar. El objetivo es destruir a todos los alienígenas antes de que aterricen pero cuidado, si dispara y falla, se unirá un nuevo alienígena a los ya presentes.

Se ha preparado una interrupción de sonido para producir el rugido de los alienígenas - un buen incentivo para destruirlos rápidamente. Tenga en cuenta que la interrupción es detenida siempre que se llama la subrutina de sonido. Cuando un alienígena es destruido se emite otro tipo de sonido. Ya que queremos que este sonido se produzca inmediatamente, en vez de ponerlo al final de la cola de sonido, eliminamos la cola de sonido del canal poniendo el séptimo bit de estado de canal. Como antes, la interrupción de sonido se detiene.

PROGRAMA 40:ATAQUE ALIENIGENA

```
10 ENV 1,1,15,5,1,0,20,5,-3,2
20 ENT 1,10,1,1,40,-1,1,60,1,5
30 SPEED KEY 1,1
40 INK 0,0 : INK 1,24 : INK 2,15 : INK 3,13,22
50 PAPER 0 : BORDER 0
60 CLS
70 z$ = CHR$(95) + CHR$(95) + CHR$(244) +
   CHR$(95) + CHR$(95)
80 p = 1 : z = 20 : pp = 800
90 t$ = ""
100 FOR k = 1 TO 20 : t$ = t$ + SPACE$(1) +
    CHR$(225) : NEXT k
110 PLOT 0,16 : DRAWR 639,0,2
120 ON BREAK GOSUB 800
130 EVERY 500 GOSUB 400
140 ON SQ(1) GOSUB 1000
150 FOR j = 1 TO 40
160 PEN 3
170 LOCATE 1,p : PRINT MID$(t$,j,40);MID$(t$,1,j-
    1)
180 PEN 2
```

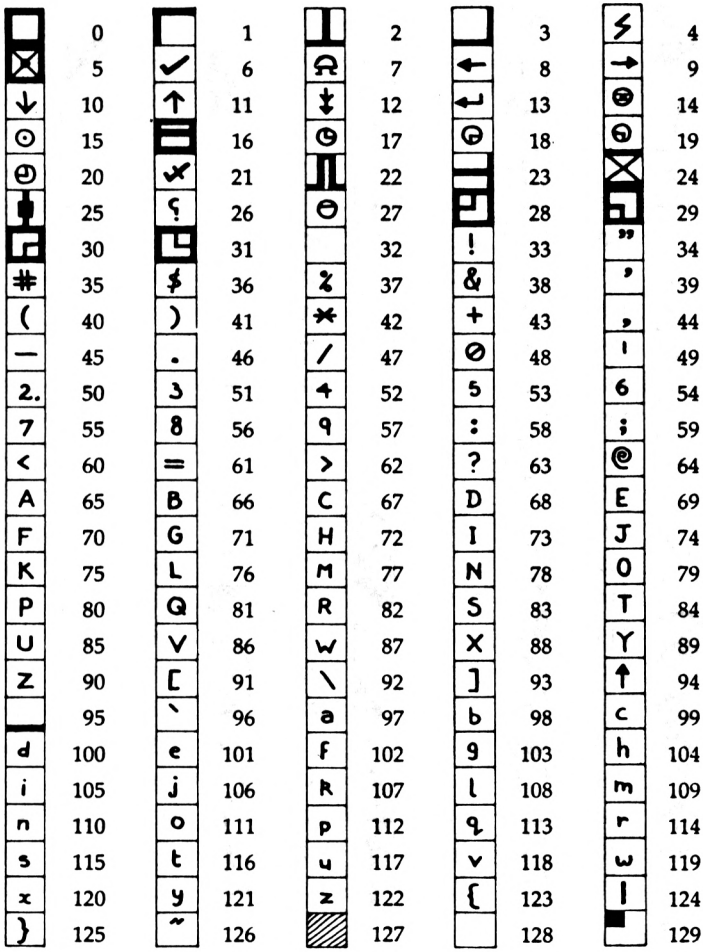
```
190 LOCATE z,24 : PRINT z$
200 k$ = INKEY$ : IF k$ = "" THEN 240 ELSE k =
    ASC(k$)
210 IF INKEY(8)<>-1 AND z>3 THEN z = z-2
220 IF INKEY(1)<>-1 AND z<36 THEN z = z+2
230 IF INKEY(9)<>-1 THEN GOSUB 600
240 NEXT j
250 GOSUB 150

400 p = p+1
410 pp = pp*0.9
420 IF p<24 THEN 460
430 LOCATE 12,25
440 PRINT "Invasiòn completa";
450 GOTO 800
460 LOCATE 1,P-1 : PRINT SPACE$(40)
470 RETURN

600 v = j+z+1 : IF v>40 THEN v = v-40
610 q = ASC(MID$(t$,v,v))
620 MOVE 16*z+22,25 : DRAWR 0,380,1
630 FOR k = 1 TO 50 : NEXT k
640 DRAWR 0,-380,0
650 IF q<>225 THEN 740
660 LOCATE z+2,p : PRINT CHR$(238)
670 SOUND 128+1,1000,40,15
680 ON SQ(1) GOSUB 1000
690 MID$(t$,v,v) = SPACE$(1)
700 IF t$<>SPACE$(40) THEN 750
710 LOCATE 12,25
720 PRINT "Invasiòn repelida";
730 GOTO 800
740 MID$(t$,v,v) = CHR$(225)
750 RETURN

800 PRINT CHR$(7)
810 INK 1,24 : PEN 1
820 SPEED KEY 10,3
830 LOCATE 1,1
840 END

1000 SOUND 1,pp,-5,15,1,1
1010 ON SQ(1) GOSUB 1000
1020 RETURN
```

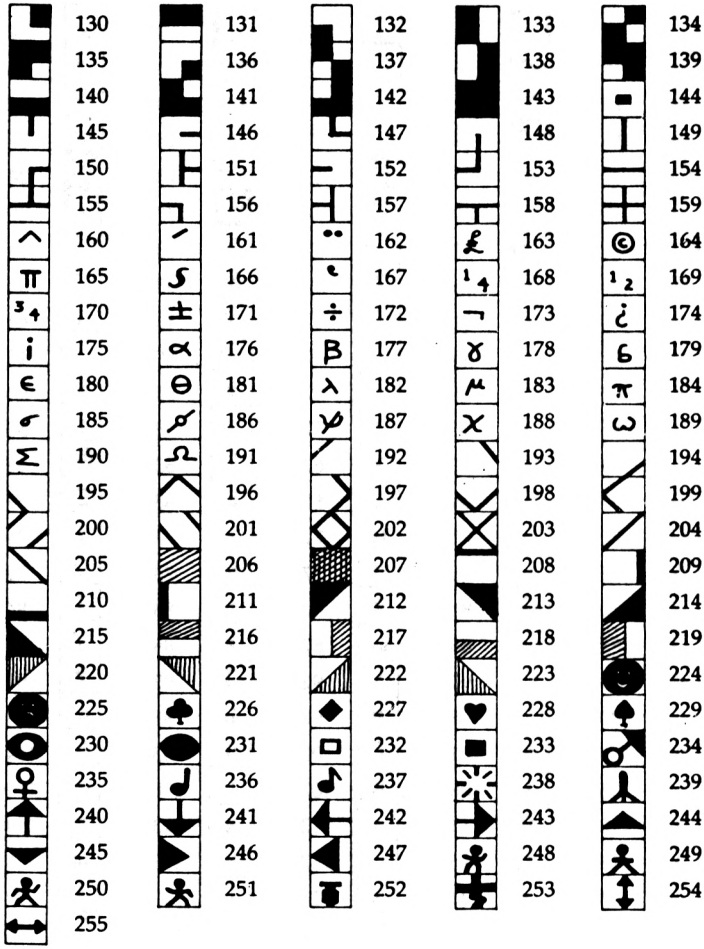
Con esto damos por terminado el capítulo y el libro. Espero que, a medida que ha ido trabajando a través de los diez capítulos, haya incrementado su repertorio de programaciòn y mejorado su tècnica. Si es así, los objetivos de este libro se han cumplido.

APENDICE A

EL JUEGO DE CARACTERES ASCII

Recuerde que del 0 al 31 son caracteres de control y solo se pueden imprimir si van precedidos por CHR$(1).

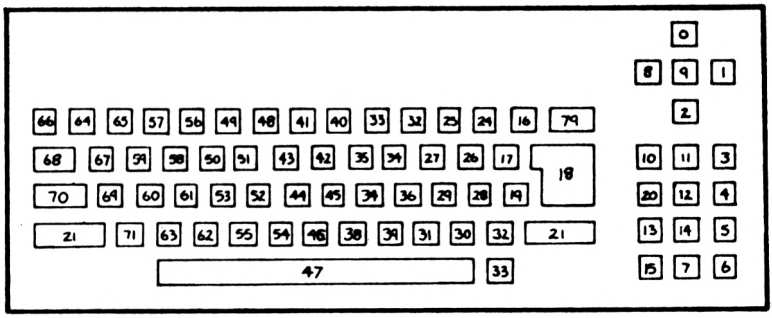



130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
131
136
141
146
151
156
161
166
171
176
181
186
191
196
201
206
211
216
221
226
231
236
241
246
251
132
137
142
147
152
157
162
167
172
177
182
187
192
197
202
207
212
217
222
227
232
237
242
247
252
133
138
143
148
153
158
163
168
173
178
183
188
193
198
203
208
213
218
223
228
233
238
243
248
253
134
139
144
149
154
159
164
169
174
179
184
189
194
199
204
209
214
219
224
229
234
239
244
249
254

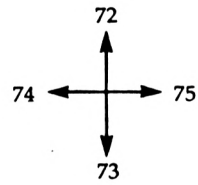
APENDICE B

CODIGOS DE MANEJO DE TECLAS

(usadas con INKEY)

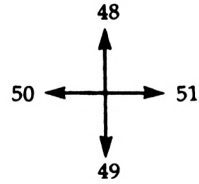


Joystick 0



fire 1 : 77
fire 2 : 76

Joystick 1



fire 1 : 53
fire 2 : 52

APENDICE C
CODIGOS DE COLOR

COD.	COLOR
0	Negro
1	Azul
2	Azul Brillante
3	Rojo
4	Magenta
5	Malva
6	Rojo Brillante
7	Púrpura
8	Magenta Brillante
9	Verde
10	"Cyan"
11	Azul Cielo
12	Amarillo
13	Blanco

COD.	COLOR
14	Azul Pastel
15	Naranja
16	Rosa
17	Magenta Pastel
18	Verde Brillante
19	Verde Mar
20	"Cyan" Brillante
21	Verde Lima
22	Verde Pastel
23	"Cyan" Pastel
24	Amarillo Brillante
25	Amarillo Pastel
26	Blanco Brillante

APENDICE D

CODIGOS DE ERROR

COD.	ERROR	
1	Unexpected NEXT	NEXT inesperado
2	Sintax error	Error de sintaxis
3	Unexpected RETURN	RETURN inesperado
4	DATA exhausted	Final de DATA
5	Improper argument	Argumento impropio
6	Overflow	Rebasamiento
7	Memory full	Memoria llena
8	Line does not exist	No existe esa línea
9	Subscript out of range	Subíndice fuera de rango
10	Array already dimensioned	Matriz ya dimensionada
11	Division by zero	División por cero
12	Invalid direct command	Comando directo invàlido
13	Type mismatch	Error al teclear
14	String space full	Cadena llena
15	String too long	Cadena demasiado larga
16	String expression too complex	Cadena muy compleja
17	Cannot CONTinue	No puede CONTinuar
18	Unknown user funcion	Función desconocida
19	RESUME missing	RESUME perdido
20	Unexpectèd RESUME	RESUME inesperado
21	Direct command found	Encontrado comando directo
22	Operand missing	Operando perdido
23	Line too long	Línea demasiado larga
24	EOF met	Encontrado EOF
25	File type error	Tipo de fichero erròneo
26	NEXT missing	NEXT perdido
27	File already open	Fichero abierto ya
28	Unknown command	Comando desconocido
29	WEND missing	WEND perdido
30	Unexpected WEND	WEND inesperado

INDICE

NOTA: Las definiciones de palabras clave del BASIC están en el capítulo 1. Las entradas en el índice son referencias seleccionadas que ilustran usos específicos de las palabras.

Volûmen 125,130

XPOS 118

YPOS 118

Z80 procesador 1
Z80A 1

SEGURAMENTE LO MEJOR PARA USTED Y SU 464

Este libro es suficientemente bueno como para haber ganado la aprobación de AMSOFT, la división de productos de ordenadores de AMSTRAD. Una simple ojeada a este nuevo y espectacular libro de nuestro autor, le dirá porqué es tan bueno, y porqué lo necesitan todos los propietarios de un 464 como compañía constante.

El libro asume que ya ha trabajado con el 464, y ha hecho algunos programas simples. Pero, aún a nivel simple, la organización del libro es atractiva, comienza con una descripción del funcionamiento del 464, como se comunica con los dispositivos externos, y una rápida revisión del BASIC. Contiene una sección de referencia para encontrar la explicación de los comandos de BASIC y las palabras clave del repertorio del Amstrad.

Otras secciones importantes del libro cubren: CADENAS Y CARACTERES; ENTRADA/SALIDA; ARITMETICA; MAPA DE MEMORIA; TIEMPO; RELOJ E INTERRUPCIONES; ESTRUCTURAS DE DATOS; PROCESO DE DATOS; GRAFICOS; SONIDO.

El libro contiene CUARENTA programas completos listos para ejecutar en el 464, desde los pequeños de demostración del funcionamiento del 464, hasta los largos que cubren por si mismos el precio del libro.

Se incluyen programas para:

Descifrar códigos;
Bases de datos simples;
Clasificar información en cualquier orden;
Trazar dibujos en tres dimensiones;
Aplicaciones comerciales.

Y, lo más espectacular de todo, un juego de tipo "arcade espacial" y un programa completo de SINTETIZADOR DE SONIDO.

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/