tomo 2

AMSTRAD

Ordenador personal y Procesador de texto

PCW8256/8512

Manual de BASIC

AMSOFT

A division of

AMSTRAD

CONSUMER ELECTRONICS PLC

Manual de Mallard BASIC

AMSOFT agradecerá el envío de comentarios y sugerencias relativos a este manual y al producto en él descrito.

Toda la correspondencia se debe dirigir a:

AMSOFT
Avda. del Mediterráneo 9
28007 Madrid
España

Toda reparación u operación de mantenimiento de este producto debe ser confiada a los distribuidores autorizados de AMSOFT. Ni AMSOFT ni AMSTRAD pueden asumir ninguna responsabilidad derivada del daño o pérdida que se pueda ocasionar como resultado de reparaciones efectuadas por personal no autorizado. El objetivo de este manual no es sino servir de ayuda al usuario en la utilización del producto; por consiguiente, AMSTRAD y AMSOFT quedan eximidos de responsabilidad por el daño o pérdida a que pueda dar lugar la utilización de la información aquí publicada o la incorrecta utilización del producto.

Escrito por Jean Gilmour, Locomotive Software
Publicado por Amsoft
Traducido por Emilio Benito Santos
Edición española producida por Vector Ediciones

Edición 1986 (1)

Introducción a BASIC

El record mundial de velocidad para locomotoras de vapor lo ostenta la locomotora LNER 4-6-2 n.º 4468, "Mallard". El 3 de julio de 1938 esta máquina arrastró a lo largo de un cuarto de milla, a la velocidad de 202 km/h, siete vagones que pesaban un total de 240 toneladas.

Mallard BASIC en el AMSTRAD PCW8256

Mallard BASIC está disponible en diferentes versiones para diversas aplicaciones y ordenadores. La versión que se suministra con el PCW8256 es la 'versión completa de Mallard BASIC para Z80-CP/M' con el sistema Jetsam de ficheros indexados. Incluye un editor de líneas que no requiere instalación. La versión 'sólo ejecución' de Mallard BASIC se puede adquirir aparte.

Este manual consta de dos partes: una de introducción y otra de referencia.

La parte de introducción describe el Mallard BASIC tal como se suministra con el PCW8256, pero indica además qué cambios hay que hacer en los programas para que éstos sean compatibles con otras versiones de Mallard BASIC.

La parte de referencia está diseñada para describir exhaustivamente todas las versiones de Mallard BASIC. En particular, se describe tanto Mallard-80 (para los microprocesadores 8080, Z80 y similares) y Mallard-86 (para 8088/8086 y similares). En los pocos casos en los que BASIC depende del tipo de microprocesador, se hace la advertencia adecuada. Mientras no se diga otra cosa, Mallard-80 y Mallard-86 producen los mismos resultados.

Editor de líneas e instalación del programa

Los 'BASIC completos' incluyen un editor de líneas basado en la pantalla. Este editor requiere un pequeño número de órdenes de pantalla y supone que la consola es una pantalla. Para utilizar el editor de líneas, en algunos casos hay que preparar ciertas tablas que BASIC necesita consultar para poder gestionar la pantalla y reconocer las órdenes que se le dan a través del teclado. Tal proceso se realiza mediante el 'programa de instalación' que se proporciona junto con BASIC; este programa está descrito en el apéndice V. No obstante, el Mallard BASIC que se suministra con el PCW8256 ya está 'instalado' en esta máquina; por consiguiente, no se incluye el programa de instalación.

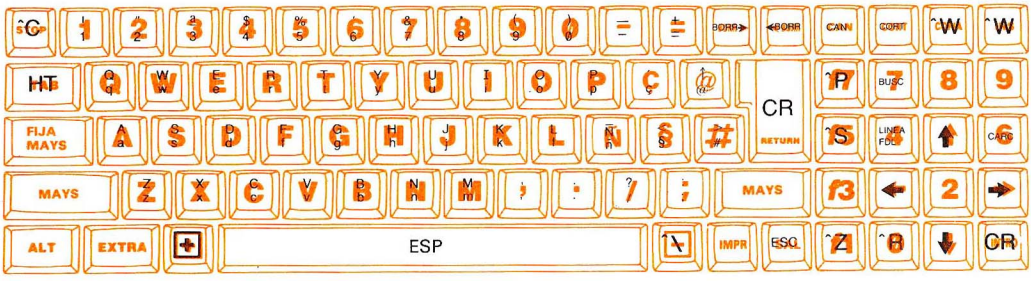
Con algunas versiones 'completas' se suministra también un segundo editor de líneas que no requiere instalación. Este editor no está basado en la pantalla, sino que funciona a base de órdenes; es el que utiliza BASIC antes de que se le instale el editor basado en la pantalla. El proceso de instalación borra el editor de órdenes y lo sustituye por el basado en la pantalla. El editor de órdenes no es necesario en el Mallard BASIC que se suministra con el PCW8256; sólo lo mencionamos por dejar completa esta descripción.

CONTENIDO

El teclado del PCW8256 con Mallard BASIC

Teclado principal

Las letras minúsculas y los caracteres inferiores se obtienen pulsando la tecla correspondiente sola. Las mayúsculas y los caracteres superiores se obtienen pulsando la tecla correspondiente en combinación con [MAYS].

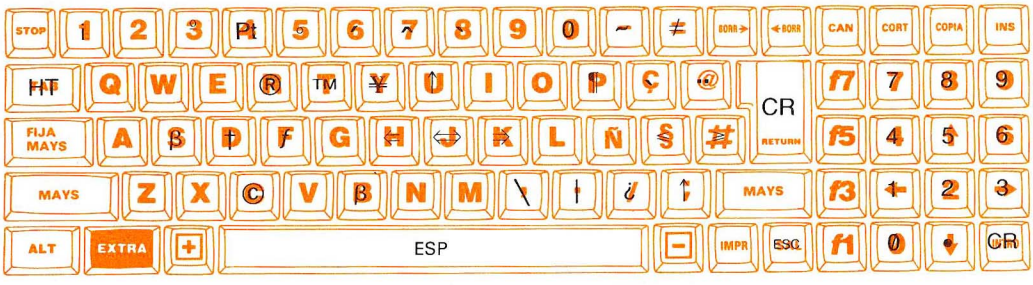


Teclado con ALT

Los caracteres inferiores se obtienen pulsando la tecla [ALT] al mismo tiempo que la tecla deseada. Para obtener los caracteres superiores se debe pulsar también [MAYS]. Para los códigos de control de CP/M (↑ *carácter*) no es necesaria la tecla [MAYS].

Teclado con EXTRA

El carácter que se muestra para cada tecla se obtiene pulsando la tecla al mismo tiempo que [EXTRA].



Combinaciones especiales de teclas

[ALT] + [INTRO] funciona como «bloqueo de mayúsculas», activando y desactivando la conversión de todas las letras (excepto las griegas) a sus correspondientes mayúsculas, pero sin afectar al resto de los caracteres.

[ALT] + [JUST] funciona como «bloqueo de números», activando y desactivando la utilización del grupo de teclas de la derecha como teclado numérico.

[MAYS] + [EXTRA] + [SAL] reinicializa el ordenador.

Introducción

Esta descripción de BASIC está diseñada para ayudar al usuario a utilizar el Mallard BASIC suministrado con el PCW8256, tanto en la ejecución como en la escritura de programas. La información de carácter general sobre el ordenador se puede encontrar en el 'Manual del usuario'. La 'Guía de CP/M Plus' describe el sistema operativo.

Este manual consta de dos partes:

- La **parte 1** es una suave introducción a BASIC, adecuada incluso para quienes no tengan ninguna experiencia en escribir programas de ordenador.
- La **parte 2** es una descripción detallada de todas las funciones de Mallard BASIC; puede ser consultada por quienes hayan leído y entendido la primera parte, o bien, directamente, por quienes ya conozcan otros dialectos de BASIC.

Si después de leer la primera parte usted no se considera preparado para empezar a programar, puede recurrir a alguno de los muchos libros que se han publicado sobre programación en BASIC; consulte en tal caso a su librero.

1.1 El método de lectura

Si usted no tiene ninguna experiencia en BASIC

Empiece por leer la parte 1. Practique con los ejemplos que le ofrecemos. Si lo desea, puede omitir el capítulo 7 en una primera lectura; este capítulo trata de los ficheros en disco. Cuando haya terminado de leer y asimilar la parte 1, estará en condiciones de escribir programas sencillos pero útiles, aunque posiblemente necesitará consultar de vez en cuando la parte 2.

A continuación, lea la parte 2, en la que explicamos cómo utilizar todas las funciones de Mallard BASIC. De hecho, la parte 1 cubre casi todas las facetas de BASIC, pero a un nivel más elemental, prestando mayor atención a la comprensión de los conceptos que a los detalles técnicos. La descripción completa de Mallard BASIC sólo se encuentra en la parte 2.

Si usted ya sabe programar en BASIC

En este caso no necesitará leer tan detenidamente la parte 1, con la posible excepción del capítulo 7 (gestión de ficheros), en el que se describe el potente sistema de ficheros indexados (Jetsam) de Mallard BASIC.

La parte 2 da un resumen completo y conciso de las funciones de Mallard BASIC. En el capítulo 9 se describen todas las palabras clave, en orden alfabético.

1.2 Para qué sirve BASIC

BASIC fue diseñado hace ya muchos años con la intención de desarrollar un lenguaje de programación que sirviera como instrumento de enseñanza y fuera fácil de aprender. Del BASIC original han derivado muchos dialectos. El Mallard BASIC es un BASIC moderno y muy potente que se suministra en varias versiones; su principal ventaja con respecto a otros es su capacidad de acceder mediante 'claves' (sistema que explicaremos más adelante) a la información grabada en disco.

Usted puede empezar a utilizar BASIC inmediatamente, considerándolo como una calculadora muy eficaz y completa. Pero, lo que es más importante, BASIC le servirá para escribir programas tras un corto aprendizaje.

Los programas pueden realizar una enorme diversidad de tareas, desde la simple suma de series de números hasta las más complejas, como pueden ser el mantenimiento de ficheros, el cálculo de nóminas o los análisis estadísticos.

Usted puede utilizar programas de BASIC escritos por otros, o bien, en cuanto haya aprendido a hacerlo, escribir programas a medida de sus necesidades.

BASIC es muy versátil; sirve para prácticamente todo trabajo que consista en manipular números y textos. No obstante, hay tareas que se realizan con mayor facilidad o rapidez fuera de BASIC, bien mediante otros programas suministrados con el ordenador (véase la 'Guía de CP/M Plus'), bien comprando programas comerciales (consulte a su distribuidor).

1.3 Convenios

Para esta descripción de BASIC hemos adoptado ciertos convenios relativos a cómo representar los diferentes tipos de información:

descripción

> descripción de la información, distinta de la información propiamente dicha. Por ejemplo, *número* significa 1, 2, 3, etc. Si la descripción consta de varias palabras, éstas se escriben unidas por guiones; por ejemplo, *nombre-de-fichero*.

[una opción]

> algo que puede ser omitido o incluido a voluntad (con la consiguiente diferencia de efectos).

algo que aparece en la pantalla o que se debe escribir en el teclado

Este tipo de letra imita los caracteres o palabras que aparecen en la pantalla y representa además lo que se introduce por el teclado.

GOTO, FIND$, OR, etc.

Las palabras clave de BASIC (es decir, las palabras que BASIC reconoce como pertenecientes a su vocabulario) se escriben siempre en mayúsculas; los nombres de las variables, en minúsculas. Aunque usted escriba las palabras clave en minúsculas, BASIC las convierte a mayúsculas cuando las almacena en la memoria.

señala una parte del texto de la línea anterior. Normalmente va seguido, en la línea siguiente, de una explicación de ese texto.

véase GOTO, véase FIND$, véase OR, etc.

remite, para ampliar la información, a la descripción de la palabra clave que se da en el capítulo 9 de la parte 2.

RETURN, CARC, etc.

representa la tecla marcada con 'RETURN', 'CARC', etc., no las letras R E T U R N, etc.

(Al principio de la parte 2 se describen otros convenios adicionales que se utilizan en ella.)

NOTA

Cuando se carga CP/M Plus, el juego de caracteres que se selecciona automáticamente es el español (juego n.º 7). Si bien esto es lo deseable cuando se va a ejecutar programas de CP/M escritos en castellano o adaptados a este idioma, sugerimos al usuario que antes de cargar BASIC seleccione el juego estándar ASCII (juego n.º 0). La orden necesaria es

LANGUAGE 0 [RETURN]

El efecto de esta orden es intercambiar las siguientes parejas de caracteres:

Pt ↔ #, ¡ ↔ [, Ñ ↔ \, ¿ ↔], ¨ ↔ {, ñ ↔ |

De esta forma, por ejemplo, el signo # se obtiene ahora con [EXTRA]+4 y es interpretado como tal por BASIC.

Véase además, en el manual de CP/M Plus, la descripción de las órdenes LANGUAGE y SETKEYS (sección 5.2) y de los juegos de caracteres de CP/M (apéndice I).

Primeros pasos en BASIC

Este capítulo enseña a cargar BASIC y a utilizarlo como calculadora. Además, presenta algunos conceptos que más adelante serán necesarios para empezar a programar en BASIC.

2.1 Entrada en BASIC

Para cargar BASIC antes hay que cargar el sistema operativo CP/M Plus. Si no está seguro de cómo hacerlo, consulte el capítulo 1 de la 'Guía de CP/M Plus'. Una vez cargado el sistema, inserte en la unidad una copia del primer disco de los suministrados con el ordenador, con la cara 2 hacia la izquierda. Finalmente, escriba BASIC[RETURN]. Al cabo de unos segundos aparecerá en la pantalla un mensaje de confirmación:

```
          A>
Escriba   BASIC[RETURN]

          Mallard-80 BASIC with Jetsam Version n.nn
          (c) Copyright 1984 Lomotive Software Ltd
          All rights reserved
          nnnnn free bytes

          Ok
```

Si el aspecto de la pantalla no es éste, compruebe que ha escrito correctamente la palabra BASIC y que el disco que ha insertado es el de BASIC, con la 'cara 2' hacia la izquierda.

Cuando aparezca el mensaje tendrá a su disposición todas las funciones de BASIC. Extraiga entonces el disco con el que ha cargado BASIC e instale el disco en el que piense grabar programas y datos.

2.2 Salida de BASIC

Cuando quiera cargar algún otro lenguaje o programa de aplicación, o sencillamente utilizar algún programa de ayuda de CP/M Plus, puede volver al sistema operativo mediante la orden SYSTEM de BASIC.

```
Escriba   SYSTEM[RETURN]

          A>
```

También puede volver a cargar CP/M Plus reinicializando la máquina (pulse [EXTRA] y [MAYS] y, antes de soltarlas, pulse también [SAL]), o bien apagando y volviendo a encender el ordenador, pero para ello es necesario que en la unidad haya un disco de CP/M Plus. Recuerde que debe extraer el disco antes de apagar el ordenador.

Cuando haya vuelto a CP/M Plus puede utilizar todas sus órdenes residentes y programas transitorios. Para volver a utilizar BASIC tendrá que cargarlo nuevamente por el procedimiento descrito.

2.3 Utilización de BASIC

Aunque BASIC es un lenguaje de programación, también se lo puede utilizar como calculadora muy potente. De hecho, ésta es una buena forma de empezar a conocer BASIC y a adquirir unos cuantos conceptos fundamentales que es imprescindible asimilar antes de poder empezar a programar.

En el resto del capítulo vamos a explorar BASIC desde este punto de vista. De la programación empezaremos a ocuparnos en el siguiente.

2.4 Cómo dar instrucciones a BASIC

Si queremos que BASIC realice algún trabajo, debemos decirle qué tiene que hacer dándole las instrucciones adecuadas a través del teclado. Una instrucción empieza siempre por una palabra que tiene un significado especial para BASIC: una *orden*. La orden va normalmente seguida de cierta información que BASIC necesita para poder obedecerla y termina con la pulsación de la tecla [RETURN].

Las órdenes son uno de los varios tipos de palabras y símbolos que tienen un significado especial para BASIC; su nombre global es *palabras clave*. Aunque usted puede escribirlas en mayúsculas o minúsculas, según le sea más cómodo, en este manual siempre las mostraremos en mayúsculas para facilitar su identificación. Además, es así como aparecen en los listados de los programas, tanto en la pantalla como en la impresora.

Las órdenes desempeñan una función similar a la de los verbos en el lenguaje ordinario: indican acciones. En BASIC se dispone de gran variedad de órdenes, que iremos presentando a lo largo de los capítulos siguientes. Una de las más fáciles de entender es PRINT (escribir). Esta orden pide a BASIC que visualice la información contenida en el resto de la instrucción.

La información que acompaña a la orden es análoga al 'complemento directo' de la oración: indica a qué objeto se debe aplicar la orden. En el caso de PRINT, especifica qué es lo que se ha de visualizar.

Por ejemplo, para que BASIC muestre el número 42 en la pantalla

escriba PRINT 42 RETURN
orden información

Observe que BASIC no intenta obedecer la instrucción mientras no se pulsa RETURN; hasta ese momento usted puede corregir los errores mecanográficos borrando con ←BORR y reescribiendo lo necesario.

Observe también que BASIC, como casi todos los demás lenguajes y programas de ordenador, sólo reconoce las órdenes que le damos si las escribimos con correción absoluta. Los frecuentes errores consistentes en omitir alguna letra o en cambiar una letra por otra suelen provocar en BASIC un mensaje del tipo 'Syntax error' (error de sintaxis). Siempre que usted se encuentre con un mensaje inesperado de esta clase, compruebe la ortografía de la orden y vuelva a escribirla correctamente. En particular, no olvide dejar un espacio en blanco entre la orden y la información que la acompaña.

Otra orden muy sencilla y útil es DIR, que tiene prácticamente el mismo efecto que la orden de CP/M Plus de igual nombre. Así, por ejemplo, para listar los nombres de todos los ficheros que hay en el disco de la unidad implícita actual,

escriba DIR RETURN

2.5 Tipos de información

La información que interviene en las instrucciones puede ser de los tipos que describimos a continuación.

2.5.1 Información fija (constantes)

El tipo de información más sencillo es el que ya hemos visto en el ejemplo anterior:

PRINT 42 RETURN
orden constante (número)

Hemos especificado un número escribiéndolo explícitamente en la instrucción. Este tipo de información se denomina *constante* porque su valor es fijo y está determinado en la propia instrucción. Las constantes pueden ser números (tales como 42, 0.0000002, −1111111111) o 'cadenas literales' (cadenas de texto, tales como "Hola!", "Beneficios del ejercicio fiscal 84/85", "42").

Escriba PRINT "Hola, amigo lector!" RETURN
orden constante (cadena de texto)

Observe que las cadenas literales se escriben entre comillas (").

2.5.2 Información cambiante (variables)

Un segundo tipo de información es lo que se denomina *variable*. Una variable es el nombre de un elemento de información, no la información propiamente dicha; cuando BASIC se encuentra con una variable, pone en su lugar el valor que actualmente tiene almacenado para esa variable (que habrá sido asignado en alguna instrucción anterior).

Pero ¿para qué poner una variable en la instrucción?; ¿por qué no poner directamente la constante deseada? La explicación es muy sencilla: al poner una variable hacemos que la instrucción sea mucho más versátil; cuando queremos realizar una serie de tareas similares, podemos usar reiteradamente la misma instrucción sin más que cambiar el valor de la variable, en lugar de tener que escribir una instrucción nueva para cada valor.

Hay diferentes tipos de variables para los distintos tipos de información. En una primera clasificación, las variables pueden ser *numéricas* (para números) o *literales* (para cadenas literales). Las variables numéricas tienen nombres formados por combinaciones de letras (de la 'A' a la 'Z' y de la 'a' a la 'z') y dígitos (del '0' al '9'); los nombres de las variables literales son de la misma forma, pero terminan con el carácter '$'. En ambos casos, el primer carácter del nombre tiene que ser una letra, no puede ser un dígito.

Nótese que, aunque nosotros distingamos entre mayúsculas y minúsculas al escribir los nombres de las variables, BASIC no hace tal distinción. Así, para BASIC las palabras 'perro', 'Perro' y 'pERRO' son un mismo nombre de variable.

En este manual hemos adoptado la norma de escribir los nombres de las variables en minúscula, para ayudar a distinguirlos de las palabras clave de BASIC. Nombres típicos de variables son los siguientes:

numéricas: a, contador, tasa.iva

literales: a$, nombre$, ciudad$

El usuario puede elegir los nombres de las variables a su gusto, con muy pocas limitaciones; la más notable es que no pueden coincidir con ninguna palabra clave y que no pueden contener espacios.

La forma principal de dar un valor a una variable es *asignárselo*, utilizando para ello el signo 'igual' (=). Por ejemplo, para dar a la variable 'iva' el valor 0.15,

Escriba iva = 0.15 RETURN

variable — valor asignado

Esta variable se puede luego incluir en una orden en la que se necesite ese valor. Por ejemplo,

Escriba PRINT iva RETURN

orden — variable

Análogamente se puede definir una variable literal, tal como nombre$.

Escriba nombre$ = "Ana Isabel Valbuena" RETURN

Este nombre se puede escribir entonces fácil y rápidamente:

Escriba PRINT nombre$ RETURN

orden variable

La asignación cambia el valor de la variable, pero no lo deja fijo. De hecho, se lo puede cambiar cuantas veces se desee. Al utilizar la variable en una instrucción, BASIC toma el valor más recientemente asignado.

Quizá sorprenda al lector la forma de la instrucción de asignación. ¿Dónde está la orden que indica a BASIC qué debe hacer? La respuesta es que BASIC analiza la instrucción, encuentra el signo igual y entonces actúa como si la instrucción fuese precedida de una orden (invisible). [En las primeras versiones de BASIC había que escribir la orden 'LET' (sea) explícitamente; en las modernas se puede incluir el LET, pero no es necesario.]

2.5.3 Información calculada (expresiones)

El tercer tipo de información que puede intervenir en las instrucciones son las *expresiones*. Una expresión es una instrucción que hace que BASIC calcule la información deseada. Consiste en una o varias constantes y/o variables, combinadas con 'operadores' o 'funciones' de BASIC. Los operadores y las funciones explican a BASIC cómo tiene que realizar los cálculos.

Operadores. Los operadores son normalmente símbolos que se escriben entre dos elementos de información. Especifican el tipo de operación que se debe realizar con los dos elementos para obtener el resultado final.

Los ejemplos más sencillos son la suma y la resta:

Escriba PRINT 42+1 RETURN

Escriba PRINT 43−1 RETURN

Los significados y efectos son obvios.

Otros dos operadores elementales son el de la multiplicación (*) y el de la división (/). Pruebe los siguientes ejemplos:

Escriba PRINT 42*2 RETURN

y

escriba PRINT 42/2 RETURN

Estos operadores pueden ser aplicados tanto a constantes como a variables. El único que tiene sentido aplicado a cadenas literales es el '+', cuyo efecto es unir las cadenas, una al final de la otra. Escriba lo siguiente exactamente como se indica; pero antes de pulsar RETURN por tercera vez trate de predecir cuál va a ser el resultado.

Escriba primero$ = "Ana "RETURN

escriba segundo$ = "Isabel"RETURN

escriba PRINT primero$+segundo$RETURN

¿Lo había adivinado?

Las expresiones no tienen por qué ser tan sencillas como éstas. Por ejemplo, para sumar 22, 44 y 90000032 y luego restar 400311 del resultado se puede escribir tres instrucciones por separado:

escriba PRINT 22+44RETURN
66
Ok

escriba PRINT 66+90000032RETURN
90000098
Ok

escriba PRINT 90000098−400311RETURN

Pero se puede ahorrar mucho trabajo combinando todas las operaciones en una sola expresión:

escriba PRINT 22+44+90000032−400311RETURN

Aunque la longitud y la complejidad de las expresiones están limitadas, no es probable que usted alcance esos límites.

Cuando una expresión contiene varios operadores, no es obvio en qué orden va a ser calculada, y sin embargo este orden es decisivo. Por ejemplo, 3+4*5 se podría calcular de la siguiente forma:

```
3+4=7
    *5=35
```

o bien

```
4*5=20
    +3=23
```

En BASIC los operadores y las funciones tienen unos niveles de prioridad que determinan el orden en que se los aplica. Por ejemplo, * y / se aplican antes que + y −, de modo que

3+4*5 se calcula de la forma 3+(4*5)=23. Los niveles de prioridad de los operadores están descritos en la parte 2 de este manual. De momento, en caso de duda, agrupe entre paréntesis las operaciones que deban ser tratadas como un bloque. Por ejemplo, (3+4)*5 se calcula como (7)*5, mientras que 3+(4*5) se calcula como 3+(20).

Funciones. Las funciones tienen nombres (lo mismo que las órdenes de BASIC) y van seguidas de la información sobre la que han de actuar, la cual se pone entre paréntesis. Una función típica es SQR (raíz cuadrada):

```
escriba     PRINT SQR(121)[RETURN]
            11
            Ok
```

Otra función típica, esta vez aplicable a cadenas literales, es LEFT$ (izquierda); esta función copia el número de caracteres especificado empezando por la izquierda de la cadena. Por ejemplo, para escribir los tres primeros caracteres de nombre$,

```
escriba     PRINT LEFT$(nombre$,3)
            Ana
            Ok
```

(Nótese que LEFT$ requiere dos elementos de información, separados por una coma.)

Hay funciones que actúan sobre información de un tipo y producen un resultado de un tipo distinto. Por ejemplo, para averiguar la longitud de nombre$, en lugar de contar nosotros mismos los caracteres podemos dejar que lo haga BASIC:

```
escriba     PRINT LEN(nombre$)
            19
            Ok
```

(LEN es abreviatura de *length*, 'longitud'.)

BASIC dispone de una gran variedad de funciones: trigonométricas, logarítmicas, de manipulación de cadenas literales, de conversión de cadenas a números y viceversa, etc. Todas estas funciones están descritas en el capítulo 6 y en la parte 2. Incluso es posible definir funciones nuevas (con DEF FN, capítulo 5).

2.6 Repaso del capítulo

En este capítulo usted ha aprendido a cargar y poner en marcha BASIC, además de algunos conceptos acerca de las instrucciones y órdenes de BASIC y de la información sobre la que éstas actúan. Ahora ya sabe cómo ordenar a BASIC que realice tareas sencillas, tales como hacer cálculos aritméticos y escribir los resultados.

Capítulo 3

Introducción a la programación en BASIC

En este capítulo vamos a presentar los conceptos fundamentales de la programación y algunas herramientas sencillas de las muchas que BASIC pone a disposición del usuario para ayudarle a escribir y ejecutar los programas.

3.1 El primer programa

Hasta ahora no hemos hecho más que aprender a usar un pequeño número de órdenes de BASIC, y nada de programación. Pero esto tiene fácil remedio. Escriba lo siguiente, exactamente según se indica:

NEW RETURN

10 PRINT 42 RETURN

¡Acaba usted de escribir su primer programa!

La primera línea es la orden NEW (nuevo). Esta orden pide a BASIC que olvide el programa que pudiera tener en la memoria y que se prepare para recibir uno nuevo.

La segunda línea es la única que hay en el programa. Consiste en una instrucción que usted ya conoce, precedida de un número: el *número de línea*. El propósito de los números de línea es doble: indican a BASIC que lo que sigue es una línea de programa y que por lo tanto no tiene que obedecer la instrucción inmediatamente, sino almacenarla; además, dan a la línea una 'etiqueta', a la que más tarde se puede hacer referencia con diversos fines.

Escriba 10 PRINT 42 RETURN

número de línea — instrucción

En esto es en lo que consisten todos los programas: una serie de instrucciones de BASIC repartidas en varias líneas, cada una de las cuales va precedida de un número (diferente).

Entonces, ¿qué diferencia hay entre este programa y la instrucción? Los programas tienen dos propiedades que los distinguen de las simples instrucciones:

- los programas pueden ser utilizados repetidamente sin requerir más que un mínimo esfuerzo por parte del usuario.
- los programas pueden ser grabados en el disco, de modo que no se los pierde cuando se apaga el ordenador.

Para que BASIC obedezca las instrucciones del programa se le da la orden RUN:

escriba RUN[RETURN]
42
Ok

(Si no es éste el resultado que usted obtiene, es que hay algún error en el programa; vuelva a empezar a partir de NEW, y esta vez sea más cuidadoso.)

Para grabar el programa en el disco se usa la orden SAVE. CP/M Plus graba los programas en forma de ficheros con nombre; especifique a continuación de SAVE el nombre que quiera dar al programa:

escriba SAVE "PROGNUM"[RETURN]
Ok

En el nombre de fichero no se debe incluir el punto, a menos que se quiera especificar también el distintivo de tipo.

Para ejecutar un programa que está grabado en disco se da la orden LOAD (cargar del disco a la memoria) y luego RUN (ejecutar):

escriba LOAD "PROGNUM"[RETURN]
Ok

escriba RUN[RETURN]
42
Ok

El mismo efecto se consigue con una sola orden, poniendo el nombre del programa a continuación de la orden RUN:

escriba RUN "PROGNUM"[RETURN]
42
Ok

Nota. Si el programa que se va a grabar o cargar está en el disco de la unidad B, se debe escribir 'B:' como prefijo del nombre del fichero. Por ejemplo, RUN "B:PROGNUM".

Una vez cargado el programa en la memoria, se lo puede ejecutar cuantas veces se desee con la orden RUN, sin necesidad de volver a escribir el nombre. Es decir, no es necesario cargar el programa del disco a la memoria cada vez que se lo ejecuta.

(Si no está muy convencido de que el programa haya quedado grabado en el disco y pueda ser cargado más tarde, haga lo siguiente: apague y vuelva a encender el ordenador, cargue CP/M Plus y BASIC y dé la orden RUN seguida del nombre del programa. No obstante, este grado de escepticismo le hará perder unos minutos. Si a pesar de todo quiere hacer la prueba, no olvide sacar el disco antes de apagar el ordenador.)

Si examina el contenido del disco con la orden DIR, observará que BASIC ha grabado el programa con el nombre especificado y que ha asignado el distintivo de tipo BAS. Este sufijo identifica el fichero como programa de BASIC. Al crear o utilizar un programa con SAVE, LOAD, RUN (o CHAIN) no hace falta mencionar el distintivo de tipo, a menos que sea distinto de BAS.

3.2 Programas algo más complejos

Si un programa sólo consta de una línea, como ocurría en el ejemplo anterior, el número que se dé a esa línea es casi indiferente. Sin embargo, si se añade más líneas al programa, los números de línea tienen una importancia fundamental. Si a una nueva línea se le da el mismo número que a otra ya existente, ésta es reemplazada por la nueva; en general, un programa consiste en cierto número de líneas, colocadas por orden de número de línea ascendente.

Por ejemplo, escriba las cuatro líneas siguientes exactamente como se indica:

```
NEW[RETURN]
30 PRINT 64*64[RETURN]
10 PRINT "64 por 64"[RETURN]
20 PRINT "es:"[RETURN]
```

Ahora puede servirse de la orden LIST para comprobar el orden en que han quedado guardadas las líneas del programa:

escriba `LIST[RETURN]`

En la pantalla se verá lo siguiente:

```
10 PRINT "64 por 64"
20 PRINT "es:"
30 PRINT 64*64
Ok
```

Como puede comprobar, a pesar del orden en que usted introdujo las líneas, BASIC las ha almacenado en orden de números crecientes. Cuando BASIC se toma la molestia de hacerlo así no es sólo por organizar mejor los listados, sino porque los números de las líneas imponen el orden en que éstas van a ser obedecidas cuando se ejecute el programa. Compruébelo ejecutando el programa:

escriba
```
RUN[RETURN]
64 por 64
es:
 4096
Ok
```

Hay una orden de BASIC que proporciona automáticamente los números de línea cuando se está introduciendo un programa nuevo. Se trata de la orden AUTO:

`AUTO[RETURN]`

genera números de línea empezando por el diez e incrementándolos de 10 en 10 cada vez que se pulsa [RETURN].

AUTO 620[RETURN]

genera números de línea empezando por el especificado (en este caso el 620) e incrementándolos de 10 en 10 cada vez que se pulsa [RETURN]

AUTO 100,20[RETURN]

genera números de línea empezando por el primer número especificado (en este caso el 100) y sumando el segundo número (20) cada vez que se pulsa [RETURN].

Para interrumpir la generación automática de números de línea se debe introducir el carácter especial llamado 'control-C'; en el PCW8256 este carácter se obtiene pulsando [ALT]+C, o bien [STOP].

En los ejemplos de este manual las líneas de programa normalmente sólo contienen una instrucción. No obstante, en general las líneas pueden contener varias instrucciones, separadas unas de otras por el signo de dos puntos (:). Así, podríamos haber escrito el programa anterior en una sola línea de la siguiente forma:

```
10 PRINT "64 por 64": PRINT "es:": PRINT 64*64
```

número de línea 1.ª instrucción 2.ª instrucción 3.ª instrucción

Hay ocasiones en las que esta posibilidad es muy interesante. Sin embargo, le recomendamos que por ahora se limite a poner una instrucción en cada línea, pues de esta forma es más fácil modificar los programas, observar su evolución cuando funcionan y detectar los errores cuando no lo hacen.

3.3 Modificación de los programas

Si usted no está satisfecho con un determinado programa, una forma de «cambiarlo» es borrarlo con NEW y escribirlo de nuevo. Este método puede servir para programas muy cortos, pero es totalmente inadecuado cuando lo que se pretende es introducir pequeños cambios en programas largos.

BASIC dispone de un buen repertorio de órdenes que facilitan la corrección de los programas:

Para exhibir en la pantalla la lista completa de las instrucciones del programa se utiliza la orden LIST, según hemos explicado antes.

Si el programa es tan largo que no cabe completo en la pantalla, se puede pulsar control-S para «congelar» el listado en el momento en que estén en la pantalla las instrucciones que interese observar. Después se puede, o bien pulsar otra vez control-S para «descongelar» el listado y permitir que éste continúe, o bien control-C para abandonar el listado definitivamente. Análogamente a lo que ocurría con control-C, control-S se obtiene pulsando [ALT]+S.

No obstante, hay otras versiones de la orden LIST que son más adecuadas para obtener listados parciales:

LIST 120[RETURN]

lista una sola línea.

LIST −80[RETURN]

lista todas las líneas desde el principio del programa hasta la especificada inclusive.

LIST 400−[RETURN]

lista todas las líneas desde la especificada hasta el final del programa.

LIST 420−440[RETURN]

lista las líneas del margen especificado.

También se puede enviar los listados a la impresora. La orden necesaria es LLIST. Funciona exactamente igual que LIST, con la única diferencia de que toma como dispositivo de salida la impresora, en vez de la pantalla.

Para borrar una línea, sencillamente se escribe su número y se pulsa [RETURN]. Por ejemplo, para borrar la línea 400,

escriba 400[RETURN]

Para borrar grupos de líneas se utiliza la orden DELETE seguida de la especificación del margen deseado, igual que en LIST.

Para sustituir una línea por una versión más moderna, se escribe la versión nueva con el mismo número de línea que la antigua.

Para rehacer la numeración de las líneas, de modo que el salto de cada una a la siguiente sea igual a lo largo de todo el programa, se utiliza la orden RENUM:

escriba RENUM[RETURN]

Esta orden cambia el número de la primera línea a 10, el de la segunda a 20, el de la tercera a 30, etc. Las referencias a líneas concretas que pueda haber en las instrucciones del programa quedan también correctamente modificadas. (Hay versiones de RENUM más complejas. Por ejemplo, RENUM 100 cambia el número de la primera línea a 100, el segundo a 110, etc. Para más detalles, consulte la palabra clave RENUM en la parte 2 de este manual.)

La última orden «de edición» es EDIT; esta orden permite modificar una línea especificada. Por ejemplo, para modificar la línea 100.

escriba EDIT 100[RETURN]

La orden EDIT exhibe en la pantalla la línea solicitada y ofrece los siguientes recursos de edición:

mover el cursor por la línea con las teclas [←], [→], [↑] y [↓].

[BUSC] buscar el carácter especificado y llevar el cursor hasta él.

[+] conmutar el modo de inserción entre 'activado' (el carácter nuevo se inserta en la posición del cursor) y 'desactivado' (el carácter nuevo sustituye al que está bajo el cursor).

[←BORR] borrar el carácter que está a la izquierda del cursor.

[BORR→] borrar el carácter que está bajo el cursor.

[CORTE] borrar desde la posición del cursor hasta la de un carácter especificado.

[CAN] abandonar EDIT conservando la versión anterior de la línea.

[RETURN] abandonar EDIT dando por válidas las modificaciones introducidas en la línea.

Por ejemplo, si usted ha escrito por error la línea 20 de la siguiente forma:

```
20 PRNT
```

puede editarla escribiendo:

```
EDIT 20[RETURN]
   20 PRNT
```

Lleve el cursor hasta la 'N' pulsando [→] dos veces:

```
20 PRNT
```

Luego escriba la 'I':

```
20 PRINT
```

Finalmente, pulse [RETURN] para almacenar la versión modificada.

Los mismos recursos de EDIT son aplicables a la línea que se está escribiendo nueva (tanto si es una orden como si una línea de programa) pulsando control-A.

3.4 Repaso del capítulo

En este capítulo usted ha aprendido a crear programas basados en las órdenes y tipos de información que presentamos en el capítulo anterior. También ha aprendido a grabar programas en disco y a recuperarlos cargándolos del disco al ordenador. Finalmente, ha aprendido a examinar y a modificar programas.

Capítulo 4

Programación útil

4.1 Diseño de programas

Como explicamos en el capítulo anterior, un programa de ordenador no es más que una colección de instrucciones numeradas que se suministra al ordenador, a través de la cual se explica a éste cómo debe realizar cierta tarea. La programación es el conjunto de actividades necesarias para elaborar un programa de ordenador.

El desarrollo de todos los programas, incluso de los más sencillos, obliga a pasar por diferentes fases. Una subdivisión típica del trabajo que hay que realizar para desarrollar un programa complejo es la siguiente:

- decidir qué se pretende que haga el programa (análisis)
- decidir cómo se quiere que el programa realice la tarea prevista (diseño)
- escribir el programa (codificación)
- comprobar que el programa funciona correctamente (verificación)
- redactar de instrucciones para el usuario (documentación)
- corregir, modificar e introducir mejoras en el programa (mantenimiento).

De todo esto hablaremos detalladamente en capítulos posteriores. La idea que debe quedar clara en este momento es que antes de empezar a escribir un programa es necesario diseñarlo.

Para diseñar un programa, lo primero que hay que hacer es analizar la tarea que se quiere que el programa realice y luego subdividirla en subtareas. Cuando se sabe cuál es la misión de las subtareas, ya se puede empezar a convertirlas en una especificación de los pasos que debe dar el programa para realizar la tarea global, expresándola con las palabras y conceptos proporcionados por el lenguaje de programación.

Todos los programas reciben una información de entrada, la manipulan convenientemente y luego la utilizan para generar los resultados. Una de las formas más sencillas de analizar una tarea es dividirla en estas tres etapas:

- captar la información
- manipular la información
- generar los resultados

Normalmente, el programa capta la información a través del teclado (la escrita por el usuario) o la lee en el propio programa o en el disco; luego la manipula para calcular sumas, promedios, para elaborar listas ordenadas, etc.; finalmente, la escribe en la pantalla o en el papel, en forma de números, texto, diagramas, etc., o la almacena en el disco para que luego pueda ser utilizada por el mismo programa o por otros.

4.2 Obtención de la información

La información utilizada por un programa puede provenir de tres fuentes principales: del interior del propio programa, de los datos proporcionados por el usuario a través del teclado o de ficheros grabados en disco. Vamos a describirlas a continuación.

4.2.1 Información almacenada en el programa

Cuando la información que necesita el programa es fija (o sea, siempre la misma), lo mejor es incorporarla al mismo programa, bien sea mediante constantes, en instrucciones del tipo

```
330 PRINT "***SUB-TOTALES***"
```

bien asignando los datos a variables y luego poniendo las variables en las instrucciones:

```
330 título$ = "***SUB-TOTALES***"
340 PRINT título$
```

Ya hemos aplicado este método en programas anteriores; de hecho, son muy pocos los programas que no lo utilizan. La idea de usar las variables para representar valores constantes, en lugar de manipular las constantes directamente, tiene las siguientes ventajas:

- Si la constante es una cadena literal complicada o un número largo, basta con escribirla (correctamente) una sola vez; después, cuando haya que mencionarla en el programa, se la sustituye por el nombre de la variable, que siempre será más corto o más fácil de recordar.
- Si la asignación de todas las variables se realiza en la misma zona del programa, éste es más fácil de modificar. Por ejemplo, si se usa la variable cliente$ para representar el nombre de un cliente en varias líneas de programa, éste será más fácilmente adaptable a otro cliente (asignando un nuevo valor a cliente$) que si hubiera que repasar el programa entero para hacer la sustitución en todos los lugares donde aparece explícitamente el nombre del cliente.
- Los nombres de las variables se pueden elegir de forma que hagan el programa más inteligible. Por ejemplo, PRINT valor*tasa.iva tiene un significado más obvio que PRINT valor*0.15.

Cuando el número de constantes que intervienen en el programa es suficientemente grande, en lugar de asignarlas una a una a las variables es preferible manejarlas más sistemáticamente mediante el par de palabras clave DATA (datos) y READ (leer). DATA va seguida de una lista de valores literales o numéricos, separados por comas. READ va seguida de una variable (o lista de variables) a la que se asignan sucesivamente los datos leídos en la lista de DATA.

Por ejemplo, para almacenar el número de días que hay en cada mes del año, podríamos escribir la siguiente sentencia DATA:

```
100 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

Esta información sería leída por instrucciones READ del siguiente tipo:

```
110 READ dias.ene
120 READ dias.feb
130 READ dias.mar
```

etc.

Para simplificar el programa y ahorrar trabajo mecanográfico, se puede incluir todas las instrucciones READ en una misma línea:

```
110 READ dias.ene,dias.feb,dias.mar,  etc.
```

El par READ/DATA es particulamente indicado cuando se asignan valores a un tipo especial de variables: las denominadas *matrices*. (Véase la sección 4.3.)

Los datos constantes que se asignen a variables por este método pueden estar distribuidos en un número arbitrario de sentencias DATA. BASIC las considera todas ellas como si formasen una sola lista de constantes, empezando por la que lleve el número de línea más bajo. Cada vez que se lee un valor con READ, el siguiente valor queda preparado para la siguiente lectura. Si se intenta leer con READ más valores de los disponibles en la lista, se provoca un error.

Hay otra palabra clave, RESTORE (restaurar), que permite controlar el orden en que se leen los datos. En la parte 2 de este manual se explica más ampliamente la utilización de DATA, READ y RESTORE.

4.2.2 Información introducida por el usuario a través del teclado

Todos los programas que hemos escrito hasta ahora adolecen de una limitación importante: tienen una misión única y predeterminada. Si queremos que hagan algo diferente, aunque sea muy parecido, tenemos que modificarlos.

Supongamos, por ejemplo, que usted es propietario de una tienda (muy especial) en la que sólo se vende un artículo y que quiere escribir un programa que le diga cuánto tiene que cobrar por el artículo para obtener un beneficio del 20%. Podría servirle el siguiente programa:

```
10 coste=10
20 beneficio=0.2
30 PRINT "El precio que se debe cobrar es"; coste+coste*beneficio
```

Introduzca y pruebe este programa. (No olvide escribir NEW[RETURN]. Puede utilizar AUTO.)

El programa funciona correctamente, pero es muy poco útil. Si cambia el coste del artículo, hay que modificar la línea 10; si usted decide cambiar el porcentaje de beneficio, hay que modificar la línea 20. Esto es muy laborioso y expuesto a errores. (Haga unas cuantas modificaciones de éstas y lo comprobará.) Por otra parte, esto representa que las únicas personas que pueden utilizar el programa cuando cambian los datos son las que sepan como modificarlo.

Todas estas dificultades radican en las líneas 10 y 20; la información que hay en ellas es **constante**, y lo que necesitamos es todo lo contrario: permitir que la información sea **variable**. BASIC nos da la solución por medio de la orden INPUT; esta orden pide la información al usuario **cada vez que se ejecuta el programa** y la asigna a las variables especificadas.

Para ver cómo funciona INPUT, modifique la línea 10:

```
10 INPUT coste
```

Ahora, cada vez que ejecutamos el programa, éste escribe un signo '?' y espera hasta que escribamos un número y pulsemos [RETURN]. Hecho esto, asigna el número a la variable 'coste' y continúa. Ejecute usted el programa; escriba un número y pulse [RETURN]. Vuelva a ejecutarlo introduciendo un número diferente.

El programa modificado es como sigue:

```
10 INPUT coste
     cambiado
20 beneficio=0.2
30 PRINT "El precio que se debe cobrar es"; coste+coste*beneficio
```

La orden INPUT también sirve para asignar textos a las variables literales; para ello basta con poner el nombre de una variable literal a la derecha de INPUT (por ejemplo, INPUT nombre$). Hay ciertas limitaciones en cuanto al tipo de texto que puede captar (por ejemplo, distingue las comas de los demás caracteres); pero estas limitaciones son superables (véase LINE INPUT en la parte 2).

Pues bien, INPUT ha introducido cierta versatibilidad en nuestro programa, a pesar de lo cual sigue siendo un programa poco «amistoso». Para quien no haya escrito el programa, ¿qué significa el signo de interrogación?; está claro que el programa debería dar alguna pista al usuario. Usted ya sabe cómo hacerlo: con la instrucción PRINT y una cadena literal.

Modifique el programa para que éste emita algún mensaje (inductor) que explique al usuario qué tiene que hacer cuando vea el '?'. Pruebe su nueva versión del programa.

Una solución podría ser la siguiente:

```
5 PRINT "Escriba el coste del artículo y luego pulse RETURN"
```

inductor (cadena de texto)

```
10 INPUT coste
20 beneficio = 0.2
30 PRINT "El precio que se debe cobrar es"; coste+coste*beneficio
```

Observe que el mensaje tiene que estar en una línea anterior a la 10. La elección del texto no afecta en absoluto al funcionamiento del programa, pero es **muy importante** para el usuario. El cuidado exquisito en la redacción de los mensajes es decisivo para la facilidad de uso de muchos programas.

Dado que INPUT suele requerir un mensaje que explique al usuario qué es lo que tiene que escribir, BASIC permite incluir el mensaje en la propia instrucción INPUT. Gracias a esta variante, las líneas 5 y 10 se pueden combinar en una sola:

```
10 INPUT "Escriba el coste del artículo y luego pulse RETURN",coste
```

inductor (cadena de texto) variable

```
20 beneficio=0.2
30 PRINT "El precio que se debe cobrar es"; coste+coste*beneficio
```

Observe que el texto del mensaje tiene que ser una cadena literal constante, no una variable literal. (Si fuese una variable, INPUT no la consideraría mensaje inductor, sino que trataría de asignarle lo que el usuario escribiese en el teclado.) Además, el mensaje tiene que estar antes que la variable, no después.

Observe también la coma que hemos puesto entre el texto y la variable. Esta coma suprime el '?'. Si usted prefiere conservar el signo de interrogación, ponga un punto y coma en lugar de la coma.

Grabe este este programa en el disco, con el nombre de 'pvp'. Volveremos a necesitarlo.

Modifique el programa de forma que capte por el teclado también el porcentaje de beneficio (no olvide incluir un mensaje inductor adecuado). Pruébelo para comprobar que funciona correctamente.

BASIC ofrece otro método de captación de la información a través del teclado. Se trata de la función INKEY$. Esta función examina el teclado casi instantáneamente para averiguar si hay alguna tecla pulsada; si la hay, genera una cadena cuyo único carácter es el correspondiente a la tecla; si no, genera la *cadena vacía*. La forma de las instrucciones en las que interviene INKEY$ es:

```
variable$=INKEY$
```

Puesto que INKEY$ no espera a que el usuario pulse [RETURN] (ni ninguna otra tecla), la forma de utilizar esta función es muy distinta de la de INPUT. Tiene dos aplicaciones principales: averiguar qué ha pulsado el usuario en el teclado al mismo tiempo que se continúa realizando otra tarea y proporcionar un método fácil y rápido de captación de información en forma de caracteres aislados.

Para ver INKEY$ en marcha tendremos que esperar hasta el siguiente capítulo, ya que ambas aplicaciones requieren otras órdenes de BASIC que aún no conocemos.

4.2.3 Información grabada en disco

La tercera fuente de información para los programas son los ficheros de datos grabados en disco. Los ficheros de datos son creados por BASIC para almacenar información numérica o literal, análoga a la que se guarda en las variables. La diferencia está en que la información grabada en disco se conserva hasta que deliberadamente decidamos borrarla o modificarla; no se pierde aunque cambiemos de programa o apaguemos el ordenador.

Todo lo relativo a los ficheros de datos está descrito detalladamente en el capítulo 7.

4.3 Almacenamiento de la información

En esta sección vamos a describir en detalle los diversos tipos de variables y de información que BASIC puede utilizar.

4.3.1 Elección de los nombres de las variables

BASIC da gran libertad al usuario en cuanto a la elección de nombres para las variables. Las principales limitaciones son las siguientes:

- Se puede incluir cualquier letra de la 'A' a la 'Z' y de la 'a' a la 'z' (en los nombres de variables BASIC no distingue mayúsculas de minúsculas).
- El punto puede figurar en cualquier posición, salvo en la primera.
- Los dígitos del '0' al '9' pueden figurar en cualquier posición, salvo en la primera.

- Los signos % ! # y $ sólo pueden figurar en última posición, y entonces especifican el tipo de variable.
- Los paréntesis sólo pueden intervenir en los nombres de las matrices.
- No están permitidos como nombres de variables los que sean idénticos a las palabras clave de BASIC, cuya lista completa se da en el apéndice IV.

Recuérdese que BASIC no distingue entre mayúsculas y minúsculas en los nombres de las variables, de modo que, por ejemplo, nombre$, Nombre$, noMBRe$, etc. son todos iguales para BASIC.

Dentro de estas limitaciones, conviene dar a las variables nombres que indiquen cuál es la naturaleza de la información que se almacena en ellas. Por ejemplo, si una variable representa un saldo provisional, su nombre debería ser 'saldo', o incluso 'saldo.provisional'; nunca nada tan misterioso como 'zz'. De esta manera se consigue que los programas sean más comprensibles cuando nos disponemos a corregirlos o mejorarlos meses después de escribirlos por primera vez.

No obstante, también interesa que los nombres sean razonablemente cortos; esto ayuda a lograr programas compactos y a minimizar los errores mecanográficos. Si no está muy convencido, escriba

```
saldo.provisional.mensual.acumulado=saldo.provisional.mensual.acumulado+mes
```

unas cuantas veces y comprenderá por qué le damos este consejo.

4.3.2 La corta vida de una variable

Las variables son, por su misma naturaleza, entes volátiles, transitorios. El valor actual de una variable se pierde por el hecho de:

- ejecutar un programa (RUN)
- cargar un programa (LOAD)
- salir de BASIC

Si se trata de usar una variable cuyo valor se ha perdido, lo que se obtiene es 0 (en el caso de las variables numéricas) o la cadena vacía "" (en el caso de las literales).

Hay varias formas de conservar la información almacenada en las variables para su utilización posterior. No obstante, por ahora, asegúrese de que todos sus programas asignen a las variables los valores correctos; no dé por supuesto que conservan los que tenían la última vez que se ejecutó el programa, porque **no los conservan.**

4.3.3 Números

BASIC puede trabajar con números de diversos tipos. Son los que vamos a describir a continuación.

Los que utilizamos habitualmente son los 'números en notación ordinaria', ya sean enteros o decimales; por ejemplo, −44444, 0, 3.1111, 3333333333.77702, etc. Estos números son los que vemos y usamos en la vida cotidiana, y también los más frecuentes en programación. Observe, no obstante, que en estos números no se pone el punto para separar los millares, y que se escriben con **punto decimal**, no con coma decimal.

Menos conocidos, salvo para los lectores que tengan una información científica, son los 'números en notación científica'. En esta notación los números se expresan en forma de un número decimal multiplicado por una potencia de 10. En BASIC tienen la forma n.nnnE*[signo]*nn, o bien n.nnnD*[signo]*nn. (Las letras E y D especifican con qué precisión debe manejarlos y almacenarlos BASIC.) Este sistema se utiliza sobre todo para expresar números muy grandes o muy pequeños.

Finalmente, BASIC puede trabajar con números expresados en sistemas de numeración distintos del decimal: en el sistema octal (base 8) y en el hexadecimal (base 16). Los números octales se escriben precedidos de &O; los hexadecimales, de &H. Estos sistemas de numeración sólo interesan a quienes estén acostumbrados a programar en lenguajes de bajo nivel en miniordenadores y microordenadores.

Veamos unos ejemplos de números de todos estos tipos:

Notación ordinaria	**Notación científica**	**Octal**	**Hexadecimal**
0	0E0	&O0	&H0
100	1E2	&O144	&H64
−100	−1E2	&O177634	&HFF9C
100.1	1.001E2	-	-
0.023	2.3E−2	-	-

Los márgenes permitidos son:

de −1.7E38	de −1.7E38	de &O0 a	de &H0 a
a +1.7E38	a +1.7E38	&O177777	&HFFFF

4.3.4 Variables numéricas

Cualquiera que sea el método que se adopte para representar los números, BASIC sólo dispone de tres tipos de variables para almacenarlos: variables numéricas enteras, de precisión sencilla y de doble precisión. Estos tipos se distinguen por el espacio de memoria que se requiere para almacenar cada número, por la velocidad a la que BASIC los maneja y por la precisión con que los almacena. Una vez asignado un número a una variable, BASIC olvida la forma en que se lo había representado originalmente.

Las variables enteras no pueden contener más que números enteros (o sea, sin parte decimal) del margen de −32768 a +32767. Si se asigna un número decimal a una variable entera, BASIC redondea el número al entero más próximo; en cambio, si se asigna un número ajeno al margen mencionado, se provoca un error.

Las variables numéricas de precisión sencilla y de doble precisión pueden almacenar números del margen de −10000000000000000000000000000000000000 a +10000000000000000000000000000000000000 (en notación científica, de -1×10^{38} a $+1\times10^{38}$). Si se asigna a una variable un número ajeno a este margen, se provoca un error. Las variables de precisión sencilla dan aproximadamente siete cifras significativas exactas; las de doble precisión dan aproximadamente dieciséis cifras significativas exactas. El menor número distinto de cero que puede almacenar una variable de doble precisión es aproximadamente 3E−39 (3×10^{-39}).

El tipo de las variables numéricas se indica mediante el último carácter del nombre:

% significa variable entera
! significa variable de precisión sencilla
# significa variable de doble precisión
$ significa variable literal (véase el siguiente apartado)

En ausencia de uno de estos distintivos, BASIC normalmente supone precisión sencilla. (Véase, no obstante, DEFINT, DEFSNG, DEFDBL, DEFSTR.)

4.3.5 Cadenas literales

Las cadenas literales o cadenas de texto son sucesiones de varios caracteres (posiblemente sólo uno, o incluso ninguno), encerradas entre comillas ('').

Cadenas típicas son las siguientes:

```
""
"Manual del usuario"
"Juan M. Fuentes"
"23"
```

Observe que la última es una cadena gracias a las comillas; sin ellas sería, por supuesto, un número.

4.3.6 Variables literales

Los nombres de las variables literales se distinguen de todos los demás tipos por el sufijo $. El máximo número de caracteres que pueden ser almacenadas en una cadena es 255; el mínimo es 0 (la cadena vacía, ""). Dentro de esta limitación, las cadenas pueden contener caracteres cualesquiera (código ASCII del 0 al 255).

4.3.7 Organización de variables (matrices)

Todas las variables que hemos descrito hasta ahora han sido independientes unas de otras. Esto es perfectamente válido cuando las variables no son muchas y el programa se limita a captar la información, manipularla y escribir algún resultado. Sin embargo, en ocasiones se necesita guardar listas o tablas de información en variables y preservar la relación que pueda haber entre los datos.

Una posibilidad es dar a las variables una serie de nombres que ponga de manifiesto esa relación. Por ejemplo, si es usted aficionado a la música y quiere guardar los títulos de sus discos, puede preparar variables literales que se llamen disco1$, disco2$, disco3$, etc.

Aunque a usted este sistema de denominación le parezca obvio, a BASIC le pasa totalmente desapercibido; no observa ningún orden en los nombres ni encuentra que tengan nada en común. Por ejemplo, para escribir el título del primer disco se necesita la instrucción

```
PRINT disco1$
```

Para el del tercer disco hace falta una instrucción diferente:

```
PRINT disco3$
```

Y así sucesivamente para todos los discos de la colección. Y lo que es aun más grave: no hay forma sencilla (con las órdenes descritas hasta ahora) de construir un programa que realice tareas tan elementales como escribir el título de un disco a partir de su número.

Como era de esperar, BASIC tiene solución para estos problemas. BASIC permite agrupar variables para formar las denominadas *matrices*. Cada variable del grupo es un *elemento de la matriz*.

Para concretar, una matriz es un grupo de variables, todas las cuales tienen el mismo nombre y son del mismo tipo. En su forma más sencilla, las matrices se reducen a *listas* o *vectores* (este último es el nombre más correcto). Los elementos del vector se distinguen unos de otros por un número que se escribe entre paréntesis a la derecha del nombre de la variable. Ese número se llama *índice*. Es el 0 para el primer elemento, el 1 para el segundo, etc. (No obstante, se puede hacer que BASIC de al primer elemento el índice 1; véase OPTION BASE.)
Las matrices se pueden formar con cualquiera de los tipos de variable descritos más arriba. Por ejemplo, disco$ es una variable literal, disco$(6) es una matriz literal y disco(3) es una matriz numérica. Observe que todas estas variables son distintas entre sí.

Las variables matriciales pueden ser utilizadas igual que las no matriciales del mismo tipo, pero su aplicación principal es la que hemos mencionado: el proceso de listas y tablas de información. Por ejemplo, para resolver el problema que planteábamos antes, valdría el siguiente programa:

```
10 disco$(0)="El pájaro de fuego"
20 disco$(1)="Las cuatro estaciones"
30 disco$(2)="Peer Gynt"
40 INPUT "¿De qué número de disco quiere saber el título";número
50 PRINT disco$(número)
```

Las líneas 10-30 asignan valores a los tres primeros elementos de la matriz disco$( ). La línea 40 pregunta al usuario el número del disco. La línea 50 escribe la cadena literal que está almacenada en la correspondiente variable de la matriz.

Si usted va a utilizar una matriz de más de diez elementos, o bien si necesita una de menos elementos y no quiere desperdiciar espacio de memoria, debe empezar por 'dimensionar' la matriz con la orden DIM. Esta orden va seguida del nombre de la matriz y del máximo valor del índice que se piensa utilizar, puesto entre paréntesis. Por ejemplo,

```
10 DIM disco$(3)
```

Esta matriz, disco$( ), es de una sola dimensión (o sea, sus elementos se identifican con un solo índice); pero en general las matrices pueden tener varias dimensiones. Por ejemplo, si quisiéramos registrar la temperatura en diferentes puntos de un recinto, podríamos usar temp(x,y,z), donde x, y, z son las coordenadas del punto. Si además queremos registrar la evolución de la temperatura con el tiempo, podemos incluir una cuarta dimensión: temp(x,y,z,t). Las matrices de más de una dimensión siempre tienen que ser dimensionadas con DIM [por ejemplo, DIM temp(10,10,15)].

Nótese que las matrices no son variables en sí; las variables son los elementos de la matriz. Una matriz no es más que una forma de organizar variables para facilitar su manejo en los programas.

El procedimiento más rápido y fácil de asignar valores a las variables de una matriz es utilizar las palabras clave DATA y READ. Compare los dos fragmentos de programa siguientes; ambos asignan valores a la matriz 'días'. El primero combina DATA y READ en un bucle de tipo FOR ... NEXT; las instrucciones son compactas y fáciles de entender. El segundo método, en cambio, es más torpe, sobre todo cuando la matriz está formada por muchas variables. El bucle FOR ... NEXT está explicado en la sección 5.2.

```
10 DIM días(12)
20 DATA 31,28,31,30,31,30,31,31,30,31,30,31
30 FOR mes=1 TO 12
40 READ días(mes)
50 NEXT
```

```
10 DIM días(12)
20 días(1)=31
30 días(2)=28
40 días(3)=31
50 días(4)=30
60 días(5)=31
70 días(6)=30
80 días(7)=31
90 días(8)=31
100 días(9)=30
110 días(10)=31
120 días(11)=30
130 días(12)=31
```

4.3.8 Elección de tipos de variables

Las variables se dividen, en una primera clasificación, en numéricas y literales. En segunda clasificación, las variables numéricas se subdividen en enteras y no enteras (precisión doble o sencilla).

Las variables literales son las únicas que pueden contener texto; también se puede almacenar en ellas información numérica, a condición de que ésta no vaya a ser manipulada en operaciones matemáticas.

Las variables enteras son las que se debe utilizar siempre que no interese la parte decimal de los números y a condición de que éstos queden dentro del margen de −32768 a +32767. Estas variables ocupan menos espacio en la memoria que las no enteras; los cálculos con ellas son más rápidos y más precisos (dentro de sus limitaciones).

En los casos en los que las variables no puedan ser enteras, las de precisión sencilla serán adecuadas para la mayor parte de las aplicaciones. Como orientación se puede decir que sólo se necesita la doble precisión cuando se requiere trabajar con precisión de más de seis cifras significativas. Las variables de doble precisión ocupan mayor espacio en la memoria y hacen los cálculos considerablemente más lentos.

Una aparente torpeza de los ordenadores consite en que, mientras que los humanos pensamos y calculamos en el sistema de numeración decimal, los ordenadores calculan en binario y sólo convierten los resultados a decimal para ofrecérselos al usuario. Esto implica que ciertas fracciones que nosotros expresamos con total precisión en decimal no son manejadas por BASIC con la misma exactitud. El análisis detallado de esta cuestión queda fuera del alcance de este manual. Lo que sí podemos recomendarle es que redondee los resultados de las operaciones (con la función ROUND de BASIC, parte 2) siempre que necesite la máxima precisión.

Las variables matriciales son las idóneas para el almacenamiento y manipulación de toda información que se presente en forma de listas o tablas. Un ejemplo típico de tabla de datos es una lista de nombres, señas y teléfonos. Tal información se puede guardar en tres matrices independientes, para el nombre, las señas y el teléfono, conectadas a través del número de índice [por ejemplo, nombre$(12), señas$(12) y tel%(12) darían toda la información relativa al duodécimo individuo].

4.4 Más sobre entradas y salidas (PRINT, LPRINT)

El objetivo de todo programa de ordenador es producir una 'salida', esto es, resultados tangibles y utilizables. Hay dos tipos principales de salida: la salida visible que se emite por la pantalla o la impresora, producida por PRINT y LPRINT, y la salida invisible, consistente en datos que otros programas pueden utilizar para producir finalmente salida visible y grabada normalmente en ficheros de datos en el disco (véase también la descripción de la orden CHAIN). En lo que queda de este capítulo vamos a explicar la utilización de PRINT y LPRINT; la salida hacia el disco está descrita en el capítulo 7.

4.4.1 Salida por la pantalla (PRINT)

La orden que envía la salida de los programas a la pantalla es PRINT (escribir, imprimir). La hemos utilizado ampliamente en ejemplos anteriores, pero en forma limitada. Usted habrá observado que los resultados de esos programas se presentaban en la pantalla un tanto desorganizados: ocupando dos líneas cuando habrían quedado mucho mejor presentados en una sola, sin la debida alineación en vertical, etc. Las instrucciones PRINT se pueden modular con otras palabras clave para controlar el aspecto de la información que se emite por la pantalla.

Toda instrucción PRINT empieza con la palabra clave PRINT, la cual puede ir seguida de los elementos escribibles y de elementos que controlan la forma de escribirlos.

La versión más simple consiste en la palabra PRINT sola. Su efecto es hacer saltar el cursor al principio de la línea siguiente; su aplicación más frecuente, producir líneas en blanco.

Los elementos escribibles van separados por comas (,) o por signos de punto y coma (;). Si un elemento va precedido de una coma, se lo escribe en la «zona de escritura» siguiente a la ocupada por el elemento anterior. En cambio, si el elemento va precedido del punto y coma, se lo escribe inmediatamente a la derecha del elemento anterior. Las zonas de escritura dividen la pantalla a lo ancho en columnas de 15 caracteres cada una; son similares a los topes de tabulación en las máquinas de escribir. Escriba y ejecute el siguiente programa:

```
10 PRINT "Zonas de escritura:"
20 PRINT "1","2","3","4","5","6"
```

La anchura de las zonas de escritura se puede modificar con la orden ZONE(n), donde n es la nueva anchura en caracteres.

Los elementos escribibles pueden ser constantes numéricas o literales, variables o expresiones, o bien los *modificadores de escritura* SPC(n) y TAB(n). SPC(n) escribe n espacios. TAB(n) lleva el cursor a la columna número n de la línea actual, o a la columna n de la línea siguiente cuando el cursor ya está a la derecha de aquélla.

Cuando se completa la instrucción PRINT, el cursor salta al principio de la línea siguiente, a menos que la instrucción termine con el signo de punto y coma o con SPC o TAB, pues en tal caso el cursor se queda donde está. Ejecute el siguiente programa y observe cómo el punto y coma mantiene juntas las salidas de las líneas 30 y 40:

```
10 PRINT "Líneas"
20 PRINT "diferentes"
30 PRINT "La misma ";
40 PRINT "línea"
```

Control del formato de escritura. El efecto de la instrucción PRINT se puede controlar más estrechamente utilizando (USING) una *plantilla de formato* en la que se especifica la distribución de los elementos escribibles. La palabra clave USING está descrita en la parte 2 de este manual. El siguiente ejemplo da una idea de la flexibilidad alcanzable.

Si un programa trabaja con sumas de dinero, será deseable tabular las cantidades de forma bien organizada. A primera vista podría parecer que para ello basta con incluir cláusulas TAB, como en la línea 50 del siguiente programa:

```
10 suma(1)=1.24
20 suma(2)=0.333
30 suma(3)=1444
40 FOR contador=1 TO 3
50 PRINT TAB(40) suma(contador)
60 NEXT
```

Introduzca y pruebe este programa. Los números quedan alineados en vertical, pero por su primer dígito, no por el punto decimal, que sería lo correcto.

Cambie la línea 50 y pruebe la nueva versión del programa:

```
50 PRINT TAB(40) USING "######,.##";suma(contador)
```

La plantilla de formato "######,.##" ha especificado que los números se deben escribir reservando seis posiciones a la izquierda del punto decimal y dos a la derecha (la tercera cifra se redondea); además, las cifras que quedan a la izquierda del punto decimal se agrupan de tres en tres y los grupos se separan por comas. (Éste es el sistema habitual de presentación de números en países de habla inglesa.)

Control de la pantalla. La orden PRINT también sirve para controlar la pantalla y el cursor «escribiendo» en la pantalla *códigos de control* y *secuencias de escape*. Los códigos de control son caracteres cuyo código ASCII está entre 0 y 31. Las secuencias de escape son sucesiones (cortas) de caracteres, todas las cuales empiezan por el código ASCII 27 (llamado 'ESC'). Los efectos de estos códigos y secuencias están descritos con todo detalle en la 'Guía de CP/M Plus' del PCW8256; aquí daremos un breve resumen.

Ciertas versiones de BASIC incluyen una orden CLS que borra la pantalla de texto y lleva el cursor a su 'posición inicial' (extremo superior izquierdo de la pantalla). En Mallard BASIC, esta acción se realiza con la secuencia de escape ESC "E" ESC "H".

ESC "H" lleva el cursor a la posición inicial.

Para llevar el cursor a cualquier otra posición se puede utilizar ESC "Y" seguida de CHR$(32+y) CHR$(32+x), donde y es la fila (0 es la superior) y x es la columna (0 es la primera por la izquierda).

Para facilitar su manejo en el programa, puede ser conveniente asignar éstas y otras secuencias de escape a variables literales cuyos nombres sean más representativos de los respectivos efectos. Por ejemplo, en las primeras líneas del programa se puede poner:

```
10 escape$ = CHR$(27)
20 inic$ = escape$ + "H"
30 borrar$ = escape$ + "E" + inic$
40 mover.derecha$ = escape$ + "C"
50 mover.cualq$ = escape$ + "Y"
```

Entonces, para borrar la pantalla, se da la orden

```
PRINT borrar$
```

Para llevar el cursor a la posición inicial,

```
PRINT inic$
```

Para mover el cursor hacia la derecha,

```
PRINT mover.derecha$
```

Para llevarlo a la fila 15 y la columna 22,

```
fila = 15
columna = 22
PRINT mover.cualq$;CHR$(32+fila);CHR$(32+columna)
```

(El punto y coma que hemos puesto al final de todas estas órdenes impide que el cursor salte al principio de la línea siguiente.)

En el capítulo 5 describiremos un método más cómodo para llevar el cursor a cualquier lugar de la pantalla, basado en una 'función definida por el usuario'.

4.4.2 Salida por la impresora (LPRINT)

El funcionamiento de LPRINT es análogo al de PRINT; de hecho, puede ser controlada por los mismos modificadores de escritura que PRINT.

Las diversas funciones de la impresora también se controlan mediante códigos de control y secuencias de escape, aunque no con los mismos que la pantalla. (Véase el apéndice II de la 'Guía de CP/M Plus'.)

Capítulo 5

Diseño de programas largos

Aunque, como hemos dicho, un programa no sea más que una serie de instrucciones escritas en cierto número de líneas, en muchos aspectos es bastante más que la suma de sus partes. Uno de los factores determinante del éxito de un programa es la forma de organizar las instrucciones que lo componen.

La organización de los programas, su estructura, es el objeto de este capítulo.

5.1 Estructuración de los programas

Los programas cortos son siempre fáciles de entender y seguir, por desorganizados que estén; basta con entender cada una de las instrucciones para comprender el funcionamiento global del programa: la línea 10 capta un número, la línea 20 lo multiplica por 3, la línea 30 escribe el resultado en la pantalla, etc.

Pero en cuanto el programa pase de unas diez líneas o las instrucciones sean relativamente complejas, las cosas empiezan a no estar tan claras. Y cuanto más difícil de entender sea el programa, más difícil será escribirlo, y mucho más modificarlo o corregirlo.

La única forma de abordar con posibilidades de éxito un programa largo o complejo es subdividirlo en porciones pequeñas y darles unos nombres que expliquen cuál va a ser su misión.

Para subdividir el programa en secciones hay que meditar cuáles son las tareas elementales que debe realizar y luego escribir las instrucciones necesarias para cada una de ellas. Finalmente, las diversas secciones se ensamblan, dejándolas separadas por una línea de programa que no contenga más que el signo de dos puntos (:).

Para indicar en el propio programa qué tarea realiza cada sección, se escribe el texto adecuado a la derecha de la palabra clave REM. Ésta es una curiosa orden: hace que BASIC ignore el resto de la línea. Así pues, a continuación de REM se puede escribir cuantas aclaraciones sean necesarias para explicar el funcionamiento de las diversas secciones del programa.

En lo que queda de esta parte 1, usted tendrá ocasión de ver programas ejemplo con diversos grados de estructuración. Para hacerse una idea de las diferencias que puede haber entre dos programas similares, compare los dos siguientes:

```
10 título$ = "Programa PVP"
20 recargo=0.5
30 PRINT título$:PRINT
40 INPUT "¿Precio del artículo";precio
50 INPUT "¿Nombre del artículo";nombre$
60 pvp.final=precio*(1+recargo)
70 iva=0.15
80 INPUT "¿Cuántas unidades";número
90 valor=número*pvp.final*(1+iva)
100 PRINT "Las existencias totales de ";nombre$;" se venderán por ";valor;
    "pesetas"
```

```
10 REM * preparación de variables **
20 :
30 iva=0.15
40 recargo=0.5
50 título$="Programa PVP"
60 :
70 REM * preparación de la pantalla **
80 :
90 PRINT título$:PRINT
100 :
110 REM ** captar entradas **
120 :
130 INPUT "¿Precio del artículo";precio
140 INPUT "¿Nombre del artículo";nombre$
150 INPUT "¿Cuántas unidades";número
160 :
170 REM ** calcular valor incluyendo recargo e iva **
180 :
190 pvp.final=precio*(1+recargo)
200 valor=número*pvp.final*(1+iva)
210 :
220 REM ** salida por la pantalla: valor total **
230 :
240 PRINT "Las existencias totales de ";nombre$;" se venderán por ";valor;
    "pesetas"
```

El segundo programa es algo más largo, pero tiene la ventaja de ser mucho más fácil de entender y corregir gracias a que está perfectamente estructurado. Por ejemplo, compruebe si ha asignado valor a la variable 'iva' antes de usarla. En el primer programa la respuesta no es inmediata; en el segundo, hemos hecho todas las asignaciones juntas, lo que facilita las comprobaciones de este tipo.

Si usted va a escribir programas de cierta complejidad o longitud, esfuércese por diseñarlos como el segundo de este ejemplo. Modificar un programa como el primero puede llegar a ser una pesadilla.

5.2 Secuencias y bucles

Todos los programas que hemos escrito hasta ahora se han ejecutado de forma secuencial, empezando por la línea de número más bajo y terminado con la de número más alto. Sin embargo, este orden de ejecución «lineal» puede ser modificado de varias formas.

La primera se basa en la orden GOTO (ir a), que va seguida por un número de línea. Cada vez que BASIC encuentra esta instrucción, salta a la línea especificada y continúa a partir de ella.

A primera vista esto no parece nada impresionante, pero observe cómo queda el programa "pvp" cuando le añadimos la línea siguiente. Cargue el programa escribiendo

```
LOAD "pvp"
```

y luego introduzca la nueva línea:

```
40 GOTO 10
```

El programa completo es ahora:

```
10 INPUT "Escriba el coste del artículo y luego pulse RETURN",coste
20 beneficio=0.2
30 PRINT "El precio que se debe cobrar es";coste+coste*beneficio
40 GOTO 10
```

Ejecute el programa y observe qué cómodo es ahora calcular el precio de venta al público de varios productos; ya no hay que ejecutarlo una vez para cada artículo, sino que el programa pregunta el costo del siguiente artículo cada vez que termina de escribir un PVP.

La repetición de un grupo de instrucciones es lo que se llama *formar un bucle*; las instrucciones que se repiten constituyen un *bucle*.

Esta nueva versión del programa "pvp" es mucho mejor que la anterior, pero tiene el defecto de que el programa no termina nunca, a menos que lo interrumpamos por la vía drástica de pulsar control-C. Este código de control se genera pulsando ALT + C, o bien STOP.

Esto sirve para interrumpir un bucle cuando no importa que se detenga el programa, pero no sirve de nada cuando lo que se pretende es salir del bucle y continuar ejecutando otras secciones del programa. Necesitamos, pues, otra forma más razonable de salir del bucle:

Los bucles se utilizan tan frecuentemente, que BASIC dispone de diversas órdenes para formarlos y controlarlos.

Hay un tipo de bucle que empieza con la orden FOR (para) y termina con la orden NEXT (siguiente). Su forma más sencilla es:

```
FOR contador=principio TO fin
  instrucciones
NEXT
```

Cuando BASIC encuentra una instrucción FOR, lo primero que hace es comprobar si *principio* es igual o menor que *fin*. Si no lo es, salta el bucle completo. Si lo es, BASIC da a la variable numérica *contador* el valor de la variable numérica *principio* y obedece las *instrucciones* hasta que llega a NEXT. Entonces suma 1 a *contador* y comprueba si este nuevo valor es mayor que el de la variable numérica *fin*. Si no lo es, vuelve a ejecutar las instrucciones internas al bucle, a incrementar *contador* y a compararlo con *fin*. El proceso se repite hasta que el valor actualizado de *contador* sea mayor que *fin*. Entonces BASIC abandona el bucle y continúa la ejecución a partir de la instrucción siguiente a la orden NEXT.

Los bucles de este tipo se llaman 'bucles FOR ... NEXT', por razones obvias.

El bucle FOR ... NEXT es adecuado cuando se sabe de antemano cuántas veces ha de repetirse la ejecución de un grupo de instrucciones. Por ejemplo, para escribir una columna de asteriscos:

```
10 INPUT "¿Cuántos asteriscos";asteriscos
20 FOR a=1 TO asteriscos
30 PRINT "*"
40 NEXT
```

También es adecuado cuando interesa utilizar la variable *contador* en alguna instrucción del bucle. Por ejemplo, en un programa que dé las «tablas de multiplicar»:

```
10 INPUT "¿Qué tabla de multiplicar desea";por%
20 FOR número%=1 TO 2
30 PRINT número%*por%
40 NEXT
```

Como puede observar, la variable del contador, no sólo sirve para controlar el bucle, sino también para generar la salida por la pantalla.

5.3 Toma de decisiones

Con la introducción de los bucles hemos dado un gran paso adelante; ya sabemos cómo utilizar el ordenador para realizar cálculos, almacenar y recuperar la información y repetir grupos de instrucciones un número concreto o infinito de veces. Pero esto no basta para ciertas aplicaciones; ¿recuerda el problema que nos planteaba la salida del bucle que creamos antes con la orden GOTO?

Lo que nos falta es la capacidad de tomar decisiones condicionadas. Sin ella, el ordenador sólo puede funcionar como máquina calculadora; con ella, podremos escribir programas realmente versátiles que adapten su funcionamiento a la información que vayan recibiendo y que incluso, en casos extremos, parezcan «inteligentes».

5.3.1 Elección de alternativas

La toma de decisiones en BASIC se apoya principalmente en dos órdenes: IF (si) y ON (según sea).

IF. La palabra clave IF va seguida de una 'expresión lógica' comprobable por BASIC y cuyo resultado sólo pueda ser 'verdadero' o 'falso'; por ejemplo, 'el contador es mayor que el límite'. En la forma más sencilla de la orden, si BASIC observa que la expresión lógica es verdadera, obedece las instrucciones que figuren a continuación de la palabra clave THEN en la misma línea. Si la expresión es falsa (por ejemplo, en el caso de que el contador sea 4 y el límite 6), BASIC no ejecuta esas instrucciones y salta a la siguiente línea del programa.

Esta versión sencilla tiene la siguiente forma:

```
      IF  expresión-lógica  THEN  instrucciones
expresión comprobada              se ejecuta si es verdadera
```

(Leído en castellano: 'si ... entonces ...'.) Nótese que los cuatro elementos de la instrucción tienen que estar en la misma línea de programa. Éste es uno de los casos en los que necesariamente hay que poner varias instrucciones (separadas por el signo de dos puntos) en una misma línea.

Volviendo al programa "pvp", podríamos decidir que el programa debe abandonar el bucle cuando el usuario introduzca el número 0 como coste del artículo. La expresión lógica que hay que comprobar es si el coste es igual a cero, o sea, si 'coste=0'. Para salir del bucle necesitamos una orden GOTO seguida del número de una línea que esté fuera de él (línea 50). Introduzca en el programa las dos líneas siguientes:

```
15 IF coste = 0 THEN GOTO 50
      expresión      se ejecuta si es verdadera
50 PRINT "Y ahora el resto del programa ..."
```

El programa completo ha quedado de esta forma:

```
10 INPUT "Escriba el coste del artículo y luego pulse RETURN",coste
15 IF coste=0 THEN GOTO 50
20 beneficio=0.2
30 PRINT "El precio que se debe cobrar es"coste+coste*beneficio
40 GOTO 10
50 PRINT "Y ahora el resto del programa ..."
```

Ejecute el programa para comprobar que funciona correctamente. Introduzca números distintos de cero y verá que el bucle se repite. Después introduzca un 0; aparecerá el mensaje de la línea 50 y el programa terminará.

En la instrucción IF se puede incluir también la palabra clave ELSE (si no) para especificar un segundo grupo de instrucciones que se ejecutarán cuando la expresión lógica sea falsa:

```
    IF expresión-lógica THEN instrucciones-1 ELSE instrucciones-2
expresión comprobada      si es verdadera     si es falsa
```

Observe, por ejemplo, el siguiente fragmento de un programa:

```
90 ...
100 saldo=saldo+operación
110 IF saldo<0 THEN PRINT "**Saldo negativo":PRINT "de ";-saldo ELSE
    PRINT "Saldo positivo de";saldo
120 INPUT "¿Qué quiere hacer ahora (F-terminar, T-transferencia, D-depósito";
    respuesta$
130 ...
```

La línea 110 utiliza una instrucción IF ... THEN ... ELSE para emitir uno de los dos mensajes posibles, en función del saldo de una cuenta:

```
                ↗ THEN PRINT "**Saldo negativo":PRINT "de "; – saldo
IF saldo < 0 ⟨
                ↘ ELSE PRINT "Saldo positivo": PRINT "de";saldo
```

Obsérvese que, igual que en el caso de IF ... THEN, la instrucción IF ... THEN ... ELSE ha de estar completa, con todos sus elementos, en una sola línea.

ON. La palabra clave ON va seguida de una expresión numérica, de GOTO y de una lista de números de línea:

ON *expresión-numérica* GOTO *línea1,línea2,línea3,*etc.

(Leído en castellano: 'según sea ... ir a ...'.) La expresión numérica se calcula en el momento en que se ejecuta la instrucción, obteniéndose como resultado el número 'n'. Si n está entre 1 y el número de líneas mencionadas en la lista, la ejecución del programa continúa a partir de la línea que figura en n-ésimo lugar en la lista. Si n es menor que 1 o mayor que el número de líneas mencionadas en la lista, la ejecución continúa en la instrucción siguiente.

ON proporciona una forma compacta de elegir entre varias alternativas. Es equivalente a una sucesión de instrucciones IF:

```
IF expresión-numérica=1 THEN GOTO línea1
IF expresión-numérica=2 THEN GOTO línea2
IF expresión-numérica=3 THEN GOTO línea3
etc.
```

(ON también puede ir seguida de GOSUB, en lugar de GOTO, según explicaremos más adelante.)

5.3.2 Comprobación de expresiones lógicas

Como hemos visto, la *expresión-lógica* es normalmente una afirmación acerca de la relación que existe entre dos variables o constantes. Tales afirmaciones se construyen a base de *operadores relacionales*:

```
a<b     verdadera si a es menor que b
a<=b    verdadera si a es igual o menor que b
a=b     verdadera si a es igual a b
a<>b    verdadera si a es distinto de b
a>=b    verdadera si a es igual o mayor que b
a>b     verdadera si a es mayor que b
```

Expresiones lógicas típicas son las siguientes:

```
contador1<contador2
nombre$=""
clave<>puntero%*2+1
indicador.no.fichero
```

Cuando BASIC evalúa las expresiones lógicas obtiene un resultado numérico que es 0 para las falsas o −1 para las verdaderas. En lugar de una expresión lógica podemos poner también una constante o una variable numérica: BASIC las considera «verdaderas» a menos que valgan 0.

Las expresiones lógicas pueden ser modificadas o combinadas con otras mediante los operadores lógicos. Los más útiles son NOT, AND y OR.

NOT invierte el valor de la expresión lógica a la que se aplica. Por ejemplo, si 'a>6' es verdadera, 'NOT a>6' es falsa.

OR combina dos expresiones lógicas de forma tal que la expresión global es verdadera si alguna de las intervinientes lo es. Así, '1=2 OR 3=3' es verdadera porque la segunda expresión es verdadera, y basta con que lo sea una.

AND combina dos expresiones lógicas de forma tal que la expresión global es verdadera solamente en el caso de que lo sean las dos expresiones intervinientes. Así, '1=2 AND 3=3' es falsa porque la primera expresión es falsa.

(En realidad, lo que hacen los operadores lógicos es realizar operaciones bit a bit con sus argumentos; pero esto, aplicado a los valores −1 y 0 con los que BASIC representa los valores lógicos 'verdadero' y 'falso', equivale a las reglas descritas.)

5.3.3 Control de bucles

IF hace que BASIC ejecute una vez un grupo de instrucciones **si** se cumple cierta condición. WHILE es una orden de BASIC que hace que se ejecute un bucle **mientras** se cumpla cierta condición.

El bucle de tipo WHILE empieza con esta palabra clave y termina con WEND. Su forma general es la siguiente:

WHILE *expresión-lógica*
  instrucciones
WEND

Cuando BASIC ejecuta la instrucción WHILE, lo primero que hace es evaluar la *expresión-lógica*; si es verdadera, ejecuta las instrucciones del bucle, hasta la orden WEND; al llegar a ésta, vuelve a evaluar la expresión lógica como si acabara de entrar en el bucle. No sale del bucle mientras la expresión lógica no deje de ser verdadera; entonces salta a la instrucción siguiente a WEND.

Los bucles WHILE son idóneos en los casos en los que no se sabe de antemano cuántas veces se va a ejecutar el bucle. Puesto que la salida del bucle depende del valor lógico de una expresión, todo lo que tiene que hacer el programador es cerciorarse de que esa expresión tome el valor 'falso' en el momento adecuado.

5.4 Interrupción de los programas

Los programas de BASIC no concluyen mientras no terminan de ejecutar la última línea, a menos que el programador o el usuario decidan otra cosa. El usuario puede detener el programa pulsando control-C, que se obtiene con [ALT]+C o con [STOP]. Esto sirve para interrupciones de emergencia.

Pero BASIC dispone de dos órdenes, END y STOP, con las que se controla la interrupción desde dentro del propio programa.

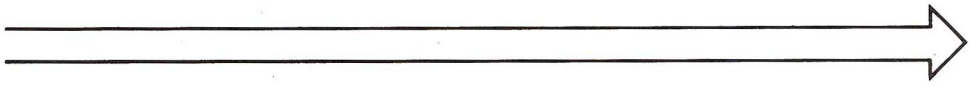
END termina el programa de la misma forma que cuando se agotan las líneas. Su aplicación principal es terminar el programa en una línea que no sea la última.

STOP detiene el programa y muestra en la pantalla el número de línea en la que ha ocurrido la interrupción. La aplicación habitual de STOP es insertar puntos de interrupción que permitan observar el funcionamiento de determinadas secciones del programa cuando se está tratando de detectar errores. (La orden CONT reanuda la ejecución del programa a partir de la situación en que fue interrumpido.)

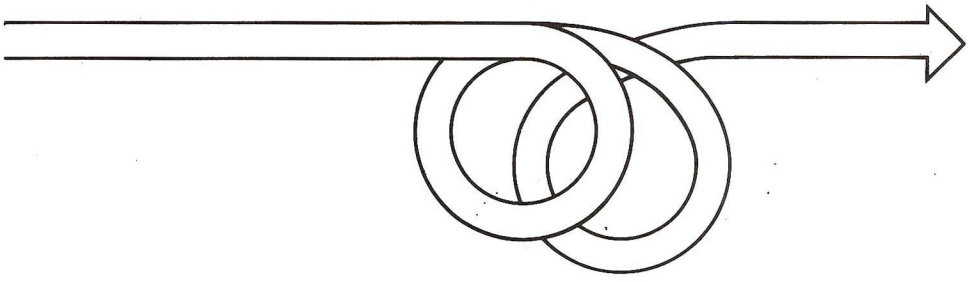
5.5 Organización de los programas

5.5.1 Repaso de la estructura de los programas

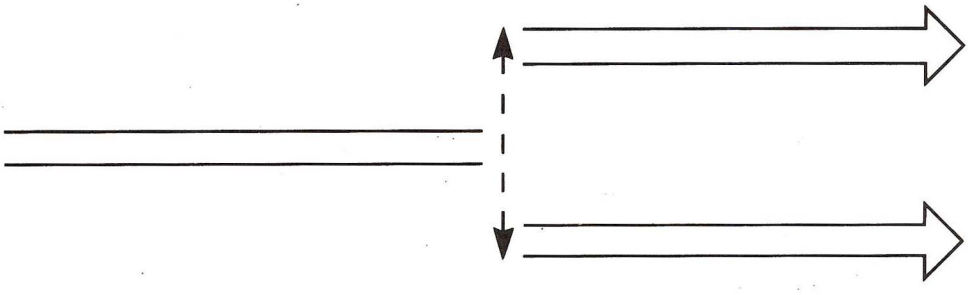
Ya hemos visto las diversas formas de controlar el orden en que se ejecutan las líneas de un programa. En los primeros ejemplos de este capítulo cada línea se ejecutaba una sola vez, en orden de números de línea ascendentes, del principio al fin del programa. Eran programas 'en línea recta':



Después vimos programas con bucles; el programa sigue evolucionando del principio al fin, pero repite algún tramo del camino:



Finalmente, hemos visto programas en los que se toman decisiones condicionadas: bifurcaciones del camino en las que el programa decide si ir por la derecha o por la izquierda dependiendo de la información que se le ha suministrado o de la que él mismo ha calculado:



Estas tres estructuras están adaptadas a diferentes tipos de tareas: la 'línea recta', para trabajos sencillos; los bucles, para trabajos repetitivos; las decisiones condicionadas, para trabajos cuyos detalles dependan de información de que el programa disponga en cada momento.

Al diseñar un programa se debe decidir qué estructura o combinación de estructuras es la más adecuada para cada una de las subtareas que el programa tenga que realizar.

5.5.2 Subrutinas

BASIC ofrece un recurso más para facilitar la organización de las secciones de los programas. Se trata de las *subrutinas*, grupos de instrucciones en las que se programan ciertas tareas comunes a distintas secciones del programa. Esta estructura se basa en las órdenes GOSUB (ir a subrutina) y RETURN (retornar).

Observe los siguientes fragmentos de un programa:

```
...
80 FOR a=1 TO veces%
90 PRINT nombre$(a)
100 NEXT
110 PRINT "***Pulse cualquier tecla para borrar la pantalla***"
120 WHILE INKEY$="":WEND
130 PRINT CHR$(27);"E"
...
200 FOR a=1 TO preguntas
210 PRINT ecuación$(a),resultado(a)
220 NEXT
230 PRINT "***Pulse cualquier tecla para borrar la pantalla***"
240 WHILE INKEY$="":WEND
250 PRINT CHR$(27);"E"
...
```

Como puede ver, las líneas 110 a 130 y 230 a 250 son idénticas, aparte de sus números de línea. Esta repetición es un despilfarro de trabajo mecanográfico y de espacio de memoria, tanto más grave cuantas más veces aparezca el mismo grupo de líneas en el programa. Lo que se debería hacer es escribir esas líneas solamente una vez, en una subrutina.

Una subrutina no es más que una sucesión de instrucciones que termina con la orden RETURN y que puede ser invocada mediante la orden GOSUB seguida del número de la primera línea de la subrutina.

La subrutina debe estar aislada del resto del programa para asegurar que sólo se llega a ella intencionadamente, a través de GOSUB. El mejor sitio para las subrutinas es el final del programa, después de una instrucción END, como en el programa siguiente.

Utilizando subrutinas, el programa queda de esta forma:

```
...
80 FOR a=1 TO veces%
90 PRINT nombre$(a)
100 NEXT
110 GOSUB 1000
...
200 FOR a=1 TO preguntas
210 PRINT ecuación$(a),resultado(a)
220 NEXT
230 GOSUB 1000
...
990 END
999 REM *Subrutina que borra la pantalla cuando se pulsa una tecla*
1000 PRINT "***Pulse cualquier tecla para borrar la pantalla***"
1010 WHILE INKEY$="":WEND
1020 PRINT CHR$(27);"E"
1030 RETURN
```

GOSUB es similar a GOTO, con la diferencia de que cuando BASIC encuentra la orden RETURN al final de la subrutina, 'retorna' a la instrucción siguiente a aquélla que contiene el GOSUB que invocó la subrutina. Si esto no queda claro, pruebe los siguientes programas:

```
10 REM ejemplo que ilustra GOTO
20 PRINT "Primero ésta, ";
30 GOTO 60
40 PRINT "Esta instrucción no se ejecuta nunca"
50 END
60 PRINT"después ésta, a causa del GOTO"
```

```
10 REM ejemplo que ilustra GOSUB
20 PRINT "Primero ésta, ";
30 GOSUB 60
40 PRINT "y finalmente ésta, cuando la rutina ya ha terminado"
50 END
60 PRINT "después ésta, escrita por la subrutina, ";
70 RETURN
```

Las subrutinas también sirven aun en los casos en los que los grupos de líneas a los que sustituyen no sean exactamente iguales; se requiere entonces utilizar inteligentemente las variables y las decisiones condicionadas. Por ejemplo, si la subrutina tiene que imprimir diferentes mensajes en función de la zona del programa que la invoque, lo que se hace es asignar a una variable el texto necesario y dejar que la subrutina imprima el valor de la variable, no un texto constante.

Pero las rutinas no sólo permiten escribir programas más compactos, sino que también facilitan la corrección y la modificación de los programas: cuando hay que cambiar algo, basta con hacerlo una vez en una subrutina, no muchas veces a lo largo de todo el programa.

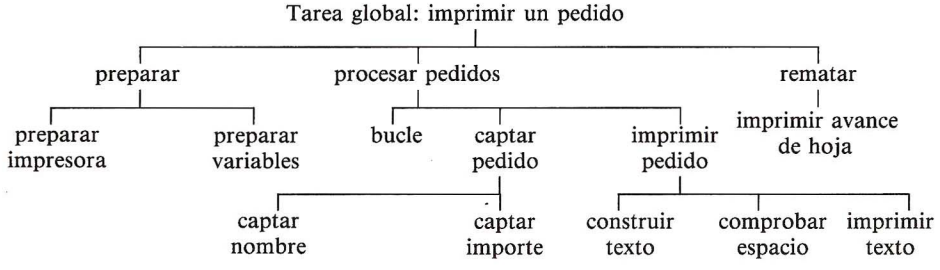
GOSUB se puede combinar también con la palabra clave ON para seleccionar, en función del valor de una expresión numérica, una de entre varias subrutinas mencionadas en una lista. La forma de la instrucción es la siguiente:

ON *expresión-numérica* GOSUB *línea1,línea2,línea3,*etc.

La expresión mumérica se calcula en el momento en que se ejecuta la instrucción, obteniéndose como resultado el número 'n'. Si n está entre 1 y el número de líneas mencionadas en la lista, se invoca la n-ésima subrutina. Si n es menor que 1 o mayor que el número de líneas mencionadas en la lista, no se invoca ninguna subrutina y la ejecución continúa en la instrucción siguiente.

5.5.3 Diseño modular de programas

Las subrutinas son útiles en todos los programas, incluso en los que no tengan muchas líneas repetidas, pues facilitan la división del programa en secciones lógicas. Los programas escritos de esta manera están desglosados en distintos niveles jerárquicos, y esto los hace más fáciles de diseñar, desarrollar y entender. Observe, por ejemplo, la siguiente tarea y su subdivisión:



Cada nivel jerárquico da lugar a un grupo de subrutinas. El nivel superior es el más fácil:

```
10 GOSUB 1000:REM preparar
20 GOSUB 2000:REM procesar pedidos
30 GOSUB 3000:REM rematar
40 END
50 REM ********* SUBRUTINAS *********
```

Ahora hay que diseñar y escribir estas subrutinas. Por ejemplo, la subrutina principal de proceso podría ser:

```
2000 REM *** procesar pedidos ***
2010 "Para preparar un pedido, pulse S y RETURN",respuesta$
2020 WHILE respuesta$="S"
2030 GOSUB 2100:REM captar pedido
2040 GOSUB 2500:REM imprimir pedido
2050 "Para preparar otro pedido, pulse S y RETURN",respuesta$
2060 WEND
2070 RETURN
```

Descendiendo al siguiente nivel,

```
2100 REM ** captar pedido **
2110 GOSUB 2200:REM captar nombre
2120 GOSUB 2300:REM captar importe
2130 RETURN
...
```

Y descendiendo uno más:

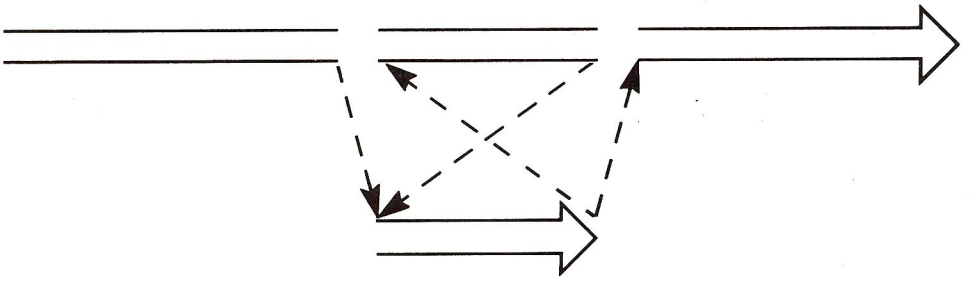
```
2200 REM *captar nombre *
2210 INPUT "Escriba el nombre del cliente y pulse RETURN ",nombre$
2220 IF nombre$="" THEN GOTO 2210
2230 RETURN
...
2300 REM *captar importe *
2310 INPUT "Escriba el importe y pulse RETURN ",importe$
2320 IF importe%="" THEN GOTO 2310
2330 importe=VAL(importe$)
2340 IF importe<0 THEN PRINT "El importe tiene que ser mayor que 0":GOTO 2310
2350 RETURN
...
```

y así sucesivamente.

Al subdividir de esta manera los programas se puede conseguir que la estructura del programa refleje la estructura lógica de la tarea, lo que facilita la búsqueda e identificación de la sección del programa que realiza una subtarea determinada.

Además, este método permite escribir los programas por tramos. Empezando por el superior, se escribe cada vez un nivel y no se pasa al siguiente mientras el actual no funcione perfectamente. Así, cuando se detecta algún error, se tiene muy bien delimitada la zona en que ha podido producirse.

El flujo de un programa basado en subrutinas puede responder a un esquema como el siguiente:



5.5.4 Funciones definidas por el usuario

Las subrutinas proporcionan un medio cómodo de «empaquetar» grupos de instrucciones. BASIC dispone además de un recurso que permite empaquetar expresiones; se trata de las *funciones definidas por el usuario*. Estas funciones pueden ser utilizadas (llamadas) en cualquier lugar del programa **después** de definirlas mediante una instrucción DEF FN.

La instrucción DEF FN consta de cuatro elementos:

DEF FN*nombre-de-función*(*lista-de-parámetros*)=*expresión-funcional*

La llamada a la función tiene la forma:

FN*nombre-de-función*(*lista-de-valores*)

El nombre de la función es elegido por el programador y puede ser cualquier nombre de variable válido.

La lista de parámetros (que tienen que estar entre paréntesis y separados por comas) consiste en uno o varios nombres de variables ('parámetros formales'). Cuando se usa la función, se asigna a estas variables los valores que figuran en las posiciones correspondientes en la llamada a la función. Nótese que estas variables son totalmente independientes de las que pudiera haber en otro lugar del programa con el mismo nombre.

La expresión funcional puede ser cualquier expresión válida para BASIC; generalmente consistirá en una descripción de los cálculos que se ha de realizar con los valores que se le suministran a través de los parámetros formales. El resultado de esos cálculos es el valor que finalmente toma la función.

La lista de valores es un conjunto de valores (constantes, variables o expresiones) separados por comas. Tiene que haber tantos valores como parámetros formales hayan intervenido en la definición; además, cada valor tiene que ser de tipo compatible con el correspondiente parámetro, o sea, ambos numéricos o ambos literales.

He aquí unas cuantas funciones y llamadas típicas:

```
10 DEF FNiva(bruto)=bruto- bruto/1.15
20 importe=100
30 PRINT "Para un importe total de";importe;"el IVA es";FNiva(importe)
```

```
10 DEF FNmedia(n1,n2)=(n1 +n2)/2
20 max%=42:min% =3
30 valor.medio=FNmedia(max%,min%)
40 PRINT valor.medio
```

```
10 esc$=CHR$(27)
20 DEF FNpantalla$(x,y,texto$)=esc$+"Y"+CHR$(32+y)+CHR$(32 +x)+texto$
30 INPUT "Mensaje";mensaje$
40 INPUT "Columna";col%
50 INPUT "Fila";fila%
60 PRINT FNpantalla$(col%,fila%,mensaje$)
70 GOTO 30
```

```
10 log2=LOG(2)
20 DEF FNlog2(x)=LOG(x)/log2
30 INPUT "Número positivo: ",número
40 PRINT "El logaritmo en base 2 es";FNlog2(número)
50 GOTO 30
```

La instruccción de definición lo único que hace es preparar la función para que **después** pueda ser utilizada por el usuario. Es conveniente agrupar todas las definiciones en un mismo sitio, bien al principio del programa, bien en una subrutina de inicialización.

Las funciones definibles por el usuario son una alternativa interesante a las subrutinas, aplicables solamente cuando las acciones requeridas pueden ser encapsuladas en una expresión.

Capítulo 6

Manipulación de la información

Hasta ahora hemos presentado las funciones y los operadores de BASIC en un orden y una profundidad adecuados a los usuarios que establecen su primer contacto con BASIC.

En este capítulo vamos a describir las funciones más detalladamente, clasificándolas en grupos lógicos, de modo que el lector pueda apreciar de qué medios dispone para cada aplicación. Seguramente, al menos en una primera lectura, no vale la pena que usted dedique demasiado tiempo a estudiar las funciones que no parezcan aplicables a sus necesidades concretas. Si más tarde descubre que sí las necesita, puede retroceder hasta este capítulo y leer la sección correspondiente.

Lo que vamos a dar aquí **no** es una descripción completa de todas las funciones y operadores disponibles. Las más complejas y utilizadas con menor frecuencia, o no las mencionamos, o sólo las describimos muy superficialmente. Para más información consúltese la parte 2.

6.1 Manipulación de la información numérica

6.1.1 Aritmética

Las operaciones que se realizan más frecuentemente con la información numérica son la suma (+), la resta (−), la multiplicación (*) y la división (/).

Estas operaciones se aplican igual a los números enteros que a los de coma flotante (precisión sencilla y doble). Hay otros dos operadores de interés especial en la artimética de enteros: se trata de \, que realiza la división entera (o sea, que da la parte entera del cociente), y MOD, que da el resto de la división entera:

```
PRINT 7/2
3.5
PRINT 7\2
3
PRINT 7 MOD 2
1
```

6.1.2 Trigonometría

BASIC dispone de tres funciones trigonométricas, COS, SIN y TAN, que dan los valores del coseno, el seno y la tangente, respectivamente, de los ángulos (expresados en radianes). Estas funciones llevan un único argumento numérico, entre paréntesis. Por ejemplo,

```
opuesto=adyacente*TAN(ángulo)
```

Hay además una función trigonométrica inversa, ATN, que da el ángulo (en radianes) cuya tangente es el número dado:

```
ángulo=ATN(opuesto/adyacente)
```

6.1.3 Signos

Para obtener el valor absoluto de un número (constante, variable o expresión), se le aplica la función ABS. Los números positivos no resultan modificados, pero los negativos se convierten en positivos.

La función SGN sirve para averiguar el signo de un número (constante, variable o expresión). Genera el valor 0 si el número vale 0; el 1 si el número es positivo; y el −1 si el número es negativo. Por ejemplo,

```
100 estado$(0)=" hay pedidos pendientes de servir"
110 estado$(1)=" no hay existencias"
120 estado$(2)=" hay existencias"
...
200 estado=SGN(existencias(artículo))+1
210 PRINT "De ";nombre$(artículo);estado$(estado)
```

6.1.4 Logaritmos y potencias

BASIC dispone de dos funciones logarítmicas: LOG, que calcula logaritmos neperianos o naturales, y LOG10, que calcula logaritmos decimales.

La función inversa de LOG es EXP.

Para elevar un número a una potencia se utiliza el operador ↑. Por ejemplo,

```
PRINT 7 ↑ 4.4
```

El signo ↑ se obtiene con [EXTRA]+U.

6.1.5 Operaciones bit a bit

Hay varios operadores que actúan sobre números enteros al nivel de bits individuales. Los más útiles son los siguientes:

NOT *número*	Invierte todos los bits del número
número1 AND *número2*	Pone a 0 cada bit del resultado, a menos que los bits correspondientes en los dos operandos sean ambos 1
número1 OR *número2*	Pone a 0 cada bit del resultado, a menos que alguno de los bits correspondientes en los operandos sea 1
número1 XOR *número2*	Pone a 0 cada bit del resultado a menos que los bits correspondientes sean 0 en un operando y 1 en el otro.

Estos operadores se utilizan frecuentemente para combinar expresiones lógicas; por ejemplo, para invertir el sentido de una comprobación:

```
WHILE NOT EOF(fichero%)
  INPUT#fichero%,nombre$
WEND
```

(Véase también EQV e IMP en la parte 2.)

6.1.6 Números aleatorios

En ocasiones interesa poder generar números cuyo valor sea impredecible (o al menos difícilmente predecible), esto es, *números aleatorios.*

Estos números se utilizan en las aplicaciones estadísticas, en la generación de claves y en la programación de juegos.

Es muy difícil generar números verdaderamente aleatorios. Lo que BASIC proporciona es 'números pseudoaleatorios', perfectamente adecuados para la mayor parte de las aplicaciones.

(Véase también RND y RANDOMIZE en la parte 2.)

6.1.7 Máximos y mínimos

MAX es una función que da el mayor de los valores que se le suministran como parámetros. Por ejemplo,

```
PRINT "El vencedor es ";MAX(puntos(1),puntos(2),puntos(3),puntos(4))
```

Si los valores son negativos, el máximo es el más próximo a 0 (o sea, el máximo en el sentido matemático).

Análogamente, MIN da el valor mínimo.

6.2 Manipulación de la información literal

6.2.1 Asignación de valores a las variables literales

Para definir una variable literal o asignarle un valor nuevo se utiliza normalmente el signo '=':

```
nueva.cadena$="Nueva, nueva, nueva"
```

Para modificar parte de una cadena literal se utiliza la orden MID$:

```
fecha$="xx de marzo de 1986"
MID$(fecha$,1,2)="21"
PRINT fecha$
21 de marzo de 1986
```

Los parámetros de MID$ especifican el nombre de la variable, la posición de partida y el número de caracteres que van a ser reemplazados.

LSET y RSET son palabras clave especiales que permiten asignar valores a una plantilla para ficheros de acceso aleatorio; cambian el texto de la variable sin modificar su longitud. Las describiremos en el capítulo 7 al hablar de los ficheros.

6.2.2 Unión de cadenas

El único operador aritmético aplicable a las cadenas literales es el '+'. Su efecto es unir las dos cadenas a las que afecta, una a continuación de la otra. Esto no tiene nada que ver con la suma de números; compruébelo observando el resultado de las dos instrucciones siguientes:

```
PRINT "123"+"456"
PRINT 123+456
```

6.2.3 Disección de cadenas

BASIC proporciona tres funciones con las que extraer diferentes secciones de una cadena dada: LEFT$, MID$ y RIGHT$.

LEFT$(texto$,n) da los n primeros caracteres de la cadena texto$.

RIGHT$(texto$,n) da los n últimos caracteres de la cadena texto$.

MID$(texto$,p,n) da n caracteres extraídos de la cadena texto$ a partir de la posición p.

MID$(texto$,p) da todos los caracteres de la cadena texto$ que hay entre la posición especificada por p y el final de la cadena.

6.2.4 Generación de cadenas

BASIC dispone de dos funciones que permiten generar cadenas por repetición de un carácter.

SPACE$(n) produce una cadena que consiste en n espacios.

STRING$(n,c) produce una cadena que consiste en el carácter cuyo código ASCII es c repetido n veces.

STRING$(n,"z") produce una cadena que consiste en la letra z repetida n veces.

Por ejemplo,

```
PRINT STRING$(20,"*");"¡Hola!";STRING$(20,42)
**********¡Hola!**********
```

6.2.5 Longitud de las cadenas

Una cadena puede contener entre 0 y 255 caracteres. Hay una función de BASIC, LEN, que da la longitud de la cadena que se le proporciona como argumento. Conocer la longitud de la cadena es necesario, por ejemplo, cuando se va a procesar sus caracteres uno a uno, como ocurre en la siguiente rutina, que escribe el texto título$ insertando espacios entre los caracteres:

```
100 caracteres=LEN(título$)
110 FOR a=1 TO caracteres
120 PRINT MID$(título$,a,1);" ";
130 NEXT
140 PRINT
```

6.2.6 Búsqueda de subcadenas

La función INSTR permite averiguar si una determinada sucesión de caracteres está contenida en una cadena. Su forma general es la siguiente:

```
INSTR(principio,cadena1$,cadena2$)
```

donde cadena2$ es la cadena buscada, cadena1$ es la cadena en la que se busca y 'principio' especifica la posición (en cadena1$) a partir de la cual comienza la búsqueda.

Si se omite 'principio', se le supone el valor 1; es decir, se busca cadena2$ en toda la cadena1$.

INSTR genera un resultado numérico. Si es 0, indica que la cadena2$ no está contenida en la cadena1$; si no, indica la posición en cadena1$ en la que empieza cadena2$.

Esta función tiene varias aplicaciones que no son evidentes. Así, en el ejemplo siguiente la utilizamos para reconocer una respuesta del usuario tanto si éste la da en mayúsculas como si en minúsculas:

```
100 PRINT "A=abandonar, E=empezar de nuevo, R=reintentar. Pulse una tecla"
110 respuesta$=INKEY$
120 IF respuesta$="" THEN 110
130 código=INSTR (1,"AaEeRr",respuesta$)
140 IF código=0 THEN GOTO 100
150 ON código GOSUB 500,500,600,600,900,900
```

6.2.7 Conversión de cadenas

Los caracteres que forman las cadenas de texto se guardan en la memoria en mayúsculas o minúsculas, según cómo se los haya escrito. (No ocurre lo mismo con las variables y las palabras clave, para las que BASIC no distingue mayúsculas de minúsculas.)

BASIC dispone de dos funciones, LOWER$ y UPPER$, que convierten mayúsculas en minúsculas, y viceversa.

UPPER$(*cadena*) convierte las letras de *cadena* a mayúsculas. LOWER$(*cadena*) realiza la conversión inversa. Por ejemplo,

```
PRINT UPPER$("Febrero 1986. Libro de Vencimientos")
FEBRERO 1986. LIBRO DE VENCIMIENTOS
```

Nótese que estas funciones sólo afectan a las letras de la 'a' a la 'z' y de la 'A' a la 'Z' (con exclusión de la 'ñ' y la 'Ñ'); los restantes caracteres no se modifican.

Los caracteres que forman las cadenas pueden tener código ASCII entre 0 y 255, si bien sólo los del margen de 0 a 127 están definidos por la norma ASCII. A veces hay que manejar cadenas cuyo bit más significativo se ha utilizado como indicador (por ejemplo, como bit de paridad); para restringir sus caracteres al margen de 0 a 127 se utiliza la función STRIP$(*cadena*), que pone a cero el bit 7 de todos los caracteres de la cadena.

6.3 Conversión entre diferentes tipos de información

En esta sección vamos a describir las funciones que convierten la información de un tipo en otro. Recordemos que los tipos principales son cuatro: literal, entera, precisión sencilla y doble precisión.

Algunas conversiones se realizan automáticamente. Por ejemplo, se puede asignar un valor entero a una variable de doble precisión, o un valor de doble precisión a una variable entera, sin necesidad de realizar ninguna conversión previa.

Las conversiones se utilizan para:

- homogeneizar datos incompatibles (literales y numéricos).
- forzar una conversión que normalmente no ocurriría.

6.3.1 Conversión entre números y cadenas literales

Los caracteres constituyentes de las cadenas se almacenan en la memoria en forma de bytes, cuyos valores pueden estar entre 0 y 255. Cada valor está asociado a un carácter diferente. El valor que corresponde a cada carácter es su código ASCII. Por ejemplo, el código ASCII de la letra 'A' es el 65; el de la 'B', el 66; etc. (Para más detalles véase el apéndice I del manual de CP/M.)

La función CHR$ genera el carácter que corresponde a un código ASCII dado, de modo que, por ejemplo, PRINT CHR$(65) escribe la letra 'A'.

A pesar de que BASIC convierte internamente los códigos en caracteres visibles antes de imprimirlos, el usuario no puede intercambiar libremente las cadenas con los números. Así, para utilizar una cadena (por ejemplo, "1234") como número, o un número (por ejemplo, 1234) como cadena, antes hay que realizar la conversión adecuada.

Para convertir un número en cadena literal se emplea la función STR$; por ejemplo, número$=STR$(número).

Para convertir una cadena literal formada por cifras en un número se usa VAL; por ejemplo, número=VAL(número$).

Hay otras funciones que convierten números en cadenas decimales dotadas de un formato especificado (DEC$), o en cadenas octales (OCT$) o hexadecimales (HEX$). (Véase la parte 2.)

Finalmente, hay funciones que realizan la conversión (en ambos sentidos) entre valores numéricos y representaciones literales compactas para su grabación en ficheros de disco. Las describiremos en el capítulo 7.

6.3.2 Conversión entre enteros, precisión sencilla y doble precisión

Cuando se asigna a una variable numérica un valor, éste se almacena automáticamente en la forma adecuada al tipo de la variable. No obstante, hay dos ocasiones en las que interesa forzar una conversión determinada: cuando no se va a asignar el valor numérico a ninguna variable (y por consiguiente no hay conversión automática) y cuando la conversión automática no es la deseada.

Por ejemplo, cuando se asigna un valor en punto flotante a una variable entera, BASIC redondea el valor al entero más próximo; en caso de duda, elige el valor más distante de cero (redondea 4.49 a 4, pero 4.5 a 5).

Para forzar la conversión de un número a entero, precisión sencilla o doble precisión se utiliza CINT, CSNG y CDBL, respectivamente. Esto puede ser necesario para especificar la precisión con que se quiere realizar un cálculo. Por ejemplo, compare

```
a#=1/10
PRINT a#
0.1000000014901161
```

con

```
a# = CDBL(1)/10
PRINT a#
0.1
```

En el primer caso, la división se realiza en precisión sencilla. El resultado que se obtiene, también de precisión sencilla, es asignado a una variable de doble precisión, a#. En el segundo caso CDBL convierte el dividendo en doble precisión **antes** de la división; ésta se realiza en doble precisión y el resultado es, por consiguiente, un número de doble precisión. (Dicho sea de paso, el primero de estos ejemplos pone de manifiesto la inexactitud de la representación de fracciones decimales mediante fracciones binarias; la inexactitud ha sido visible en este caso porque hemos escrito el resultado con 16 cifras, en lugar de con las 7 habituales.)

También se dispone de varias funciones para redondear de diversas formas la parte decimal de los números en punto flotante. Se trata de INT, FIX y ROUND. Nótese que estas funciones no convierten el número en entero.

INT redondea al valor entero inmediatamente inferior. Por ejemplo, INT(3.9) produce 3; INT(−3.9) produce −4.

FIX suprime la parte decimal. Por ejemplo, FIX(3.9) da 3 y FIX(−3.9) da −3.

ROUND redondea el valor con el número de decimales especificado. Su forma es ROUND(*número,decimales*). Por ejemplo, ROUND(3.335,2) da 3.34; ROUND(−3.335,2) da −3.34. Si se omite el parámetro *decimales*, BASIC lo toma como 0. Así, ROUND(3.335) produce 3.

Si *decimales* es negativo, entonces indica el número de cifras que se han de redondear por la izquierda del punto decimal. Por ejemplo, ROUND(12345,−2) da 12300.

Capítulo 7

Almacenamiento de la información en los discos

Las funciones, órdenes y otros recursos de BASIC que hemos descrito hasta ahora cubren la mayor parte de las necesidades de la programación, pero no son adecuados a una tarea fundamental para la que se puede utilizar BASIC: el almacenamiento y proceso de grandes cantidades de información, como pueden ser los archivos de personal de una empresa, las listas de existencias, los ficheros de clientes, etc.

Los discos no sólo sirven para almacenar programas, sino también la información con la que los programas trabajan. Se puede considerar, pues, que los discos son una especie de ampliación de la memoria del ordenador. La información se almacena en los discos por el procedimiento de grabar en ellos los llamados 'ficheros de datos'.

Por consiguiente, la información utilizada por los programas se puede almacenar en los ficheros o en las variables. Cada método tiene sus ventajas e incovenientes:

Ficheros	**Variables**
La información se conserva mientras no se la modifique deliberadamente o se borre el fichero	La información se pierde fácilmente; por ejemplo, cuando se apaga el ordenador
Capacidad de almacenamiento limitada por la capacidad del disco (unos 170000 caracteres)	Capacidad de almacenamiento limitada por la memoria del ordenador (unos 30000 caracteres)
Fácil de compartir por varios programas	Difícil de compartir por varios programas
El tiempo de acceso varía entre casi instantáneo y varios segundos	Acceso casi instantáneo
La información debe ser transferida a las variables para que pueda ser utilizada por BASIC	La información es directamente utilizable en las instrucciones de BASIC

En resumen, los ficheros de datos permiten almacenar grandes cantidades de información y la hacen fácilmente accesible al programa que la ha grabado o a otros distintos, pero su manejo es algo más engorroso y lento que el de las variables.

BASIC proporciona tres sistemas de archivo de datos: ficheros de *acceso secuencial*, ficheros de *acceso aleatorio* o *directo* y ficheros de *acceso por claves*.

Un fichero secuencial es algo similar a la sentencia DATA: la información está en el orden en que se la ha grabado y sólo puede ser leída en ese orden.

Un fichero de acceso aleatorio es más parecido a una matriz. Los elementos están numerados; para leer o modificar un elemento se accede a él mencionando su número.

Los ficheros de acceso por claves son parecidos a los de acceso aleatorio, pero permiten seleccionar los elementos más comodamente, a través de 'claves' (cadenas de texto) tales como un apellido o la descripción de un artículo.

A continuación describiremos estos tres tipos de ficheros de datos.

7.1 Órdenes de gestión de discos

BASIC dispone de varias órdenes para la gestión de los discos; estas órdenes se utilizan, no sólo en la manipulación de ficheros de datos, sino también en otras situaciones.

El sistema operativo CP/M Plus proporciona las órdenes necesarias para listar el directorio del disco, borrar ficheros, cambiarles el nombre o examinar su contenido. En BASIC estas funciones se realizan mediante palabras clave que tienen los mismos nombres y llevan los mismos parámetros que en CP/M: DIR, ERA, REN y TYPE:

— Para listar los ficheros de un disco: DIR *constante*; por ejemplo, DIR *.BAS

— Para borrar ficheros: ERA *constante*; por ejemplo, ERA PRUEBA.*

— Para cambiar el nombre de un fichero: REN *constante=constante*; por ejemplo, REN NUEVO=ANTIGUO

Para exhibir en la pantalla el contenido de un fichero: TYPE *constante*: por ejemplo, TYPE INFO.DOC

Para más amplia información sobre estas órdenes y el significado de sus argumentos, consúltese el manual de CP/M Plus.

Estas órdenes de BASIC van seguidas de textos constantes encerrados entre comillas (igual que en CP/M Plus, salvo las comillas). Pero BASIC dispone de órdenes equivalentes cuyos argumentos pueden ser expresiones literales, no necesariamente constantes. El significado de los argumentos sigue siendo el mismo. Las órdenes son FILES, KILL, NAME AS y DISPLAY:

— Para listar los ficheros de un disco: FILES *cadena*; por ejemplo, FILES lista.fichero$

— Para borrar ficheros: KILL *cadena*; por ejemplo KILL "*.TMP"

— Para cambiar el nombre de un fichero: NAME *cadena* AS *cadena*; por ejemplo, NAME nombre.nuevo$ AS nombre.antiguo$

— Para exhibir en la pantalla el contenido de un fichero: DISPLAY *cadena*; por ejemplo, DISPLAY nombre$+tipo$

Hay una orden de BASIC, RESET, que se debe ejecutar siempre que se cambie de disco (igual que control-C en CP/M). Esta orden informa a CP/M de que se va a cambiar el disco y «cierra» todos los ficheros que estén abiertos (véase la sección 7.2).

La función FIND$ sirve para buscar ficheros en el disco. Si el fichero especificado no está en el disco, FIND$ genera la cadena vacía; si lo está, genera una cadena que contiene cierta información acerca del fichero (véase la parte 2). Por ejemplo, para averiguar si un determinado fichero está en el disco,

```
IF FIND$(fichero$)="" THEN PRINT "No encuentro ";fichero$
```

7.2 Ficheros de acceso secuencial

Los ficheros de datos más sencillos son los secuenciales, así que empezaremos por ellos. Describiremos las palabras clave necesarias y luego daremos un programa ejemplo para ilustrar cómo se las combina.

Los ficheros secuenciales pueden ser creados y leídos, pero no modificados (directamente).

7.2.1 Creación de un fichero secuencial

La creación de un fichero secuencial se desarrolla en tres etapas:

- abrir el fichero
- escribir información en el fichero
- cerrar el fichero

No se puede leer el fichero mientras no haya sido cerrado.

Apertura del fichero. Antes de poder escribir en un fichero es necesario pedirle a BASIC, con la orden OPEN, que lo prepare:

```
OPEN "O",número-del-fichero,nombre-del-fichero
```

La letra "O" especifica el modo de acceso y representa 'Output' (salida). Esta orden hace que BASIC cree un fichero completamente nuevo, borrando, si es necesario, el fichero que pueda haber en el disco con el nombre especificado. Hay otros modos de acceso ("I", "R" y "K") que describiremos más abajo.

Nótese que los ficheros secuenciales no pueden ser modificados parcialmente.

El número del fichero es un entero del margen 1-3, por el que se lo identificará en todas las instrucciones que en el futuro hagan referencia al fichero. Este número tiene que ser distinto del que tengan asignados los demás ficheros que estén abiertos en este momento. (El límite superior de este margen, 3, puede ser modificado mediante la orden MEMORY, o bien especificado tras el parámetro /F en la orden de carga de BASIC; véase la parte 2.)

El nombre del fichero es la especificación completa (incluyendo posiblemente el distintivo de tipo) del nombre con el que el sistema operativo creará el fichero. No puede haber un fichero abierto con ese mismo nombre. Si existe un fichero con ese nombre, el sistema operativo lo borra.

Si se desea abrir un fichero en el disco de la unidad B, se debe escribir 'B:' como prefijo del nombre del fichero.

En todo momento se puede tener abiertos tres ficheros como máximo (pero este tope se puede modificar con MEMORY o con el parámetro /F de la orden de carga de BASIC; véase la parte 2). Si se alcanza este límite y se necesita abrir otro fichero, antes será necesario cerrar alguno de los anteriores (véase más abajo la orden CLOSE).

El fichero permanece abierto mientras no se lo cierre explícitamente (con CLOSE) o implícitamente (con RUN, BUFFERS, RESET o SYSTEM). No se debe extraer el disco mientras no se haya cerrado los ficheros que estén abiertos en él.

Escritura de información en el fichero. Una vez abierto el fichero, la información se puede escribir en él mediante las órdenes PRINT # y WRITE #.

Estas órdenes van seguidas de un número de fichero, una coma y la lista de los datos (constantes, variables o expresiones) que van a ser escritos en el fichero. Por ejemplo,

```
PRINT ficheroseñas%,nombre$(i),señas$(i),teléfono%(i)
```

PRINT # funciona casi igual que PRINT; en particular, permite la utilización de USING, SPC y TAB para controlar el formato de la salida hacia el fichero. La única diferencia importante es que PRINT # no expande el código de control 'Tab' (ASCII 9), es decir, no lo convierte en espacios.

WRITE # es similar a PRINT #, con la diferencia de que escribe los elementos de la lista separados por comas, ignora las zonas de escritura y pone las cadenas entre comillas. WRITE # no admite TAB, SPC ni USING.

Estas diferencias se hacen más obvias si se observan los siguientes ejemplos, en los que hemos utilizado constantes para mayor claridad.

```
PRINT "Nombre","Señas";"Teléfono";1234
```

produce en la pantalla el siguiente efecto:

```
Nombre          SeñasTeléfono 1234
```

La orden

```
PRINT # fichero%,"Nombre","Señas";"Teléfono";1234
```

graba los datos con el mismo formato:

```
Nombre          SeñasTeléfono 1234
```

En cambio,

```
WRITE#fichero%,"Nombre","Señas","Teléfono";1234
```

los graba de la siguiente forma:

```
"Nombre","Señas","Teléfono",1234
```

La importancia de estos dos formatos de salida se hará patente cuando expliquemos cómo se lee la información de los ficheros secuenciales.

Le sugerimos que al principio utilice WRITE#, pues esta orden produce automáticamente ficheros fácilmente legibles con INPUT#.

Cada vez que se ejecuta una instrucción PRINT# o WRITE# con un fichero secuencial, la nueva información se añade al final del fichero.

(En realidad, la información no se graba inmediatamente en el disco, sino que se la almacena temporalmente en una pequeña zona de la memoria denominada *tampón*. El contenido del tampón se vuelca sobre el disco cada vez que se llena el tampón o cuando se cierra el fichero. Este método hace los programas más eficaces, al reducir el número de veces que se debe acceder a la unidad de disco.)

Cierre del fichero. Cuando se termina de escribir en el fichero, se debe ejecutar una orden CLOSE para cerrarlo. La acción de cerrar el fichero tiene los cuatro efectos siguientes:

- Asegura que la información que pueda haber en el tampón sea enviada al fichero.
- Permite que más tarde se pueda abrir el fichero para leerlo.
- Libera el número del fichero para que el usuario pueda asignarlo a otro fichero.
- Libera los recursos asignados a ese fichero (por ejemplo, el tampón) para que un nuevo fichero pueda utilizarlos.

Para cerrar todos los ficheros se ejecuta CLOSE sin parámetros. Para cerrar ficheros concretos se ejecuta CLOSE seguida de los nombres de los ficheros, separados entre sí por comas. Por ejemplo,

```
CLOSE fichero.clientes,fichero.artículos
```

BASIC cierra todos los ficheros cuando se ejecuta cualquiera de las siguientes órdenes: RESET, RUN, BUFFERS y SYSTEM.

7.2.2 Lectura de un fichero secuencial

La lectura de un fichero secuencial se desarrolla en tres etapas: abrir el fichero, escribir la información y cerrar el fichero.

Apertura del fichero. Antes de poder leer un fichero es necesario pedirle a BASIC, con la orden OPEN, que lo prepare:

OPEN "I",*número-del-fichero,nombre-del-fichero*

La letra "I" especifica el modo de acceso y representa 'Input' (entrada). Hay otros modos de acceso: "O", descrito en el apartado anterior, y "R" y "K", que describiremos más adelante.

El número del fichero es un entero del margen 1-3, por el que se lo identificará en todas las instrucciones que en el futuro hagan referencia al fichero. Este número tiene que ser distinto del que tengan asignados los demás ficheros que estén abiertos en este momento. (El límite superior de este margen, 3, puede ser modificado según hemos explicado antes.)

El nombre del fichero es la especificación completa (incluyendo posiblemente el distintivo de tipo) del nombre con el que el sistema operativo buscará el fichero. No se puede abrir un fichero en dirección de entrada si no existe en el disco o si está abierto en dirección de salida (no obstante, puede estar abierto en dirección de salida con otro número de fichero).

Si se desea abrir un fichero en el disco de la unidad B, se debe escribir 'B:' como prefijo del nombre del fichero.

En todo momento se puede tener abiertos tres ficheros como máximo (pero este tope se puede modificar según hemos explicado). Si se alcanza este límite y se necesita abrir otro fichero, antes será necesario cerrar alguno de los anteriores (véase más abajo la orden CLOSE).

El fichero permanece abierto mientras no se lo cierre explícitamente (con CLOSE) o implícitamente (con RUN, BUFFERS, RESET o SYSTEM). No se debe extraer el disco mientras no se hayan cerrado los ficheros que estén abiertos en él.

Lectura de la información grabada en un fichero secuencial. Una vez abierto el fichero, la información se puede leer mediante la orden INPUT#.

Esta orden va seguida de un número de fichero, una coma y la lista de las variables a las que se va a asignar la información leída. Por ejemplo,

```
INPUT fichero%,nombre$(i),señas$(i),cuenta(i)
```

La información se lee en forma secuencial (de ahí el nombre de este tipo de ficheros). El primer dato grabado en el fichero será asignado a la primera variable mencionada en la primera instrucción INPUT#, el segundo valor a la segunda variable, etc. No es posible leer dos veces el mismo dato (para ello habría que cerrar el fichero, volver a abrirlo y leer datos hasta llegar al que interesa).

La información grabada en un fichero secuencial se puede utilizar de dos formas:

— Se puede asignar todos los datos del fichero a matrices adecuadas al tiempo que se los va leyendo; luego se ignora el fichero y sólo se utiliza las matrices. Este método sólo es aplicable en el caso de que toda la información quepa en la memoria del ordenador, pero por lo demás es muy deseable, ya que entonces todos los datos es-

tán accesibles en cualquier orden y casi instantáneamente. Es adecuado en tareas en las que se necesite consultar la información almacenada en un fichero o reorganizar la información antes de imprimirla o de rehacer el fichero.

— Leer los datos uno a uno o en grupos y utilizar cada dato o grupo antes de leer el siguiente. Este método es aplicable a ficheros de longitud arbitraria, ya que en la memoria del ordenador sólo hay que almacenar un dato, o como mucho un grupo de datos. Pero tiene el inconveniente de que en cada momento no se puede procesar más que la información «actual». Es adecuado para tareas tales como realizar sumas de muchas cantidades, o para copiar o imprimir ficheros completos.

El tipo de las variables que sigan a PRINT# y el orden en que se las coloque dependen en gran medida de la estructura del fichero, esto es, de las instrucciones con las que fue creado.

La forma más fácil de manejar ficheros secuenciales es crearlos con WRITE # y luego leerlos con PRINT#, utilizando en las dos instrucciones variables del mismo tipo y en el mismo orden. Por ejemplo, si un fichero ha sido creado con

```
FOR i=1 TO 10
WRITE# fichero%,nombre$(i),señas$(i),teléfono%(i)
NEXT
```

se lo puede leer con

```
FOR veces=1 TO 10
INPUT# fichero%,cliente$(veces),dirección$(veces)
INPUT#fichero%,tel%(veces)
```

Nótese que no es necesario que los nombres de las variables sean iguales; basta con que lo sean los tipos. Además, tampoco hace falta que las variables estén agrupadas de la misma forma, siempre que se las lea en el mismo orden.

En realidad, tampoco es imprescindible que los tipos de las variables sean idénticos, a condición de que sean compatibles (literales o numéricos) y de que el usuario esté dispuesto a aceptar los posibles redondeos.

(Incluso es posible, escribiendo con PRINT#, crear un fichero cuya información pueda ser asignada posteriomente tanto a variables literales como a variables numéricas. Sin embargo, esta técnica es relativamente compleja y no la describiremos aquí; consúltese la parte 2.)

Si al leer un fichero se intenta sobrepasar el último dato grabado en él, se provoca un error. Esto no plantea ningún problema si se sabe de antemano cuántos datos hay en el fichero, como ocurría en el ejemplo anterior; de lo contrario, el programador puede grabar sistemáticamente al final de los ficheros cierto dato especial de su elección y luego, en el programa de lectura, comparar cada dato que se lee con ese dato especial.

No obstante, BASIC ofrece una solución más elegante a través de la función EOF ('fin de fichero'). EOF(número.fichero%) genera el valor 0 si todavía quedan datos por leer en el fichero, y el valor −1 si ya se ha leído el último. Por ejemplo, para imprimir todos los datos del fichero "prueba.sec", se puede utilizar las siguientes instrucciones:

```
número.fichero% = 1
OPEN"I",número.fichero%,"prueba.sec"
WHILE NOT EOF(número.fichero%)
  INPUT#número.fichero%,elemento$
  PRINT elemento$
WEND
```

(Hemos utilizado el operador NOT para «negar» el valor generado por EOF, de modo que el bucle WHILE continúe hasta que alcance el final del fichero.)

Cierre del fichero. Cuando se termina de leer un fichero (o parte de él), se debe ejecutar una orden CLOSE para cerrarlo. La acción de cerrar un fichero que está abierto en dirección de entrada tiene los tres efectos siguientes:

- Permite que más tarde se pueda abrir el fichero en modo de escritura.
- Libera el número del fichero para que el usuario pueda asignarlo a otro fichero.
- Libera los recursos asignados a ese fichero (por ejemplo, el tampón de entrada) para que un nuevo fichero pueda utilizarlos.

Para cerrar todos los ficheros se ejecuta CLOSE sin parámetros. Para cerrar ficheros concretos se ejecuta CLOSE seguida de los nombres de los ficheros, separados entre sí por comas. Por ejemplo,

```
CLOSE fichero.clientes,fichero.artículos
```

BASIC cierra todos los ficheros cuando se ejecuta cualquiera de las siguientes órdenes: RESET, RUN, BUFFERS y SYSTEM.

7.2.3 Modificación de un fichero secuencial

BASIC no permite modificar los ficheros secuenciales directamente. Para añadir, borrar o alterar la información de un fichero secuencial es necesario leer el fichero, modificar o ampliar la información en la memoria del ordenador y grabarla en un fichero completamente nuevo.

Si el fichero es suficientemente pequeño, se lo puede cargar completo en la memoria, modificar la información y grabar en el disco la versión nueva, como se ilustra en el ejemplo 2 del apartado siguiente.

Si el fichero es demasiado grande, se lo puede leer por tramos y grabar en varias etapas la nueva versión (con nombre diferente).

No obstante, hay cambios que se pueden realizar con facilidad mediante las funciones de control de ficheros aleatorios que describiremos en la sección 7.3.

7.2.4 Programas de ejemplo

Los siguientes programas ilustran un par de aplicaciones típicas de los ficheros de acceso secuencial. No se sienta defraudado por su aparente trivialidad: los hemos simplificado deliberadamente para poder resaltar los aspectos que ahora nos interesan.

Ejemplo 1: análisis estadístico sencillo

En este ejemplo se utiliza un fichero secuencial de la forma más fácil posible: leyendo los datos uno a uno y utilizándolos inmediatamente.

Objetivo. La misión del programa es sumar, contar y promediar los datos numéricos grabados en un fichero secuencial, para luego imprimir esta información junto con los valores máximo y mínimo encontrados en el fichero. El valor medio se calculará con los valores absolutos, esto es, ignorando el signo de los datos; en cambio, todos los demás resultados tendrán en cuenta los signos.

El fichero de datos contendrá información sobre transacciones comerciales, números de personas, resultados de partidos de fútbol; esto es indiferente para el programa. Los datos serán todos numéricos, positivos o negativos, enteros o en punto flotante, y estarán en el margen de −99999.999 a +99999.999. El fichero puede contener entre 0 y 20000 datos. El nombre del fichero será suministrado por el usuario en el momento de ejecutar el programa.

Si usted desea autoexaminarse para comprobar si ha entendido todo lo que hemos explicado hasta ahora, puede intentar diseñar y escribir el programa antes de seguir leyendo.

Diseño. Dado que ninguno de los cálculos y operaciones va a requerir el acceso a datos anteriores, no hay necesidad de guardar los datos leídos en una matriz, lo que en todo caso sería imposible, ya que los ficheros largos no cabrían en la memoria.

El programa se puede dividir en las siguientes etapas:

- preparar las variables y la pantalla
- elegir disco
- elegir fichero
- leer y procesar los datos
- completar los cálculos e imprimir los resultados
- terminar ordenadamente

El programa. Ésta es una de las posibles soluciones para el problema planteado:

```
1 REM *** ejemplo 1 - análisis estadístico sencillo ***
10 REM *** preparar variables y pantalla ***
20 :
30 escape$=CHR$(27):pit$=CHR$(7):borra$=escape$+"E"+escape$+ "H":fichero%=1
40 PRINT borra$;"Programa de análisis estadístico"
50 :
100 REM *** elegir disco ***
110 :
120 DIR
130 PRINT"Si el fichero que usted necesita no está en este disco, cambie el disco y luego
 pulse C."
140 PRINT"Si éste es el disco correcto, pulse cualquier otra tecla para continuar."
150 r$=""
160 WHILE r$="":r$=INKEY$
170 WEND
180 r$=UPPER$(r$):IF r$="C" THEN RESET:GOTO 120
190 :
200 REM *** elegir fichero ***
210 :
220 INPUT"Escriba el nombre del fichero y pulse RETURN: ",nombrefi$
230 IF FIND$(nombrefi$)="" THEN PRINT pit$;nombrefi$;" NO ESTA EN ESTE DISCO.
 PRUEBE OTRA VEZ.":GOTO 220
240 :
300 REM *** leer y procesar datos ***
310 :
320 PRINT borra$;"Analizando ";nombrefi$;" - POR FAVOR, ESPERE":PRINT
330 OPEN"I",fichero%,nombrefi$
340 WHILE NOT(EOF(fichero%))
350   cuenta%=cuenta%+1
360   INPUT#fichero%,numero
370   total=total+numero
380   total.absoluto=total.absoluto+ABS(numero)
390   minimo=MIN(numero,minimo)
400   maximo=MAX(numero,maximo)
410 WEND
420 :
500 REM *** terminar cálculos e imprimir resultados ***
510 :
520 IF cuenta%=0 THEN PRINT nombrefi$;" está vacío":GOTO 620
530 media=total.absoluto/cuenta%
540 PRINT"RESULTADOS PARA EL FICHERO ";nombrefi$:PRINT
550 PRINT"Total final:";TAB(40);total
```

```
560 PRINT"Valor medio de los valores absolutos:";TAB(40);media
570 PRINT"El valor mínimo ha sido:";TAB(40);minimo
580 PRINT"El valor máximo ha sido:";TAB(40);maximo
590 :
600 REM *** terminar ordenadamente ***
610 :
620 CLOSE fichero%
630 END
```

Este programa no requiere explicación. La parte más compleja es la de las líneas 150-170. INKEY$ examina el teclado continuamente hasta que se pulsa una tecla. El carácter correspondiente se asigna a r$; esta cadena se procesa en la línea 180. Además, CHR$(7) es el código 'Bell'; al enviarlo a la pantalla produce un pitido.

Aunque sea una solución razonable, este programa no es perfecto, ni mucho menos. Por ejemplo, la única forma de detenerlo es pulsar control-C, las variables quizá debieran ser de doble precisión y se podría mejorar la presentación de los resultados. El lector puede tratar de mejorarlo.

Comprobación. La etapa final del desarrollo de un programa debe ser probarlo en una situación la más realista posible. Como no disponemos de ningún fichero con el que pueda trabajar este programa, tendremos que crearlo. Podemos crear un fichero de prueba con el siguiente programa:

```
1 REM prueba1
10 OPEN "O",1,"prueba1.sec"
20 FOR a=1 TO 10
30 WRITE #1,a,-a
40 NEXT
50 CLOSE
```

Genere el fichero con este programa y luego analícelo con el programa anterior.

La prueba ideal es aquélla que produce unos resultados fácilmente comprobables por otros medios y que lleva el programa hasta el límite de sus posibilidades. Con una sola prueba es difícil cumplir ambos objetivos.

Este fichero de prueba da unos resultados que podemos comprobar mediante unos simples cálculos aritméticos. Pero no incluye datos suficientes, ni en número ni en variedad, como para comprobar todas las especificaciones de diseño.

Para ahorrar al lector el trabajo de realizar los cálculos, vamos a facilitarle los resultados:

```
Total final:                          0
Valor medio de los valores absolutos: 5.5
El valor mínimo ha sido:             -10
El valor máximo ha sido:              10
```

Ejemplo 2: listín telefónico

Este programa ilustra cómo se puede leer un fichero secuencial para asignar sus datos a una matriz que luego se procesa en la memoria y cómo se devuelve la información al disco después de procesarla.

Objetivo. La misión del programa es proporcionar un listín telefónico electrónico que dé el número de teléfono cuando se escriba un nombre y que permita añadir nuevas fichas (nombres y números) a la lista. Para no complicar el programa excesivamente, prescindiremos de funciones tales como la modificación de fichas existentes y la impresión de la lista ordenada por nombres.

Cuando se escribe un nombre, el programa debe entenderlo igual en mayúsculas que en minúsculas; también debe entenderlo aunque no esté completo y debe dar todos los nombres que concuerden con los caracteres escritos.

El número máximo de fichas será 100. Los números de teléfono (incluidos los posibles prefijos), constarán como máximo de 12 caracteres. Los prefijos se almacenarán como parte integral del número. Los nombres tendrán como máximo 30 caracteres.

El fichero tendrá un nombre fijo.

Si usted desea autoexaminarse para comprobar que ha entendido todo lo que hemos explicado hasta ahora, puede intentar diseñar y escribir el programa antes de seguir leyendo.

Diseño. Como el número de fichas es bastante pequeño, el fichero entero cabe en la memoria; lo asignaremos a una matriz para hacer su manejo más fácil y rápido. Por esta misma razón, es necesario ordenarlo; el programa puede recorrer la matriz muy deprisa en busca del nombre deseado.

Parte del diseño consiste en decidir de qué forma se va a almacenar la información. En este caso hay tres tipos de información: los nombres y los números de teléfono, evidentemente, y además la **relación que hay entre unos y otros**. Esta información se puede grabar de diversas formas:

- Como lista de nombres y lista de números correspondientes (el tercer nombre tiene el tercer número, etc.).
- Como lista de fichas, cada una de las cuales contiene un nombre y un número.
- Como lista de fichas que consisten en un nombre y una serie de índices que hacen referencia a los números de teléfono, los cuales están en otra lista distinta (por ejemplo, "Enter, S.A.",4,7 significaría que los teléfonos de Enter, S.A. son los que figuran en la lista de números en los lugares cuarto y séptimo.)

Los dos primeros métodos son iguales en eficacia y dificultad de programación. El tercero es más complicado, pero también más útil en la práctica (porque puede haber un nombre que tenga varios números de teléfono y un teléfono que sea compartido por varios nombres).

El programa se puede dividir en las siguientes etapas:

- preparar las variables y la pantalla.
- leer el fichero y asignar la información a las variables.
- pedir instrucciones al usuario y obedecerlas (buscar un nombre, añadir una ficha nueva, terminar).
- escribir la versión actualizada del fichero y terminar ordenadamente.

El programa. El siguiente programa realiza las funciones especificadas. Nótese que el programa **no puede modificar el fichero**, sino borrarlo y sustituirlo por una versión actualizada.

```
1 REM ejemplo 2 – listín telefónico
5 :
10 REM **** preparar variables y pantalla ***
20 DIM nombre$(100),teléfono$(100)
30 escape$=CHR$(27):pit$=CHR$(7):borra$=escape$+"E"+escape$+"H"
40 fichero.tele$="telefono.sec":fichero%=1
50 falso=0:verdadero=-1
60 fichero.cambiado=falso
70 PRINT borra$;"Listín telefónico"
80 :
90 REM **** leer datos ****
100 GOSUB 320
110 :
120 REM **** captar ordenes ****
130 orden=0
140 WHILE orden<5
150   PRINT borra$;"Listín telefónico ";fichas;"fichas"
160   PRINT:PRINT"Buscar número, Añadir ficha o Fin (B/A/F)"
170   letra$="BbAaFf":GOSUB 900:orden=respuesta
180   ON orden GOSUB 500,500,600,600:GOSUB 1000
190 WEND
200 :
210 REM **** fin ****
220 IF fichero.cambiado=verdadero THEN GOSUB 700
230 END
240 :
290 REM ****** SUBRUTINAS ******
300 REM *** leer fichero ***
310 REM ** antes de nada, averiguar si está el fichero **
320 IF FIND$(fichero.tele$)<>"" THEN GOTO 410
330 PRINT"No hay fichero de listín telefónico en este disco. ¿Quiere cambiar el disco (C)
  o crear un listín nuevo (N)?"
340 letra$="CcNn":GOSUB 900:IF respuesta=3 OR respuesta=4 THEN fichas=0:
  RETURN
350 GOSUB 800:GOTO320
```

```
360 :
400 REM ** leer fichas **
410 PRINT:PRINT"Leyendo el listín telefónico. Por favor, espere."
420 OPEN "I",fichero%,fichero.tele$
430 fichas=0
440 WHILE NOT(EOF(fichero%))
450   fichas=fichas+1
460   INPUT#fichero%,nombre$(fichas),telefono$(fichas)
470 WEND
480 CLOSE fichero%
490 RETURN
500 REM *** buscar ***
510 IF fichas=0 THEN PRINT"¡El listín telefónico esta vacío!":RETURN
520 INPUT"Escriba el nombre que quiere buscar y pulse RETURN, o sólo RETURN para
  no buscar ninguno: ",busca$
530 IF busca$="" THEN RETURN
540 cuenta=1
550 WHILE cuenta<=fichas AND nombre$(cuenta)<>busca$:cuenta=cuenta+1
560 WEND
570 IF cuenta>fichas THEN PRINT "No encuentro ";busca$:RETURN
580 PRINT"Número de teléfono: ";telefono$(cuenta):RETURN
590 :
600 REM *** añadir ficha ***
605 IF ficha>=100 THEN PRINT"¡Listín lleno!":RETURN
610 INPUT"Escriba el nombre que quiere añadir y pulse RETURN, o sólo RETURN para
  no añadir ninguno: ",nombre$
620 IF nombre$="" THEN RETURN
630 INPUT"Escriba el número y pulse RETURN, o sólo RETURN para no añadir esta ficha:
",telefono$
640 IF telefono$="" THEN RETURN
650 fichas=fichas+1
660 nombre$(fichas)=nombre$
670 telefono$(fichas)=telefono$
680 fichero.cambiado=verdadero:RETURN
690 :
700 REM *** escribir fichero ***
710 PRINT:PRINT"Escribiendo el listín telefónico. Por favor, espere."
720 OPEN "O",fichero%,fichero.tele$
730 FOR cuenta=1 TO fichas
740   WRITE#fichero%,nombre$(cuenta),telefono$(cuenta)
750 NEXT
760 CLOSE fichero%
770 RETURN
790 :
800 REM * cambio de disco *
810 RESET
```

```
820 PRINT"Inserte el disco donde está el listín telefónico y pulse RETURN",a$
830 RETURN
890 :
900 REM * leer tecla *
910 respuesta$=INKEY$
920 WHILE respuesta$=""
930  respuesta$=INKEY$
940 WEND
950 respuesta=INSTR(letra$,respuesta$)
960 IF respuesta=0 THEN GOTO 910
970 RETURN
990 :
1000 REM * esperar por tecla *
1010 PRINT:PRINT"Pulse cualquier tecla para continuar"
1020 WHILE INKEY$=""
1030 WEND
1040 RETURN
```

También este programa dista mucho de ser perfecto. Por ejemplo, no es capaz de encontrar un nombre si el usuario lo escribe en minúsculas y el nombre ha sido archivado en mayúsculas, o si se escribe el nombre incompleto. Tampoco comprueba si ya existe una ficha con el mismo nombre antes de permitir que se añada una ficha nueva al final de la lista.

El lector puede tratar de modificar el programa para corregir estos defectos. (*Sugerencias*: se puede utilizar UPPER$ o LOWER$ para homogeneizar los nombres antes de compararlos, y LEN y LEFT$ para buscar fichas partiendo de nombres incompletos. Antes de añadir una ficha nueva se debe examinar el fichero completo para comprobar si ya existe una ficha con el mismo nombre; si existe, se sustituye el teléfono antiguo por el nuevo; si no, se añade la ficha al final del fichero. En el segundo caso habrá que actualizar la variable 'fichas'.)

Comprobación. Ejecute el programa para crear un listín con unas cuantas fichas. Vuelva a ejecutarlo para comprobar que encuentra el fichero. Pruebe las tres órdenes posibles. Si tiene paciencia, compruebe que puede manejar las 100 fichas correctamente.

7.3 Ficheros de acceso aleatorio

Los ficheros de acceso aleatorio son similares a los secuenciales: tienen nombres asociados, antes de utilizarlos es necesario abrirlos, se puede leer y escribir información en ellos y deben ser cerrados en cuanto se termina de usarlos.

Sin embargo, en comparación con los ficheros secuenciales, los de acceso aleatorio tienen dos ventajas principales: la información se puede **leer y escribir** en cualquier orden (de ahí el nombre de 'aleatorio' en lugar de 'secuencial') y además se la puede **modificar** (también en cualquier orden).

(La información de los ficheros secuenciales sólo puede ser leída en el orden en que se la escribió; como no es posible modificarla directamente sobre el fichero, hay que cargarla en la memoria, completa o por bloques, realizar los cambios necesarios en la memoria y luego grabar la versión nueva del fichero completo en el disco.)

Estas diferencias hacen que los ficheros de acceso aleatorio sean más eficaces y más fáciles de usar que los secuenciales en muchas aplicaciones. Las únicas desventajas consisten en que son más difíciles de entender y utilizar correctamente y que ocupan algo más de espacio en el disco para una cantidad de información dada.

Los métodos de almacenamiento de la información son diferentes en los dos sistemas. En un fichero secuencial, cada dato se lee o escribe por separado y puede tener una longitud diferente de la de los demás. Los datos que escribimos o leemos en un fichero de acceso aleatorio son los llamados *registros*. Todos los registros de un fichero han de tener **la misma longitud**. Cada registro puede consistir en un solo elemento (como en los ficheros secuenciales) o varios elementos que se manipulan en bloque.

Los registros proporcionan una buena forma de almacenar datos que tienen algo en común o que están relacionados entre sí. Por ejemplo, en un fichero de datos del personal de una empresa, se dedicaría un registro a cada empleado, con los siguientes datos:

— Nombre (30 caracteres)
— Señas (50 caracteres)
— Teléfono (10 caracteres)
— Número de afiliación a la SS (9 caracteres)
— Fecha de nacimiento (6 caracteres)
— Escala salarial (1 carácter)
— Escala de seguro (1 carácter)

Evidentemente, tiene sentido mantener toda esta información agrupada para cada persona.

Para especificar un registro concreto en el que se desea leer o escribir se menciona la posición que ocupa en el fichero, esto es, su *número de registro*. El primer registo del fichero es el número 1; el segundo, el 2; etc. Esto es análogo a la forma en que se manejan las matrices, cuyos elementos se especifican mediante un índice (por ejemplo, nombre$(2)). De hecho, se puede utilizar los ficheros de acceso aleatorio del mismo modo que las matrices, pero sin la limitación impuesta por la capacidad de la memoria y con la posibilidad de que un fichero sea compartido por varios programas.

Los ficheros de acceso aleatorio se utilizan frecuentemente para almacenar en un solo registro, o en un grupo de registros consecutivos, toda la información relativa a un elemento (persona, empresa, artículo, etc.), reservando el mismo espacio para todos los elementos. Los programas se diseñan de forma que permitan al usuario acceder fácil y rápidamente al registro (o grupo de registros) deseado para examinarlo, modificarlo, imprimirlo, etc.

La mejor forma de organizar un fichero para aplicaciones de este tipo es colocar los registros en algún orden que sea fácilmente correlacionable con alguna información numérica contenida en ellos; por ejemplo, el número del empleado dentro de la empresa o el código de almacén de un artículo. A falta de información de este tipo, habrá que anotar en una

lista qué número de registro tiene cada elemento; el usuario especificará el elemento deseado mediante su número de registro, de forma que, por ejemplo, "Enter, S.A." será la empresa número 1, "Seguros del Hogar" será la empresa número 2, etc. (Jetsam proporciona una solución mucho más elegante, como enseguida veremos.)

7.3.1 Creación de un fichero de acceso aleatorio

El proceso de creación de un fichero de acceso aleatorio consta de las siguientes fases:

- abrir el fichero
- definir los formatos de los registros
- asignar datos a cada registro
- escribir cada registro en el fichero
- cerrar el fichero

A continuación vamos a describir cada una de estas etapas.

Apertura del fichero. Antes de poder escribir datos en un fichero de acceso aleatorio es necesario pedirle a BASIC, con la orden OPEN, que lo prepare:

```
OPEN "R",número-de-fichero,nombre-de-fichero
```

Por ejemplo,

```
OPEN "R",3,"person2.ale"
```

La "R" indica que el fichero se abre para acceso aleatorio. (Las otras posibilidades son: "O", que lo abre para salida secuencial, "I", para entrada secuencial, y "K", para acceso por claves.)

El significado de los otros parámetros, número del fichero y nombre del fichero, es el mismo que en los ficheros secuenciales.

También se puede añadir otro parámetro para especificar la longitud de los registros cuando no interese utilizar la longitud implícita (128 bytes):

```
OPEN "R",número-de-fichero,nombre-de-fichero,longitud-de-registro
```

De esta forma se aprovecha mejor el espacio en el disco en los casos en los que la longitud de los registros puede ser menor que 128 caracteres. Además, este último parámetro es necesario cuando los registros tienen que ser de más de 128 bytes (pero entonces hay que modificar este valor máximo mediante la orden MEMORY; véase la parte 2).

El hecho de abrir un fichero para acceso aleatorio equivale a crearlo nuevo si no existe todavía (igual que en el caso de los ficheros secuenciales). Si existe, la orden OPEN no lo borra; gracias a esto es posible modificar la información de los ficheros de acceso aleatorio, según

explicamos más abajo. Por consiguiente, cuando se desea crear un fichero de acceso aleatorio nuevo, primero hay que comprobar si ya existe uno con el mismo nombre y, en tal caso, borrarlo (con FIND$ y KILL, según explicaremos más adelante).

Cuando se abre un fichero para acceso aleatorio BASIC le reserva un espacio en la memoria, el denominado *tampón de registros*, de la misma longitud que el registro (normalmente 128 bytes). Es en el tampón donde BASIC forma y modifica la información antes de escribirla en el fichero. Cada fichero abierto para acceso aleatorio tiene su propio tampón de registros.

Aunque es posible abrir para acceso aleatorio un fichero que ya esté abierto, tal acción no es recomendable.

Definición de los formatos de los registros. La información se coloca en el tampón de registros (previamente a su grabación en el fichero) asignándola a unas variables especiales, denominadas *variables de campo*. Se trata de una forma especial de variables literales que se definen en una instrucción FIELD.

La instrucción FIELD construye el formato del registro dividiendo el tampón en varias zonas (campos), cada una de las cuales se identifica con una variable de campo. La forma de la instrucción es la siguiente:

FIELD *número-de-fichero,longitud-de-campo* AS *variable-de-campo*
[,*longitud-de-campo* AS *variable-de-campo*]

Por ejemplo,

```
FIELD 3, 10 AS nombre$, 30 AS señas$, 10 AS teléfono$
```

El número de fichero es el mismo que se especificó al abrir el fichero; nótese que el formato de registro es válido solamente para el fichero para el que ha sido definido, y que no es utilizable automáticamente por los demás ficheros.

La longitud de campo es el número de caracteres que se reservarán para cada variable de campo en el registro. La suma de todas las longitudes de campo no puede exceder de la longitud del registro (normalmente 128, pero véase también la descripción de OPEN en la parte 2).

La instrucción FIELD no escribe ninguna información en el fichero, sino que se limita a «etiquetar» los diferentes campos con los nombres de las variables de campo. La información que pudiera haber antes en esas variables se pierde y es sustituida por el contenido actual de sus correspondientes campos en el registro.

Para cada fichero se puede definir varios formatos distintos, de forma que en un mismo fichero se pueda grabar registros con formatos diferentes. Por ejemplo, en el fichero de personal del que hablábamos antes se puede necesitar de vez en cuando un registro especial: un 'registro de continuación' para los empleados cuyas señas sean tan largas que no quepan en los caracteres reservados para ellas en el registro principal. En tal caso habrá que definir dos formatos de registro, uno para el registro principal:

```
FIELD perfil%, 30 AS nombre$, 50 AS señas$, 10 AS teléfono$,
9 AS número.SS$, 6 AS fecha.nacim$, 1 AS escala.salarial$,
1 AS escala.seguro$, 1 AS registro.cont
```

y otro para el registro de continuación:

```
FIELD perfil%, 108 AS cont.señas$
```

Las variables de campo son especiales en el sentido de que su longitud es fija (la especificada en la instrucción FIELD); sólo se les debe asignar valores por el procedimiento que se explica a continuación.

Asignación de información a los registros. La información se coloca en los registros asignándola a las variables de campo que han sido definidas en una instrucción FIELD anterior.

La asignación se debe realizar de alguna de las tres formas siguientes:

```
LSET variable-de-campo=expresión-literal
RSET variable-de-campo=expresión-literal
MID$ (variable-de-campo,principio,longitud)=expresión-literal
```

Si la asignación se realiza de cualquier otra forma, se rompe la relación que hay entre la variable y el tampón de registros; de esta forma se puede volver a utilizar la variable de campo como variable literal normal.

LSET asigna los caracteres de la *expresión-literal* a la variable de campo, alineándolos por la izquierda (la 'L' proviene de *left*=izquierda). Esto quiere decir que si la expresión literal es más corta que la variable de campo, ésta se rellena con espacios por la derecha. Si la expresión literal es demasiado larga, se ignoran los caracteres sobrantes (por la derecha) y no se produce error.

RSET funciona de forma similar, pero rellenando con espacios por la izquierda cuando la expresión literal es demasiado corta.

MID$ toma de la expresión literal un número de caracteres especificado por *longitud* y los asigna a la variable de campo, empezando por la posición dada por *principio*.

Por ejemplo, si con el formato

```
FIELD fichero%, 20 AS nombre$
```

se ejecuta la orden

```
LSET nombre$ = "**********************************"
```

el valor de nombre$ pasa a ser:

```
"********************"
```

Si se ejecuta	LSET nombre$="Ana Isabel"
nombre$ vale	"Ana Isabel "
Si se ejecuta	RSET nombre$="María Teresa"
nombre$ vale	" María Teresa"
Y si luego se ejecuta	MID$(nombre$,2,7)="********************"
nombre$ vale	" *******María Teresa"

La información numérica puede ser asignada a los campos convirtiéndola previamente en una cadena literal equivalente:

```
LSET número$ = STR$(cuenta)
```

Pero esta forma de almacenar números es muy ineficaz; por ejemplo, un entero tal como −30000, que BASIC almacena en sólo dos bytes, ocuparía 6 bytes en el fichero. Por otra parte, al pasar de la representación binaria interna a la forma de cadena literal siempre se empeora la precisión de los números. BASIC resuelve estos dos problemas mediante las funciones MKD$, MKS$ y MKI$, que convierten la información numérica en una representación literal compacta:

- MKD$(*número*) convierte un número de doble precisión en una cadena de 8 caracteres.
- MKS$(*número*) convierte un número de precisión sencilla en una cadena de 4 caracteres.
- MKI$(*número*) convierte un entero en una cadena literal de 2 caracteres.

Por ejemplo,

```
LSET número$ = MKI$(total)
```

Estas cadenas especiales no pueden ser exhibidas en la pantalla ni impresas correctamente.

Las funciones inversas son CVD, CVS, CVI. Convierten las cadenas en números de doble precisión, de precisión sencilla y enteros, respectivamente. Las describiremos más adelante.

(También es posible asignar información a un registro mediante las órdenes PRINT *#n* y WRITE *#n*; véase la parte 2.)

Escritura de registros. Después de abrir el fichero para acceso aleatorio, de definir un formato para los registros y de asignar datos al registro, la etapa siguiente es escribir el registro en el fichero.

Los registros se escriben en el fichero mediante una instrucción PUT, que puede tener la forma

PUT *número-de-fichero* o bien PUT *número-de-fichero,número-de-registro*

El número de fichero es el mismo que se especificó al abrir el fichero.

La primera versión de la instrucción PUT escribe en el fichero como si se tratase de un fichero secuencial; cada registro se escribe a continuación del último que se ha escrito con PUT (o del último que se ha leído con GET; véase más abajo). Ésta es la forma más rápida de escribir en un fichero de acceso aleatorio, pues no se pierde el tiempo que de otra forma tardaría la unidad de disco en buscar las diferentes zonas del fichero.

La segunda versión es la que se utiliza normalmente para acceder a registros individuales de un fichero ya existente o para crear un fichero de estructura compleja. El *número-de-registro* especifica la posición en que se va a encontrar el registro dentro del fichero. Nótese que el tamaño total del fichero depende de la longitud de los registros (la misma para todos, normalmente igual a 128) y del número de registro más alto que se haya utilizado. Por ejemplo, si escribimos un solo registro en un fichero con la orden PUT fichero%,1000, el fichero constará de 1000 registros y los 999 primeros estarán vacíos.

Por el hecho de escribir un registro en el disco no se modifica el contenido del tampón de registros; esto sólo ocurre cuando se asigna nuevos valores a las variables de campo o cuando se lee el fichero con el mismo número de fichero.

Cierre del fichero. Cada vez que se termina de escribir en un fichero de acceso aleatorio se lo debe cerrar con la orden CLOSE para asegurar que toda la información quede grabada en el disco.

Para cerrar todos los ficheros se ejecuta CLOSE sin parámetros. Para cerrar ficheros concretos se utiliza CLOSE seguida de los nombres de los ficheros, separados entre sí por comas. Por ejemplo,

```
CLOSE fichero.clientes,fichero.artículos
```

7.3.2 Lectura de un fichero de acceso aleatorio

La lectura de un fichero de acceso aleatorio es un proceso que se desarrolla en las cinco etapas siguientes:

- abrir el fichero
- definir los formatos de los registros
- leer los registros del fichero
- utilizar la información leída
- cerrar el fichero

Apertura del fichero. Antes de poder leer un fichero de acceso aleatorio es necesario abrirlo con la orden OPEN:

```
OPEN "R",número-del-fichero,nombre-del-fichero
```

Por ejemplo,

```
OPEN "R",3,"person2.ale"
```

La "R" indica que el fichero se abre para acceso aleatorio. (Las otras posibilidades son: "O", que lo abre para salida secuencial, "I", para entrada secuencial, y "K", para acceso por claves.)

El significado de los otros parámetros, número del fichero y nombre del fichero, es el mismo que en los ficheros secuenciales.

También se puede añadir otro parámetro para especificar la longitud de los registros cuando no interesa utilizar la longitud implícita (128 bytes):

```
OPEN "R",número-del-fichero,nombre-del-fichero,longitud-de-registro
```

Este parámetro será necesario normalmente para leer un fichero que haya sido creado con una longitud de registro distinta de la implícita; es habitual (aunque no imprescindible) leer los ficheros con la misma longitud de registro con que fueron creados.

Definición de los formatos de los registros. La forma más sencilla de leer los ficheros es utilizar la misma longitud de registro y el mismo formato con que se los creó. Así se garantiza que la información se lee con la estructura con la que fue escrita. No obstante, hay ocasiones en las que para la lectura interesa cambiar la longitud o el formato de los registros, pero tales técnicas quedan fuera de los objetivos de esta introducción y no las describiremos.

Los formatos se definen mediante la instrucción FIELD, tal como explicamos al hablar de la creación de los ficheros.

Lectura de los registros del fichero. Los datos grabados en el fichero se leen transfiriendo un registro del disco al tampón de registros con la orden GET:

```
GET número-de-fichero
```

o bien

```
GET número-fichero,número-de-registro
```

El número de fichero es el mismo que se especificó al abrir el fichero.

La primera versión de la instrucción GET lee el fichero como si se tratase de un fichero secuencial; cada registro se lee a continuación del último leído con GET (o del último escrito con PUT). Si se acaba de abrir el fichero, lee el primer registro (el número 1).

La segunda versión permite leer cualquier registro.

Detectar el final del fichero es algo más difícil en los ficheros de acceso aleatorio que en los secuenciales. Se puede utilizar la función EOF, pero antes es necesario leer el fichero

al menos una vez. Además, EOF puede dar el valor 'verdadero' aunque no se haya llegado al final del fichero; esto ocurre cuando el registro que se acaba de leer está vacío (no se ha escrito nunca sobre él).

En la práctica, esto obliga a que sea el programa el que se encargue de saber en todo momento dónde está el final del fichero; para ello el fichero tiene que ser de longitud fija (conocida por el programa) o, de lo contrario, en el momento de crear el fichero se debe escribir todos los registros, rellenando con espacios o con ceros binarios los registros que deban quedar vacíos.

Utilización de la información leída en el registro actual. La información leída en el último registro está disponible a través de las variables de campo. El contenido de estas variables se altera en cuanto se lee otro registro con el mismo número de fichero.

Si se ha definido varios formatos de registro para un fichero, se puede utilizar las variables de campo de los diversos formatos en cualquier combinación.

La información numérica que se haya grabado en el fichero en forma literal compacta (MKD$, MKS$, MKI$) debe ser convertida a forma numérica normal con CVD, CVS y CVI:

- CVD(*cadena*) convierte una cadena de 8 bytes en un número de doble precisión.
- CVS(*cadena*) convierte una cadena de 4 bytes en un número de precisión sencilla.
- CVI(*cadena*) convierte una cadena de 2 bytes en un número entero.

(La información también se puede leer con INPUT #, pero no describiremos aquí este método.)

Cierre del fichero. Cuando se termina de leer un fichero de acceso aleatorio, se lo debe cerrar mediante la orden CLOSE.

Para cerrar todos los ficheros se ejecuta CLOSE sin parámetros. Para cerrar ficheros concretos se ejecuta CLOSE seguida de los nombres de los ficheros, separados entre sí por comas. Por ejemplo,

```
CLOSE personal%,pagas%
```

7.3.3 Lectura, modificación y escritura de ficheros de acceso aleatorio

Como ya hemos indicado, las instrucciones que abren el fichero y definen los formatos de los registros son las mismas tanto si se va a leer el fichero como si se va a escribir en él. De hecho, cuando se trabaja con ficheros de acceso aleatorio es muy fácil leer, modificar y reescribir registros individuales sin más que combinar adecuadamente las instrucciones que acabamos de describir. Estas técnicas serán ilustradas en el siguiente ejemplo.

7.3.4 Ejemplo

Ejemplo 3: fichero de personal

Objetivo. Se trata de crear, mantener y consultar un sencillo fichero de datos del personal de una empresa. El fichero debe tener capacidad para 500 fichas, en cada una de las cuales se guardará la siguiente información:

- Número del empleado (3 cifras)
- Nombre (30 caracteres)
- Señas (50 caracteres)
- Teléfono (10 caracteres)
- Número de afilicación a la Seguridad Social (9 caracteres)
- Fecha de nacimiento (6 caracteres)
- Escala salarial (1 carácter)
- Escala de seguro (1 carácter)

Las funciones del programa serán las siguientes:

- Añadir un registro nuevo
- Borrar un registro
- Reemplazar un registro
- Mostrar un registro

Los registros serán seleccionados por el 'número del empleado'

Si usted desea autoexaminarse para comprobar si ha entendido todo lo que hemos explicado hasta ahora, puede intentar diseñar y escribir el programa antes de seguir leyendo.

Diseño. La longitud máxima del fichero (unos 50000 caracteres) descarta la posibilidad de crearlo como fichero secuencial y manipular toda la información a la vez en la memoria, pues esto es lo que habría que hacer para poder modificar fácilmente los registros.

Así pues, el fichero tendrá que ser de acceso aleatorio. Lo siguiente que hay que decidir es el formato de los registros. Dado que se va a acceder a la información a través del número del empleado, es evidente que los registros se deben ordenar según ese número. Puesto que el número de empleado se escribe con 3 cifras, si se identifica este número con el número de registro el fichero podrá contener 999 registros como máximo, límite que es superior al especificado (500); así pues, haremos coincidir el número de registro con el número de empleado. (Nótese que de esta forma no tenemos que guardar el número de empleado en el registro.)

Las etapas globales del diseño son, pues, las siguientes:

- preparar las variables y la pantalla
- abrir el fichero y definir el formato de los registros
- pedir al usuario las instrucciones y obedecerlas
- cerrar el fichero y terminar ordenadamente

El programa. El siguiente programa es una solución sencilla, en la que no hemos incluido refinamientos para no oscurecer los aspectos que aquí nos interesan, cual es la correcta utilización de las órdenes de gestión de ficheros de acceso aleatorio.

```
1 REM ejemplo 3 – fichero de personal
5 :
10 REM **** preparar variables ****
20 escape$=CHR$(27):pit$=CHR$(7):borra$=escape$+"E"+escape$+"H":
   crlf$=CHR$(13)+CHR$(10)
30 fichero%=1: limite=500
40 :
50 REM **** funciones definidas por el usuario ****
60 :
70 DEF FNcabecera$(titulo$)=borra$+STRING$((88-LEN(titulo$))/2,"*")+" "
   +titulo$+" "+STRING$((87-LEN(titulo$))/2,"*")+crlf$
80 :
90 REM **** preparar pantalla ****
100 :
110 PRINT FNcabecera$("Fichero de personal")
120 :
130 REM **** preparar fichero ****
140 :
150 IF FIND$("personal.ale")="" THEN GOSUB 1200 ELSE OPEN "R",fichero%,
   "personal.ale"
160 FIELD fichero%, 30 AS nombre.reg$, 50 AS domicilio.reg$, 10 AS telefono.reg$,
   9 AS SS.reg$, 6 AS fecha.reg$, 1 AS salario.reg$, 1 AS seguro.reg$
170 FIELD fichero%, 128 AS registro$
180 :
200 REM **** leer registro ****
210 :
220 PRINT:PRINT
230 INPUT"Escriba el número del empleado (1 a 500) y pulse RETURN: ",persona
240 IF persona < 1 OR persona > limite THEN GOTO 230
250 :
260 PRINT FNcabecera$("Número de empleado"+STR$(persona))
270 PRINT
```

```
280 GET fichero%,persona
290 :
300 REM **** pedir y obedecer una instrucción ****
310 :
320 IF nombre.reg$ = STRING$(30," ") THEN GOSUB 500 ELSE GOSUB 600:PRINT:
    PRINT
330 inductor$="Pulse T para terminar o C para continuar":letras$="TtCc":GOSUB
1400
340 IF respuesta > 2 THEN GOTO 220
390 :
400 REM **** terminar ordenadamente ****
410 :
420 CLOSE
430 PRINT borra$
440 END
490 :
500 REM **** subrutinas ****
510 :
520 REM **** opciones en caso de registro vacío ****
530 :
540 PRINT"Empleado desconocido."
550 letras$="SsNn":inductor$ = "¿Añadir registro nuevo (S/N)?":GOSUB 1400
560 IF respuesta=1 OR respuesta=2 THEN GOSUB 900
570 RETURN
590 :
600 REM *** opciones en caso de registro existente ***
610 :
620 REM ** imprimir el registro en la pantalla **
630 :
640 PRINT"Nombre: ";nombre.reg$
650 PRINT"Señas: ";domicilio.reg$
660 PRINT"Teléfono: ";telefono.reg$;TAB(20);"Fecha de nacimiento: ";
    LEFT$(fecha.reg$,2);"/";MID$(fecha.reg$,3,2);"/";RIGTH$(fecha.reg$,2)
670 PRINT
680 PRINT"Numero afil. S.S.: ";SS.reg$;TAB(30);"Escala salarial: ";salario.reg$;
    TAB(60);"Escala de seguro: ";seguro.reg$
690 :
700 REM ** ofrecer y obedecer opciones **
710 :
720 letras$="MmBbliSs":inductor$ = "¿Modificar, borrar, imprimir o seguir
    (M/B/I/S)? ":GOSUB 1400
730 ON respuesta GOSUB 900,900,800,800,1500,1500
740 RETURN
790 :
800 REM *** borrar registro actual ***
810 :
```

```
820 letras$="SsNn":inductor$="Preparado para borrar el registro. Pulse S para
    confirmar o N para cancelar ":GOSUB 1400
830 IF respuesta > 2 THEN RETURN
840 LSET registro$ = ""
850 PUT fichero%,persona
860 PRINT"Registro borrado"
870 RETURN
880 :
900 REM *** añadir o modificar registro ***
910 :
920 INPUT "Nombre";nombre$
930 INPUT "Señas";domicilio$
940 INPUT "Número de teléfono";telefono$
950 INPUT "Fecha de nacimiento: día";dia$
960 INPUT "                     mes";mes$
970 INPUT "                     año (2 dígitos)";año$
980 INPUT "Número afil. S.S.";SS$
990 INPUT "Escala salarial";salario$
1000 INPUT "Escala de seguro";seguro$
1010 :
1020 letras$="SsNn":inductor$="¿Conforme? Pulse N para volver a escribir, o S para
    continuar ":GOSUB 1400
1030 IF respuesta > 2 THEN 920
1040 :
1050 LSET nombre.reg$=nombre$
1060 LSET domicilio.reg$=domicilio$
1070 LSET SS.reg$=SS$
1080 LSET salario.reg$=salario$
1090 LSET seguro.reg$=seguro$
1100 MID$(fecha.reg$,1,2)=RIGHT$("0"+dia$,2)
1110 MID$(fecha.reg$,3,2)=RIGHT$("0"+mes$,2)
1120 MID$(fecha.reg$,5,2)=RIGHT$("0"+año$,2)
1130 LSET telefono.reg$=telefono$
1140 PUT fichero%,persona
1150 PRINT"Registro escrito"
1160 RETURN
1190 :
1200 REM *** crear fichero vacío ***
1210 :
1220 PRINT"Creando fichero vacío. Por favor, espere."
1230 OPEN "R",fichero%,"personal.ale"
1240 FIELD fichero%,128 AS registro$
1250 LSET registro$ = ""
1260 FOR a=1 TO 500
1270    PUT fichero%
1280 NEXT
```

```
1290 PRINT"Fichero creado"
1300 RETURN
1390 :
1400 REM ** leer tecla **
1410 :
1420 PRINT:PRINT inductor$
1430 respuesta$=INKEY$
1440 WHILE respuesta$ = ""
1450   respuesta$=INKEY$
1460 WEND
1470 respuesta=INSTR(letras$,respuesta$):IF respuesta=0 THEN GOTO 1430
1480 RETURN
1490 :
1500 REM ** imprimir el registro en la impresora **
1510 :
1520 LPRINT "Datos del empleado número";persona
1530 LPRINT "**********************************":LPRINT
1540 LPRINT "Nombre: ";nombre.reg$
1550 LPRINT "Señas: ";domicilio.reg$
1560 LPRINT "Teléfono: ";telefono.reg$;TAB(40);"Fecha de nacimiento: ";
   LEFT$(fecha.reg$,2);"/";MID$(fecha.reg$,3,2);"/";RIGHT$(fecha.reg$,2)
1570 LPRINT
1580 LPRINT "Número afil. S.S: ";SS.reg$;TAB(35);"Escala salarial: ";salario.reg$;
   TAB(60);"Escala de seguro: "seguro.reg$
1590 RETURN
```

Hemos omitido muchas funciones deseables; por ejemplo, comprobar la validez de las fechas de nacimiento, listar el fichero completo o por tramos, buscar registros por el nombre o algún otro dato del empleado, etc. Usted puede tratar de mejorar el programa para incluir estas mejoras.

Comprobación. Procure comprobar que el programa funciona correctamente en todas las situaciones posibles: que crea el fichero cuando no hay ninguno en el disco, que no lo crea cuando ya existe, que la presentación de los datos en la pantalla es la correcta, que funcionan bien las opciones de borrar, modificar e imprimir los registros.

7.4 Ficheros de acceso aleatorio por claves (Jetsam)

Los ficheros de acceso sencuencial son los indicados cuando la información se necesita normalmente en un solo orden preestablecido. Los de acceso aleatorio son adecuados para almacenar información que pueda ser consultada o actualizada en cualquier orden; pero, a pesar de todo, la información sólo es accesible a través del número de registro.

BASIC proporciona un sistema de gestión de ficheros de acceso aleatorio, denominado 'Jetsam', que permite el acceso a la información a través de *claves*. Aunque frecuente en los ordenadores grandes y medianos, este método de gestión de ficheros es muy raro en los microordenadores.

Jetsam crea 'ficheros de acceso por claves' cuya información es accesible en cualquier orden, lo mismo que en los ficheros de acceso aleatorio ordinarios. La diferencia consiste en que, en lugar de seleccionar los registros por el número de registro, se los selecciona mediante cualquiera de sus 'claves'. Las claves son etiquetas de texto que se asignan al registro en el momento de crearlo o que se añaden posteriormente.

Utilizando Jetsam, habría sido muy fácil escribir el programa del ejemplo anterior de forma que buscara los registros, no sólo por su número, sino también por el nombre del empleado, por su número de afiliación a la Seguridad Social, etc.; o incluso por alguna otra característica no grabada en los registros (tal como sexo o departamento).

Si comparamos un fichero con un libro, para buscar un dato en un fichero de acceso aleatorio hay que saber en qué página se encuentra; en cambio, en un fichero de acceso por claves, podemos buscarlo en el índice.

7.4.1 Claves, rangos y ficheros de índices

Las claves son cadenas literales que pueden constar de hasta 31 caracteres. Cada fichero de acceso por claves tiene ocho juegos de claves; cada juego es lo que se llama un *rango*. Los rangos se numeran del 0 al 7. En realidad, un rango es un índice de las claves del fichero. Dentro de cada rango las claves están en orden alfabético; las claves repetidas están en orden cronológico; un grupo de claves repetidas es un *conjunto de claves*. De hecho, los rangos son contiguos, de modo que si un programa trata de leer claves más allá del final del rango 1, lo que hace es leer las primeras del rango 2, etc.

Normalmente en cada rango se almacena un tipo de claves; por ejemplo, claves de nombres en el rango 1, claves de escalas salariales en el rango 2, departamentos en el rango 3, etc. Dentro de cada rango las claves pueden ser únicas o estar repetidas (pero véase RANKSPEC). Se puede poner claves para un mismo registro en diferentes rangos, y también diferentes claves para un determinado registro en un mismo rango.

Los programas que manejan ficheros de acceso por claves eligen el rango al que deben añadir las claves (o por cuyas claves deben buscar) en función del tipo de clave.

Un fichero de acceso por claves consiste en realidad en dos ficheros: uno de datos y otro de índices. El fichero de datos es un fichero de acceso aleatorio en el que los dos primeros bytes de cada registro están reservados por Jetsam y los restantes contienen la información de la misma forma que los ficheros de acceso aleatorio ordinarios. El fichero de índices es un fichero especial utilizado por Jetsam para almacenar las claves ordenadas en rangos.

Nótese que el orden de los registros de un fichero de acceso por claves es casi indiferente; puesto que a los registros se accede a través de sus claves, es el **orden de las claves dentro de cada rango** el que establece el «orden» de los registros.

En muchos aspectos se podría considerar que un fichero de acceso por claves dotado de 'n' rangos equivale a 'n' ficheros independientes que tienen ciertos datos compartidos. Ahora bien, las ventajas del fichero de acceso por claves frente a los 'n' ficheros independientes son muy numerosas: las modificaciones de los datos afectan inmediatamente a todos los rangos, mientras que en el sistema de ficheros separados habría que modificar los datos en todos los ficheros; un fichero de acceso por claves es más compacto y la información se puede 'trasladar' o 'copiar' de un rango a otro con gran facilidad.

En resumen, los ficheros de acceso por claves proporcionan todas las ventajas de los ficheros de acceso ordinarios, pero son mucho más adecuados cuando la información debe ser buscada por diversos criterios y no sólo por el número de registro.

7.4.2 Ideas generales

Casi todas las palabras clave que se emplean en la gestión de ficheros de acceso por claves son funciones: generan un valor numérico que informa sobre el resultado de la acción que se ha intentado realizar. Ese valor puede ser asignado a alguna variable numérica, aunque lo más normal es examinarlo directamente para comprobar si la operación ha llegado a buen término. El número generado es normalmente (pero no siempre) 0 para indicar que la operación ha terminado con éxito o algún otro para explicar la razón del fracaso. Para más amplia información véase la descripción de las palabras clave en la parte 2.

Vamos a explicar el uso de los ficheros de acceso por claves desglosándolo en tres fases: creación de un fichero nuevo, lectura y modificación de un fichero. En la práctica los programas combinarán estas tres técnicas.

7.4.3 Creación de un fichero de acceso por claves

La creación de un fichero de acceso por claves se desarrolla en cinco etapas:

- crear un fichero vacío
- definir los formatos de los registros
- asignar datos a cada registro
- escribir cada registro en el fichero
- cerrar el fichero

Creación de un fichero vacío. A diferencia de lo que ocurre con los ficheros secuenciales y los de acceso aleatorio, para crear un fichero de acceso por claves no basta con abrirlo, sino que es necesario ejecutar una orden CREATE. Antes de eso, no obstante, hay que reservar espacio en la memoria para los tampones de índices de Jetsam mediante la orden BUFFERS:

BUFFERS *número-de-tampones*

El parámetro es el *número-de-tampones*, de 128 caracteres cada uno, que deben ser reservados. Cuantos más tampones se reserven, más deprisa se procesarán los ficheros y menos espacio quedará en la memoria para el programa. Recomendamos reservar como mínimo seis, a menos que se esté muy escaso de memoria.

La orden CREATE crea un fichero de acceso por claves nuevo y vacío. Su forma es la siguiente:

CREATE *número-de-fichero,nombre-fichero-datos,nombre-fichero-índices*,2

o bien

CREATE *número-de-fichero,nombre-fichero-datos,nombre-fichero-índices*,2, *longitud-de-registro*

El significado de estos parámetros no requiere explicación. Nótese el número 2, que no puede ser omitido. (Es necesario por razones de compatibilidad con las versiones multiusuario de Jetsam, en las que este número indica el bloqueo de registro que se ha de aplicar.)

Esta orden crea dos ficheros: el fichero de datos y el de índices. Ninguno de ellos puede existir ya en el disco.

Cuando se lo crea por primera vez, el fichero de índices permite poner claves repetidas en cualquiera de los rangos. Esta situación se puede cambiar con RANKSPEC; véase la parte 2.

Nótese que en la longitud de registro están incluidos los dos bytes reservados por Jetsam.

Definición de los formatos de los registros. Los formatos de los registros de un fichero de acceso por claves se definen con FIELD, exactamente igual que en el caso de los ficheros de acceso aleatorio. Por ejemplo,

FIELD fichero%, 10 AS nombre$, 50 AS señas$

Téngase en cuenta que el número de bytes disponibles para los campos es igual a la longitud de registro **menos dos.**

Asignación de datos al registro. Los datos se colocan en el registro asignándolos a las variables de campo, igual que en el caso de los ficheros de acceso aleatorio.

Escritura de los registros. Una vez asignada la información a un registro, éste se escribe en el fichero mediante la palabra clave ADDREC. El efecto es, no sólo escribir los datos en el fichero de datos, sino también escribir una clave para el registro en el fichero de índices.

Como casi todas las palabras clave de Jetsam, ADDREC es una función, es decir, genera un valor para indicar el éxito o la razón del fracaso de la operación. Por consiguiente, lo normal es ponerla como segundo miembro de una sentencia de asignación:

```
resultado = ADDREC(número-de-fichero,2,rango,clave)
```

Si el resultado es 0, la operación ha concluido sin incidentes.

El número de fichero es el mismo que se especificó cuando se creó el fichero.

La clave es una expresión literal (de 31 caracteres como máximo) que será insertada en el fichero de índices, en el rango especificado. Ésta es la clave que se debe utilizar más tarde para acceder a este registro.

Cuanto más corta sea la clave, más deprisa podrá trabajar Jetsam con ella. Si es numérica, no conviene convertirla a literal con STR$ o DEC$, sino preferiblemente a alguna de las formas «comprimidas» siguientes:

- Si la clave es un entero del margen de 0 a 255, se la convierte con CHR$(*clave*)
- Si es un entero del margen de −32768 a +32767, se utiliza MKIK$
- Si es un entero del margen de 0 a 65535, se utiliza MKUK$

Si ya existe un registro con la misma clave en el mismo rango, normalmente la clave para el nuevo registro será aceptada y colocada a continuación de la ya existente (pero véase RANKSPEC). Hay palabras clave especiales de Jetsam con las que se puede leer grupos de registros que tengan una misma clave.

Si es necesario, se puede añadir más tarde otras claves para el mismo registro, según explicamos más abajo.

Cierre del fichero. Todo fichero de acceso por claves **tiene que ser cerrado explícitamente** mediante una orden CLOSE en la que se especifique su número. Por ejemplo,

```
CLOSE porclaves1%
```

No se puede permitir que el fichero se cierre implícitamente con ninguna de las siguientes órdenes:

```
BUFFERS
CLOSE  sin parámetros
SYSTEM
RESET
RUN
```

Si así ocurriera, el fichero quedaría inutilizable. Esto se debe a que en el proceso de actualización del fichero la información del fichero de índices frecuentemente se desfasa con respecto a los registros de datos; la única forma de garantizar que el fichero de acceso por claves sea consistente es cerrarlo con una orden CLOSE explícita o ejecutar una orden CONSOLIDATE.

7.4.4 Lectura de un fichero de acceso por claves

La lectura de un fichero de acceso por claves es un proceso que se desarrolla en las seis etapas siguientes:

- abrir el fichero
- definir los formatos de los registros
- buscar el registro (normalmente por su clave)
- leer el registro
- utilizar la información leída
- cerrar el fichero

Apertura del fichero. Antes de poder leer un fichero de acceso por claves es necesario abrirlo mediante la orden OPEN. Pero antes hay que reservar espacio en la memoria para los tampones de índices de Jetsam mediante la orden BUFFERS:

```
BUFFERS número-de-tampones
```

El parámetro es el *número-de-tampones*, de 128 caracteres cada uno, que deben ser reservados. Cuanto más espacio se reserve, más deprisa se procesarán los ficheros y menos espacio quedará en la memoria para el programa. Recomendamos reservar como mínimo seis tampones, a menos que se esté muy escaso de memoria.

Para abrir un fichero de acceso por claves se ejecuta la orden:

```
OPEN "K",número-de-fichero,nombre-fichero-datos,nombre-fichero-índices,2
```

o bien

```
OPEN "K",número-de-fichero,nombre-fichero-datos,nombre-fichero-índices,2,
longitud-de-registro
```

Al igual que ocurría con los ficheros de acceso aleatorio, esta segunda forma sólo es necesaria en los casos en los que el fichero haya sido abierto con una longitud de registro distinta de la implícita (128 caracteres).

Definición de los formatos de los registros. Los formatos de los registros se especifican mediante la orden FIELD, según se ha descrito antes.

Búsqueda de un registro. Hay diversos métodos para buscar un registro en un fichero de acceso por claves. Esta variedad puede parecer desalentadora para el principiante, pero es a ella precisamente a lo que se debe la flexibilidad de los ficheros de este tipo; siempre se puede elegir el método más adecuado e ignorar los restantes.

Cada palabra clave es una función que genera un valor numérico mediante el cual indica si la operación ha concluido con éxito o si ha fracasado. Normalmente el valor 0 indica éxito, mientras que los demás informan sobre la razón del fracaso (no obstante, véase la parte 2).

Para buscar un registro por la clave que tiene en un rango determinado se utiliza la función SEEKKEY:

resultado=SEEKKEY(*número-de-fichero*,0,*rango*,*clave*)

El valor 0 indica que la clave ha sido encontrada en el rango especificado.

Lo que se ha encontrado en realidad es el **primer** registro que tiene esa clave en el rango; si la clave es única, no hay ambigüedad. Pero si la clave está repetida (es decir, si hay varios registros que tienen la misma clave en ese rango) habrá que comprobar, después de leer el registro, que es el que verdaderamente se buscaba. Si no lo es, se busca el siguiente con SEEKNEXT.

Para buscar el registro que tiene la siguiente clave en el mismo rango se utiliza SEEKNEXT:

resultado=SEEKNEXT(*número-de-fichero*,0)

El resultado significa lo siguiente:

0	la nueva clave tiene el mismo valor que la anterior
101	la nueva clave es distinta de la anterior, pero está en el mismo rango
102	la nueva clave está en rango diferente
otros	no se ha encontrado una nueva clave (véase la parte 2)

Para buscar la siguiente clave que es distinta de la actual se utiliza SEEKSET:

resultado=SEEKSET(*número-de-fichero*,*rango*,0)

El resultado 101 significa que la nueva clave se ha encontrado en el mismo rango; el 102, que la nueva clave está en un rango diferente; los demás valores indican fracaso de la operación (véase la parte 2).

Para buscar el registro cuya clave es la anterior a la actual (única o repetida) se utiliza SEEKPREV:

resultado=SEEKPREV(*número-de-fichero*,*rango*,0)

El resultado significa lo siguiente:

0	la nueva clave tiene el mismo valor que la anterior
101	la nueva clave es distinta de la anterior, pero está en el mismo rango
102	la nueva clave está en rango diferente
otros	no se ha encontrado una nueva clave (véase la parte 2)

[También es posible buscar claves (registros) referidos a una clave o registro especificados, no necesariamente a la clave actual. No describiremos aquí esta técnica; véase SEEKREC, SEEKNEXT, SEEKPREV y SEEKKEY en parte 2.]

La utilidad de estas funciones depende de cómo esté estructurado el fichero. Las ilustraremos mediante unos ejemplos sencillos.

Nótese que encontrar un registro **no** es lo mismo que leer su contenido. Para ello se debe ejecutar la orden GET.

Ejemplo 4

Supongamos que disponemos de un fichero que contiene información sobre diversos productos, clasificados por nombre (rango 0), número (rango 1) y precio (rango 2). A partir de él queremos imprimir una lista de artículos y una lista de precios. La lista de artículos debe estar en orden numérico; la de precios, en orden alfabético de los nombres.

Para elaborar la lista de artículos, el programa busca la primera clave partiendo del principio del rango de números de artículos. Después utiliza SEEKNEXT y, mientras el resultado sea 0 (éxito) o 101 (SEEKNEXT ha leído una clave distinta en el mismo rango), el programa lee el registro, lo imprime y busca la siguiente clave:

```
resultado = SEEKRANK(fichero%,0,1):REM buscar la primera clave del rango de
    números de artículos.
WHILE resultado = 0 OR resultado = 101
    (leer registro, extraer información e imprimirla)
    resultado = SEEKNEXT(fichero%,0,1):REM buscar la siguiente clave del rango
        de números de artículos
WEND
```

Análogamente, para la lista de precios,

```
resultado = SEEKRANK(fichero%,0,0):REM buscar la primera clave del rango de
    nombres de artículos.
WHILE resultado = 0 OR resultado = 101
    (leer registro, extraer información e imprimirla)
    resultado = SEEKNEXT(fichero%,0,0):REM buscar la siguiente clave del rango
        de nombres de artículos
WEND
```

Nótese que la única diferencia entre las dos rutinas está en el número del rango: 1 para los números y 0 para los nombres de los artículos.

Ejemplo 5

Supongamos que tenemos un fichero de datos sobre el personal de una empresa en el que la amplitud de la información varía mucho de unas personas a otras. En lugar de hacer todos los registros de la misma longitud, la máxima requerida, podemos adoptar una longitud de registro normal y utilizar registros adicionales **con las mismas claves** cuando la información relativa a una persona no quepa en un registro. Supongamos que se utiliza como clave el nombre de las personas y que estas claves se almacenan en el rango 7. En tal caso, para escribir la ficha completa de una persona llamada nombre$ se utilizaría la siguiente rutina:

```
resultado = SEEKKEY(fichero%,0,7,nombre$)
IF resultado < > 0 THEN PRINT "No lo encuentro":RETURN
WHILE resultado = 0:REM terminar cuando se cambie de clave o de rango
    (leer registro, extraer la información e imprimirla)
    resultado = SEEKNEXT(fichero%,0)
WEND
```

El resultado generado por SEEKNEXT solamente será 0 mientras la clave siguiente sea igual a la actual, de modo que el bucle WHILE asegura que sólo se lea e imprima los registros que tienen la clave nombre$.

Lectura del registro. Contra lo que pudiera parecer, todavía no hemos leído ningún registro: lo único que hemos hecho es decir a Jetsam qué registro queremos leer.

Para leer el registro se ejecuta la orden GET:

```
GET número-de-registro
```

Esta orden lee el registro cuya clave hemos encontrado con alguna de las funciones 'SEEK' antes descritas.

(También se puede utilizar la forma GET *número-de-fichero,número-de-registro* si previamente se ha averiguado el número de registro con FETCHREC. No describiremos aquí esta técnica; véase la parte 2.)

Utilización de la información leída. La información leída en el registro se utiliza exactamente igual que en el caso de los ficheros de acceso aleatorio, es decir, a través de las variables de campo definidas en una instrucción FIELD.

Cierre del fichero. Los ficheros de acceso por claves deben ser cerrados explícitamente mediante una orden CLOSE en la que se especifique su número. Por ejemplo,

```
CLOSE porclaves1%
```

No se puede permitir que el fichero se cierre implícitamente con ninguna de las siguientes órdenes:

BUFFERS
CLOSE sin parámetros
SYSTEM
RESET
RUN

Si así ocurriera, el fichero quedaría inutilizable. Esto se debe a que en proceso de modificación del fichero la información del fichero de índices puede quedar desfasada con respecto a los registros de datos; la única forma de garantizar que el fichero de acceso por claves quede actualizado y marcado como 'consistente' es cerrarlo con una orden CLOSE explícita (o 'consolidarlo' con la función CONSOLIDATE de Jetsam).

(Si un programa se limita a leer y no modifica el fichero de índices, puede cerrar el fichero de cualquiera de las formas mencionadas; pero es fácil confundirse, y por lo tanto conviene acostumbrarse a cerrar los ficheros explícitamente.)

7.4.5 Modificación de un fichero de acceso por claves

La modificación de un fichero de acceso por claves es un proceso que consta de las cuatro etapas siguientes:

- abrir el fichero
- definir los formatos de los registros
- efectuar el cambio
- cerrar el fichero

Las técnicas para abrir el fichero, definir los formatos y cerrar el fichero son como las descritas en el apartado anterior para la lectura de ficheros.

Los cambios posibles son los siguientes:

- añadir un registro nuevo
- borrar un registro
- modificar un registro
- añadir una nueva clave
- borrar una clave
- modificar una clave

o cualquier combinación de éstos.

A continuación vamos a describir los procedimientos por los que se realiza cada uno de estos cambios.

En cuanto un programa introduce un cambio en un fichero, Jetsam marca el fichero como 'inconsistente'. Esto se debe a que, mientras que los registros modificados se escriben en el disco, las claves modificadas se almacenan provisionalmente en el tampón, de forma que los ficheros de datos y de índices quedan desfasados. Vuelven a estar en fase y a ser marcados como 'consistentes' cuando se cierra el fichero adecuadamente, ya que esta acción actualiza el fichero de índices.

Si el fichero de acceso por claves no se cierra correctamente (por ejemplo, porque se interrumpe el suministro de energía eléctrica), quedará en un estado inconsistente y marcado como tal.

Puesto que Jetsam no permite utilizar ficheros que estén marcados como inconsistentes, se debe tomar dos medidas para proteger los ficheros de acceso por claves contra accidentes de este tipo: copiar siempre los ficheros de datos y de índices antes de empezar a modificarlos y «consolidarlos» (con la función CONSOLIDATE) cada vez que se realice un cambio o grupo de cambios.

La función CONSOLIDATE actualiza los ficheros de datos y de índices, hace que sean consistentes el uno con el otro y los marca como tales. Su forma es:

resultado=CONSOLIDATE(*número-de-fichero*)

El resultado es siempre 0.

Adición de un registro nuevo. Para añadir un registro nuevo se utiliza la función ADDREC, según hemos explicado en el apartado 7.4.3. Esta función añade un registro de datos y una clave.

Borrado de un registro. Para borrar un registro hay que borrar todas sus claves con la función DELKEY (descrita más abajo). Cuando se borra la última clave del registro se pierde también la información correspondiente en el fichero de datos.

Modificación de un registro. Para modificar un registro hay que buscarlo (SEEKKEY, etc.), leerlo (GET), realizar los cambios a través de las variables de campo y volver a escribirlo en el fichero (PUT). Todas estas etapas ya han sido descritas, a excepción de esta modalidad de PUT.

Se utiliza PUT en lugar de ADDREC, pues ADDREC no reescribiría el registro, sino que lo crearía como registro nuevo. La forma de la instrucción es

PUT *número-de-fichero*

Nótese que es imprescindible leer el registro antes de escribirlo, aunque no se vaya a utilizar la información.

Otro método es borrar el registro y añadirlo como si fuera nuevo (con ADDREC).

Adición de una clase nueva. Cuando se utiliza ADDREC para añadir un registro a un fichero de acceso por claves sólo se puede especificar una clave. Si se necesita especificar otras claves para el mismo registro, hay que empezar por averiguar su número de registro.

Esto se consigue con FETCHKEY cuando el registro que interesa es el registro actual (si no, se utiliza SEEKKEY, etc.). La nueva clave se añade entonces con ADDKEY:

resultado=ADDKEY(*número-de-fichero*,0,*rango*,*clave*,*número-de-registro*)

El resultado 0 indica que la clave ha sido añadida correctamente; los demás resultados indican que la operación ha fracasado (véase su interpretación en la parte 2).

Las claves adicionales se pueden añadir al mismo rango en el que se puso la primera con ADDREC (salvo que esto esté prohibido por RANKSPEC, véase la parte 2) o a rangos diferentes.

Borrado de una clave. Para borrar una clave se necesita, o bien que esa clave sea la clave actual (lo que se consigue con SEEKKEY, etc.), o bien conocer la clave, el rango y el número del registro al que está asociada. En cualquier caso, la clave se borrar con DELKEY:

resutado=DELKEY(*número-de-fichero*,0)

borra la clave actual, mientras que

resultado=DELKEY(*número-de-fichero*,0,*rango*,*clave*,*número-de-registro*)

borra la clave especificada.

Los resultados 0, 101, 102 y 103 indican que la clave ha sido borrada (véase la parte 2).

¡ADVERTENCIA! Si la clave es la única del registro, se borra también la información grabada en el fichero de datos para ese registro.

Modificación de una clave. Para modificar una clave se debe añadir la clave nueva y **después** borrar la antigua, utilizando para ello ADDKEY y DELKEY, según hemos explicado antes. (Si se invierte el orden, es decir, si se empieza por borrar la clave antigua, se borrará también el registro si éste no tenía más que esa clave.)

7.4.6 Ejemplo

Listín telefónico v2. Este ejemplo ilustra las técnicas de creación, lectura y modificación de un fichero de acceso por claves.

Objetivo. Se trata de crear y mantener un fichero de listín telefónico que cumpla las especificaciones que dimos en el ejemplo de los ficheros secuenciales y además las siguientes:

- capacidad para 1000 fichas
- posibilidad de anotar varios teléfonos para un mismo cliente
- generar listas de nombres y teléfonos ordenadas alfabéticamente por nombres
- generar listas de nombres y teléfonos ordenadas por tipo de cliente

Los tipos de cliente son 'A', 'B' y 'C'.

Si usted desea autoexaminarse para comprobar si ha entendido todo lo que hemos explicado hasta ahora, puede intentar diseñar y escribir el programa antes de seguir leyendo.

Diseño. Puesto que el número de fichas es bastante grande, no cabe en una matriz en la memoria. La necesidad de acceder al fichero de dos formas (por nombre y tipo de cliente) sugiere que la solución debe ser un fichero de acceso por claves.

Parte del diseño consiste en decidir cómo se va a almacenar la información y con qué claves. La información que hay que guardar es:

— Nombre
— Número de teléfono
— Tipo de cliente

Las formas en las que el usuario pedirá la información son:

— por el nombre
— por el tipo de cliente

Se necesitará una clave por cada vía de acceso a la información, así que habrá que usar dos rangos, uno para los nombres y otro para los tipos de cliente. Por otra parte, ¿habrá que guardar esta misma información también en el fichero de datos? Esto depende de si la información va a ser necesitada cuando se acceda al registro a través de una clave distinta y de si la información cabe completa en la clave. En este ejemplo, la información se grabará en las claves y en el fichero de datos.

Para que el programa sea más fácil de usar, las claves se limitarán a los seis caracteres primeros de cada nombre, de modo que al usuario le baste con recordar el principio del nombre del cliente para buscar el número.

El programa se puede dividir en las siguientes etapas principales:

- preparar las variables y la pantalla
- abrir o crear el fichero de listín telefónico
- pedir al usuario instrucciones (buscar, añadir una ficha, terminar)
- cerrar el fichero y terminar ordenadamente

A continuación vamos a describir una forma de convertir estas instrucciones en un programa. Muchas de estas técnicas ya han sido descritas antes. Dejamos como ejercicio para el lector la conversión de esta descripción en líneas de programa.

- Preparar las variables y la pantalla
- Si ya existen los ficheros de datos y de índices, abrir el fichero de acceso por claves
- Si no existen, dar la opción de cambiar de disco o de crear un fichero nuevo

- Mostrar las opciones en la pantalla: buscar, añadir, listar por nombre, listar por tipo de cliente, fin
- Captar la opción elegida por el usuario y obedecerla:

Buscar: Captar el nombre, abreviado hasta la longitud de la clave, convertir a mayúsculas (o a minúsculas), utilizar el resultado como clave

Buscar en el rango de nombres con esta clave (SEEKKEY)

Si no se encuentra, informar al usuario y pedir otro nombre

Si se encuentra, leer el registro y mostrar la información en la pantalla

Preguntar si quiere buscar otro

Si quiere, SEEKNEXT

Si se encuentra la misma clave en el mismo conjunto, leer registro, mostrar información y preguntar si quiere buscar otro

Si no se encuentra la misma clave, informar al usuario y pedir otro nombre

Si no quiere buscar otro, salir de la opción

Añadir: Captar nombre, número de teléfono, tipo de cliente

Abreviar nombre hasta la longitud de la clave, convertir a mayúsculas y usar el resultado como clave

Añadir registro: ADDREC con clave de nombre al rango de nombres

Añadir la otra clave: ADDKEY con tipo de cliente al rango de tipos

Salir de la opción

Listar por nombres: SEEKRANK en rango de nombres

Leer registro e imprimirlo

SEEKNEXT

Si se encuentra clave en el mismo rango, imprimir y repetir

Si no, salir de la opción

Listar por tipos: Preguntar tipo de cliente

SEEKKEY con el tipo de cliente especificado

Leer registro e imprimirlo

SEEKNEXT

Si se continúa en el mismo conjunto de claves, imprimir y repetir

Si se ha salido del conjunto de claves o del rango, salir de la opción

Terminar: Cerrar el fichero

7.5 Elección del tipo de fichero

El tipo de fichero que se debe adoptar en cada aplicación viene impuesto normalmente por la complejidad de la información que se ha de almacenar y por la forma de acceso requerida.

Los ficheros secuenciales son los más sencillos y fáciles de usar al principio, pero limitan considerablemente la eficacia de los programas. Los ficheros de acceso por claves son, en cambio, los más difíciles de entender y usar para el principiante, pero a la larga resultan los más adecuados para realizar tareas complejas.

En general, los ficheros secuenciales se utilizan en los casos en que la información se lee siempre en el mismo orden, o bien cuando toda la información cabe en la memoria y puede ser manipulada en ella. Son los idóneos para el almacenamiento de información de naturaleza intrínsecamente secuencial (por ejemplo, texto), a la que nunca se va a acceder más que secuencialmente. Los ficheros secuenciales no pueden ser modificados directamente.

Los ficheros de acceso aleatorio permiten agrupar la información en bloques lógicos (registros) y acceder a los diversos registros por su posición en el fichero. Se puede añadir, modificar y cambiar registros individuales con gran facilidad. Estos ficheros son adecuados para almacenar información que contenga algún dato numérico fácilmente correlacionable con la posición de los registros en el fichero; por ejemplo, índices, control de existencias y otras aplicaciones empresariales.

Los ficheros de acceso por claves (Jetsam) gozan de todas las ventajas de los ficheros de acceso aleatorio, y además permiten buscar los registros, no sólo por su posición, sino también a través de las «etiquetas» que se les ha asignado. Son, con gran diferencia, los ficheros más versátiles, perfectos para almacenar información a la que deban acceder de diferentes formas varios programas. Un fichero de acceso por claves bien diseñado puede ser el núcleo de un complejo paquete de programas que realice tareas integradas tales como gestión de nóminas o contabilidad.

En el apéndice VII (pág. 385) damos el listado completo de un programa de 'Lista de direcciones y teléfonos' que ilustra la utilización de los ficheros de Jetsam.

Índice

Mallard BASIC

CONTENIDO

Capítulo 4: El editor de líneas. Captación de líneas 119

Capítulo 5: Resumen de las órdenes y funciones 129

Capítulo 1

Introducción

El intérprete de BASIC es capaz de ejecutar las órdenes descritas en el capítulo 9. Cada orden está formada por una o varias palabras clave, seguidas en ocasiones por uno o varios parámetros. En general, los parámetros pueden ser expresiones consistentes en alguna combinación de constantes, variables y funciones. BASIC puede manejar cadenas literales e información numérica de diversos tipos, así como ficheros secuenciales y de acceso aleatorio.

BASIC incluye además un sistema de ficheros de acceso por claves, denominado Jetsam, de aplicación general. Este sistema permite crear, leer y modificar ficheros de acceso por claves. El acceso a la información se realiza, de forma indexada o secuencial, a través de los valores de las claves.

Se puede utilizar la versión multiusuario de Jetsam siempre que el sistema operativo esté preparado para ello. En esta versión se dispone de las funciones de bloqueo de ficheros y de registros, tanto en los ficheros secuenciales y aleatorios ordinarios como en los ficheros de acceso por claves de Jetsam.

Las órdenes se le dan a BASIC en líneas. Cada línea puede contener varias órdenes, separadas entre sí por signos de dos puntos, sin más límite que la longitud de la línea. En 'modo directo' las líneas se introducen por la consola (el teclado). En 'modo de programa' BASIC toma las líneas del programa que haya en la memoria.

Cuando se está en modo directo se puede añadir, borrar o modificar líneas del programa. BASIC incluye un editor de líneas que permite corregir la línea actual antes de introducirla en la memoria (y antes de que BASIC la obedezca en modo directo).

Existe además una versión de 'sólo ejecución' que, como su nombre indica, sólo sirve para ejecutar programas, no para escribirlos ni depurarlos. No obstante, cuando el programa está terminado, tiene sus ventajas utilizar esta versión de BASIC.

La versión de 'solo ejecución' es un subconjunto de BASIC en el que se pierden las siguientes funciones:

- El modo directo, y por consiguiente sus órdenes específicas, tales como EDIT
- El editor de líneas
- Control-A (repetir línea anterior)
- Control-C (interrumpir)
- Control-S (suspender BASIC)

Cuando en la descripción de una orden se dice que provoca el retorno a modo directo, se debe entender que lo que esa orden provoca en la versión de 'sólo ejecución' es el retorno al sistema operativo (salvo en los casos en que se entra en el 'modo directo especial'; véase la sección 3.4).

1.1 Metalenguaje

Para la descripción de las órdenes y sus parámetros hemos adoptado un metalenguaje sencillo. La forma de cada orden indica cómo se la debe escribir en la práctica; las variables y partes opcionales están sustituidas por unos «representantes» que hacen referencia a conceptos definidos en algún otro sitio.

Los conceptos se representan por su nombre escrito en *cursiva*. Por ejemplo, donde se requiera una expresión que dé un valor numérico, se la representa por:

expresión-numérica

Lo que aquí aparezca en letra Univers debe ser escrito literalmente en el teclado. Por ejemplo, la forma de la orden STOP es:

STOP

Las partes opcionales se escriben en cursiva y entre corchetes. Por ejemplo, una expresión numérica opcional se representa por:

[expresión-numérica]

Si la parte opcional puede repetirse (es decir, puede figurar un número cualquiera de veces, o incluso ninguna), se le pone un asterisco a la derecha del signo de cerrar corchetes. Por ejemplo, una cadena de dígitos en la que tenga que figurar al menos uno se representa por:

*dígito[dígito]**

En ocasiones se especifica una lista de objetos separados por comas. Estas listas tienen el significado que ilustran los siguientes ejemplos:

lista-de:expresión	equivale a	*expresión[,expresión]**
lista-de:[#]número	equivale a	*[#]número[,[#]número]**

Nótese que cada lista puede constar de un solo objeto. Si una lista contiene varios objetos, éstos van separados por comas.

El capítulo 2 contiene las definiciones de los elementos fundamentales de BASIC. Al principio del capítulo 9 daremos las definiciones de otros conceptos frecuentes, muchas de las cuales estarán basadas en los elementos fundamentales.

Capítulo 2

Los elementos de BASIC

2.1 El juego de caracteres

BASIC supone que se está utilizando el juego de caracteres ASCII.

Los caracteres de código inferior a 32 se consideran 'no imprimibles' o 'no visibles'; muchos de ellos tienen significado especial para BASIC.

Al procesar líneas de órdenes BASIC no distingue mayúsculas de minúsculas, salvo en las cadenas literales.

2.2 El espacio

Los espacios en blanco, los tabuladores y los avances de línea (así como todas sus combinaciones posibles) cuentan como si fuesen un *espacio*. El *espacio* es ignorado en las líneas de órdenes, salvo cuando sirve para señalar el final de algún elemento de la línea o cuando está dentro de una cadena literal. También se ignora el retorno del carro cuando va precedido inmediatamente de un avance de línea.

Cuando hace falta, BASIC realiza una 'compresión de espacios'; esta operación consiste en suprimir todos los *espacios* innecesarios en las líneas según se las va añadiendo al programa. BASIC genera, cuando es necesario, un espacio para separar las palabras clave de los nombres de las variables. La compresión de espacios reduce el tamaño del programa, y por consiguiente el espacio que éste ocupa en la memoria. El programa de instalación habilita o inhibe esta compresión de espacios (véase el apéndice V).

2.3 Nombres

Las variables y las palabras clave tienen *nombres*. El primer carácter de cada *nombre* tiene que ser una letra; los siguientes pueden ser letras, números o el punto. La longitud máxima de los nombres es de 40 caracteres, todos ellos significativos. El *nombre* termina con cualquier carácter que por su naturaleza no pueda formar parte de un *nombre*.

Las palabras clave pueden contener otros caracteres; algunas se expresan en forma de varias palabras.

A diferencia de otros dialectos de BASIC, éste exige que los nombres de las variables estén separados de las palabras clave por al menos un carácter que no pueda formar parte de un nombre. Esto significa que frecuentemente se ha de utilizar el *espacio* como separador.

(Si no fueran necesarios los separadores, entonces los nombres de las variables no podrían contener palabras clave, cosa que sí se permite en este dialecto del BASIC.

2.4 Números

Un *número* puede ser:

número-en-notación-ordinaria[indicador-de-tipo]
o, abreviadamente, *número-ordinario[indicador-tipo]*
número-en-notación-científica[indicador-de-tipo]
o *número-científico[indicador-tipo]*
número-no-decimal
número-de-línea

Véase en la sección 2.6 la descripción de los indicadores de tipo.

Un *número-ordinario* puede tener cualquiera de las siguientes formas:

dígitos
dígitos.[dígitos]
[dígitos].dígitos

donde *dígitos* es una o varias cifras decimales. En los *números-ordinarios* se permiten *espacios* entre los dígitos.

Un *número-científico* puede tener cualquiera de las siguientes formas:

*número-ordinario*E*[signo]dígitos*
*número-ordinario*D*[signo]dígitos*

donde *signo* puede ser el signo 'más' o el 'menos' y *dígitos* es una o varias cifras decimales. Se permite el *espacio* a los lados de los caracteres de la parte exponencial que no son dígitos, pero no entre éstos. El valor del número es el valor del *número-ordinario* multiplicado por la potencia de 10 especificada por *signo* y *dígitos*, es decir, por la parte exponencial. (Nótese que el valor absoluto del exponente no puede exceder de 99.) La diferencia entre la E y la D en la parte exponencial se explica más abajo.

Un *número-no-decimal* puede tener cualquiera de las siguientes formas:

&dígitos-octales
&Odígitos-octales
&Hdígitos-hexadecimales

donde *dígitos-octales* son cifras del margen de 0 a 7 y *dígitos-hexadecimales* son cifras del margen de 0 a 9 o letras del margen A a F (o a a f). El equivalente decimal de un *número-no-decimal* no puede ser mayor que 65535; los valores se interpretan como enteros en 'complemento a dos' en el margen de −32768 a +32767. Se permite el *espacio* a los lados de los caracteres que no son dígitos, pero no entre éstos.

Un *número-de-línea* tiene la forma:

dígitos

donde *dígitos* es al menos una cifra decimal. Los números de línea tienen que estar en el margen de 0 a 65534.

Los *números* tienen que terminar en un carácter que no pueda formar parte de ningún *número*.

2.5 Constantes literales

Una *constante-literal* es una colección de caracteres cualesquiera escritos entre comillas, o bien escritos empezando con comillas y terminando con un código de fin de línea (pero en este caso no se incluye en la cadena el *espacio* que se pueda haber escrito al final). En las constantes literales se puede incluir caracteres ajenos al margen normal de ASCII, a excepción de 'Null' (carácter número 0).

2.6 Tipos de datos

BASIC maneja dos tipos de datos: numéricos y literales.

Las cadenas literales pueden tener una longitud de entre 0 y 255 caracteres. Los caracteres de las cadenas se representan internamente mediante enteros del margen de 0 a 255 (es decir, mediante bytes sencillos). Se supone que se está utilizando el juego de caracteres ASCII, pero no obstante casi todas las operaciones de proceso de cadenas literales son independientes de los valores de los caracteres.

Los datos numéricos pueden ser de tres clases: números enteros, números de precisión sencilla y números de doble precisión.

Los enteros se representan internamente mediante dos bytes en complemento a dos, de modo que sus valores pueden estar entre −32768 y +32767. En algunas ocasiones BASIC los interpreta como enteros sin signo del margen de 0 a 65535 (véase la sección 2.10).

Los números de precisión sencilla se almacenan en formato de punto flotante en cuatro bytes: tres bytes para la mantisa y uno para el exponente binario. El máximo valor absoluto representable es aproximadamente 1.7E+38; el menor número distinto de cero es aproximadamente 2.9E−39. Con la mantisa de tres bytes se consigue una precisión de algo más de siete cifras significativas.

Los números de doble precisión se almacenan en formato de punto flotante en ocho bytes: siete bytes para la mantisa y uno para el exponente binario. El máximo valor absoluto representable es aproximadamente 1.7E+38; el menor número distinto de cero es aproximadamente 2.9E−39. Con la mantisa de siete bytes se consigue una precisión de algo más de dieciséis cifras significativas.

El tipo de un *número* viene impuesto por su forma:

- Los *números-no-decimales* son siempre enteros; los valores sin signo son interpretados como su equivalente en complemento a dos (lo que equivale a restar 65536 de los valores mayores que 32767).
- Los *números-ordinarios* (números en notación ordinaria) son tratados como enteros siempre que carezcan de parte decimal y su valor encaje en el margen admisible para los enteros. De lo contrario, se los trata como números de precisión sencilla si no tienen más de siete cifras significativas, y como números de doble precisión si tienen ocho o más cifras significativas.
- Los *números-científicos* (números en notación científica) son tratados en precisión sencilla o en doble precisión, dependiendo del número de cifras significativas y de la letra que preceda a la parte exponencial. La E especifica precisión sencilla; la D, doble precisión.
- El tipo de los *números-ordinarios* y los *números-científicos* se puede especificar con un *indicador-de-tipo* para que sea distinto del implícito, es decir, del que le correspondería en ausencia de otra especificación. Los indicadores de tipo son los siguientes:

  % que fuerza el redondeo del número al entero más próximo. Si el valor queda fuera del margen permitido para los enteros, da lugar a un error de sobrepasamiento.

  ! que fuerza la conversión del número a precisión sencilla.

  # que fuerza la conversión del número a doble precisión.

  (Nótese que estos indicadores de tipo son los mismos que los de los nombres de las variables; véase más abajo.)

En ocasiones BASIC necesita un número entero del margen de 0 a 65535 almacenado en forma de entero de dos bytes. En tales casos interpreta los enteros en complemento a dos habituales como si fueran enteros sin signo (sumando 65535 a los enteros en complemento a dos negativos). Véase la sección 2.10.

2.7 Variables

Las variables tienen tres atributos: nombre, tipo y organización. Un *nombre-de-variable* tiene la forma *nombre[indicador-de-tipo]*. El tipo de los datos que pueden ser asignados al *nombre-de-variable* viene determinado por el *indicador-de-tipo*, o bien por el tipo implícito actual. Los *indicadores-de-tipo* son:

% para variables enteras
! para variables de precisión sencilla
# para variables de doble precisión
$ para variables literales

El tipo implícito depende del primer carácter del *nombre-de-variable* y puede ser establecido dinámicamente con las órdenes DEFINT, DEFSGL, DEFDBL y DEFSTR. El tipo implícito inicial para todas las variables es 'precisión sencilla'.

Las variables pueden ser *variables-sencillas* o *variables-matriciales*. Estas últimas son colecciones de variables de un mismo tipo. En la sección siguiente describiremos las matrices.

Nótese que se puede utilizar el mismo *nombre* para designar variables distintas de tipos distintos.

En BASIC no es necesario declarar las variables antes de utilizarlas, sino que empiezan a existir cuando se las menciona por primera vez (con valor inicial 0 en el caso de las numéricas o igual a la cadena vacía en el caso de las literales).

2.8 Matrices

BASIC puede manejar matrices de datos de cualquier tipo. Una matriz es una colección de variables, todas del mismo tipo y con un nombre común, en la que cada variable particular está identificada por uno o varios subíndices:

nombre-de-variable(*subíndices*)
nombre-de-variable[*subíndices*]

donde *subíndices* es *lista-de: expresión-entera* y la *expresión-entera* debe tomar un valor dentro del margen correcto.

El tipo de los datos que pueden ser almacenados en la matriz viene dado por el tipo del *nombre-de-variable*.

Las matrices pueden tener una o varias dimensiones. Pueden ser declaradas explícitamente con DIM, o implícitamente por el hecho de utilizarlas. Al declarar una matriz explícitamente se establece el número de dimensiones y el valor máximo del subíndice en cada dimensión (en la declaración implícita ese límite máximo se supone igual a 10). Las posteriores referencias a la misma matriz deben incluir el mismo número de subíndices, todos ellos en el margen adecuado. Una vez declarada la matriz, explícita o implícitamente, ya no se le puede cambiar el número de dimensiones.

Si no se especifica otra cosa, el valor mínimo de todos los subíndices es 0; pero se lo puede establecer en 0 o en 1 mediante la orden OPTION BASE. Esta orden no puede ser ejecutada mientras exista alguna matriz.

2.9 Compatibilidad de tipos y conversión

BASIC normalmente trata todos los tipos numéricos como compatibles unos con otros. El tipo literal sólo es compatible consigo mismo.

Cuando maneja información numérica, BASIC realiza automáticamente todas las conversiones de tipo que necesita. Si no se le impone otra cosa, siempre elige el tipo más «amplio»

de los que intervengan en la operación. Así, convierte enteros a precisión sencilla, y precisión sencilla a doble precisión. Esta conversión no puede fallar nunca (es decir, BASIC siempre la realiza sin error).

También se puede especificar que un número sea tratado en una representación que de otra forma no le correspondería. El paso por este procedimiento de doble precisión a precisión sencilla no entraña más que un redondeo de 16 a 7 cifras significativas; aunque la pérdida de precisión es considerable, raramente se produce error de sobrepasamiento. Para poder convertir cualquiera de las dos representaciones en punto flotante a entero es necesario que el valor del número, después de redondeado, quede dentro del margen de los enteros, que va de -32768 a +32767; de lo contrario, se produce el error n.º 8 (sobrepasamiento). La conversión de cualquier forma de representación a 'entero sin signo' está descrita en la sección 2.10.

2.9.1 Redondeo

Hay varias maneras de redondear un número de una forma a otra más restrictiva. Si el número X de la forma más amplia no puede ser representado exactamente en la forma más restrictiva, BASIC determina entre qué valores, X1 y X2, exactamente representables, queda el número X, donde X1 y X2 son los valores más próximos a X tales que X1<X<X2. A continuación el redondeo se puede realizar por alguno de los métodos siguientes:

- Truncamiento (redondeo hacia cero):
  se suprimen los dígitos sobrantes; X→X1 para valores negativos y X→X2 para los positivos
- Redondeo hacia $-\infty$: X→X1
- Redondeo hacia $+\infty$: X→X2
- Redondeo hacia el más próximo: X→X1 o X→X2, dependiendo de cuál de estos valores sea más próximo a X.

Cuando X está a medio camino entre X1 y X2, BASIC redondea hacia el más alejado de cero.

Si no se indica otra cosa, el término «redondeo» se debe entender como «redondeo al más próximo».

2.10 Enteros sin signo

En ocasiones BASIC requiere un valor entero del margen de 0 a 65535 que esté en forma de entero de dos bytes; por ejemplo, en la función PEEK, cuyo argumento es una dirección de memoria.

Los valores enteros habituales (en complemento a dos) se interpretan como si fuesen sus equivalentes sin signo, lo que equivale a sumar 65536 a los enteros negativos.

Si BASIC recibe un número en punto flotante cuando lo que necesita es un entero sin signo, primero redondea el número a entero (pero todavía no de dos bytes); su valor tiene que quedar entre −32768 y 65535. A continuación lo pasa de la representación actual, que sigue siendo punto flotante, a entero en complemento a dos, para lo cual tiene que restar 65536 de todos los números mayores que 32767. Una vez obtenido este entero en complemento a dos, BASIC lo interpreta como entero sin signo, según se ha explicado antes.

2.11 Expresiones

Hay cuatro tipos de expresiones: literales, numéricas, de relación y lógicas. Los niveles de prioridad de los operadores están diseñados de forma que las expresiones se calculen en un orden lo más parecido al habitual en los cálculos a mano. Cuando el nivel de prioridad es el mismo para dos operadores, la ambigüedad en el orden de ejecución se resuelve forzando el cálculo mediante paréntesis; en su defecto, las expresiones se calculan de izquierda a derecha.

2.11.1 Expresiones numéricas

La sintaxis de las expresiones numéricas es como sigue:

expresión-numérica es *elemento-numérico[operador expresión-numérica]*
elemento-numérico es *[operador-monádico]elemento-numérico*
o *constante-numérica*
o *variable-numérica*
o *función-numérica*
o (*expresión-numérica*)
o (*expresión-de-relación*)

donde *variable-numérica* es una referencia a una variable sencilla o a un elemento de una matriz de cualquiera de los tipos numéricos. Una *función-numérica* es una función que genera un valor de cualquiera de los tipos numéricos.

Los *operadores* y los *operadores-monádicos* (u *operadores-unarios*) son, en orden de prioridad, los siguientes:

↑	Potenciación	Eleva el valor que hay a su izquierda a la potencia especificada por el valor que hay a su derecha. Fuerza ambos valores a precisión sencilla antes de realizar la operación y produce un resultado en precisión sencilla.
+	Más monádico	Nótese que todos los *operadores-monádicos* tienen prioridad inferior a la potenciación.
−	Menos monádico	
*	Multiplicación	Multiplicación en punto flotante o de enteros.

/	División	División en punto flotante. Si alguno de los operandos es entero, se lo fuerza a precisión sencilla antes de realizar la operación.
\	División entera	Fuerza ambos operandos a representación entera. El resultado se trunca a un entero.
MOD	Módulo	Fuerza ambos operandos a representación entera. El resultado es el resto de la división entera.
+	Suma	Suma de enteros o en punto flotante.
−	Resta	Resta de enteros o en punto flotante.

Nótese lo siguiente:

i. La potenciación tiene la prioridad máxima.

ii. El más monádico y el menos monádico tienen la misma prioridad.

iii. La multiplicación y la división tienen la misma prioridad.

iv. La división entera y MOD tienen la misma prioridad, que es inferior a la de la multiplicación y la división ordinaria. Esto implica que:

```
x\y*y=x\(y*y)
```

y que

```
x\y*y <> (x\y)*y
```

v. MOD se define de forma tal que

```
x MOD y=x-((x\y)*y)
```

Por consiguiente, x MOD y tiene siempre el mismo signo que x (a menos que sea igual a cero).

vi. La suma y la resta tienen igual prioridad.

vii. A no ser que el operador requiera otra cosa, se fuerza el tipo de los operandos para que ambos tengan el mismo (el más amplio).

La división por cero genera el error n.° 11. Si en el momento de producirse este error está activa una orden ON ERROR GOTO, BASIC salta a la línea especificada; si no, se emite el mensaje Divison by zero y además:

- En el caso de la división entera (\) se interrumpe la ejecución de la línea de órdenes o del programa.
- En el caso de la división en punto flotante (/), cualquiera que sea el tipo de los operandos, la ejecución no se interrumpe, ya que el cero podría provenir de una operación anterior cuyo resultado fuese menor (en valor absoluto) que el mínimo representable (2.9E−39). BASIC supone que el resultado de la división ha sido el mayor número representable (aproximadamente 1.7E+38), con el signo que corresponda, y continúa.

Si se produce sobrepasamiento cuando los argumentos son enteros, BASIC los pasa a precisión sencilla y repite la operación.

Si se produce sobrepasamiento cuando los argumentos están en punto flotante, BASIC genera el error n.° 6 ("Overflow"). Si en ese momento está activa una orden ON ERROR GOTO, la ejecución salta a la línea especificada. De lo contrario, BASIC emite el mensaje de error "Overflow", asigna al resultado de la operación el mayor número representable (aproximadamente 1.7E+38) y continúa.

2.11.2 Expresiones literales

La sintaxis de las expresiones literales es como sigue:

expresión literal es *elemento-literal*[+*expresión-literal*
elemento-literal es *variable-literal*
o *constante-literal*
o *función-literal*
o (*expresión-literal*)

donde *variable-literal* es una referencia a una variable sencilla o a un elemento de una matriz literal. Una *función-literal* es una función que genera una cadena literal.

El efecto del operador + sobre las cadenas es unirlas («concatenarlas»): produce una cadena nueva que consta de la primera seguida inmediatamente de la segunda. Si la suma de las longitudes de los operandos es mayor que 255, no se realiza la concatenación y se genera el error n.° 14.

2.11.3 Expresiones de relación

Las expresiones de relación comparan dos valores numéricos o dos cadenas literales. Pueden tener las siguientes formas:

expresión-de-relación es *expresión-numérica operador-de-relación expresión-numérica*
o *expresión-literal operador-de-relación expresión-literal*

Los *operadores-de-relación* tienen prioridad inferior a la de los operadores que intervienen en las expresiones numéricas y literales, pero superior a la de todos los operadores lógicos.

Los *operadores-de-relación* son los siguientes:

<	Menor que
<= =<	Igual o menor que
=	Igual a
>= =>	Igual o mayor que
>	Mayor que
<>	Distinto de

Si los argumentos son numéricos, se homogeneizan sus tipos según se ha explicado en el apartado 2.9.1. El significado de las relaciones es el que tienen habitualmente en el lenguje ordinario y en el científico.

Sin embargo, cuando los argumentos son cadenas literales, el significado requiere alguna aclaración:

- Dos cadenas son iguales si sus longitudes son iguales y contienen los mismos caracteres en las mismas posiciones.
- Una cadena es menor que otra si:
  - o bien son iguales hasta el final de la primera cadena y la segunda es más larga
  - o bien el primer carácter en el que no coinciden las dos cadenas es menor en la primera que en la segunda (la comparación se realiza con los códigos de los caracteres).

BASIC no dispone de un tipo de información 'booleana'; las expresiones de relación dan el valor −1 cuando son 'verdaderas' y el 0 cuando son 'falsas'. Obsérvese que las órdenes IF y WHILE tratan el valor 0 como 'falso' y todos los demás como 'verdadero'.

2.11.4 Expresiones lógicas

BASIC no dispone del tipo 'booleano' de información. Las expresiones lógicas realizan operaciones booleanas bit a bit entre enteros. Tienen la siguiente forma:

argumento [operador-lógico argumento]

donde *argumento* es NOT *argumento*
o *expresión-numérica*
o *expresión-de-relación*
o (*expresión-lógica*)

Los argumentos de los operadores lógicos se fuerzan a representación entera; si alguno de ellos no encaja en el margen permitido para enteros, se produce el error nº 6. Una vez convertidos los valores a enteros en complemento a dos, se realiza la operación para cada uno de los 16 bits.

El operador monádico NOT invierte todos los bits de su argumento (el 0 a 1 y el 1 a 0).

Los operadores diádicos, en orden de prioridad, y sus efectos sobre los bits son los siguientes:

AND Resultado 0 a menos que los bits correspondientes en ambos operandos sean 1.
OR Resultado 1 a menos que los bits correspondientes en ambos operandos sean 0.

XOR Resultado 1 a menos que los bits correspondientes en ambos operandos sean iguales.

EQV Resultado 0 a menos que los bits correspondientes en ambos operandos sean iguales.

IMP Resultado 1 a menos que el bit correspondiente en el operando de la derecha sea 1 y el de la izquierda sea 0.

El valor de toda operación de relación es −1 o 0. El valor −1 se representa mediante un entero cuyos bits son todos '1'; el 0, con un entero cuyos bits son todos '0'. Las operaciones lógicas aplicadas a argumentos de este tipo sólo pueden dar los resultados −1 y 0.

2.12 Funciones

Las funciones son subrutinas que toman cierto número de parámetros de entrada y generan un valor de salida. Las funciones se invocan al utilizarlas como argumentos en las expresiones; el valor del argumento es el generado por la función. Hay tres clases de funciones: intrínsecas, definidas por el usuario y externas.

2.12.1 Funciones intrínsecas

Las funciones intrínsecas forman parte del lenguaje BASIC; están descritas en el capítulo 1. A diferencia de lo que ocurre con las definidas por el usuario, las funciones intrínsecas generan resultados cuyo tipo no está necesariamente determinado por el nombre de la función. Por ejemplo, el resultado de la función ABS es del mismo tipo que el argumento. Algunas funciones intrínsecas pueden llevar parámetros opcionales; otras llevan parámetros de tipos diferentes; ninguna de estas dos cosas es posible en las funciones definidas por el usuario.

2.12.2 Funciones definidas por el usuario

Estas funciones se definen mediante DEF FN; esta orden asocia un nombre a una lista de parámetros formales y a una expresión.

Los nombres de las funciones tienen la forma FN*nombre[indicador-de-tipo]*. Las letras FN sirven solamente para distinguirlos de los nombres de las variables y no cuentan a efectos de la longitud máxima del *nombre*, que es de 40 caracteres. Al igual que ocurre con los nombres de las variables, las funciones que tienen el mismo nombre y distinto indicador de tipo son funciones distintas.

La lista de parámetros formales declara cierto número de variables locales para la función. Cuando se invoca la función se debe proporcionar la lista de parámetros concretos sobre los que la función deba actuar; estos parámetros tienen que concordar en número y tipo con los parámetros formales, si bien BASIC realizará la conversión de unos tipos de valores numéricos a otros cuando sea necesario. Los parámetros concretos para cada invocación de

la función son expresiones; BASIC las calcula automáticamente antes de realizar la invocación y asigna sus valores a los parámetros formales.

El cuerpo de la función es una expresión única. No es posible incluir estructuras de control de ningún tipo. La expresión puede hacer referencia a otras funciones y a variables globales, así como incluir los parámetros formales. (Aunque, por supuesto, un parámetro formal que tenga el mismo nombre que una variable global hace que la variable global sea inaccesible.)

Cuando se invoca una función, el resultado se convierte al tipo especificado por el nombre de la función. El valor generado por la función es el que resulta de esta conversión.

2.12.3 Funciones de usuario externas

Una función de usuario externa es una rutina en código de máquina cuya dirección inicial en la memoria se declara en una orden DEF USR. Las funciones externas pueden llevar un parámetro de cualquier tipo y generan un valor de cualquier tipo. En el apéndice III se describe detalladamente la gestión de rutinas externas desde BASIC.

2.13 Fracciones decimales y punto flotante binario

Es un hecho lamentable (e inevitable) que muy pocas fracciones decimales tienen una representación exacta cuando se las convierte a números binarios en punto flotante. No obstante, BASIC redondea los números en punto flotante antes de imprimirlos, de modo que esta inexactitud puede quedar oculta.

Este efecto se puede observar comparando valores calculados por procedimientos distintos y que, de no ser por la inexactitud mencionada, debieran ser iguales. Aunque sus versiones impresas sean iguales, las representaciones binarias internas pueden ser diferentes. También se observa el efecto cuando se restan los dos valores y la diferencia no es cero.

Esta inexactitud se pone de manifiesto cuando se convierten los números de precisión sencilla a doble precisión. Por ejemplo, si se convierte 0.1! a doble precisión, el resultado es 0.1000000014901161#, parecido pero no idéntico.

A menos que se eviten todas las fracciones decimales (por ejemplo, multiplicando todos los números por las potencias de 10 adecuadas), este problema es irresoluble. Se puede redondear los números explícitamente con la función intrínseca ROUND; también se puede operar con los resultados inexactos y tener en cuenta las pequeñas diferencias cuando se compare o reste los números.

Capítulo 3

Modo directo y modo de programa

BASIC puede trabajar en dos modos principales: modo directo y modo de programa.

En modo directo, BASIC capta las líneas de órdenes a través de la consola (el teclado) y las procesa en cuanto recibe el retorno del carro con que termina cada línea. En modo de programa, BASIC toma las líneas del programa que hay actualmente en la memoria y ejecuta las órdenes que encuentra en ellas. Salvo raras excepciones, todas las órdenes de BASIC son válidas para ambos modos de funcionamiento.

El modo directo es posible en las versiones de 'sólo ejecución'; no obstante, véase la sección 3.4.

3.1 Modo directo

El modo directo se subdivide en los siguientes submodos:

- Órdenes directas.

  Las órdenes introducidas por el teclado se ejecutan inmediatamente.

- Introducción de programas.

  Se introducen líneas de programa nuevas, o en sustitución de las ya existentes, o se borran líneas del programa actual.

- Modificación de programas.

  Se modifican líneas ya existentes.

En el submodo de órdenes directas se puede utilizar variables. Sin embargo, se pierden los valores de todas las variables cuando se entra en el modo de programa mediante la orden RUN.

En el submodo de introducción de programas las líneas se escriben precedidas de un número de línea. Cuando se termina la línea (con un retorno del carro), BASIC la incorpora al programa actual, en el que sustituye a la línea que pudiera haber con el mismo número. (Si una línea consiste solamente en el número de línea, borra la línea que pudiera haber en el programa con el mismo número.) La orden AUTO hace que BASIC genere automáticamente los números de línea para este submodo.

El submodo de modificación de programas se activa mediante la orden EDIT (véase el capítulo 4).

BASIC no realiza ninguna comprobación de las líneas en el momento en que se las introduce o modifica.

3.2 Modo de programa

En el modo de programa, BASIC pasa automáticamente a la línea siguiente en cuanto termina de obedecer las instrucciones que encuentra en la línea actual. El orden de ejecución es el dictado por los números de línea, salvo cuando alguna estructura de control obligue a alterarlo.

El 'modo de gestión de errores' es un submodo del modo de programa que se activa cuando se produce un error y está activa una orden ON ERROR GOTO.

A excepción de la versión de 'sólo ejecución', el programa se interrumpe cuando se introduce por la consola un código control-C; BASIC vuelve entonces al modo directo e imprime un mensaje "Break". La ejecución se puede reanudar (posiblemente tras comprobar, por ejemplo, el valor de alguna variable) con la orden CONT, a condición de que no se haya modificado el programa.

Al volver del modo de programa al modo directo, BASIC preserva los valores de las variables, y esto puede ser útil en la depuración de programas.

3.3 Suspensión de BASIC

El código control-S introducido por la consola hace que BASIC suspenda momentáneamente la ejecución del programa o la salida hacia la impresora o la consola. Un segundo control-S hace que BASIC continúe como si no hubiese ocurrido nada. Si después del primer control-S se introduce cualquier otro carácter, BASIC continúa, pero el carácter no se pierde y puede ser procesado más tarde.

Al igual que ocurre con control-C, el código control-S no ejerce ningún efecto en las versiones de 'sólo ejecución'.

3.4 Modo directo especial en 'sólo ejecución'

Hay dos variantes de las versiones de 'sólo ejecución': una con 'modo directo especial' y otra sin él.

La variante que carece del modo directo especial retorna inmediatamente al nivel de consola del sistema operativo en cuanto el programa fracasa o se detiene por cualquier otra razón.

En la otra variante el modo directo especial se activa cuando BASIC detecta un error o cuando ejecuta una orden STOP o END. En esa situación BASIC emite el mensaje "Direct Mode" y puede aceptar un número limitado de órdenes, a saber,

DIR	MID$	GOTO
PRINT	LSET	GOSUB
LET	RSET	SYSTEM

BASIC no acepta ninguna otra orden ni permite la introducción de nuevas líneas en el programa. La asignación de valores se debe hacer con LET.

Este modo está diseñado para que permita examinar (PRINT) el estado de un programa que ha fracasado, y quizá también introducir algún cambio (LET, MID$, LSET, RSET). Si se consigue remediar el fallo, se puede volver a poner en marcha el programa (con GOTO o GOSUB); si no, se puede abandonar el programa (con SYSTEM).

Para utilizar este modo es necesario conocer a fondo el programa que ha fracasado, o hacerse asesorar por alguien que lo conozca. Cuando un programa fracasa puede interesar volver a ponerlo en marcha para evitar la pérdida de los datos.

Capítulo 4

El editor de líneas. Captación de líneas

El editor de líneas se activa cuando se ejecuta las órdenes EDIT y AUTO, cuando se introduce el código control-A o cuando BASIC detecta un error de sintaxis (error n.º 2). BASIC está implícitamente en el editor de líneas mientras está recibiendo una línea en modo directo, por lo que la línea actual puede ser editada antes de terminarla (con el retorno del carro).

En las demás ocasiones en que BASIC espera la introducción de una línea por la consola, utiliza una rutina de captación de líneas en la que el único recurso de edición es borrar el último carácter escrito.

La orden EDIT activa el editor de líneas y ofrece la línea especificada, lista para su modificación. Cuando BASIC detecta un error de sintaxis, entra en el editor de líneas exactamente igual que si se hubiera ejecutado una orden EDIT para la línea errónea. Cuando AUTO genera el número de una línea que ya existe, BASIC la ofrece para su edición.

Siempre que se está introduciendo líneas de programa se las puede editar antes de terminarlas. En modo directo las líneas de órdenes también pueden ser editadas antes de entregárselas a BASIC para que las ejecute. Pulsando control-A inmediatamente después de un retorno del carro, en modo directo, se recupera la última línea de órdenes, la cual queda en la pantalla lista para ser modificada y ejecutada.

Si al editar una línea de programa se modifica el número de línea, la resultante se inserta en el lugar que le corresponda en el programa, sustituyendo a la que pudiera haber con el mismo número; la línea original permanece inalterada.

El editor de líneas no está incluido en las versiones de 'sólo ejecución', en las cuales los errores de sintaxis hacen que BASIC vuelva al sistema operativo o entre en el 'modo directo especial' (véase la sección 3.4).

BASIC ofrece dos editores de líneas. Uno es el 'editor de líneas basado en la pantalla', que muestra siempre la última versión de la línea que está siendo editada. El otro es un 'editor basado en órdenes', que no muestra la línea actual en la pantalla.

Para poder gestionar la edición basada en la pantalla, BASIC necesita conocer ciertos datos sobre los códigos que mueven el cursor. Con objeto de facilitar su uso, el editor puede ser adaptado a los diferentes teclados. Dentro de BASIC hay una tabla que define la forma de las órdenes de pantalla y de las órdenes de edición. La preparación de esta tabla es un proceso denominado 'instalación de BASIC' que se realiza mediante un programa de instalación suministrado con el lenguaje. En el apéndice V se describe el programa de instalación. Las versiones de BASIC que ya han sido instaladas en un ordenador concreto se suministran sin programa de instalación.

La principal ventaja del editor de líneas basado en la pantalla es su facilidad de uso. La desventaja es que una versión instalada de BASIC es específica de la máquina y el editor puede no funcionar en otras. En cambio, el editor basado en órdenes no requiere instalación, y por lo tanto es independiente de la máquina; su desventaja es que resulta más incómodo de usar.

Puesto que las versiones de 'sólo ejecución' no incluyen ningún editor de líneas, no requieren instalación. No obstante, puede ser conveniente establecer la anchura de la pantalla y la columna en la que se realiza automáticamente el CR/LF; en ausencia de otra especificación, ambos valores son 80.

4.1 Captación de líneas

Cuando BASIC espera una línea, como consecuencia de INPUT, LINE INPUT o RANDOMIZE, utiliza una sencilla rutina de captación de líneas. (En las versiones de 'sólo ejecución' ésta es la única rutina de entrada por la consola.) Esta rutina acepta todos los caracteres que no sean códigos de control y los coloca al final de la línea actual. Reconoce además los siguientes caracteres de control:

- Control-C (&H03), que abandona la entrada e ignora la captación de la línea. Interrumpe el programa.
- Retorno del carro (&H0D), que completa la línea y se la entrega a BASIC.
- Retroceso del cursor (&H08) o DEL (&7F), que borran el último carácter de la línea; a la consola se envía un espacio (&H20), un retroceso del cursor (&H08) y un espacio (&H20).
- Tab (&H09), que se inserta como tal en la línea; a la consola se envían los espacios necesarios para llevar el cursor hasta el siguiente tope de tabulación; hay un tope de tabulación cada ocho columnas.
- Avance de línea (&H0A), que se inserta como tal en la línea; a la consola se envía un retorno del carro seguido de un avance de línea.

Todos los demás caracteres de control (margen de &H00 a &H1F) son ignorados. Todos los demás caracteres (incluidos los del margen &H80 a &HFF) son colocados al final de la línea y exhibidos en la pantalla.

4.2 El editor de líneas basado en la pantalla. Instalación

Este editor muestra la línea actual en la pantalla y permite que el cursor se mueva por la línea, así como la inserción y el borrado de texto en la posición del cursor. En todo momento se puede ver en la pantalla la última versión de la línea.

Aquí vamos a describir las órdenes que acepta el editor de líneas. Las mencionaremos por sus nombres, ya que las teclas que las generan son algo que se decide en el proceso de instalación de BASIC. Las órdenes pueden ser invocadas por cualquier combinación de caracteres, pero es deseable que se adopten códigos de control sencillos, o bien códigos de control seguidos de uno o dos caracteres.

La línea que está siendo editada puede no caber en una sola línea de la pantalla. El término «línea de pantalla» hace referencia a la porción de la línea actual que se muestra en una sola fila de caracteres en la pantalla. El editor necesita conocer la anchura de la pantalla y si ésta tiene activada la función de CR/LF automático. Esta información se le suministra a BASIC en el proceso de instalación, pero también puede ser modificada mediante la orden WIDTH.

4.2.1 Movimientos del cursor

El cursor se puede mover a lo largo de la línea a derecha y a izquierda, y también arriba y abajo cuando la línea ocupa varias líneas de pantalla (pero sin sobrepasar la primera y la última). Además, también se lo puede llevar hasta la siguiente aparición de un carácter especificado.

El cursor no está limitado a las posiciones de la línea que están ocupadas por caracteres; por ejemplo, puede estar en el «hueco» que hay entre un tabulador y el carácter siguiente. Si en el momento de insertar o borrar texto el cursor está sobre un carácter, BASIC lo mueve a la izquierda o a la derecha antes de realizar la inserción o el borrado.

Las órdenes de movimiento son:

Cursor a la izquierda. Desplazar una posición hacia la izquierda.

Si ya está en la primera columna de la primera línea de pantalla, ningún efecto.

Si está en la primera columna de la segunda línea de pantalla (o de alguna posterior), retrocede hasta la última columna de la anterior línea de pantalla.

Cursor a la derecha. Desplazar una posición hacia la derecha.

Si ya está en la última columna de la última línea de pantalla, ningún efecto.

Si está en la última columna de la penúltima línea de pantalla (o de alguna anterior), avanza hasta el principio de la siguiente línea de pantalla.

Cursor arriba. Subir a la misma columna de la anterior línea de pantalla.

Si ya está en la primera línea de pantalla, ningún efecto.

Cursor abajo. Bajar a la misma columna de la siguiente línea de pantalla.

Si ya está en la última línea de pantalla, ningún efecto.

Buscar carácter. Avanzar hasta la siguiente aparición del carácter especificado.

La orden toma el siguiente carácter introducido (que puede ser cualquiera) y lo busca desde la posición actual hacia delante. Antes de iniciarse la búsqueda, el cursor se desplaza una posición hacia la derecha para no incluir el carácter que está en la posición actual. Si no se encuentra el carácter, el cursor queda al final de la línea.

4.2.2 Adición de texto

Todos los caracteres escritos en el teclado son añadidos a la línea actual, a excepción de los caracteres de control (&H7F y &H00 a &H1F) que, salvo 'tab' y 'avance de línea', son rechazados. La forma en que el carácter nuevo se incorpora a la línea depende de si está en vigor el modo de 'inserción' o el de 'sobreescritura'. El editor se pone en modo de inserción cada vez que se inicia la edición de una línea.

Las órdenes relacionadas con la adición de texto son las siguientes:

Conmutador inserción/sobreescritura. Bascular entre los modos de inserción y sobreescritura.

Cuando se introduce un carácter en modo de inserción, el carácter que está en la posición actual y todos los que quedan a su derecha se desplazan hacia la derecha para dejar sitio para el nuevo.

Cuando se introduce un carácter en modo de sobreescritura, el carácter que está en la posición actual se borra y es sustituido por el nuevo.

El modo de inserción se selecciona automáticamente cada vez que se inicia la edición de una línea.

Carácter literal: "\". El carácter que se introduce inmediatamente después de la orden 'carácter literal' no es tratado como orden para el editor, sino incorporado a la línea. Tal carácter puede ser 'tab' (&H09), 'avance de línea' (&H0A) o cualquier otro cuyo valor esté en el margen de &H20 a &HFF, salvo el &H7F.

La principal aplicación de esta orden es incluir en la línea los caracteres 'tab' y 'avance de línea' cuando están siendo utilizados como órdenes del editor.

El propio carácter '\' puede ser incluido en la línea escribiéndolo dos veces.

Los caracteres 'tab' y 'avance de línea' pueden aparecer en la línea, y también ser añadidos a ella. Tienen los efectos especiales que describimos a continuación.

Tab aparece en forma de espacios hasta la siguiente posición de tabulación. Hay una posición de tabulación cada ocho columnas. Si el cursor está sobre el primer espacio, se dice que está **en** el tab; si está sobre un espacio posterior, se dice que está **después** del tabulador.

El **avance de línea** se representa por un espacio y un salto a la siguiente línea de pantalla. Si el cursor está sobre ese espacio o sobre alguno posterior de la misma línea de pantalla, se dice que está en el avance de línea.

4.2.3 Borrado de texto

Los caracteres se borran implícitamente cuando se añade texto nuevo en modo de sobreescritura. Las siguientes órdenes los borran explícitamente:

Borrar hacia delante. Si no hay ningún carácter bajo el cursor, desplazar éste hacia delante hasta que llegue a un carácter. Borrar el carácter que está bajo el cursor.

Borrar hacia atrás. Desplazar el cursor una posición hacia atrás. Si entonces no hay ningún carácter bajo el cursor, seguir desplazándolo hacia atrás hasta que lo haya. Borrar el carácter que está bajo el cursor.

Borrar hasta el próximo carácter. Borrar desde la posición actual hasta aquélla donde se encuentre el carácter especificado.

La orden toma el siguiente carácter introducido (que puede ser cualquiera) y lo busca desde la posición actual hacia delante, borrando al mismo tiempo todos los caracteres por los que pasa. Antes de iniciarse la búsqueda, el cursor se desplaza una posición hacia la derecha para no incluir el carácter que está en la posición actual. Si no se encuentra el carácter, se borra todo el resto de la línea.

4.2.4 Fin de la edición

La edición puede concluir de dos formas: entregando a BASIC la nueva versión de la línea o abandonando la línea y las modificaciones introducidas. Las órdenes son las siguientes:

Completar la edición y validar la línea. Si la línea tiene número de línea, se la inserta en su posición en el programa, sustituyendo la que pudiera haber antes con ese mismo número. Si está activa la orden AUTO, se genera el siguiente número de línea y se vuelve a entrar en el editor. Si durante la edición se ha modificado el número de línea, la línea original no se modifica.

Si la línea no va precedida de número, se la trata como línea de órdenes para modo directo y por lo tanto se la ejecuta inmediatamente.

(Es aconsejable que esta orden sea realizada con el retorno del carro.)

Abandonar la edición. Se ignoran todos los cambios introducidos. Se desactiva AUTO y BASIC vuelve al modo directo.

(Es aconsejable que esta orden sea realizada con control-C.)

4.3 El editor basado en órdenes

Este editor no requiere instalación. Esto es posible porque el editor no trata de mostrar en la pantalla la versión actualizada de la línea y porque además se utiliza un reducido número de códigos de control como órdenes del editor.

El editor de órdenes está siempre en alguno de los tres modos siguientes: modo de órdenes, modo de inserción y modo de sobreescritura.

- En **modo de órdenes** todas las pulsaciones de teclas son interpretadas como órdenes para el editor.
- En **modo de inserción** todo carácter «legal» introducido por el teclado es insertado en la posición actual dentro de la línea y visualizado en la posición actual del cursor. Los caracteres que están a la derecha del cursor se desplazan a la derecha, pero esto no es observable en la pantalla. Si la línea está llena y se intenta insertar otro carácter, suena un pitido y se ignora el carácter.

  En este modo el editor acepta y obedece un número limitado de códigos de control.
- En **modo de sobreescritura** todo carácter «legal» introducido por el teclado es incluido en la línea en sustitución del que está en la posición actual y visualizado en la posición actual del cursor. Si la posición actual es el final de la línea, los caracteres se incluyen igual que en el modo de inserción.

  En este modo el editor acepta y obedece un número limitado de códigos de control.

Cuando se invoca el editor, el contenido inicial de la línea que se va a editar se muestra en la pantalla (a menos que la línea esté vacía); el cursor se sitúa al principio de la siguiente línea de pantalla. Si la línea no está vacía, se emite un mensaje inductor. Si la línea está vacía, o si sólo consta de un número de línea, el editor entra en el modo de inserción; de lo contrario, entra en el modo de órdenes. El editor puede ser invocado:

- por una orden EDIT o al detectarse un error de sintaxis
  cuando está activa la orden AUTO y se encuentra una línea no vacía
  mediante control-A en modo directo cuando la última línea ha sido una línea de programa

  En todos estos casos se muestra el número de línea como mensaje inductor. El editor entra en el modo de órdenes.
- cuando está activa la orden AUTO y la línea no existe
  mediante control-A cuando la línea anterior constaba solamente de un número de línea

  En todos estos casos se muestra el número de línea como mensaje inductor. El editor entra en el modo de inserción.
- mediante control-A cuando la línea anterior no era vacía y no empezaba con un número de línea.

  En este caso se muestra la cadena "?" como mensaje inductor. El editor entra en el modo de órdenes.

- en modo directo con una línea vacía

  En este caso no se emite mensaje inductor. El editor entra en el modo de inserción.

ADVERTENCIA

Las órdenes de 'cursor a la izquierda' y 'borrar hacia atrás' intentan mantener actualizada la representación de la línea en la pantalla; para ello envían a la consola espacios y retrocesos del cursor. Este mecanismo puede fallar cuando la línea que está siendo editada ocupa varias líneas de pantalla, y también cuando se han visualizado con las órdenes D, H o K los caracteres editados, rodeados de signos de barra a la izquierda (\). En tales casos la consola no muestra el estado actual de la línea y por consiguiente la representación puede inducir a error. Recomendamos al usuario que utilice la orden L para exhibir la línea actualizada, la cual quedará con el cursor al principio o a continuación del número de línea (si la hay).

4.3.1 Órdenes comunes a todos los modos

Los siguientes códigos de control son tratados como órdenes en todos los modos:

Retorno del carro (&H0D)	Terminar la edición. Se muestra el resto de la línea, a partir de la posición actual. Se almacena la línea, o se la obedece.
Escape (&H1B)	Entrar en el modo de órdenes.
Control-C (&H03)	Abandonar la edición. Se muestra el mensaje 'Break'. No se obedece ni se almacena la línea, pero su contenido actual no se pierde (de forma que pueda volver a ser editada si se pulsa control-A).

4.3.2 Órdenes específicas del modo de órdenes

Espacio (&H20)	Avanzar un carácter en la línea actual. Si la posición actual es la última de la línea, no se avanza, sino que se envía un código 'bell' (pitido, &H07) a la consola. De lo contrario, el editor envía a la consola el carácter actual y avanza hasta el carácter siguiente en la línea actual.
DEL (borrar, &H7F) o retroceso (&H08)	Retroceder un carácter en la línea actual. Si la posición actual es la primera de la línea, no se retrocede, sino que se envía un código 'bell' (pitido, &H07) a la consola. De lo contrario, el editor envía a la consola un retroceso (&H08) y retrocede hasta el carácter anterior en la línea actual.

B o b	Borrar hacia atrás. Si la posición actual es la primera de la línea, no se modifica la posición y se envía a la consola un código 'bell' (&H07). De lo contrario, el editor envía un retroceso (&H08), un espacio (&H20) y un retroceso (&H08) a la consola y elimina el carácter anterior en la línea actual.
D o d	Borrar hacia delante. Si la posición actual es la última de la línea, no se modifica la posición y se envía a la consola un código 'bell' (&H07). De lo contrario, el editor envía a la consola el carácter actual entre signos '\' y lo elimina de la línea actual.
H o h	Borrar hasta el final de la línea y entrar en el modo de inserción. El editor muestra entre signos '\' todos los caracteres borrados.
I o i	Entrar en el modo de inserción.
K o k	Borrar hasta el carácter especificado. Esta orden borra todos los caracteres que hay entre la posición actual y la próxima ocurrencia del caracter pulsado después de K o k. Este carácter no se borra. El carácter de la posición actual no se incluye en la búsqueda, por lo que se borra en todos los casos. Los caracteres borrados se muestran entre signos '\'.
L o l	Muestra en la pantalla la línea actual. Reinicia el editor con la línea actual; esta orden se emite cuando se desea listar la línea, cosa que ocurre cada vez que se reinicia el editor.
O u o	Entrar en el modo de sobreescritura.
S o s	Buscar carácter. Avanza hasta la próxima ocurrencia del carácter pulsado después de S o s. El carácter de la posición actual no se incluye en la búsqueda. Todos los caracteres por los que se pasa son enviados a la consola.
X o x	Ampliar la línea actual. Se avanza hasta el final de la línea y se entra en el modo de inserción. Todos los caracteres por los que se pasa son enviados a la consola.

4.3.3 Caracteres válidos para los modos de inserción y sobreescritura

Los siguientes caracteres son aceptados como texto y por lo tanto pueden ser incluidos en la línea, tanto en el modo de inserción como en el de sobreescritura.

- Tab (&H09) — Los tabuladores se representan en forma de espacios hasta la siguiente posición de tabulación.
- Avance de línea (&H0A) — Los avances de línea se representan en forma de un retorno del carro (&H0D) (&H0A) seguido de un avance de línea (&H0A).
- Caracteres imprimibles ASCII, margen de &H20 a &H7E.
- Caracteres adicionales, margen de &H80 a &HFF.

Además de los códigos de control citados en el apartado 4.3.1, los modos de inserción y de sobreescritura obedecen el retroceso (&H08) y el DEL (borrar, &H7F), con los siguientes efectos:

- Modo de inserción — Borrar hacia atrás.

  Si la posición actual es la primera de la línea, no se modifica la posición y se envía a la consola un código 'bell' (&H07). De lo contrario, el editor envía un retroceso (&H08), un espacio (&H20) y un retroceso (&H08) a la consola y elimina el carácter anterior en la línea actual.
- Modo de sobreescritura — Retroceder un carácter en la línea actual.

  Si la posición actual es la primera de la línea, no se modifica la posición y se envía a la consola un código 'bell' (&H07). De lo contrario, el editor envía un retroceso (&H08) a la consola y retrocede hasta el carácter anterior en la línea actual.

Todo intento de introducir un carácter de control (margen de &H00 a &H1F) distinto de los mencionados aquí y en la sección 4.3.1 será rechazado; además, el editor ignora el carácter y envía un código 'bell' (&H07) a la consola.

4.3.4 Comportamiento supuesto para la consola

El editor basado en órdenes envía a la consola un reducido número de códigos de control. La consola debe funcionar de tal manera que los efectos de esos códigos sean los siguientes:

- El retorno del carro seguido del avance de línea (&H0D, &H0A) debe llevar el cursor al principio de la siguiente línea de pantalla.
- El retroceso (&H08) debe hacer retroceder el cursor una posición sin borrar el carácter. Si el cursor ya está en la primera columna de la consola, deberá dejarlo donde está o bien llevarlo a la última columna de la línea de pantalla anterior.
- El código 'bell' (&H07) debe producir algún tipo de ruido de aviso.

Estos son, por otra parte, los significados habituales de estos códigos de control.

Capítulo 5

Resumen de las órdenes y funciones

En este capítulo vamos a hacer una revisión rápida de las órdenes y funciones disponibles en BASIC, agrupándolas en bloques lógicos para así dar una idea de su utilidad y situarlas en su contexto. La descripción detallada se da en el capítulo 9.

5.1 Creación de programas

AUTO, DELETE, EDIT, LIST, LLIST, NEW, RENUM, SAVE.

Las órdenes de este grupo se aplican a la creación de programas nuevos y a la modificación de los existentes. En las versiones de 'sólo ejecución' se pierden todas estas órdenes, salvo SAVE y DELETE.

AUTO ayuda en la introducción de una nueva sección de programa. Genera automáticamente los números de línea a medida que se va introduciendo líneas por el teclado.

DELETE borra grupos de líneas del programa.

EDIT permite corregir líneas concretas del programa actual.

LIST y LLIST generan listados totales o parciales del programa a través de la consola o la impresora.

NEW borra la zona de memoria donde está almacenado el programa actual y la deja preparada para la introducción de un nuevo programa.

RENUM renumera las líneas del programa, total o parcialmente. Las referencias a números de línea incluidas dentro de las instrucciones se renumeran también correctamente.

SAVE graba en un fichero el programa actual. La grabación se puede hacer en cualquiera de las siguientes modalidades:

- ASCII — fichero de texto, con la misma forma que produce LIST.
- Comprimido — de la forma en que el ordenador guarda internamente el programa; ocupa menos espacio en el disco y se carga más deprisa en la memoria.
- Protegido — similar a 'comprimido', pero deja el programa indescifrable. Cuando se carga un programa protegido, dejan de funcionar todas las órdenes que normalmente permiten al usuario examinar el programa. Una vez protegido un programa, ya no es posible desprotegerlo.

5.2 Carga y ejecución

CLEAR, CHAIN, CHAIN MERGE, COMMON, COMMON RESET, HIMEM, LOAD, MEMORY, MERGE, RUN

CLEAR borra todas las variables, cierra todos los ficheros y, más en general, reinicializa BASIC, pero conserva el programa actual. Con esta orden se puede también establecer el tamaño de la pila, el número máximo de ficheros simultáneos y el tamaño máximo de los registros de acceso aleatorio.

MEMORY permite modificar el espacio de memoria disponible para BASIC, el tamaño de la pila, el número máximo de ficheros simultáneos y el tamaño máximo de los registros de acceso aleatorio.

HIMEM es una función que da la dirección del byte más alto de la memoria utilizable por BASIC.

CHAIN, CHAIN MERGE, COMMON y COMMON RESET. CHAIN y CHAIN MERGE permiten que un programa cargue e inicie la ejecución de otro. Con CHAIN MERGE se conserva el programa actual, completo o en parte. Los valores de las variables se conservan para que puedan ser utilizados por el nuevo programa (se conservan todos o solamente los especificados en sentencias COMMON). En cualquier momento se puede borrar con COMMON RESET todas las variables que no hayan sido especificadas en sentencias COMMON.

MERGE mezcla con el programa actual un programa grabado en el disco. Esta orden se utiliza normalmente en el desarrollo de programas; no está incluida en las versiones de 'sólo ejecución'.

LOAD borra el programa actual y los valores de las variables, reinicializa todos los indicadores de estado y carga un nuevo programa desde el disco. Opcionalmente puede poner en marcha el nuevo programa.

RUN. En su forma más sencilla, esta orden inicia la ejecución del programa actual. Las demás formas inician la ejecución a partir de una línea especificada o funcionan de forma similar a LOAD.

5.3 Conclusión del programa

END, SYSTEM

END. Los programas terminan (y BASIC vuelve al modo directo) cuando se ha ejecutado la última línea del programa. La orden END se puede utilizar para terminar el programa en otro lugar (o lugares).

SYSTEM hace que BASIC abandone el programa, cierre todos los ficheros y vuelva al nivel de órdenes del sistema operativo.

5.4 Varios

OPTION RUN, OPTION STOP, VERSION.

OPTION RUN inhibe las acciones de control-C y control-S, que normalmente (salvo en las versiones de 'sólo ejecución') interrumpen o suspenden el programa actual.

OPTION STOP rehabilita esas acciones.

VERSION es una función que genera información sobre la versión de BASIC que se está utilizando, sobre el sistema operativo, etc.

5.5 Estructuras de control

FOR, GOSUB, GOTO, IF, ON x GOSUB, ON x GOTO, RETURN, WHILE.

FOR, WHILE. BASIC proporciona dos tipos de bucle: FOR y WHILE. El bucle FOR tiene asociada una variable de control que recorre un margen de valores (avanzando un 'paso' por cada ejecución del bucle) hasta que se alcanza el valor final especificado. La magnitud del 'paso' puede ser especificada; incluso puede tener un valor negativo. El bucle FOR no se ejecuta ninguna vez si el valor inicial de la variable de control es mayor que el valor final especificado. La palabra clave NEXT señala el final del bucle FOR. El bucle WHILE se repite hasta que la condición especificada sea 'falsa'. El bucle no se ejecuta ninguna vez si la condición es falsa en el primer intento. Ambos bucles pueden ser anidados en cualquier combinación.

IF permite elegir entre acciones alternativas u opcionales. La palabra clave IF va seguida de una expresión. Si la expresión da un valor distinto de cero, se emprenden las acciones especificadas tras la palabra clave THEN. Si el valor de la expresión es cero, se elige lo especifiacado detrás de ELSE. Los bloques especificados tras THEN y ELSE pueden constar de varias órdenes, pero la instrucción IF ... THEN ... ELSE ... tiene que estar completa en una sola línea de programa. Las instrucciones IF pueden ser anidadas en una misma línea.

GOSUB, RETURN son órdenes con las que se gestionan las subrutinas de BASIC. GOSUB invoca la rutina que empieza en la línea especificada. RETURN señala el final de la subrutina y devuelve el control a la orden inmediatamente posterior al correspondiente GOSUB. Todas las variables de BASIC son globales. Las subrutinas pueden ser recurrentes.

GOTO desvía incondicionalmente el programa a la línea especificada.

ON x GOSUB, ON x GOTO evalúan la expresión **x** y eligen, en función del resultado, una línea de las mencionadas en una lista para invocar una subrutina o saltar.

5.6 Variables

DEFINT, DEFSNG, DEFDBL, DEFSTR, DIM, ERASE, OPTION BASE.

DEFINT, DEFSNG, DEFDBL, DEFSTR. Cuando un nombre de variable no incluye ningún indicador de tipo, BASIC supone que es del tipo 'implícito'. Pues bien, el tipo implícito depende del primer carácter del nombre (que tiene que ser una letra). DEFINT, DEFSNG, DEFDBL y DEFSTR especifican qué tipo se debe considerar como implícito para cada letra. Estas especificaciones se pueden modificar sin ninguna restricción, pero el potencial para las confusiones es enorme.

DIM, OPTION BASE. Si no se especifica otra cosa explícitamente en una orden DIM, BASIC supone, cuando se utiliza por primera vez una matriz (declaración implícita), que el límite superior del índice es 10. El valor mínimo del índice se establece con OPTION BASE; puede ser 0 o 1. Se produce un error si se intenta modificar este valor después de haber sido establecido, y cuando se intenta establecerlo cuando ya hay matrices declaradas (implícita o explícitamente.)

ERASE. Las matrices ocupan grandes zonas de la memoria. La orden ERASE recupera ese espacio cuando ya no se necesitan las matrices.

5.7 Entrada y salida por la consola

INKEY$, INPUT, INPUT$, LINE INPUT, POS, PRINT, WIDTH, WRITE, ZONE, OPTION PRINT, OPTION INPUT, OPTION NO TAB, OPTION TAB

INKEY$. Esta función lee un carácter en el teclado, si se ha pulsado alguna tecla. El carácter no aparece en la pantalla. INKEY$ se puede utilizar en rutinas que capten respuestas de una sola letra.

INPUT. lee una línea introducida por la consola. Puede interpretar partes de la línea como información numérica y otras partes como información literal, dependiendo de los parámetros que acompañen a la palabra clave INPUT.

INPUT$. es similar a INKEY$, pero lee un número especificado de caracteres, no necesariamente sólo uno. INPUT$ espera hasta que se hayan introducido todos los caracteres.

LINE INPUT. lee una línea introducida por la consola, sin interpretarla, y la asigna a una variable literal.

WIDTH. Toda la salida hacia la consola interacciona con la anchura de la pantalla, modificable con la orden WIDTH (la anchura implícita es 80). BASIC empieza una línea nueva cada vez que es necesario para mantener la congruencia entre la anchura de la consola y la longitud de las líneas. La posición actual en horizontal en la consola se averigua con la función POS. Si la propia consola inicia líneas nuevas cuando se alcanza cierta columna, se puede utilizar WIDTH para informar a BASIC de este hecho, de modo que no inicie líneas indebidamente.

PRINT envía números y cadenas de texto a la consola. Los números se emiten en «formato libre». PRINT divide la consola en «zonas de escritura», con las que establece cierta organización en la salida. La orden ZONE sirve para modificar la anchura de las zonas de escritura. La palabra clave USING define plantillas de formato, con las que se controla la salida de números y cadenas sin más referencia a las zonas de escritura. En el capítulo 6 se estudia más a fondo la orden PRINT y otras asociadas con ella.

WRITE envía números y cadenas de texto a la consola. Inserta comas entre cada dos elementos y encierra las cadenas entre comillas. Los números se emiten en «formato libre».

OPTION PRINT permite al usuario especificar la dirección de una subrutina en código de máquina que BASIC deba utilizar para enviar caracteres a la consola (en lugar de llamar al sistema operativo). OPTION INPUT permite al usuario especificar la dirección de las subrutinas en código de máquina que BASIC deba utilizar para leer las entradas por el teclado.

OPTION TAB, OPTION NO TAB permite al usuario que active o inhiba la conversión que BASIC hace de los caracteres 'tab' (&H09) a espacios cuando los envía a la consola o a la impresora.

5.8 Salida hacia la impresora

LPOS, LPRINT, WIDTH LPRINT, OPTION PRINT

Estas órdenes son equivalentes a las análogas para la consola.

5.9 Ficheros

BASIC puede leer y escribir ficheros en disco, en acceso secuencial o aleatorio. Para que un fichero sea accesible, antes tiene que haber sido abierto. En el momento de abrir el fichero se especifica el modo de acceso y se lo asocia con un *número-de-fichero*. En lo sucesivo todas las referencias al fichero se hacen a través de ese número. Un *número-de-fichero* determinado sólo puede representar un fichero, pero puede ser asociado a otro fichero cuando se haya cerrado el primero.

Si no se especifica otra cosa cuando se carga BASIC (véase el apéndice I), o bien mediante las órdenes CLEAR o MEMORY, los números de fichero tienen que estar en el margen de 1 a 3. Es decir, BASIC puede tener abiertos simultáneamente tres ficheros como máximo.

En el capítulo 7 daremos una descripción completa de la gestión de ficheros realizable desde BASIC.

5.9.1 Acceso al directorio

DEL, DIR, ERA, FILES, KILL, NAME, REN, FIND$, OPTION FILES

Las órdenes DIR y FILES listan por la consola el directorio del disco.

Las órdenes DEL, ERA y KILL borran ficheros del disco.

Las órdenes NAME y REN cambian el nombre de los ficheros.

La función FIND$ sirve para averiguar el nombre completo de un fichero dado, incluidos los atributos del fichero. En el nombre de fichero suministrado a FIND$ se puede incluir «símbolos comodín»; además, se puede especificar que FIND$ busque el n-ésimo fichero que concuerda con el nombre especificado.

OPTION FILES permite establecer la unidad de disco y el número de usuario implícitos.

5.9.2 Inspección de ficheros

DISPLAY, TYPE

Estas órdenes escriben en la consola el contenido del fichero especificado.

5.9.3 Órdenes de aplicación general

CLOSE, OPEN, RESET

OPEN. Esta orden especifica el nombre del fichero, el modo de acceso y el número de fichero por el que se lo va a identificar en el futuro. El nombre del fichero se da en forma de cadena literal; su formato tiene que ser compatible con los requisitos impuestos a este respecto por el sistema operativo.

CLOSE realiza la operación inversa de OPEN. Si el fichero está abierto en modo de salida secuencial, esta orden escribe en el disco los datos que pudiera haber todavía en el tampón. Cierra el fichero y libera el número de fichero.

RESET. Esta orden cierra todos los ficheros y reinicializa el sistema de ficheros. Suele ser necesario ejecutar esta orden antes de cambiar los discos.

5.9.4 Acceso secuencial en dirección de entrada

EOF, INPUT, LINE INPUT

INPUT y LINE INPUT. El acceso secuencial en dirección de entrada permite la lectura de ficheros de texto ordinarios. Las órdenes INPUT y LINE INPUT llevan en este caso *#número-de-fichero* y funcionan de la misma forma que las análogas para la entrada por consola, con la diferencia de que leen las líneas en el fichero.

EOF. Si se intenta leer más allá del final de un fichero se produce un error. La función EOF detecta el final del fichero.

5.9.5 Acceso secuencial en dirección de salida

PRINT, WRITE

PRINT. El acceso secuencial en dirección de salida permite escribir ficheros de texto ordinarios. La orden PRINT lleva en este caso *#número-de-fichero* como primer argumento y funciona igual que la análoga para la salida por consola, con la diferencia de que escribe en el fichero. Esta orden supone que las líneas son de longitud infinita.

WRITE. Esta orden lleva, en esta aplicación, *#número-de-fichero* como primer argumento; es particularmente útil en la preparación de ficheros que más tarde vayan a ser leídos con INPUT, ya que INPUT espera que los elementos que lee estén separados por comas y además las comillas que rodean las cadenas literales evitan toda ambigüedad.

5.9.6 Acceso aleatorio

CVD, CVI, CVS, CVUK, CVIK, FIELD, GET, LSET, MID$, MKD$, MKI$, MKS$, MKUK$, MKIK$, PUT, RSET

GET y PUT. BASIC mantine un tampón para cada fichero que está abierto en modo de acceso aleatorio. GET lee un registro del fichero y lo transfiere al tampón. PUT lee el contenido actual del tampón y lo escribe en el fichero.

FIELD. La orden FIELD define un formato al que se han de ajustar los datos del tampón de registros. Para un mismo fichero se puede definir varios de estos formatos, los cuales permiten entonces dar al fichero diversas organizaciones.

CVD, CVI, CVS, CVUK, CVIK, MKD$, MKI$, MKS$, MKUK$, MKIK$. Estas órdenes realizan un mecanismo de conversión mediante el cual la información numérica es convertida en cadenas literales adecuadas para su grabación en los ficheros. MKD$, MKS$ y MKI$ producen cadenas a partir de números de doble precisión, precisión sencilla y enteros, respectivamente. CVD, CVS y CVI realizan la conversión inversa, de cadenas a números. CVUK y CVIK son similares a CVI, y MKUK$ y MKIK$ lo son a MKI$, pero actúan sobre cadenas adecuadas para su utilización como claves en Jetsam.

5.10 Datos constantes

DATA, READ, RESTORE

Todas las sentencias DATA del programa, consideradas en orden ascendente de números de línea, forman una especie de fichero. Este «fichero» puede ser leído mediante órdenes READ. La posición dentro del «fichero» se puede reajustar con órdenes RESTORE.

5.11 Funciones aritméticas

ABS, ATN, COS, EXP, FIX, INT, LOG, LOG10, MAX, MIN, RANDOMIZE, RND, ROUND, SGN, SIN, SQR, TAN

ATN, COS, SIN, TAN son las funciones trigonométricas habituales (ATN es el arco tangente); los ángulos se entienden en radianes.

EXP da el valor del número *e* elevado a la potencia dada. LOG da el logaritmo natural (o neperiano, de base *e*) de su argumento. LOG10 da el logaritmo decimal. SQR da la raíz cuadrada de su argumento.

RND da el siguiente número de una sucesión pseudoaleatoria. RANDOMIZE establece una nueva «semilla», es decir, un nuevo punto de arranque para la sucesión de números pseudoaleatorios.

MIN, MAX. MIN da el valor mínimo de entre los valores que se le suministran como argumentos. MAX da el valor máximo.

FIX, INT generan valores enteros a partir de sus argumentos. FIX redondea hacia el entero más próximo a cero. INT redondea hacia $-\infty$. ROUND redondea su argumento con un número de cifras decimales especificado; utiliza el método de «redondeo hacia el más próximo». (Véase el apartado 2.9.1.)

ABS da el valor absoluto de su argumento. SGN da el signo de su argumento.

5.12 Funciones literales

OCT$, DEC$, HEX$, INSTR, LEFT$, LEN, LOWER$, MID$, RIGHT$, SPACE$, STR$, STRING$, STRIP$, UPPER$, VAL

**LEFT$, RIGHT$, MID$** son funciones que entregan una parte de las cadenas que se les suministran como argumentos. LEFT$ da el número de caracteres especificado, tomado del principio de la cadena. RIGHT$ da el número de caracteres especificado, tomado del final

de la cadena. MID$ da una porción de la cadena, tomada a partir de una posición especificada.

MID$ permite sustituir una parte especificada de una cadena por otra cadena.

INSTR busca la primera ocurrencia de una cadena dentro de otra.

LEN da la longitud de la cadena que se le suministra como argumento.

LOWER$, UPPER$, STRIP$. LOWER$ genera una copia de su argumento en la que todas las letras mayúsculas han sido convertidas a minúsculas. UPPER$, análogamente, genera una copia de su argumento en la que todas las letras minúsculas han sido convertidas a mayúsculas. STRIP$ genera una copia de su argumento después de poner a cero el bit 7 (el más significativo) de cada carácter.

STRING$** da una cadena que consiste en el carácter dado repetido un número dado de veces. **SPACE$ da una cadena que consiste en el número especificado de espacios.

OCT$, DEC$, HEX$, STR$ producen representaciones literales de sus argumentos numéricos. OCT$ genera una cadena de dígitos octales; HEX$, una cadena de dígitos hexadecimales. STR$ da una cadena de dígitos decimales de formato libre. DEC$ produce una cadena de dígitos decimales de formato especificado (mediante plantillas de formato similares a las de PRINT USING). VAL toma una cadena, la interpreta como número (de forma análoga a como lo hace INPUT) y da su valor numérico.

5.13 Funciones de conversión entre tipos

ASC, CHR$, CDBL, CINT, CSNG, UNT

ASC toma el primer carácter de una cadena literal y da su valor numérico entero.

CHR$ toma un valor entero y lo convierte en una cadena literal que consiste en el carácter correspondiente.

CINT toma un valor numérico y lo redondea a un entero por el método de «redondeo hacia el más próximo». El resultado tiene que estar en el margen de −32768 a 32767.

CSNG toma un valor numérico y lo convierte a precisión sencilla. **CDBL** toma un valor numérico y lo convierte a doble precisión.

UNT toma un valor numérico y lo convierte en entero sin signo. (Véase la sección 2.10.)

5.14 Operaciones a nivel de máquina

CALL, DEF USR, DEF SEG, INP, INPW, OUT, OUTW, PEEK, POKE, USR, WAIT, WAITW, OPTION INPUT, OPTION LPRINT, OPTION PRINT

CALL, DEF USR, USR. BASIC puede utilizar dos tipos de rutina externa. CALL invoca la rutina que empieza en la dirección especificada y puede entregarle varios parámetros. USR invoca una función programada en una rutina de código de máquina cuya dirección inicial ha sido especificada previamente en una orden DEF USR. USR entrega un parámetro a la rutina y recibe de ella un valor, que es el valor generado por la función. Véase en el apéndice III una descripción más detallada.

PEEK, POKE leen y escriben bytes en la memoria de la máquina.

DEF SEG se utiliza en Mallard-86 para especificar la componente de segmento de las direcciones usadas en CALL, USR PEEK y POKE. Esta orden se ignora en Mallard-80.

INP, INPW, OUT, OUTW, WAIT, WAITH. INP, OUT y WAIT dan acceso al espacio de entrada/salida de la máquina. INPW, OUTW y WAITW dan acceso al espacio de entrada/salida de la máquina, organizado en forma de palabras.

OPTION INPUT, OPTION LPRINT, OPTION PRINT permiten que el usuario especifique qué rutinas de código de máquina debe invocar BASIC, en sustitución de las del sistema operativo, para gestionar las entradas por el teclado y las salidas por la consola y la impresora.

5.15 Gestión de errores

ERL, ERR, ERROR, ON ERROR GOTO, OSERR, RESUME

ERL, ERR, ERROR, ON ERROR GOTO, RESUME. Cuando BASIC detecta un error, emite un mensaje adecuado y vuelve al modo directo, a menos que se le haya indicado que debe hacer otra cosa con la orden ON ERROR GOTO. Esta orden especifica el número de línea donde empieza la subrutina de gestión de errores que se ha incluido en el programa. Cuando se produce un error, ERL toma un valor que es el número de la línea donde se ha detectado el error; ERR da el número del error. La rutina de gestión de errores puede examinar estos valores y decidir si puede subsanar el error o no. La orden RESUME puede servir para reanudar la ejecución.

OSERR. Algunos sistemas operativos pueden informar a BASIC de diversos errores. BASIC, en lugar de manejar una tabla de errores distinta para cada sistema operativo, genera el error n.º 21 y proporciona la función OSERR para examinar los errores. Cuando se ha producido un error n.º 21, la función OSERR da un número que depende del sistema operativo y que identifica el error.

5.16 Desarrollo de programas

CONT, STOP, TRON, TROLL

De todas estas órdenes, solamente STOP está incluida en las versiones de 'sólo ejecución', en las que ejerce el mismo efecto que END.

STOP, CONT. La orden STOP detiene el programa de forma tal que pueda ser reanudado con CONT (a condición de que no se haya modificado el programa durante la interrupción). Las órdenes STOP se insertan en el programa en lugares clave donde interese poder observar el estado del programa y los valores de las variables.

TRON, TROFF activan y desactivan el mecanismo de seguimiento del programa. Cuando este mecanismo está activado, BASIC escribe en la pantalla, entre corchetes, el número de la línea que va a ejecutar. Estas órdenes proporcionan un sencillo método de rastreo de los programas.

Capítulo 6

Introducción a las instrucciones y formatos de escritura

En este capítulo vamos a describir todos los recursos que BASIC proporciona para el control de la salida de números y cadenas literales por la pantalla y la impresora. Estos recursos se basan en las diversas órdenes de escritura: PRINT, PRINT # y LPRINT.

Si no se las complementa con la cláusula USING, todas estas órdenes escriben los números y las cadenas en 'formato libre', el cual consiste en que BASIC:

- escribe todos los números con el mínimo número de caracteres posible.
- escribe las cadenas literalmente, pero expande el código 'tab' en la consola y en la impresora (no en la salida hacia los ficheros).
- organiza la salida en «zonas de escritura» cuando así se le pide.
- genera una línea nueva cuando encuentra que un elemento no cabe en la actual.

Combinando USING con estas órdenes se puede escribir los números y las cadenas encajándolos en unas plantillas de formato diseñadas por el usuario, de modo que:

- se puede escribir los números con un número de cifras especificado.
- se puede escribir los números con comas que separen los millares, millones, etc.
- se puede escribir los números con un formato «científico» especificado.
- se puede escribir parte de las cadenas, o escribirlas completas; el código 'tab' se expande en la consola y la impresora, pero no en la salida hacia los ficheros.
- se ignoran las zonas de escritura.
- se genera una línea nueva sólo cuando la actual está completamente llena.

Las tres formas de escritura, en la consola (PRINT), en la impresora (LPRINT) y en los ficheros (PRINT #) funcionan todas igual. A fin de gestionar las zonas de escritura, la expansión de 'tab' y la función TAB, BASIC mantiene actualizada una *posición lógica*, para lo cual lleva la cuenta de los caracteres que ha emitido (véase la sección 6.3).

La consola y la impresora tienen anchuras asociadas. BASIC genera una línea nueva cada vez que la posición lógica sobrepasa la anchura del dispositivo. En el modo de formato libre, BASIC genera una línea nueva antes de escribir un elemento siempre que éste no quepa entre la posición lógica y la anchura del dispositivo. La anchura de los ficheros se supone infinita. Las anchuras de la consola y de la impresora pueden ser establecidas por el usuario; se les puede dar anchura infinita.

6.1 Estructura de una sentencia de escritura

Las órdenes de escritura pueden ir seguidas de una serie de «elementos imprimibles», separados por comas o por signos de punto y coma. Cada elemento puede ser una expresión literal o numérica, o bien una función TAB o SPC. BASIC evalúa las expresiones y las escribe en «formato libre». Si hay una coma después de la expresión, BASIC avanza hasta el principio de la siguiente «zona de escritura» cuando termina de escribirla. El signo de punto y coma sirve para separar los elementos en la orden, pero no ejerce ningún efecto sobre la salida.

Hay dos funciones especiales, TAB y SPC, que sólo funcionan dentro de las instrucciones de escritura. TAB escribe espacios hasta llegar a la posición de escritura especificada. SPC escribe el número de espacios especificado. BASIC trata estas dos funciones como si fuesen seguidas de un signo de punto y coma.

Los formatos de escritura se controlan con la palabra clave USING, seguida de una cadena literal que especifica la plantilla de formato. Después de la plantilla se ponen las expresiones, literales o numéricas, cuyos valores se desee escribir. Las expresiones se pueden separar con comas o signos de punto y coma, pero ninguno de estos signos afecta al formato de la salida.

Cuando BASIC termina de escribir todos los argumentos de una orden, salta a la línea siguiente, a menos que la orden haya terminado con TAB, SPC, coma o punto y coma.

6.2 Gestión de los caracteres de control

Cuando BASIC recibe un código 'tab' (&H09) para escribirlo en la consola o en la impresora, normalmente lo transforma en cierto número de espacios, los suficientes para avanzar hasta el siguiente tope de tabulación. (Los topes de tabulación están distribuidos inicialmente de forma tal que hay uno cada ocho columnas.) La expansión de 'tab' es independiente de los caracteres anteriores, de modo que si el valor &H09 pertenece a una secuencia de escape, BASIC no se da cuenta y lo transforma. Las órdenes OPTION NO TAB y OPTION TAB inhiben y habilitan, respectivamente, la expansión de 'tab'.

Todos los demás caracteres de control son enviados al dispositivo de salida lo más directamente posible, evitando la interferencia con el sistema operativo siempre que éste lo permite.

6.3 Posición lógica

La posición lógica se cuenta a partir de la columna n.º 1. Cada vez que escribe un carácter, BASIC la incrementa en una unidad, a menos que el carácter sea alguno de los siguientes:

- El retroceso (&H08), que cuenta como −1.
- 'Tab' (&H09); si está permitida su expansión, cada espacio cuenta como 1.
- El retorno del carro (&H0D), que vuelve a poner a 1 la posición lógica.
- Los restantes caracteres de control del margen de &H00 a &H1F, que cuentan como 0.

La posición lógica puede no coincidir con la posición física cuando se han emitido secuencias de control.

6.4 Escritura en formato libre

Los resultados de las expresiones numéricas se escriben según se explica más abajo. Los de las expresiones literales se escriben carácter a carácter, después de que BASIC compruebe que caben en el resto de la línea. Los efectos de SPC y TAB se describen más abajo.

Si hay una coma a la derecha de un elemento, BASIC avanza hasta el principio de la siguiente zona de escritura antes de procesar la siguiente expresión. El signo de punto sirve para separar los elementos, pero no afecta a la salida.

6.4.1 Zonas de escritura y anchura de los dispositivos

BASIC divide la línea de salida en zonas. La anchura de las zonas se puede establecer mediante la orden WIDTH; el valor implícito es 15 caracteres. Cuando una orden impone el avance hasta la siguiente zona de escritura, BASIC salta a la línea siguiente si el espacio que queda hasta el final de la línea es menor que la anchura de la zona de escritura.

Antes de escribir un elemento, BASIC comprueba si entre la posición actual y el final de la línea del dispositivo (establecido con WIDTH) hay espacio suficiente para aquél, a menos que la posición actual sea el principio de una línea. Si no queda espacio suficiente, pasa a la línea siguiente antes de escribir el elemento. Para realizar esta comprobación en el caso de las cadenas literales, BASIC utiliza la longitud de la cadena. Nótese que es muy probable que la longitud de la cadena no coincida con el espacio físico necesario para escribirla en el caso de que la cadena contenga caracteres de control.

6.4.2 Escritura de números en formato libre

Los números positivos se escriben con un espacio delante; los negativos, con un signo menos. Todos los números van seguidos de un espacio.

Todos los números se escriben con el menor número de cifras posible. No se escribe el punto decimal cuando no hay cifras decimales significativas. Antes del punto decimal se escribe como mínimo un dígito, de modo que los números menores que 1 llevan un cero a la izquierda del punto.

Los números de precisión sencilla se escriben en notación científica cuando no es posible representarlos exactamente con siete cifras o menos (sin contar el posible cero por la izquierda).

Los números de doble precisión se escriben en notación científica cuando no es posible representarlos exactamente con dieciséis cifras o menos (sin contar el posible cero por la izquierda).

6.4.3 Función SPC

SPC escribe el número especificado de espacios, siempre a partir de la posición actual. Esto no es lo mismo que escribir SPACE$(*n*), ya que BASIC comprueba si la cadena SPACE$(*n*) cabe en el resto de la línea y, de no ser así, realiza un retorno del carro y avance de línea antes de escribir los *n* espacios.

SPC no necesita ir seguida de ningún separador (coma o punto y coma); BASIC siempre da por supuesto el punto y coma (incluso cuando SPC es el último elemento de la orden).

6.4.4 Función TAB

TAB escribe los espacios necesarios para llegar hasta la posición especificada. Si la posición especificada coincide con la actual o es posterior a ella, escribe los espacios requeridos (en el caso de que ambas posiciones sean la misma, ninguno). Si la posición especificada es anterior a la actual, BASIC salta a la línea siguiente antes de escribir los espacios necesarios.

TAB no necesita ir seguida de ningún separador (coma o punto y coma); BASIC siempre da por supuesto el punto y coma (incluso cuando TAB es el último elemento de la orden).

6.5 Control del formato con USING

El control de los formatos se realiza incluyendo en la orden de escritura la palabra clave USING seguida de una cadena literal adecuada, que es la que especifica la plantilla de formato. Después de ésta se ponen las expresiones literales o numéricas, separadas por comas o signos de punto y coma. Las expresiones se evalúan sucesivamente; para cada una se examina la plantilla, con objeto de buscar la especificación de formato correspondiente. Si se agota la plantilla cuando se está buscando la especificación de formato, se reinicia el proceso a partir del principio de la plantilla. Si no se encuentra un formato adecuado, se genera el error n.° 5.

Nótese que, a diferencia de lo que ocurre con el formato libre, al escribir con formato especificado por USING se ignoran las zonas de escritura y no se comprueba el espacio restante en la línea actual.

6.5.1 Plantillas de formato

Una plantilla de formato es una cadena literal que BASIC interpreta carácter a carácter para controlar la forma en que debe escribir el resultado de las expresiones. A continuación describimos el significado de los caracteres que forman la plantilla. Los caracteres siguientes son los reconocibles como especificadores de formato:

```
!  \  &  #  .  +  -  $  ↑  ,
```

Si BASIC encuentra algún otro carácter cuando está examinando la plantilla en busca de una especificación de formato, lo escribe y sigue buscando. Lo mismo ocurre cuando encuentra alguno de estos caracteres fuera de contexto; en este caso, el único carácter que no se escribe es '–', porque BASIC interpreta que se lo ha incluido para especificar que el siguiente carácter debe ser escrito literalmente.

6.5.2 Especificadores de formato adecuados para cadenas literales

!	Se escribe sólo el primer carácter de la cadena.
espacios\	Se escriben los *n* primeros caracteres de la cadena, siendo *n* la longitud de la cadena *espacios*\.
&	Se escribe la cadena completa sin modificarla.

6.5.3 Especificadores de formato adecuados para números

Los números se pueden escribir en notación ordinaria o en notación científica (mantisa y exponente). La especificación de formato de un número no puede exceder de 24 caracteres, sin contar las opciones de signo por la derecha y de signo para el exponente.

Los números se redondean al número de dígitos especificado.

Para el cuerpo del número

#	Cada # representa una posición (para un dígito). Es necesario incluir al menos un #.
.	Especifica la posición del punto decimal. No puede haber más de un punto en la especificación de formato para cada número.
,	Sólo puede aparecer antes del punto decimal. Representa una posición y especifica que los dígitos que queden a la izquierda del punto decimal se separen mediante comas en grupos de tres.

Opciones con signo de dólar y asterisco. Las siguientes opciones son mutuamente excluyentes y han de figurar inmediatamente a la izquierda de la especificación para el cuerpo del número:

$$	Representa dos posiciones (para dos dígitos). Especifica que se debe escribir un signo $ inmediatamente antes del primer dígito o del punto decimal, después del signo + o – (si lo hay). Nótese que el $ ocupará una de las dos posiciones para dígitos.
**	Representa dos posiciones. Especifica que los espacios libres a la izquierda deben ser rellenados con asteriscos.
**$	Representa tres posiciones. Actúa como las dos opciones anteriores combinadas.

Opciones de signo. En ausencia de otra especificación, se escribe el signo – delante del número (y del posible signo $) cuando el número es negativo, y no se escribe signo cuando el número es positivo. El – ocupa una de las posiciones reservadas para los dígitos a la izquierda del punto decimal.

Se puede especificar que se escriba el signo + con los números positivos y que el signo se escriba después del número.

+	Especifica que siempre se debe escribir el signo + o el – (el que corresponda). Si se incluye el + al principio de la plantilla, el signo se escribirá inmediatamente antes del número (y del signo $, si se lo ha especificado). Si el + figura al final de la plantilla, el signo se escribe a la derecha del número (y del exponente, si lo hay).
–	Este signo sólo puede figurar al final de la plantilla. Especifica que, si el número es negativo, se debe escribir el signo – a su derecha (y a la derecha del exponente, si lo hay). Si el número es positivo, en lugar del – se escribe un espacio.

Opción del exponente

↑↑↑↑	Especifica que el número se debe escribir en forma de mantisa y exponente. Los signos ↑↑↑↑ se ponen a la derecha de los especificadores del cuerpo del número, pero a la izquierda de los signos + y – (si los hay).

El cuerpo del número se escribe entonces con el máximo número posible de cifras antes del punto decimal, reservando una posición para el signo (aunque el número sea positivo) si no se ha especificado otra cosa a este respecto.

Desbordamiento. Cuando un número no puede ser escrito en el formato especificado, BASIC se aproxima todo lo que puede al formato en cuestión y escribe el signo % antes del número para indicar el desbordamiento.

En notación ordinaria no es posible escribir números que en valor absoluto sean mayores que 10↑24. Si se intenta hacerlo, BASIC escribe el número en formato libre, precedido del signo % para indicar el «fallo del formato».

Capítulo 7

Introducción a la gestión de ficheros

Una faceta fundamental de BASIC es su capacidad de gestión de ficheros. Las posibilidades de BASIC en este área son, en resumen, las siguientes:

- Operaciones con el disco y el directorio.

  BASIC puede actuar sobre los ficheros o sobre sus nombres.

- Gestión de ficheros secuenciales.

  Los ficheros son tratados como una sucesión de caracteres que sólo se pueden leer o escribir en un orden determinado. Su manejo es similar al de las entradas y salidas por la consola.

  Los ficheros así manejados son ficheros de texto ASCII ordinarios.

- Gestión de ficheros de acceso aleatorio.

  Los ficheros son considerados como una colección de registros de tamaño fijo, identificados por su número de registros y accesibles en cualquier orden. Cada registro puede estar subdividido en campos o consistir sencillamente en texto ASCII.

  Salvo que se tomen medidas al efecto, los ficheros aleatorios producidos por BASIC no pueden ser procesados por otros programas.

- Gestión de ficheros de acceso por claves (Jetsam).

  BASIC está dotado de Jetsam, que es un sistema de manejo de ficheros en la modalidad de acceso por claves. Los ficheros de acceso por claves son similares a los aleatorios en el sentido de que están divididos en registros de longitud fija y pueden ser leídos y escritos en cualquier orden. El acceso se realiza a través de un índice, en el cual se guardan ordenadas las claves asignadas por el usuario a cada registro.

  Los ficheros de acceso por claves sólo pueden ser procesados por BASIC.

- Gestión de ficheros en modalidad «multiusuario» (Jetsam).

  Las versiones de BASIC para ciertos sistemas operativos pueden incluir un sistema de bloqueo de ficheros y de registros para su utilización en sistemas multiusuario. Este sistema complementa todos los métodos de gestión de ficheros, no sólo los de acceso por claves.

En este capítulo describimos todos los sistemas de gestión de fichero menos los de acceso por claves, a los que está dedicado el capítulo 8. El acceso por claves es una ampliación del acceso aleatorio, así que la información suministrada en este capítulo es básica para la comprensión del siguiente.

7.1 Nombres de ficheros

Cada vez que BASIC procesa el nombre de un fichero, pone a cero el bit 7 de todos los caracteres y convierte todas las letras en mayúsculas (compárense las funciones UPPER$ y STRIP$). Los nombres de los ficheros sólo pueden contener caracteres ASCII del margen de &H21 a &H7E, a excepción de los siguientes:

```
[ ] < > / \ = ;
```

El punto se puede utilizar para separar el nombre del distintivo de tipo. El signo de dos puntos se puede utilizar como sufijo del nombre de la unidad de disco y, en las versiones para CP/M, como sufijo del número de usuario. Si hay espacios antes o después del nombre, o a los lados de ":" o de ".", BASIC los suprime. En algunas órdenes son aceptables los símbolos comodín "?" y "*", con el significado habitual.

7.2 Operaciones con el disco y el directorio

Estas operaciones actúan sobre los ficheros propiamente dichos o solamente sobre sus nombres. Algunas órdenes están diseñadas para su uso en modo directo y tienen una sintaxis análoga a la de sus equivalentes en el sistema operativo.

La orden RESET tiene un efecto doble: cierra todos los ficheros y ordena al sistema operativo la reinicialización del sistema de disco. En algunos sistemas es necesario ejecutar una orden RESET antes de cambiar un disco, en particular cuando se va a escribir en el disco nuevo.

Las órdenes DIR y FILES emiten por la consola un listado del directorio del disco. Para listar solamente parte del directorio, se puede incluir en la orden un especificador de ficheros, que puede contener los «símbolos comodín» habituales. Estas dos órdenes pueden ser abandonadas pulsando control-C. Pulsando control-S se suspende la emisión del listado, la cual se reanuda pulsando control-S por segunda vez (excepto en las versiones de 'sólo ejecución', en las que control-C y control-S no ejercen efecto alguno).

La función FIND$ da el nombre completo del primer (o del n-ésimo) fichero que concuerda con el nombre dado. Si no existe tal fichero, FIND$ genera la cadena vacía. Mediante los símbolos comodín se puede hacer que FIND$ lea todo el directorio, o parte de él, para que el programa lo procese según convenga. Especificando un nombre no ambiguo (sin símbolos comodín) se puede averiguar si existe un fichero determinado. El nombre generado es el nombre completo proporcionado por el sistema operativo; incluso puede tener marcados algunos caracteres a través del bit 7.

Las órdenes DEL, ERA y KILL sirven para borrar ficheros. El nombre del fichero puede incluir símbolos comodín para borrar grupos de ficheros a un tiempo; no obstante, como medida de seguridad se rechaza la forma "*.*". La orden DEL sólo está disponible en las versiones de BASIC para los sistemas operativos del tipo MS-DOS (PC-DOS). Análogamente, la orden ERA sólo se incluye en las versiones de BASIC para los sistemas operativos de tipo CP/M.

Las órdenes REN y NAME sirven para cambiar los nombres de los ficheros. Llevan dos parámetros, que son los nombres antiguo y nuevo del fichero. En ninguno de ellos pueden figurar símbolos comodín. Si se cita el nombre de una unidad, ambos nombres de fichero deben especificar la misma. Es obligatorio que exista un fichero con el nombre antiguo y que no exista ninguno con el nombre nuevo.

Las órdenes TYPE y DISPLAY leen el fichero especificado y escriben su contenido en la consola. En modo directo estas órdenes permiten examinar el contenido de un fichero. Cuando se las incluye en un programa, sirven para que éste imprima cierta información (por ejemplo, un fichero de ayuda) sin necesidad de tenerla ocupando espacio en la memoria. Al leer el fichero, el carácter SUB (control-Z, &H1A) se considera como indicador de fin de fichero. Las órdenes TYPE y DISPLAY se abandonan pulsando control-C. Pulsando control-S se suspende la salida por la consola, la cual se reanuda pulsando control-S por segunda vez (excepto en las versiones de 'sólo ejecución', en las que control-C y control-S no ejercen efecto alguno).

7.3 Apertura y cierre de ficheros. Números de fichero

Antes de poder procesar un fichero es necesario abrirlo. La orden que realiza esta operación es OPEN; en ella hay que especificar la siguiente información:

i. Modo de acceso.

   El fichero puede ser procesado en alguno de los siguientes modos:

   - entrada secuencial
   - salida secuencial
   - acceso aleatorio (entrada y/o salida)
   - acceso por claves (entrada y/o salida)

ii. Nombre del fichero (o de los ficheros, en el caso de ficheros de acceso por claves).

   Debe ser un nombre válido, sin símbolos comodín. La cuestión de si el fichero debe existir o no, así como otras restricciones, depende del modo de acceso.

iii. Un número de fichero.

Una vez abierto el fichero, toda referencia a él se hace ya a través de un número, no del nombre. La orden OPEN asocia el fichero con el número de fichero; todas las órdenes posteriores utilizan ese número. El número de fichero se especifica en la orden OPEN; tiene que ser un número que no esté siendo utilizado.

Los números de ficheros son enteros pequeños, del margen 1 ... *máximo*, donde *máximo* es inicialmente 3, pero puede ser modificado mediante las órdenes MEMORY y CLEAR, o bien especificado en la orden de carga de BASIC.

iv. Un bloqueo de fichero.

Se puede especificar un modo de bloqueo para el fichero (en los de acceso por claves es obligatorio). En los sistemas multiusuario BASIC intenta establecer el bloqueo especificado; si no lo consigue, la orden OPEN fracasa. Si no se especifica bloqueo, BASIC intenta uno implícito.

El modo de bloqueo implícito y las restricciones sobre los bloqueos aplicables en cada caso dependen de modo de acceso.

Los modos de acceso aleatorio y de acceso por claves pueden admitir o requerir más información. Véanse las secciones correspondientes y la descripción de la orden OPEN en el capítulo 9.

Una vez abierto el fichero, está preparado para su lectura o escritura (o ambas cosas), según permita el modo de acceso. En las secciones siguientes se describe el proceso de ficheros secuenciales y de acceso aleatorio. Los ficheros de acceso por claves serán estudiados en el capítulo 8.

Cuando se ha terminado de leer o escribir en el fichero, es necesario cerrarlo. El acto de cerrar un fichero ejerce los siguientes efectos:

- Si en el tampón de salida quedan datos, se los escribe en el fichero.
- Se informa al sistema operativo de que se ha terminado de procesar el fichero, para que pueda realizar las operaciones de gestión de discos pendientes.
- Se libera el número de fichero para que pueda volver a ser utilizado más tarde.

Se puede utilizar la orden CLOSE para cerrar todos los ficheros que están abiertos o sólo algunos de ellos. Otras órdenes cierran implícitamente todos los ficheros (por ejemplo, RESET y RUN). El cierre de ficheros de acceso por claves y el cierre de cualquier fichero en sistemas multiusuario pueden tener efectos adicionales, que describiremos en el capítulo 8.

7.4 Gestión de ficheros secuenciales

Los ficheros secuenciales que BASIC maneja son ficheros ordinarios de texto ASCII. El fichero puede contener caracteres cualesquiera del margen de &H00 a &HFF. En las operaciones de lectura los únicos caracteres que tienen algún significado especial para BASIC son los siguientes:

- 'Null' (&H00). BASIC ignora estos caracteres y no los entrega al programa.
- 'Retorno del carro' (&H0D). BASIC lo considera como fin de línea (salvo cuando va después de un 'avance de línea', véase más abajo) y no lo entrega al programa; el fin de línea tiene, no obstante, otros efectos. BASIC ignora el 'avance de línea' (&H0A) que pueda venir inmediatamente después de un 'retorno del carro'.
- 'Avance de línea' (&H0A). BASIC lo trata como un carácter más, salvo cuando venga inmediatamente después de un 'retorno del carro', véase más arriba. BASIC ignora el 'retorno del carro' (&H0D) que pueda venir inmediatamente después de un 'avance de línea'.
- 'SUB' (control-Z, &H1A) es considerado como fin de línea y fin de fichero. Se puede detectar el fin del fichero mediante la función EOF. Se produce un error si se intenta seguir leyendo el fichero cuando ya se ha llegado a su final.

En las operaciones de salida, BASIC escribe los caracteres tal como los recibe. Los únicos que añade son los siguientes:

- Pares 'retorno del carro'/'avance de línea' (&H0D, &H0A) para iniciar líneas nuevas.

  Antes de generar una línea nueva, BASIC escribe en el fichero este par de caracteres. En las operaciones de salida hacia un fichero secuencial, BASIC sólo genera líneas nuevas cuando así se le pide y al final de cada orden de salida (tal como PRINT #). En esto la salida hacia un fichero es distinta de la salida hacia la consola o la impresora, en las cuales se genera una línea nueva cada vez que se alcanza o sobrepasa la anchura del dispositivo.

- 'SUS' (&H1A, control-Z) al final del fichero.

  Cuando se cierra el fichero, BASIC escribe este carácter, que es el indicador de fin de fichero estándar.

Obsérvese que, a diferencia de lo que ocurre en la salida por la consola o la impresora, BASIC no expande los caracteres 'tab' (&H09) que envía a los ficheros.

La lectura secuencial de un fichero consiste en leer sus caracteres empezando por el primero y terminando, como muy tarde, con el indicador de fin de fichero. No es posible alterar el orden en que se leen los caracteres. Con la orden EOF se puede averiguar si se ha llegado al final del fichero. Todas las órdenes y funciones de entrada por la consola tienen sus análogas en la lectura de ficheros (salvo INKEY$). INPUT # lee secuencias de números en forma textual o cadenas de caracteres. LINE INPUT # lee líneas completas.

El hecho de abrir un fichero para salida secuencial equivale a crear un fichero nuevo y vacío, borrando el que pueda haber en el disco con el mismo nombre. Los caracteres enviados al fichero son escritos al final de él. Cuando se cierra el fichero, se escribe al final el carácter 'SUB' (&H1A, control-Z), que es el indicador de fin de fichero. Todas las órdenes de salida por la consola tienen sus análogas en la escritura de ficheros. WRITE # es la orden adecuada para escribir datos que vayan a ser leídos con PRINT #, en particular cuando los datos son cadenas literales.

7.5 Gestión de ficheros de acceso aleatorio

Los ficheros de acceso aleatorio de BASIC son colecciones de registros de longitud constante, cada uno de los cuales se identifica por un número del margen de 1 a 32767. Todos los registros de un fichero tienen la misma longitud, pero ésta puede variar de un fichero a otro. Los registros pueden ser leídos y escritos en cualquier orden. El contenido de los registros puede ser texto, leído y escrito con PRINT #, INPUT #, etc., o bien puede ser una información estructurada en campos, en los cuales los bytes del registro se consideran como cadenas de longitud fija.

Cuando se abre un fichero aleatorio se tiene que especificar la longitud de los registros (a menos que sea la implícita, 128 bytes). Si no existe ningún fichero con el nombre especificado, se crea uno nuevo, vacío. También se puede especificar un bloqueo de fichero. En los sistemas multiusuario, al abrir el fichero BASIC intenta establecer el bloqueo dado; si no se ha especificado bloqueo, abre el fichero desbloqueado (véase el capítulo 8). Una vez abierto el fichero, ya se puede leer o escribir sus registros (a menos que el fichero esté bloqueado en lectura, porque entonces lo único que se puede hacer con los registros es leerlos, no escribirlos).

BASIC mantiene un tampón de registros para cada fichero aleatorio que abre. Las órdenes GET y PUT transfieren datos desde el registro y hacia él, respectivamente. El programa puede leer y escribir en el tampón de diversas formas. Aunque BASIC recuerda el número del último registro leído o escrito, no impone ninguna correspondencia obligatoria entre el contenido del tampón y ese registro. De hecho, se puede leer el contenido de un registro para luego escribirlo, modificado o no, en el mismo registro o en otro diferente. También se puede preparar el contenido del tampón para luego escribirlo en un registro sin necesidad de previamente leer el registro.

Lo que no se recomienda es leer un registro que nunca haya sido escrito. El resultado depende del sistema operativo y de la historia de escrituras y lecturas anteriores. La reserva de espacio para un registro se hace la primera vez que se lo escribe. Los ficheros de acceso aleatorio no pueden disminuir de tamaño.

BASIC no impone límite alguno en el uso de los números de registro. La posición del principio de un registro viene dada por el número de registro menos uno multiplicado por la longitud del registro. En algunos sistemas operativos la existencia de un registro determinado está condicionada a la existencia de todos los de número inferior. Por consiguiente, cuando se asigna número a un registro, se debe utilizar el número más bajo disponible.

7.5.1 INPUT #, PRINT # y órdenes afines

Las órdenes INPUT #, PRINT # y otras afines pueden ser utilizadas para leer y escribir en el tampón de registros aleatorios. De hecho, estas órdenes consideran el registro como una serie de líneas.

BASIC mantiene un puntero que señala siempre el próximo byte que va a ser leído en el tampón de registros. Este puntero se reorienta hacia el principio del tampón cuando se abre el fichero y después de ejecutar una orden GET o PUT. Cada vez que se lee o escribe un byte, el puntero avanza hasta el siguiente. Se produce un error si se intenta leer más allá del final del registro. Cuando se lo trata de esta forma, el registro puede ser considerado como un pequeño fichero aleatorio, con la particularidad de que en el registro no hay ningún indicador de fin de fichero.

7.5.2 La orden FIELD y otras órdenes y funciones afines

La orden FIELD define una plantilla para los datos almacenados en el tampón de registros del fichero aleatorio dado. Para un mismo registro se puede definir varias plantillas, cada una de las cuales organiza los datos del tampón de manera diferente. La forma de la orden FIELD es la siguiente:

FIELD *referencia-de-fichero, lista-de: campo*

El parámetro *referencia-de-fichero* especifica a qué fichero se va a asociar la plantilla. Cada parámetro *campo* tiene la siguiente forma:

tamaño-campo AS *variable-literal*

El tamaño del campo especifica su longitud en bytes. La variable literal es la que en el futuro será identificada con el contenido del campo. El primer campo se establece al principio del registro; el segundo, inmediatamente a continuación del primero; y así sucesivamente.

La ejecución de una orden FIELD hace que se pierda el valor actual de las variables de campo para ser sustituido por el contenido actual del campo correspondiente. (Obsérvese que una orden FIELD puede revocar una definición anterior de variables de campo.) Las variables de campo pueden ser incluidas en expresiones literales como si fuesen variables «normales»; su valor es siempre el contenido del campo con que está identificada cada variable. El contenido de un campo sólo se puede modificar con una de las órdenes siguientes:

LSET, MID$ o RSET

pues éstas son las únicas que pueden asignar valores a cadenas de longitud fija. (Si se intenta asignar un valor a una variable de campo por cualquier otro método, se deshace la conexión entre la variable y el campo y se crea una nueva variable ordinaria sin que el contenido del tampón resulte afectado. Gracias a esto se puede volver a usar el nombre de las variables de campo, cuando ya no se las necesita, para darlo a otras variables.)

Para almacenar la información numérica de forma eficaz se dispone de un mecanismo que convierte los números en cadenas compactas, y viceversa. Las órdenes son:

Tipo de número	Entero	Doble precisión	Precisión normal
Convertir número en cadena	MKI$	MKS$	MKD$
Longitud de la cadena	2 bytes	4 bytes	8 bytes
Convertir cadena en número	CVI	CVS	CVD

Las cadenas producidas de esta forma son copias exactas de la representación interna de los números. Así pues, los números se recuperan tal como eran originalmente. Este mecanismo es diferente de otros procesos de entrada y salida, en los que siempre se produce la conversión entre las representaciones binaria y decimal.

Capítulo 8

Jetsam

Gestión de ficheros de acceso por claves y en sistemas multiusuario

Jetsam es un gestor de ficheros de acceso por claves incluido en el lenguaje BASIC. En este capítulo vamos a explicar brevemente para qué sirve Jetsam. Las órdenes y funciones de Jetsam están descritas detalladamente en el capítulo 9, junto con otras ampliaciones de las órdenes estándar de BASIC.

Las versiones multiusuario de Jetsam proporcionan, cuando funcionan bajo el control de un sistema operativo adecuado, mecanismos de bloqueo de ficheros y registros para el manejo de ficheros de acceso por claves y para el manejo de ficheros de acceso aleatorio y secuencial ordinarios.

Para cada fichero de acceso por claves, Jetsam de hecho mantiene dos ficheros. A lo largo de este capítulo a este par de ficheros le llamaremos «fichero de acceso por claves». Los dos ficheros del par son el «fichero de datos» y el «fichero de índices», cuyas misiones son las siguientes:

- El fichero de datos contiene los datos del usuario.

  Es un fichero normal de acceso aleatorio, con la particularidad de que los 128 primeros bytes del fichero (aproximadamente) y los dos primeros bytes de cada registro están reservados por Jetsam. Para leer y escribir se utilizan las órdenes GET y PUT habituales, pero los números de registros son proporcionados por Jetsam, no por el programa del usuario. Al definir la longitud de registro el usuario debe tener en cuenta los dos bytes requeridos por Jetsam. Por todo lo demás, estos dos bytes no son controlables ni accesibles para el usuario.

- El fichero de índices contiene las claves.

  Es un fichero de formato especial mantenido por Jetsam. Cada reseña del fichero de índices consta de tres componentes:

  i. Valor de la clave.

     Cada clave es una cadena literal de BASIC, de 31 caracteres como máximo. Las claves se almacenan en orden alfabético (la comparación de unas claves con otras se realiza según hemos explicado en la sección 2.11.3). Puede haber varias claves con el mismo valor. Una colección de claves de igual valor es lo que se denomina un *conjunto de claves.*

ii. Número de registro.

Cada clave hace referencia a un registro del fichero de datos. Para buscar un registro del fichero de datos, el programa tiene que hacer que Jetsam busque en el fichero de índices la clave que se le ha suministrado. Jetsam entrega al programa el número de registro, que luego puede ser utilizado para acceder al registro del fichero de datos.

iii. Rango.

Las claves se organizan en ocho rangos. Cada rango puede ser considerado como índice para el fichero de datos independiente de los otros rangos. Visto de otra forma, cada rango puede ser asimilado al primer carácter de la clave, de modo que todas las claves del rango 0 son «menores» que las del rango 1, etc.

Las órdenes y funciones de Jetsam incluyen los parámetros requeridos por los sistemas multiusuario para realizar las operaciones de bloqueo de ficheros y registros. En las versiones uniusuario, Jetsam exige que esos parámetros estén dentro de los márgenes correctos, pero los ignora a todos los efectos.

8.1 Relación entre BASIC y Jetsam

Las versiones de BASIC dotadas de Jetsam están ampliadas con las órdenes y funciones específicas de este sistema de gestión de ficheros. Algunas órdenes y funciones ordinarias de BASIC admiten parámetros adicionales y opcionales para su utilización con Jetsam.

En las versiones multiusuario, Jetsam obedece los parámetros que controlan el bloqueo de ficheros y registros. Algunas órdenes y funciones ordinarias de BASIC llevan parámetros adicionales y opcionales para controlar el bloqueo de ficheros y registros en los ficheros de acceso aleatorio y secuencial.

8.2 Consistencia de los ficheros de datos y de índices

Durante el proceso de escritura de un fichero de acceso por claves hay momentos en los que el fichero de datos y el fichero de índices son inconsistentes el uno con el otro. La primera vez que se escribe en cualquiera de los dos ficheros, Jetsam pone un marcador en ambos. Estos marcadores residen en áreas reservadas en el fichero de datos y en la cabecera del fichero de índices; indican que los ficheros se encuentran en un estadio intermedio indeterminado. Cuando se cierra el fichero de acceso por claves, Jetsam borra los marcadores; pero, si el sistema falla antes, los marcadores quedan activos. No es posible abrir un fichero de acceso por claves en el que los marcadores estén activos; de esta manera se garantiza que Jetsam no trabaje con ficheros inconsistentes.

Jetsam utiliza un sistema de memoria «caché» para el almacenamiento temporal de los ficheros de índices. Las modificaciones del fichero de índices se realizan en la memoria interna y se las escribe en el disco lo más tarde posible. En cambio, para el fichero de datos no hay tal almacenamiento temporal. Debido a esto, es posible que las modificaciones que se han introducido hasta un momento dado hayan sido grabadas en el fichero de datos y no en el de índices.

Aparte del problema de la inconsistencia física, en ocasiones puede haber también inconsistencias lógicas en el fichero de acceso por claves cuando un programa lo está modificando; en particular cuando los datos son interdependientes. El sistema de marcadores impide el uso de datos cuya modificación no haya sido completada, suponiendo que el programa garantice que los datos son lógicamente consistentes antes de cerrar el fichero.

Jetsam asegura la consistencia interna de los programas de acceso por claves, pero no la consistencia de unos con otros. El programador debe elaborar su propio sistema de marcadores para garantizar esa consistencia cuando el programa maneje varios ficheros simultáneamente.

El sistema de marcadores de Jetsam se basa en unos números de «generación» grabados en el fichero de datos y en el de índices. Para marcarlos como inconsistentes, incrementa en una unidad el número de generación de un fichero y en dos unidades el del otro. Los ficheros solamente son consistentes cuando son de la misma generación, es decir, cuando coinciden los dos números. Se recomienda que, antes de empezar a modificar un fichero de acceso por claves, se haga una copia del fichero de índices y del de claves, de modo que se pueda retroceder a la anterior versión consistente en caso de fallo del sistema. (Nótese que las copias de seguridad se marcan como inconsistentes con los ficheros originales.) Al cerrar el fichero de acceso por claves, se escriben en el disco todos los datos modificados, se actualiza el fichero de índices y se anulan los marcadores de inconsistencia. La función CONSOLIDATE ejerce el mismo efecto que cerrar y volver a abrir el fichero, y además retiene la situación de bloqueo actual. En las aplicaciones en que la lectura de datos sea mucho más frecuente que la modificación del fichero de acceso por claves, se aconseja utilizar CONSOLIDATE después de cada modificación (o grupo de modificaciones) y hacer inmediatamente una copia de los nuevos ficheros.

8.3 Sistemas multiusuario: bloqueo de registros y ficheros

Cuando en un sistema multiusuario hay varios usuarios utilizando simultáneamente el mismo fichero, es necesario controlar de alguna forma el acceso que les está permitido para asegurar que todos dispongan de datos consistentes. Esto es muy sencillo si lo único que hacen los usuarios es leer el fichero, pero más complejo si alguno de ellos lo modifica.

Cuando un usuario modifica un fichero, las áreas afectadas tienen que ser bloqueadas; cuando concluye la modificación ya se las puede desbloquear. El bloqueo impide que otros usuarios accedan a áreas que están cambiando, en las que de otra forma podrían encontrar información incompleta o inconsistente.

En otras ocasiones lo que hay que impedir es que la información cambie mientras no se haya terminado de realizar alguna operación. El bloqueo tiene que consistir entonces en impedir las modificaciones.

El bloqueo se puede realizar a dos niveles: fichero y registros. Cuando se abre el fichero se establece uno de los siguientes bloqueos de fichero: desbloqueado, bloqueado en lectura o bloqueado en escritura.

El bloqueo de registros no es necesario cuando el fichero se ha abierto especificando algún tipo de bloqueo; es imposible en los ficheros de acceso secuencial. Los registros pueden estar desbloqueados, bloqueado en lectura y bloqueados en escritura.

En las versiones uniusuario, Jetsam exige que los parámetros de bloqueo tengan valores correctos, pero los ignora a todos los efectos. En particular, no comprueba si el bloqueo especificado es razonable. Por consiguiente, las versiones uniusuario ejecutan correctamente los programas escritos para versiones multiusuario. En cambio, cuando un programa se desarrolla en una versión uniusuario, puede no funcionar en la versión multiusuario, pues no se habrán comprobado las instrucciones que gestionan todo lo relativo al bloqueo de ficheros y de registros.

8.3.1 Bloqueo de fichero en ficheros de acceso por claves

Un fichero de acceso por claves de Jetsam puede ser abierto desbloqueado, bloqueado en lectura o bloqueado en escritura:

Desbloqueado	Los demás usuarios también pueden abrir el fichero (desbloqueado o bloqueado en lectura). A menos que algún otro usuario abra el fichero bloqueado en lectura, todos pueden escribir en él. Puede ser necesario bloquear los registros para controlar el acceso al fichero cuando se lo va a modificar.
Bloqueado en lectura	Los demás usuarios también pueden abrir el fichero (desbloqueado o bloqueado en lectura). Ningún usuario puede escribir en el fichero, luego no es necesario bloquear los registros.
Bloqueado en escritura	Ningún otro usuario puede abrir el fichero; la operación falla si el fichero ya está abierto. El propietario del bloqueo en escritura tiene acceso en exclusiva al fichero y no necesita bloquear los registros.

8.3.2 Bloqueo de fichero en ficheros de acceso aleatorio

Un fichero de acceso aleatorio puede ser abierto desbloqueado, bloqueado en lectura o bloqueado en escritura:

Desbloqueado	Los demás usuarios también pueden abrir el fichero, pero sólo desbloqueado. Todos los usuarios pueden leer y escribir en el fichero. Puede ser necesario bloquear los registros para controlar el acceso al fichero cuando se lo va a modificar.
Bloqueado en lectura	Los demás usuarios también pueden abrir el fichero, pero sólo desbloqueado. Ningún usuario puede escribir en el fichero, luego no es necesario bloquear los registros.

Bloqueado en escritura	Ningún otro usuario puede abrir el fichero; la operación falla si el fichero ya está abierto. El propietario del bloqueo en escritura tiene acceso en exclusiva al fichero y no necesita bloquear los registros.

8.3.3 Bloqueo de registros en ficheros de acceso por claves y de acceso aleatorio

Los ficheros de acceso por claves y los de acceso aleatorio pueden estar desbloqueados, bloqueados en lectura o bloqueados en escritura:

Desbloqueado	No hay ninguna limitación en cuanto a lo que otros usuarios pueden hacer con el registro y sus claves. Toda información sobre el registro y cualquiera de sus claves puede dejar de ser válida en cualquier momento.
Bloqueado en lectura	Los demás usuarios también pueden bloquear el registro en lectura. El bloqueo en lectura falla si el registro ya está bloqueado en escritura. Ningún usuario puede bloquear el registro en escritura. Si un registro está bloqueado en lectura, no es posible modificarlo, ni tampoco añadirle ni suprimir claves.
Bloqueado en escritura	Ningún otro usuario puede bloquear el registro; la operación falla si algún otro usuario ya ha bloqueado el registro. El propietario del bloqueo en escritura tiene acceso en exclusiva al registro. Si un registro está bloqueado en escritura (o si lo está el fichero), no es posible modificarlo, ni tampoco añadirle ni suprimir claves.

En los ficheros de acceso por claves de Jetsam, el bloqueo de registros se aplica cuando se encuentra una clave del registro, no cuando se va a leer el registro. De esta forma se asegura que el registro no pueda ser borrado ni modificado de ninguna forma en el intervalo que media entre el momento en que se obtiene del índice el número de registro y el momento en que se accede al registro.

En los ficheros ordinarios de acceso aleatorio, los registros se pueden bloquear en cualquier momento mediante la función LOCK, y también inmediatamente antes de leerlos con GET o inmediatamente después de escribirlos con PUT. Las órdenes GET y PUT, cuando se las utiliza en un fichero aleatorio, aplican y liberan implícitamente un bloqueo temporal en escritura sobre el registro si el programa no bloquea explícitamente los registros.

8.3.4 Bloqueo temporal en escritura en los ficheros de acceso aleatorio

Este sistema facilita el proceso de los ficheros aleatorios que han sido abiertos desbloqueados; gracias a él no es necesario bloquear explícitamente los registros (a condición de que éstos se procesen uno a uno). BASIC mantiene un bloqueo temporal en escritura para un registro de cada fichero que haya sido abierto sin bloqueo:

- Cuando se ejecuta una orden GET:

  i. se abandona el bloqueo temporal en lectura, si lo hay;
  ii. se procesa el parámetro de bloqueo, si lo hay;
  iii. si el registro no está bloqueado, BASIC trata de aplicarle un bloqueo temporal en escritura;
  iv. se lee el registro.

- Cuando se ejecuta una orden PUT:

  i. se escribe el registro;
  ii. si hay algún parámetro de bloqueo, se aplica el bloqueo especificado y se olvida el bloqueo temporal en escritura;
  iii. si no hay parámetro de bloqueo, se abandona el posible bloqueo temporal en escritura.

- Cuando se ejecuta una orden LOCK:

  Si hay algún bloqueo temporal en escritura, BASIC lo olvida cuando aplica el nuevo bloqueo.

8.4 Búsqueda de los registros y posición actual dentro de un fichero de acceso por claves

El fichero de índices se puede considerar como una lista de claves dispuestas en orden ascendente (alfabético), de tal forma que todas las claves del rango 0 son anteriores a las del rango 1, etc. Cuando está procesando un fichero, Jetsam mantiene una «posición actual», que es la clave actual. Es importante tener en cuenta que la posición actual está caracterizada por una clave del fichero de índices; aunque esta posición identifica un registro del fichero de datos, lo hace indirectamente.

Cuando se abre un fichero de acceso por claves sin ningún bloqueo en un sistema multiusuario, otros usuarios pueden modificar el fichero de índices en cualquier momento. Por eso se proporciona un mecanismo de bloqueo de registros, para mantener constantes ciertas áreas del fichero mientras se las está procesando, aun cuando el resto del fichero pueda estar siendo modificado por otros usuarios. Aunque la posición actual esté definida por una clave, es el registro lo que se bloquea, de modo que queda protegido tanto el registro como todas las claves asociadas a él.

Para acceder a los datos de un fichero de acceso por claves, el programa tiene que utilizar las órdenes de Jetsam que desplazan la posición actual dentro del índice y luego utilizar las órdenes GET y PUT para leer y escribir los registros cuyos números se han obtenido a través de las claves. El programa puede recordar las posiciones de determinados registros para utilizarlas posteriormente (suponiendo, en los sistemas multiusuario, que establezca los bloqueos adecuados).

Hay diversas formas de desplazar la posición actual por el índice:

- Ir al principio de un rango dado.
- Restablecer la posición actual a un valor almacenado previamente.
- Buscar la primera clave que tiene el valor especificado (en un rango dado).
- Ir a la siguiente clave.
- Ir a la clave anterior.
- Ir al siguiente conjunto de claves.

Estas funciones permiten utilizar el fichero de acceso por claves tanto en modo aleatorio como en modo secuencial.

En los sistemas multiusuario, si la posición actual no está bloqueada, la clave actual puede ser borrada en cualquier momento por otro usuario. Cada vez que se realiza una función de búsqueda a partir de la posición actual, lo primero que hace Jetsam es comprobar que la clave todavía existe.

En ciertas condiciones la posición actual no está determinada, esto es, no corresponde a ninguna clave.

8.5 Resumen de las órdenes y funciones de Jetsam y recomendaciones para su utilización

En esta sección vamos a dar un breve resumen de los recursos de gestión de ficheros proporcionados por Jetsam y a hacer algunas recomendaciones sobre la forma de utilizarlos. Algunas órdenes de BASIC tienen versiones especiales para su uso con Jetsam, e incluso otras versiones para Jetsam en sistemas multiusuario. Todas las funciones y órdenes están descritas exhaustivamente en el capítulo 9.

8.5.1 Limitaciones de tamaño y otras restricciones en Jetsam

El número máximo de ficheros de acceso por claves que pueden estar abiertos simultáneamente sólo está limitado por el número máximo de ficheros de cualquier tipo (véase las órdenes MEMORY y CLEAR y el apéndice I).

La longitud de registro mínima para el fichero de datos es de 2 bytes, ambos reservados por Jetsam para su uso interno. La longitud de registro máxima se establece igual que para los otros tipos de fichero (véase las órdenes MEMORY y CLEAR y el apéndice I). La longitud de registro debe ser igual a la suma de las longitudes de los campos (orden FIELD) más dos.

El valor máximo del número de registro es 32767, igual que en los ficheros aleatorios de BASIC. Los primeros 128 bytes del fichero de datos están reservados por Jetsam; el primer número de registro depende de la longitud de registro: si la longitud de registro es igual o mayor que 128, el primer número de registro es 2; si no, el primer número de registro es igual a

2+INT(127/longitud de registro)

El tamaño máximo de un fichero de acceso por claves está limitado, por lo que a Jetsam se refiere, por el máximo número de registro y la longitud de registro; en último extremo, está limitado por el sistema operativo y por la capacidad de almacenamiento externo.

Los valores de las claves son cadenas literales cuya longitud máxima es 31 bytes. Las claves se comparan byte a byte, de izquierda a derecha. Se puede utilizar las funciones UPPER$ y LOWER$ para homogeneizar las claves, convirtiendo todas las letras a mayúsculas o a minúsculas. Las cadenas producidas por MKI$, MKS$ y MKD$ no son adecuadas para su uso como valores de claves (véase el capítulo 9).

Los rangos posibles son ocho, numerados del 0 al 7.

Jetsam utiliza un tampón («caché») para el almacenamiento temporal de los índices. El tamaño de este tampón se puede modificar mediante la orden BUFFERS.

En los sistemas multiusuario, BASIC requiere algún espacio para guardar información sobre todos los bloqueos de registro activos en cada momento. El número máximo de bloqueos de registro para cada fichero se establece a través del segundo parámetro de la orden BUFFERS.

8.5.2 Creación de un fichero de acceso por claves

CREATE, RANKSPEC

CREATE. Antes de poder utilizar un fichero de acceso por claves, es necesario crear e inicializar los ficheros de datos y de índices. La orden CREATE crea ambos ficheros, inicializa la cabecera del fichero de datos y prepara un índice vacío. CREATE abre inmediatamente el fichero recién creado. Cuando se abre (con OPEN) un fichero nuevo para salida o para acceso aleatorio, BASIC lo crea automáticamente. No ocurre lo mismo con los ficheros de acceso por claves, que tienen que existir antes de que se ejecute la orden OPEN.

RANKSPEC. Los diversos rangos de un fichero de acceso por claves se pueden marcar para impedir que se le pueda añadir claves que tengan el mismo valor que otras ya existentes. RANKSPEC prohibe o permite esta «repetición de claves» dentro de un rango.

8.5.3 Apertura y cierre de ficheros de acceso por claves en sistemas uniusuario y de ficheros de todo tipo en sistemas multiusuario

OPEN, CLOSE, CONSOLIDATE, BUFFERS

OPEN. La orden OPEN de BASIC se amplía para su uso en los ficheros de acceso por claves de Jetsam. En este caso la orden requiere los nombres de los ficheros de datos y de índices, así como la especificación de bloqueo. La operación falla si no existe alguno de los ficheros, si son inconsistentes o si no se puede lograr el bloqueo solicitado.

CLOSE. Es extremadamente importante cerrar (CLOSE) el fichero de acceso por claves en cuanto el programa ha terminado de procesarlo. Si se ha escrito algo en el fichero, al cerrarlo se envían al disco todos los datos pendientes de grabar en el fichero de datos y se actualiza el fichero de índices; además se marca el fichero como consistente. Se liberan todos los bloqueos de registro y el bloqueo de fichero, si lo hay. Si no se cierra el fichero explícitamente, los ficheros siguen marcados como inconsistentes y los bloqueos activos persisten.

En los sistemas multiusuario, la orden OPEN lleva, cuando se la aplica a ficheros ordinarios de acceso aleatorio o secuencial, un parámetro opcional para el bloqueo del fichero. En estos casos la orden libera todos los bloqueos de registro.

CONSOLIDATE. La función CONSOLIDATE se aplica a los ficheros de acceso por claves cuando están abiertos. Su efecto consiste en escribir en el disco los datos almacenados en el tampón y los índices, y marcar el fichero como consistente, todo ello sin cerrar el fichero y sin liberar los bloqueos de registro que estén activos.

BUFFERS. Jetsam utiliza un sistema de memoria «caché» para el almacenamiento temporal de los ficheros de índices. El tamaño de este tampón se puede establecer con la orden BUFFERS. Cuanto mayor sea el tampón, más índices puede almacenar Jetsam en la memoria, con la consiguiente mejora de las prestaciones.

El espacio reservado a cada tampón se resta del disponible para el programa. Con la orden BUFFERS se puede elegir el punto de equilibrio más adecuado a cada caso.

En los sistemas multiusuario, BASIC requiere algún espacio para guardar información sobre todos los bloqueos de registro activos en cada momento. El número máximo de bloqueos de registro para cada fichero se establece a través del segundo parámetro de la orden BUFFERS. (Puesto que el valor implícito es cero, es necesario ejecutar una orden BUFFERS antes de poder establecer cualquier bloqueo de registro.)

8.5.4 Lectura, escritura y bloqueo de los registros de datos

GET, PUT, LOCK

Ficheros de acceso por claves. Las órdenes GET y PUT funcionan también en los ficheros de acceso por claves. Sin embargo, la forma de utilizarlas es algo diferente de la habitual en los ficheros de acceso aleatorio ordinarios:

i. Creación de registros.

En los ficheros de acceso aleatorio ordinarios, un registro nuevo se crea preparando su contenido y escribiéndolo con la orden PUT, en la cual se especifica el número de registro explícita o implícitamente. El procedimiento es similar en los ficheros de acceso por claves; el nuevo registro se escribe con la función ADDREC, con la cual Jetsam asigna el número de registro y simultáneamente escribe en el índice una clave para el nuevo registro.

i. Número de registro.

Los números de los registros están controlados enteramente por Jetsam; sirven como punteros desde las claves hacia los datos asociados. Los únicos números de registro utilizables por el programa son los suministrados por Jetsam. Si un programa ha de funcionar en un sistema multiusuario, sólo debe recordar los registros que tenga bloqueados en lectura o en escritura.

iii. Actualización de los registros.

El hecho de que Jetsam utilice los dos primeros bytes de cada registro de datos obliga a dos cosas: en primer lugar, el programa debe preservar esos dos bytes; en segundo lugar, debe leer (GET) el registro antes de modificarlo, aun cuando vaya a cambiarlo completamente (salvo, por supuesto, los dos bytes reservados por Jetsam).

iv. Bloqueos en escritura.

En los sistemas multiusuario, el programa no debe escribir en un registro que no haya bloqueado en escritura previamente.

Se puede usar la función LOCK para cambiar el estado de bloqueo de un registro. Para ello es necesario que el registro ya esté bloqueado; es decir, LOCK puede cambiar el tipo de bloqueo, de escritura a lectura y viceversa, o desbloquear el registro.

Ficheros de acceso aleatorio en sistemas multiusuario. Las órdenes GET y PUT aceptan un parámetro opcional para el bloqueo de registros cuando se las utiliza en ficheros aleatorios en sistemas multiusuario. En el caso de GET, el bloqueo se aplica inmediatamente antes de leer el fichero. En el caso de PUT, inmediatamente después de escribirlo.

La función LOCK también es utilizable en los ficheros de acceso aleatorio ordinarios. Si el programa no aplica explícitamente ningún bloqueo, BASIC establece implícitamente un bloqueo temporal en escritura antes de leer (GET) el registro y lo libera después de escribirlo (PUT); véase el apartado 8.3.4.

8.5.5 Creación y eliminación de registros y claves en ficheros de acceso por claves

ADDREC, ADDKEY, DELKEY

ADDREC crea un registro nuevo en el fichero de datos e inserta la correspondiente clave en el índice. Jetsam asigna un número de registro al nuevo registro. Ésta es la única forma posible de crear registros. La creación del registro y la inclusión de su clave en el índice han de realizarse simultáneamente, porque de lo contrario no habría manera de acceder al registro. Los datos que se escriben en el nuevo registro son los que actualmente están en el tampón de registros; Jetsam inicializa los dos primeros bytes del registro, que reserva para su uso.

ADDKEY. Un registro puede tener asociadas varias claves de uno o varios rangos. La primera clave es la que escribe ADDREC. Las demás se añaden con ADDKEY.

DELKEY borra del índice una clave del registro especificado. Si la clave era la única que tenía el registro, entonces se borra también el registro y el espacio que éste ocupaba puede ser utilizado por un ADDREC posterior. Por consiguiente, para modificar una clave el procedimiento correcto es añadir la nueva con ADDKEY y después borrar la antigua con DELKEY.

8.5.6 Búsqueda de claves y modificación de la posición actual en ficheros de acceso por claves

SEEKRANK, SEEKKEY, SEEKNEXT, SEEKPREV, SEEKSET, SEEKREC

Todas estas funciones buscan una clave, especificada de diversas formas. Si la búsqueda tiene éxito, la clave se convierte en la posición actual y se aplica el bloqueo solicitado al correspondiente registro de datos. **Nota:** estas funciones no leen ni escriben datos; tal tarea la realizan las órdenes GET y PUT.

SEEKRANK busca la primera clave del rango especificado. Esta función deja a Jetsam preparado para procesar en modo secuencial los datos cuyas claves están en ese rango.

SEEKKEY busca la primera clave del rango especificado cuyo valor coincide con el especificado. Esta función permite el acceso aleatorio a los registros de datos a través de los valores de las claves. La clave así encontrada puede ser la primera de un «conjunto de claves».

SEEKNEXT busca la clave siguiente a la posición actual o a la posición especificada. Esta función permite avanzar por el fichero recorriendo las claves.

SEEKPREV busca la clave inmediatamente anterior a la posición actual o a la posición especificada. Esta función permite retroceder por el fichero recorriendo las claves.

SEEKSET busca la siguiente clave o conjunto de claves a partir de la posición actual o de la posición especificada. SEEKSET es similar a SEEKNEXT, pero se salta las claves cuyo valor (y rango) sea igual al de la clave a partir de la cual empieza a buscar.

SEEKREC busca en un rango concreto una clave concreta que haga referencia a un registro determinado. SEEKREC sirve para retroceder a una posición recordada por el programa; en un sistema multiusuario el programa debería haber bloqueado el registro.

8.5.7 Búsqueda de la posición actual en ficheros de acceso por claves

FETCHREC, FETCHKEY$, FETCHRANK

Cada una de estas funciones genera un componente de la posición actual.

FETCHREC da el número de registro que Jetsam ha asignado al registro referenciado por la clave que hay en la posición actual dentro del índice. Si el registro está bloqueado en lectura o en escritura, se puede utilizar ese número para acceder al registro de datos.

FETCHKEY$ da el valor de la clave que hay en la posición actual.

FETCHRANK da el rango de la posición actual.

8.6 Observaciones generales acerca de las funciones y órdenes de Jetsam

Casi todas las operaciones de Jetsam se realizan mediante funciones, muy pocas mediante órdenes. Las funciones generan un código que informa sobre el resultado de la operación. Por consiguiente, para que Jetsam realice una de esas operaciones, la función tiene que aparecer formando parte de una expresión. Se recomienda utilizar expresiones del tipo

```
código.generado% = SEEKNEXT(#1,1)
```

Hay ocasiones en que interesa que la función intervenga en una expresión condicional construida con IF. En el ejemplo anterior Jetsam va a buscar la siguiente clave en el fichero #1 y a aplicarle un bloqueo en lectura. Si la operación termina con éxito, el programa querrá saber si la clave encontrada está en el siguiente conjunto o en el siguiente rango. Por otra parte, la operación puede fallar por diversas razones. Cada uno de estos resultados produce un código que lo caracteriza unívocamente; el programa puede utilizar ese código para decidir, en una instrucción IF ... THEN ... , qué debe hacer a continuación.

8.6.1 Errores y códigos generados

Los errores detectados en la ejecución de una orden o una función de Jetsam tienen asociado un número de error, que puede ser utilizado en una orden ON ERROR GOTO para que el programa responda adecuadamente. Ejemplos de errores son los siguientes:

- parámetros incorrectamente especificados
- operaciones «ilegales»
- intentos de leer o escribir registros sin los bloqueos adecuados
- problemas en el sistema de disco, tales como 'disco lleno'

Jetsam genera algunos errores que son desconocidos para el BASIC ordinario. Estos errores están relacionados en el apéndice II, junto con todos los demás mensajes de error.

Los valores de los códigos generados por las funciones de Jetsam tienen los siguientes significados generales:

0	éxito sin más calificativos
101	se ha sobrepasado el final del conjunto de claves
102	se ha sobrepasado el final del rango
103	se ha sobrepasado la última clave del índice
105	no se ha encontrado la clave
115	no está definida la posición actual
130	el fichero ya está bloqueado por otro usuario
131	no se ha podido bloquear en escritura el registro correspondiente a la clave que se va a añadir o suprimir
132	no se ha podido bloquear en lectura el registro
133	no se ha podido bloquear en escritura el registro

En las descripciones del capítulo 9 se dan los posibles códigos para cada función. El significado preciso depende en ocasiones de la función. Los códigos se dividen en 'códigos de éxito' y 'códigos de fracaso'. En general, las funciones generan un código de fracaso cuando no han sido capaces de modificar el fichero, la posición actual o la situación de bloqueo. A veces el fallo tiene efectos adicionales, y entonces se los describe explícitamente.

Si una función genera un 'fallo de bloqueo' (códigos 130 a 133), las acciones recomendadas son las siguientes:

130 El fichero ha sido bloqueado por otro usuario. El programa debe informar de este hecho al operador y esperar instrucciones, ya que el otro usuario puede seguir teniendo bloqueado el fichero durante algún tiempo.

131 Cuando se añade o suprime una clave, Jetsam necesita escribir en el fichero de datos. Si el registro no está bloqueado en escritura (y si tampoco lo está el fichero), Jetsam trata de bloquearlo temporalmente. Este código se genera si Jetsam no consigue realizar tal bloqueo temporal en escritura. Puede ocurrir que el registro esté bloqueado por otro usuario y que éste lo libere al cabo de unos instantes; el programa debería reintentar la operación unas cuantas veces antes de solicitar la intervención del operador.

132, 133 Las funciones ADDKEY, DELKEY, LOCK y las diversas SEEK llevan un parámetro *bloqueo*. Si la función no consigue establecer el bloqueo solicitado por ese parámetro, puede ocurrir que el registro esté bloqueado por otro usuario y que éste lo libere al cabo de unos instantes; el programa debería reintentar la operación unas cuantas veces antes de solicitar la intervención del operador.

En el apéndice VII (pág. 385) damos el listado completo de un programa de 'Lista de direcciones y teléfonos' que ilustra la utilización de los ficheros de Jetsam.

Capítulo 9

Órdenes y funciones intrínsecas

En este capítulo vamos a dar la descripción completa de todas las órdenes y funciones intrínsecas, por orden alfabético de palabras clave.

9.1 Definición de la terminología

A lo largo de este capítulo utilizaremos los siguientes términos:

cadena-entrecomillada	Un conjunto de entre 0 y 255 caracteres puestos entre comillas, o bien que empiezan por comillas y terminan con un retorno del carro.
constante-numérica	Véase la sección 2.4.
expresión	Una expresión que produce un valor de cualquier tipo.
expresión-de-dirección	Una *expresión-numérica* que se convierte en entero sin signo; véase la sección 2.10.
expresión-entera	Una *expresión-numérica* cuyo valor, después de redondeado a un entero, pertenece al margen de −32768 a +32767.
expresión-literal	Una expresión que toma un valor de tipo literal (cadena de texto).
expresión-lógica	BASIC no maneja variables de tipo booleano. Considera el entero 0 como 'falso' y todos los demás enteros como 'verdadero'. Por lo tanto, una *expresión-lógica* no es más que una *expresión-entera* cualquiera.
expresión-de-nombre-de-fichero	Una *expresión-literal* cuyo valor es un *nombre-de-fichero* válido.

expresión-numérica — Una expresión que toma un valor de cualquiera de los tipos numéricos: entero, precisión sencilla y doble precisión.

Caso particular son las expresiones lógicas y las expresiones enteras.

expresión-de-número-de-fichero — Una *expresión-entera* que da un valor perteneciente al margen 1 ... *ficheros*, donde *ficheros* es el máximo número de ficheros manejables simultáneamente, establecido cuando se cargó BASIC (véase el apéndice I).

marcador-de-tipo — Uno de los caracteres siguientes:

%, que indica 'entero'
!, que indica 'precisión sencilla'
#, que indica 'doble precisión'
$, que indica 'cadena literal'

margen-de-números-de-línea — Especifica todas las líneas comprendidas en el margen especificado, incluidos los extremos. Puede tener cualquiera de las formas siguientes:

número-de-línea — La línea citada es la única que pertenece al margen.

número-de-línea—*número-de-línea* — Las líneas citadas y las comprendidas entre ellas.

número-de-línea— — Desde la línea citada hasta el final del programa.

—*número-de-línea* — Desde el principio del programa hasta la línea citada.

nombre-de-fichero — Una colección de caracteres que puede ser admitida como nombre de fichero por el sistema operativo. BASIC convierte las minúsculas a mayúsculas antes de entregar el nombre al sistema operativo.

nombre-de-variable — El nombre de una *variable-sencilla*, que puede incluir su *marcador-de-tipo*.

nombre-de-variable-matricial — El nombre de una matriz, que puede incluir su *marcador-de-tipo*.

número-de-línea — Un número decimal del margen de 0 a 65534. En modo directo se puede poner en su lugar un punto, con el significado de «línea actual».

número-de-registro	Una *expresión-entera* que toma un valor perteneciente al margen de 1 a 32767. Los registros de los ficheros aleatorios y de los ficheros de acceso por claves (Jetsam) se identifican por su *número-de-registro*.
referencia-de-fichero	*[#]expresión-de-número-de-fichero*
variable	Una referencia a una variable de cualquier clase. Puede ser también un elemento de una matriz, y en tal caso se debe incluir también los subíndices adecuados.
variable-literal	Una referencia a una variable, que también puede ser un elemento de una matriz, cuyo valor es de tipo literal (cadena de texto).
variable-numérica	Una referencia a una variable, que también puede ser un elemento de una matriz, cuyo tipo es alguno de los numéricos.
variable-sencilla	Una referencia a una variable cualquiera que no sea un elemento de una matriz. Puede incluir un *marcador-de-tipo*.

9.2 Definición de términos de uso frecuente en los sistemas de Jetsam y multiusuario

Los siguientes términos se utilizan en las descripciones de las órdenes y funciones de Jetsam y en las versiones de las órdenes y funciones corrientes de BASIC ampliadas para su uso en Jetsam.

bloqueo

Es una *expresión-entera* que especifica el bloqueo de un fichero o de un registro. La expresión tiene que dar un valor del margen de 0 a 2. Su significado es el siguiente:

0, sin bloqueo
1, bloqueo en lectura
2, bloqueo en escritura

posición-de-índice

Equivale a *rango,valor-de-clave,número-de-registro*. Los valores de todos los componentes tienen que haber sido generados por Jetsam y, preferiblemente, deben hacer referencia a un registro que esté bloqueado.

rango

Es una *expresión-entera* que da un valor del margen de 0 a 7 y especifica uno de los ocho rangos posibles.

valor-de-clave

Es una *expresión-literal* que da una cadena cuya longitud está en el margen de 0 a 31.

Valor absoluto. Función

Aplicación

Calcular el valor absoluto de la expresión dada.

Forma

ABS(*expresión-numérica*)

Notas

ABS aplicada a un valor negativo da el valor cambiado de signo. Aplicada a un valor positivo, da el mismo valor.

Palabras clave asociadas

SGN

ADDKEY

Añadir una nueva clave a un fichero existente. **Función de Jetsam**

Aplicación

Añadir una nueva clave al fichero de índices, haciéndola corresponder a un registro que ya existe en el fichero de datos. La nueva clave pasa a ser la 'clave actual'.

Forma

ADDKEY(*referencia-de-fichero,bloqueo,rango,clave,número-de-registro*)

La *referencia-de-fichero* es el número de fichero de un fichero de acceso por claves que esté abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se debe aplicar al registro una vez añadida la nueva clave.

El *número-de-registro* especifica el registro al que se va a añadir la nueva clave. El número tiene que haber sido generado antes por Jetsam; el registro tiene que estar bloqueado en escritura.

Los parámetros *rango* y *clave* especifican el resto de la nueva clave que va a ser insertada en el índice.

Códigos generados

0	Éxito	Se ha añadido la clave; la nueva clave ha pasado a ser la posición actual; se ha establecido el bloqueo requerido.
116	Fracaso	Ya existe una clave con el mismo valor en el rango especificado, y el rango está marcado para no admitir claves repetidas (con RANKSPEC). Se ha establecido la posición actual en la primera clave del rango que coincide con el valor dado. No se ha aplicado bloqueo.
130	Fracaso	El fichero está bloqueado en lectura por otro usuario.
131	Fracaso	No se ha podido bloquear en escritura el registro al que iba a pertenecer la clave nueva.
132	Fracaso	No se ha podido aplicar el bloqueo de lectura al registro.
133	Fracaso	No se ha podido aplicar el bloqueo de escritura al registro.

Notas

Si el registro al que va a hacer referencia la clave nueva no está bloqueado en escritura (y tampoco lo está el fichero), ADDKEY automáticamente establece un bloqueo temporal en escritura. Si no consigue establecer este bloqueo, genera el código 131.

Cuando se creó el registro, se escribió para él una clave en el índice. ADDKEY sirve para añadir nuevas claves que hagan referencia a ese mismo registro (por ejemplo, en un rango distinto; cada rango se considera como un índice independiente de los demás). También sirve para modificar una clave: primero se inserta la versión nueva con ADDKEY y **después** se suprime la antigua con DELKEY. (El orden ADDKEY → DELKEY es muy importante; véase DELKEY.)

Si la operación concluye con éxito, la posición actual se identifica con la clave recién añadida y se aplica el *bloqueo* solicitado; de lo contrario, no se modifica la posición actual, a menos que el fallo sea el 116.

Palabras clave asociadas

ADDREC, DELKEY, RANKSPEC

ADDREC

Añadir un registro y una clave nuevos. Función de Jetsam

Aplicación

Añadir una nueva clave al fichero de índices y el correspondiente registro al fichero de datos. La nueva clave pasa a ser la 'clave actual'.

Forma

ADDREC(*referencia-de-fichero,bloqueo,rango,valor-de-clave*)

La *referencia-de-fichero* es el número de fichero de un fichero de acceso por claves que esté abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se debe aplicar al nuevo registro.

Los parámetros *rango* y *valor-de-clave* especifican la clave que va a ser insertada en el índice. El número de registro es suministrado por Jetsam.

Códigos generados

0	Éxito	Se ha añadido la clave y el registro; la nueva clave ha pasado a ser la posición actual; se ha establecido el bloqueo requerido.
116	Fracaso	Ya existe una clave con el mismo valor en el rango especificado, y el rango está marcado para no admitir claves repetidas (con RANKSPEC). Se ha establecido la posición actual en la primera clave del rango que coincide con el valor dado. No se ha aplicado bloqueo.
130	Fracaso	El fichero está bloqueado en lectura por otro usuario.
131	Fracaso	No se ha podido bloquear en escritura el registro que se iba a crear.

Notas

El contenido actual del tampón de registros del fichero se escribe en el nuevo registro; Jetsam suministra el número de registro.

Si la operación concluye con éxito, la posición actual se identifica con la clave recién insertada; de lo contrario, no se modifica la posición actual, a menos que el fallo sea el 116.

Ésta es la única forma de añadir datos nuevos a un fichero de acceso por claves. En otros tipos de fichero esta operación se puede realizar con PUT, pero no en éstos. La razón es doble: primero, porque Jetsam tiene que suministrar el número de registro; segundo, porque Jetsam tiene que inicializar los dos primeros bytes del registro.

Se puede producir un error 131 en los sistemas operativos en los que al bloquear un registro se bloquea implícitamente algún otro (generalmente los adyacentes). Cuando Jetsam asigna un registro a los nuevos datos, intenta bloquearlo en escritura antes de escribir en él los datos; tal intento puede fallar.

Palabras clave asociadas

ADDKEY, DELKEY, PUT, RANKSPEC

Obtener el valor ASCII de un carácter. Función

Aplicación

Obtener el valor numérico del primer carácter de una cadena, suponiendo que los caracteres estén codificados según la norma ASCII.

Es la función inversa de CHR$.

Forma

ASC(*expresión-literal*)

donde la *expresión-literal* da una cadena que consta de al menos un carácter.

Palabras clave asociadas

CHR$

Arco tangente. Función

Aplicación

Calcular el arco tangente del número especificado. El resultado se da en radianes, en el margen de −PI/2 a +PI/2.

Forma

ATN(*expresión-numérica*)

Notas

La *expresión-numérica* puede ser de cualquier tipo, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado de ATN es también un número de precisión sencilla.

Palabras clave asociadas

SIN, COS, TAN

Numeración automática de líneas. Orden

No disponible en las versiones de 'sólo ejecución'

Aplicación

Para facilitar la introducción de líneas de programa por el teclado. BASIC genera los números de línea de forma automática.

Forma

AUTO [*número-de-línea*][,[*incremento*]]

Si se omite el *número-de-línea*, se supone el 10.

Si se omite el *incremento* y la coma, se supone un incremento de 10. Si se omite el *incremento* pero se mantiene la coma, se supone el incremento que se utilizó en la anterior orden AUTO (a menos que ésta sea la primera, pues en tal caso se supone un incremento de 10). Un *incremento* de valor 0 es «ilegal» (error 5).

Notas

Mientras está activa la orden AUTO, los números de línea se generan automáticamente cuando se está introduciendo el programa. Cada vez que AUTO genera un número de línea se entra en el editor de líneas (véase el capítulo 4). Si ya existe una línea con el nuevo número, aparece en la pantalla lista para ser editada. Si no existe ninguna línea con el nuevo número, se ofrece solamente el número de línea.

AUTO continúa en vigor hasta que se abandona la edición de una línea (en lugar de terminarla normalmente). Cuando se sale de AUTO, BASIC vuelve al modo directo.

Establecer el espacio de memoria utilizable por Jetsam. Orden de Jetsam

Aplicación

Controlar el espacio de memoria que Jetsam puede dedicar al «caché» de almacenamiento temporal de índices; en sistemas multiusuario, controlar el espacio dedicado a las listas de BASIC de bloqueos pendientes.

Forma

BUFFERS *número-de-tampones[,número-de-bloqueos]*

o

BUFFERS *,número-de-bloqueos*

El *número-de-tampones* es una *expresión-entera* que da el número de tampones que puede utilizar Jetsam.

El *número-de-bloqueos* es una *expresión-entera* que da el número máximo de bloqueos que se pueden establecer simultáneamente por cada fichero; su valor debe estar en el margen de 0 a 255. Este parámetro se ignora en los sistemas uniusuario.

Notas

En la primera forma de la orden, si se omite el parámetro *número-de-bloqueos*, no se modifica el número máximo de registros que pueden estar bloqueados en cada fichero. En la segunda forma, no se modifica el número de tampones que puede utilizar Jetsam.

Cuantos más tampones se dedique al «caché» de índices de Jetsam, más eficaz será el sistema. Se recomienda un mínimo de seis. Cada uno de estos tampones ocupa 128 bytes. El espacio dedicado a ellos se resta del disponible para el programa. El «caché» es compartido por todos los ficheros de acceso por claves que están abiertos en cada momento. Esta memoria queda reservada aunque no haya ningún fichero abierto.

Cuando se carga BASIC no se asigna ningún tampón a Jetsam. Por consiguiente, todo programa que utilice Jetsam tiene que contener al menos una orden BUFFERS.

El número de tampones dedicados al «caché» se puede modificar en cualquier momento, pero el cambio es más fácil si no hay ningún fichero abierto.

Cuando se carga BASIC, el número máximo de bloqueos de registro es cero. Así pues, para especificar cualquier bloqueo de registro es necesario ejecutar antes una orden BUFFERS.

Esto es válido también aun en el caso de los ficheros de acceso aleatorio, para los que no se requiere «caché». El sistema operativo puede imponer su propio límite sobre el número de bloqueos de registro pendientes.

Cada bloqueo de registro requiere 7 bytes de memoria por fichero. Este espacio se resta del disponible para el programa y queda reservado aunque no se esté utilizando ningún bloqueo.

Al cambiar el número máximo de bloqueos de registro se cierran implícitamente todos los ficheros, sin marcar como consistentes los ficheros de acceso por claves que se encuentren abiertos en ese momento. Por lo tanto, es imprescindible cerrar todos los ficheros de acceso por claves antes de realizar este cambio.

Palabras clave asociadas

CLEAR, CLOSE, OPEN, MEMORY

Llamar una subrutina externa. **Orden**

Aplicación

Invocar desde BASIC subrutinas externas.

Forma

CALL *variable-sencilla[(lista-de-argumentos)]*

El valor de la *variable-sencilla* da la dirección de la subrutina. La *variable-sencilla* tiene que haber sido igualada antes a una *expresión-de-dirección*.

La *lista-de-argumentos* es una *lista-de: variable*. Las direcciones de las variables mencionadas en la lista de argumentos son suministradas a la subrutina.

Notas

Véase el apéndice III, donde se explica detalladamente el mecanismo de llamada, los formatos de las variables, etc.

En Mallard-86, la dirección dada es la componente de desplazamiento de la dirección completa de la subrutina. La componente de segmento viene dada por la base de segmentos de usuario más recientemente establecida por DEF SEG.

Palabras clave asociadas

DEF USR, USR, UNT, DEF SEG

Convertir a doble precisión. Función

Aplicación

Toma una expresión dada y entrega su valor en forma de número de doble precisión.

Forma

CDBL(*expresión-numérica*)

Notas

Muchas fracciones decimales no tienen una representación exacta en binario en punto flotante. Esta imprecisión se pone de manifiesto cuando se convierte a doble precisión un número que está en punto flotante en precisión sencilla. Por ejemplo,

```
100 DATA 0.1
110 READ A!
120 PRINT A!,CDBL(A!)
```

produce el siguiente resultado:

```
.1.          1000000014901161
```

Palabras clave asociadas

CSNG, CINT, CVD, MKD$.

Cargar y ejecutar un nuevo programa. Orden

Aplicación

Permite que un programa cargue y ponga en marcha otro programa, con la opción de conservar el programa actual y las variables, en todo o en parte.

Forma

CHAIN *expresión-literal[,expresión-de-número-de-línea]*

o

CHAIN *expresión-literal,[expresión-de-número-de-línea]*,ALL

CHAIN MERGE *expresión-literal[,[expresión-de-número-de-línea]*

o

CHAIN MERGE *expresión-literal,[expresión-número-de-línea]*,ALL

o

CHAIN MERGE *expresión-literal,[expresión-de-número-de-línea][*,ALL*]*,DELETE *margen-de-números-de-línea*

donde:

La *expresión-literal* da el nombre del fichero que contiene el programa requerido. Si no se especifica distintivo de tipo, se supone ".BAS".

En la *expresión-literal* se puede incluir la especificación de la unidad de disco (por ejemplo, B:) como prefijo.

La *expresión-de-número-de-línea* tiene la forma de una *expresión-de-dirección* y da el número de línea a partir del cual se debe iniciar la ejecución cuando se termine de cargar el nuevo programa. Este número se interpreta de la siguiente manera:

0 o ninguno	la ejecución empieza en la línea de número más bajo.
1 a 65534	la ejecución empieza en la línea indicada; si no existe esa línea, se produce el error 8.
65535	se retorna al sistema operativo.

La cláusula ALL hace que se conserven los valores de todas las variables del programa actual. Si se omite ALL, no se conserva ninguna variable, a menos que se haya especificado otra cosa en el programa actual en alguna sentencia COMMON.

DELETE hace que se borren las líneas especificadas por *margen-de-números-de-línea* del programa actual antes de mezclar el nuevo. (DELETE se puede especificar en la orden CHAIN, pero es una redundancia.)

Notas

La orden RENUM renumera correctamente los números del *margen-de-números-de-línea* que acompaña a la cláusula DELETE, pero no puede afectar a la *expresión-de-número-de-línea*. En CHAIN, esta expresión hace referencia a una línea de un programa distinto del actual, y por lo tanto RENUM no puede actuar sobre ella. En CHAIN MERGE, BASIC no sabe si la *expresión-de-número-de-línea* se refiere a una línea que no va a ser afectada por la mezcla del nuevo programa; por consistencia con CHAIN, no la modifica.

Las diferencias entre CHAIN y CHAIN MERGE son:

CHAIN	CHAIN MERGE
Borra todas las líneas del programa actual	Borra sólo las líneas especificadas en la cláusula DELETE (si la hay)
Reinicializa todas las caracteristicas establecidas con DEFINT, DEFSNG, DEFDBL y DEFSTR (Esto obliga a que todas las variables comunes tengan su propio marcador de tipo, o bien a que el nuevo programa emita nuevas órdenes DEF)	Conserva todas las características establecidas con DEFINT, DEFSNG, DEFDBL y DEFSTR
Reinicializa OPTION BASE, a menos que se entregue alguna matriz al nuevo programa	Conserva la situación de OPTION BASE

Acciones comunes a CHAIN y CHAIN MERGE son las siguientes:

— Se olvidan todas las funciones definidas por el usuario.
— Se cancela la intercepción de errores establecida con ON ERROR GOTO.
— Se conservan todos los ficheros que estén abiertos.
— Se ejecuta implícitamente una orden RESTORE.
— Se olvidan todas las órdenes FOR, WHILE y GOSUB que estén activas.

En el caso de CHAIN MERGE, las líneas del nuevo programa sustituyen a las que pudiera haber en el programa actual con el mismo número de línea.

Si se mezcla con CHAIN MERGE un programa protegido con uno desprotegido, el programa resultante estará protegido.

Palabras clave asociadas

COMMON, LOAD, SAVE, RUN, MERGE, COMMON RESET

Convertir a carácter. Función

Aplicación

Convertir un valor numérico en su carácter equivalente, suponiendo que los caracteres estén codificados según la norma ASCII.

Es la función inversa de ASC.

Forma

CHR$(*expresión-entera*)

donde *expresión-entera* debe dar un valor perteneciente al margen de 0 a 255.

Palabras clave asociadas

ASC

Convertir a entero. Función

Aplicación

Convertir el valor dado a representación entera, si es posible.

Forma

CINT(*expresión-numérica*)
La *expresión-numérica* se redondea a un entero, que tiene que quedar en el margen de −32768 a +32767.

Palabras clave asociadas

CSNG, CDBL, MKI$, CVI, INT, FIX, ROUND, UNT

Borrar todas las variables y ficheros. Orden

Aplicación

Igualar todas las variables numéricas a cero y todas las literales a la cadena vacía. Borrar todas las matrices. Olvidar todas las funciones de usuario. Cerrar todos los ficheros. Esta orden sirve también para especificar el tamaño de la pila y el espacio de memoria utilizable por BASIC, así como el número máximo de ficheros y el tamaño máximo de los registros aleatorios.

Forma

CLEAR
CLEAR,*memoria-alta*
CLEAR,*[memoria-alta],tamaño-pila*
CLEAR,*[memoria-alta],[tamaño-pila],número-de-ficheros*
CLEAR,*[memoria-alta],[tamaño-pila],[número-de-ficheros],longitud-de-registro-máxima*

memoria-alta es una *expresión-de-dirección* que da la dirección del byte más alto de la zona de memoria que puede ser utilizada por BASIC. Si se omite este parámetro, se mantiene la situación actual.

tamaño-de-pila es una *expresión-de-dirección* que da el número de bytes que puede ocupar la pila de BASIC. Si se omite este parámetro, se mantiene la situación actual. Si se lo incluye, la expresión tiene que dar un valor de, como mínimo, 256.

número-de-ficheros es una *expresión-de-dirección* que da el número máximo de ficheros que pueden ser manejados simultáneamente. Si se omite este parámetro, se mantiene la situación actual.

longitud-de-registro-máxima es una *expresión-de-dirección* que especifica el tamaño máximo de los registros. Si se omite este parámetro, se mantiene la situación actual.

Notas

Sólo es necesario especificar valores para las características que se desea cambiar. Si se incluye alguno, hay que poner una coma antes del primero.

Los parámetros opcionales permiten establecer el número máximo de ficheros, el tamaño máximo de los registros y la memoria total de forma muy similar a como se hace en la orden del sistema operativo con la que se carga BASIC (véase el apéndice I).

CLEAR hace que BASIC olvide las órdenes FOR, WHILE y GOSUB que puedan estar activas.

La pila de BASIC se utiliza durante la evaluación de expresiones y para almacenar la información necesaria para los bucles FOR y WHILE y para los retornos de las subrutinas. El tamaño implícito es 512 bytes, que debe de ser adecuado para la mayor parte de los programas, salvo los que tengan un grado extremo de anidamiento de bucles o muchos GOSUB.

Palabras clave asociadas

MEMORY, HIMEM, FRE, OPEN, COMMON RESET

CLOSE

Cerrar uno o varios ficheros. **Orden**

Ampliada en las versiones Jetsam

Aplicación

Para concluir la utilización de los ficheros.

Forma

CLOSE *[lista-de: referencia-de-fichero]*

Si se omite la lista, se cierran todos los ficheros.

Cada *referencia-de-fichero* da el número de uno de los ficheros que van a ser cerrados.

Notas

Cuando se cierra un fichero secuencial que estaba abierto en dirección de salida se graban en el fichero todos los datos pendientes.

Después de ejecutar CLOSE, los números de fichero afectados dejan de estar asociados a los ficheros y se los puede volver a utilizar.

Si en CLOSE se menciona un número de fichero que no tenga ningún fichero asignado, la orden ignora ese número.

Ampliaciones para Jetsam

En el caso de los ficheros de acceso por claves de Jetsam, CLOSE cierra los ficheros de índices y los de datos.

En los sistemas multiusuario, CLOSE libera los bloqueos de registro y de fichero que pueda haber activos en ficheros de cualquier tipo.

Esta operación es muy importante, pues garantiza que se escriban todos los índices y registros de datos en el disco y que se marquen como consistentes los ficheros de índices y de datos.

Nota. Los ficheros de acceso por claves que no se cierren explícitamente con una orden CLOSE no serán marcados como consistentes. Las órdenes siguientes dejan los ficheros marcados como inconsistentes, a pesar de cerrarlos implícitamente:

BUFFERS (cuando se cambia el número máximo de bloqueos de registro)
CLOSE (sin parámetros)
SYSTEM
RUN (cuando se reinicia un programa)

Palabras clave asociadas

OPEN, RESET, CONSOLIDATE (Jetsam), CREATE (Jetsam)

Declarar las variables que han de ser conservadas cuando se ejecute una orden CHAIN o COMMON RESET. Orden

Aplicación

La sentencia COMMON permite especificar qué variables se han de retener cuando se ejecuten órdenes CHAIN, CHAIN MERGE y COMMON RESET.

Forma

COMMON *lista-de: referencia-de-variable*

donde *referencia-de-variable* es *nombre-de-variable*

o bien *nombre-de-variable-matricial*

Notas

Cuando se ejecuta una orden CHAIN, CHAIN MERGE o COMMON RESET, BASIC busca y obedece todas las órdenes COMMON que haya en el programa (antes de borrar ninguna línea); mientras tanto, las órdenes COMMON no tienen ningún efecto.

Palabras clave asociadas

CHAIN, CHAIN MERGE, COMMON RESET

Borrar todas las variables que no estén declaradas en alguna sentencia COMMON.

Orden

Aplicación

COMMON RESET es similar a CLEAR, pero no borra las variables que figuren en una sentencia COMMON en algún lugar del programa actual.

Forma

COMMON RESET

Notas

Cuando se ejecuta una orden COMMON RESET, se borran todas las variables que no estén mencionadas en alguna sentencia COMMON del programa actual.

COMMON RESET hace que BASIC olvide todas las órdenes FOR, WHILE y GOSUB que estén activas.

Palabras clave asociadas

CHAIN, CHAIN MERGE, CLEAR, COMMON

Marcar como consistente un fichero de acceso por claves. **Función de Jetsam**

Aplicación

En cuanto se modifica un fichero de acceso por claves, Jetsam lo marca como inconsistente. La función CONSOLIDATE hace que se escriba en el disco toda la información pendiente y que el fichero quede marcado como consistente.

Forma

CONSOLIDATE(*referencia-de-fichero*)

La *referencia-de-fichero* tiene que dar el número de un fichero que esté abierto.

Notas

CONSOLIDATE es una función que genera un valor entero:

En los sistemas uniusuario el valor es 0.

En los sistemas multiusuario el valor es el número de usuarios (distintos del actual) que han marcado el fichero como inconsistente.

El efecto de CONSOLIDATE es similar al de cerrar (CLOSE) y abrir (OPEN) inmediatamente el fichero, pero con la diferencia de que CONSOLIDATE preserva los bloqueos que estén en vigor.

Palabras clave asociadas

CLOSE, OPEN

Continuar después de 'break', STOP o END. Orden

No disponible en las versiones de 'sólo ejecución'

Aplicación

Cuando un programa ha sido detenido por control-C ('break') o por una orden STOP o END, la ejecución se puede reanudar con la orden CONT.

Forma

CONT

Notas

CONT es imposible si el programa ha sido modificado después de la interrupción.

Coseno. Función

Aplicación

Calcular el coseno de un ángulo dado en radianes.

Forma

COS(*expresión-numérica*)

La *expresión-numérica* da el ángulo, expresado en radianes; su valor debe pertenecer al margen de (aprox.) −200000 a +200000.

Notas

Cuando los ángulos son mucho mayores que 2*PI, la precisión de esta función se deteriora como consecuencia de la necesaria transformación del ángulo para que quede en el margen de −PI a +PI. En vez de dar un resultado inexacto, BASIC rehusa calcular el coseno cuando el argumento está fuera del margen antes citado.

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado es también un número de precisión sencilla.

Palabras clave asociadas

SIN, TAN, ATN

Crear un fichero de acceso por claves. Orden de Jetsam

Aplicación

Para crear un fichero de acceso por claves. La operación consiste en crear el fichero de índices y el fichero de datos, inicializarlos y abrirlos.

Forma

CREATE *referencia-de-fichero,nombre-de-fichero-de-datos,nombre-de-fichero-de-índices, bloqueo[,longitud-de-registro]*

CREATE *referencia-de-fichero,nombre-de-fichero-de-datos,nombre-de-fichero-de-índices, bloqueo,[longitud-de-registro],cadena-de-usuario*

La referencia-de-fichero da el número de fichero con el que se debe abrir el fichero de acceso por claves después de crearlo.

El *nombre-de-fichero-de-datos* es una *expresión-de-nombre-de-fichero* que especifica el nombre que se ha de dar al fichero de datos.

El *nombre-de-fichero-de-índices* es una *expresión-de-nombre-de-fichero* que especifica el nombre que se ha de dar al fichero de índices.

El *bloqueo* especificado es el que se aplica al fichero de acceso por claves en el momento de abrirlo.

La *longitud-de-registro* es una *expresión-entera* que especifica la longitud de los registros del fichero de datos. Su valor tiene que estar en el margen de 1 a *máx*, donde *máx* es el límite máximo establecido en las órdenes MEMORY o CLEAR, o en la orden de carga de BASIC desde el sistema operativo (valor implícito: 128). Si no se especifica este parámetro, se supone el valor 128. Los dos primeros bytes de cada registro están reservados para Jetsam.

La *cadena-de-usuario* es una *expresión-literal* cuyo valor se graba en el fichero de índices. Puede tener cualquier longitud, pero sólo se graban los primeros 64 bytes. Para Jetsam esta cadena no tiene ningún significado, pero puede tenerlo para el programa.

Notas

Los ficheros se abren y bloquean inmediatamente después de su creación. La posición actual no queda establecida. Se produce un error (error 58) si ya existe alguno de los dos ficheros (de datos o de índices). Si la creación tiene éxito en uno de los dos ficheros, la orden CREATE trata de borrar el único fichero que ha creado. Un fichero sin el otro no es más que una pérdica de espacio en el disco.

Cuando se crea un fichero, en todos los rangos queda permitida la repetición de claves. La orden RANKSPEC sirve para marcar rangos concretos de forma que rechacen todo intento de añadir claves nuevas cuyo valor sea igual al de alguna clave ya existente en el rango.

Palabras clave asociadas

OPEN, CLOSE, RANKSPEC

Convertir un valor a precisión sencilla. Función

Aplicación

Toma una expresión dada y entrega su valor en forma de número de precisión sencilla.

Forma

CSNG(*expresión-numérica*)

Palabras clave asociadas

CINT, CDBL, MKS$, CVS

Convertir una cadena a forma numérica en doble precisión. Función

Aplicación

Reconvertir una cadena producida por MKD$ para recuperar el valor original en doble precisión.

Forma

CVD(*expresión-literal*)

donde la *expresión-literal* tiene que dar una cadena de 8 bytes.

Notas

MKD$ crea una cadena de 8 bytes que es una copia idéntica de la representación interna del valor en doble precisión y en cuya obtención no interviene ninguna conversión de binario a decimal. La cadena así formada se puede usar, por ejemplo, para escribir valores de doble precisión en registros aleatorios.

Palabras clave asociadas

MKD$, CVI, CVS

Convertir una cadena a forma entera. Función

Aplicación

Reconvertir una cadena producida por MKI$ para recuperar el valor original entero.

Forma

CVI(*expresión-literal*)

donde la *expresión-literal* tiene que dar una cadena de 2 bytes.

Notas

MKI$ crea una cadena de 2 bytes que es una copia idéntica de la representación interna del valor entero y en cuya obtención no interviene ninguna conversión de binario a decimal. La cadena así formada se puede usar, por ejemplo, para escribir valores enteros en registros aleatorios.

Palabras clave asociadas

MKI$, CVS, CVD

Convertir una cadena a forma entera. Función

Aplicación

Reconvertir una cadena producida por MKIK$ para recuperar el valor original entero.

Forma

CVIK(*expresión-literal*)

donde la *expresión-literal* tiene que dar una cadena de 2 bytes.

Notas

MKIK$ crea una cadena de 2 bytes que es una copia idéntica de la representación interna del valor entero y en cuya obtención no interviene ninguna conversión de binario a decimal. La cadena así formada se puede usar como clave para Jetsam.

Palabras clave asociadas

MKIK$, MKUK$, CVUK

Convertir una cadena a forma numérica en precisión sencilla. Función

Aplicación

Reconvertir una cadena producida por MKS$ para recuperar el valor original en precisión sencilla.

Forma

CVS(*expresión-literal*)

donde la *expresión-literal* tiene que dar una cadena de 4 bytes.

Notas

MKS$ crea una cadena de 4 bytes que es una copia idéntica de la representación interna del valor en precisión sencilla y en cuya obtención no interviene ninguna conversión de binario a decimal. La cadena así formada se puede usar, por ejemplo, para escribir valores de precisión sencilla en registros aleatorios.

Palabras clave asociadas

MKS$, CVI, CVD

Convertir una cadena a forma de entero sin signo. Función

Aplicación

Reconvertir una cadena producida por MKUK$ para recuperar el valor original entero (sin signo)

Forma

CVUK(*expresión-literal*)

donde la *expresión-literal* tiene que dar una cadena de 2 bytes.

Notas

MKUK$ crea una cadena de 2 bytes que es una copia idéntica de la representación interna del valor entero y en cuya obtención no interviene ninguna conversión de binario a decimal. La cadena así formada se puede usar como clave para Jetsam.

Palabras clave asociadas

MKUK$, MKIK$, CVIK

Declarar datos constantes. Orden

Forma

DATA *lista-de: constante*

constante puede ser cualquier forma de *constante-numérica*, una *cadena-entrecomillada* o una *cadena-sin-comillas*.

Una *constante-numérica* puede tener un *espacio* o varios a cada lado; tales espacios son ignorados.

Una *cadena-entrecomillada* puede tener un *espacio* o varios a cada lado; tales espacios son ignorados. Se puede omitir las últimas comillas si la cadena es el último objeto de la lista.

Una *cadena-sin-comillas* es un grupo de entre 0 y 255 caracteres que se considera como cadena. En este contexto, la *cadena-sin-comillas* termina en una coma, un signo de dos puntos o un retorno del carro. La *cadena-sin-comillas* puede tener un *espacio* o varios a cada lado; tales espacios son ignorados.

Notas

Todas las sentencias DATA del programa, consideradas en orden de números de línea ascendentes, forman una lista de constantes que pueden ser asignadas a variables mediante la orden READ. Cada vez que se lee un dato, el «puntero» avanza para señalar el dato siguiente. La orden RESTORE hace que el puntero señale el primer elemento de una línea especificada.

Las órdenes DATA no tienen efecto inmediato. BASIC no comprueba la validez de las constantes mientras no las lee (con READ). Las constantes numéricas sólo pueden ser asignadas a variables numéricas. Las literales, de cualquier tipo, sólo pueden ser asignadas a variables literales. De ser necesario, las constantes numéricas son tratadas como *cadenas-sin-comillas*.

Palabras clave asociadas

READ, RESTORE

Generar una representación de un número en forma de cadena. decimal de formato especificado. Función

Aplicación

Crear una representación en forma de cadena del valor de una expresión dada, de acuerdo con una plantilla de formato.

Forma

DEC$(*expresión-numérica,plantilla-de-formato*)

El valor de la *expresión-numérica* se convierte en una cadena decimal de formato especificado por *plantilla-de-formato*, que es una *expresión-literal* que se interpreta de la misma forma que en PRINT USING.

Notas

La plantilla de formato no puede contener todos los caracteres de las plantillas que se utilizan con PRINT USING, sino solamente los siguientes:

```
+  -  $  *  #  ,  .  ↑
```

Véase en el apartado 6.5.3. la descripción de las especificaciones de formato que son adecuadas para los números.

Palabras clave asociadas

STR$, HEX$, OCT$, PRINT USING

Definir función de usuario.

Orden

Ilegal en modo directo

Aplicación

BASIC permite que el programa defina y utilice funciones que generen valores numéricos o literales. DEF FN realiza la primera parte de este mecanismo: la definición de la función.

Forma

DEF FN*nombre[(parámetros-formales)]* = *expresión-general*

El nombre tiene la forma de un *nombre-de-variable* (posiblemente incluyendo un *marcador-de-tipo*; el nombre de la función es FN*nombre*. El valor generado por la función es del mismo tipo que el *nombre*.

Los *parámetros-formales* tienen la forma de *lista-de: nombre-de-variable-sencilla*.

La *expresión-general* puede contener, no sólo los parámetros formales, sino también otras variables o funciones. El resultado de esta expresión tiene que ser compatible con el tipo de la función.

Notas

La función de usuario se invoca cuando se la menciona como argumeto de una expresión. La invocación tiene la forma:

FN*nombre[(parámetros-concretos)]*

Los *parámetros-concretos* tienen la forma de *lista-de: expresión-general*. Tiene que haber tantos parámetros concretos en la invocación como parámetros formales en la definición. El tipo de cada parámetro concreto tiene que ser compatible con el del correspondiente parámetro formal.

Los parámetros formales son locales a la función. Las variables globales de igual nombre que los parámetros formales no resultan afectadas por la invocación de la función, ni tampoco son accesibles desde dentro de la función.

Si no se indica explícitamente el tipo de la función, se supone el tipo implícito vigente en el momento de ejecutar DEF FN. Análogamente ocurre con el tipo de cada uno de los parámetros formales.

DEF SEG

Definir la dirección de la base del segmento. **Orden**

Aplicación

En Mallard-86, esta orden establece la base del segmento que se va a utilizar en PEEK, POKE, USR, CALL, OPTION INPUT, OPTION LPRINT y OPTION PRINT.

Mallard-80 ignora esta orden.

Forma

DEF SEG *[=expresión-de-dirección]*

Notas

El valor implícito de la base del segmento de usuario es la base del segmento de datos de BASIC (en el cual residen el programa y los datos).

DEF SEG establece la base del segmento de usuario en el valor dado. Si se omite el argumento, se restablece el valor implícito.

Las direcciones que utilizan las órdenes PEEK y POKE están formadas por dos componentes: el desplazamiento especificado en la orden y la base del segmento de usuario establecida en la orden DEF SEG más reciente.

Las direcciones en que se encuentran las funciones USR y las subrutinas invocadas con CALL constan de dos componentes: el desplazamiento especificado en la orden y la base del segmento de usuario establecida en la orden DEF SEG más reciente.

Las órdenes OPTION INPUT, OPTION LPRINT y OPTION PRINT tienen un parámetro de dirección que especifica la dirección de una subrutina en código de máquina. La componente de base del segmento de la dirección completa de la subrutina es la base del segmento de usuario que esté en vigor en el momento de ejecutarse la orden.

Palabras clave asociadas

CALL, USR, DEF USR, OPTION INPUT, OPTION LPRINT, OPTION PRINT, PEEK, POKE

DEF USR

Definir función externa. Orden

Aplicación

Proporciona una alternativa a CALL como mecanismo para invocar subrutinas externas. Este mecanismo sólo admite un parámetro, pero permite que la función externa genere un valor anónimo.

Forma

DEF USR*[dígito]=expresión-de-dirección*

Se puede definir hasta diez funciones distintas, de la USR0 a la USR9. Si se omite el dígito, se supone el 0.

La *expresión-de-dirección* da la dirección de la función externa.

Notas

La función externa se invoca cuando se la menciona como argumento en una expresión. La invocación tiene la siguiente forma:

USR*[dígito](expresión-general)*

donde *dígito* puede ser cualquiera del 0 al 9; si se lo omite, se supone el 0. Sólo interviene un parámetro, y es obligatorio.

Las funciones externas pueden ser redefinidas.

En el apéndice III se dan más detalles sobre las rutinas externas y sus limitaciones y se hacen advertencias sobre su utilización.

En Mallard-86, la dirección especificada es la componente de desplazamiento de la dirección completa. En el momento de invocarse la función, la componente de base del segmento viene dada por la base del segmento de usuario que se estableció en la orden DEF SEG más reciente.

Palabras clave asociadas

CALL, VARPTR, DEF SEG

DEFINT DEFSNG DEFDBL DEFSTR

Establecer el tipo implícito para los nombres. **Orden**

Aplicación

Todos los nombres de variables y funciones tienen asociado un tipo. Este tipo puede estar indicado explícitamente en el nombre, o ser el implícito establecido por estas órdenes.

Forma

DEFINT *lista-de: margen-de-letras*
DEFSNG *lista-de: margen-de-letras*
DEFDBL *lista-de: margen-de-letras*
DEFSTR *lista-de: margen-de-letras*

donde *margen-de-letras* es *letra*
o bien *letra-letra*

La forma *letra-letra* define un margen de letras en el que se incluyen las dos mencionadas.

Notas

Cuando BASIC encuentra un nombre de variable sin *marcador-de-tipo*, supone que es del tipo implícito actual. El tipo implícito depende del primer carácter del nombre. El tipo implícito para cada letra es el establecido en la última de estas órdenes en cuya *lista-de-márgenes-de-letras* está la letra en cuestión.

DEFINT hace que el tipo implícito sea 'entero'.
DEFSNG hace que el tipo implícito sea 'precisión sencilla'.
DEFDBL hace que el tipo implícito sea 'doble precisión'.
DEFSTR hace que el tipo implícito sea 'cadena'

El estado inicial del tipo implícito es, para todas las letras, 'precisión sencilla'. A este estado se vuelve con:

LOAD, RUN, CHAIN, NEW y CLEAR

Borrar ficheros. **Orden de MS-DOS (PC-DOS)**

Aplicación

Borrar un fichero o una lista de ficheros.

Forma

DEL *lista-de: nombre-de-fichero*

Notas

Esta orden sólo está incluida en las versiones de BASIC para sistemas operativos de tipo MS-DOS (PC-DOS).

La orden borra los ficheros especificados. Si el nombre del fichero contiene «símbolos comodín», se borran todos los ficheros concordantes.

DEL considera que todo el resto de la línea es su argumento, a pesar de los posibles signos de dos puntos (:) o apóstrofos (').

No se recomienda borrar un fichero que esté abierto. En los sistemas multiusuario esta orden puede fallar cuando el fichero está siendo utilizado por otros usuarios o está bloqueado.

Palabras clave asociadas

ERA, KILL

Borrar líneas de programa. Orden

Aplicación

Borrar parte del programa actual.

Forma

DELETE *margen-de-números-de-línea[,número-de-línea]*

Notas

Borra todas las líneas del margen especificado; además tiene los siguientes efectos secundarios:

— Se olvidan todas las funciones de usuario.
— Se desactiva ON ERROR GOTO.
— Se ejecuta implícitamente una orden RESTORE.
— Se olvidan todas las órdenes FOR, WHILE y GOSUB.

Si se omite el parámetro *número-de-línea*, BASIC retorna al modo directo en cuanto termina de ejecutar DELETE. De lo contrario, BASIC inicia la ejecución en la línea especificada; si *número-de-línea* es cero, la ejecución empieza por la línea de número más bajo del programa.

Palabras clave asociadas

NEW

DELKEY

Borrar una clave. **Función de Jetsam**

Aplicación

Borrar una clave del fichero de índices. Si la clave es la única que hace referencia al correspondiente registro de datos, éste se borra también automáticamente. La posición actual se establece en la clave siguiente a la borrada.

Forma

DELKEY(*referencia-de-fichero,bloqueo[,posición-en-índice]*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves que está abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al registro al que hace referencia la clave inmediatamente posterior a la que se va a borrar.

La *posición-en-índice* especifica qué clave se ha de borrar. Si se omite este parámetro, se supone la posición actual.

Códigos generados

0	Éxito	La nueva posición actual está en el mismo conjunto de claves que la borrada (o sea, tiene el mismo valor que la clave borrada).
101	Éxito	La nueva posición actual no está en el mismo conjunto de claves que la borrada, pero sí en el mismo rango (o sea, tiene un valor distinto del de la clave borrada).
102	Éxito	La nueva posición actual está en el siguiente rango que existe.
103	Éxito	La clave borrada era la última del índice; la posición actual queda indeterminada.
105	Fracaso	No se ha podido encontrar la posición actual o la especificada. La posición actual se establece en la primera clave del conjunto que sería inmediatamente posterior a la clave especificada si ésta estuviera en el índice.
115	Fracaso	Se ha omitido el parámetro *posición-en-índice* y la posición actual queda indeterminada.
131	Fracaso	No se ha podido bloquear en escritura el registro correspondiente a la clave que iba a ser borrada.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro siguiente.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro siguiente.

Notas

Si el registro al que hace referencia la clave que se va a borrar no está bloqueado en escritura, DELKEY automáticamente establece un bloqueo temporal en escritura. Si no consigue establecer este bloqueo, genera el código 131.

DELKEY se puede utilizar en conjunción con ADDKEY para cambiar el valor de una clave. Primero se añade la versión nueva (ADDKEY) y después se suprime la antigua (DELKEY). El orden es muy importante. Si se procediese de otra forma, esto es, aplicando primero DELKEY, el registro de datos se perdería si la clave borrada fuese la única asociada al registro.

Para borrar un registro hay que borrar todas sus claves. Una vez hecho esto, el registro ya no será accesible; el espacio que ocupaba en el disco será reutilizado cuando se cree un registro nuevo (con ADDREC).

Palabras clave asociadas

ADDKEY, ADDREC

Declarar dimensiones de matrices. Orden

Aplicación

Para poder reservar espacio para una matriz, BASIC necesita saber cuántas dimensiones tiene y cuál es el valor máximo que puede tener el índice en cada una de ellas. La orden DIM establece explícitamente ambos datos.

Forma

DIM *lista-de: variable-indexada*

donde *variable-indexada* es *nombre-de-variable(lista-de-dimensiones)*
y *lista-de-dimensiones* es *lista-de: expresión-entera*

Notas

Cada *expresión-entera* de la *lista-de-dimensiones* especifica el máximo valor permitido para el correspondiente índice. El valor mínimo es 0 o 1, dependiendo de lo que se haya establecido con la orden OPTION BASE (en ausencia de especificación, el 0).

Las dimensiones de las matrices se pueden establecer explícitamente con una orden DIM, o implícitamente cuando BASIC encuentra la matriz por primera vez. En las matrices declaradas implícitamente el valor máximo de todos los índices es 10.

Se provoca un error cuando se intenta cambiar las dimensiones de una matriz que ya ha sido declarada (explícita o implícitamente).

No hay límite en cuanto al número de dimensiones que puede tener una matriz, aparte de la longitud de línea y la memoria disponible.

Palabras clave asociadas

OPTION BASE

Listado del directorio. Orden

Aplicación

Escribir un listado del directorio en la consola. Puede ser el listado total o un listado parcial que dependerá de los nombres de fichero que se especifiquen, los cuales pueden contener «símbolos comodín».

Forma

DIR *[lista-de: nombre-de-fichero]*

Notas

Si se omite la lista de nombres de fichero, el listado que se obtiene es el listado de todos los ficheros que hay en la unidad implícita. Si se incluyen varios nombres, la orden los va procesando uno a uno y listando los nombres de los ficheros concordantes. Se puede utilizar los «símbolos comodín» ? y * de la forma habitual.

DIR considera que todo el resto de la línea es su argumento, a pesar de los posibles signos de dos puntos (:) o apóstrofos (').

Palabras clave asociadas

FILES, FIND$

Mostrar en la consola el contenido de un fichero. Orden

Aplicación

Escribir en la consola el contenido del fichero especificado.

Forma

DISPLAY *expresión-literal*

La *expresión-literal* da una cadena que es el nombre del fichero que se ha de enviar a la consola. El nombre no puede contener «símbolos comodín».

Notas

Esta orden lee el fichero y lo envía directamente a la consola. Pulsando control-C se abandona la operación y se vuelve al modo directo (salvo en las versiones de 'sólo ejecución', en las que control-C se ignora).

Cuando el fichero está siendo enviado a la consola, pulsando control-S se suspende la salida (salvo en las versiones de 'sólo ejecución', en las que control-C se ignora). La salida se puede reanudar de la forma habitual (véase la sección 3.3).

Los tabuladores y retornos del carro se procesan como de costumbre; se respeta la anchura lógica de la consola. Todos los demás caracteres se envían a la consola sin modificación, tal como figuran en el fichero. Por consiguiente, en el fichero se puede incluir caracteres de control y caracteres ajenos al margen normal ASCII para producir efectos especiales en la consola.

Palabras clave asociadas

TYPE

Editar una línea de programa. **Orden**

No disponible en las versiones de 'sólo ejecución'

Aplicación

Corregir líneas individuales del programa.

Forma

EDIT *número-de-línea*

Notas

Invoca el editor de líneas y ofrece la línea especificada para su corrección.

Véase en el capítulo 4 la descripción del editor de líneas.

Fin del programa. Orden

Aplicación

Señala el final del programa y lo concluye ordenadamente.

Forma

END

Notas

Puede haber varias órdenes END en diversos lugares del programa; BASIC supone un END cuando acaba de obedecer la última línea del programa.

END cierra todos los ficheros y retorna al modo directo.

Palabras clave asociadas

STOP

Detección del final del fichero. Función

Aplicación

Para averiguar si se ha alcanzado el final del fichero dado. Genera el valor −1 ('verdadero') si se ha llegado al final del fichero, y 0 ('falso') en caso contrario.

Forma

EOF(*expresión-de-número-de-fichero*)

Notas

El fichero puede no estar abierto en dirección de salida y puede no ser un fichero de acceso por claves.

EOF es particularmente útil en el trabajo con ficheros secuenciales. Se puede utilizar un bucle WHILE NOT EOF para leer hasta el final del fichero.

También se puede utilizar EOF con los ficheros de acceso aleatorio, pero con cuidado. EOF no da un resultado válido si no se ha ejecutado una orden GET después de abrir (OPEN) el fichero. A continuación de un GET, EOF genera:

- 'falso' si ya se ha escrito alguna vez en el registro.
- 'verdadero' si el número del registro leído es mayor que los números de todos los registros en los que se ha escrito.
- 'verdadero' o 'falso' si nunca se ha escrito en el registro pero hay algún otro registro de número mayor sobre el que se ha escrito.

Así pues, EOF permite leer secuencialmente un fichero aleatorio en el que todos los registros, hasta el del número máximo, hayan sido escritos alguna vez. EOF genera el valor 'verdadero' cuando se ha leído el primer registro posterior al final del fichero. Sería adecuado un bucle WHILE NOT EOF precedido de un GET para el primer registro, y con un GET para el siguiente registro al final.

Palabras clave asociadas

LOC, LOF

Borrar ficheros. — Orden de CP/M

Aplicación

Borrar un fichero o una lista de ficheros.

Forma

ERA *lista-de: nombre-de-fichero*

Notas

Esta orden sólo está incluida en las versiones de BASIC para sistemas operativos de tipo CP/M.

La orden borra los ficheros especificados. Si el nombre del fichero contiene «símbolos comodín», se borran todos los ficheros concordantes.

DEL considera que todo el resto de la líneas es su argumento, a pesar de los posibles signos de dos puntos (:) o apóstrofos (').

No se recomienda borrar un fichero que esté abierto. En los sistemas multiusuario esta orden puede fallar cuando el fichero está siendo utilizado por otros usuarios o está bloqueado.

Palabras clave asociadas

KILL

ERASE

Borrar matrices. **Orden**

Aplicación

Cuando ya no se necesita una matriz, se la puede borrar con la orden ERASE para liberar el espacio de memoria que está ocupando.

Forma

ERASE *lista-de: nombre-de-variable*

donde cada *nombre-de-variable* es el nombre de una matriz (sin índice).

Notas

Se produce un error cuando se intenta borrar (con ERASE) una matriz que no ha sido declarada (ni explícita ni implícitamente).

Una vez borrada una matriz, es posible declararla de nuevo con otras dimensiones (o con las mismas).

Palabras clave asociadas

DIM

Número de línea con error. Función

Aplicación

Esta función puede ser utilizada en las subrutinas de gestión de errores para averiguar èl número de la línea que se estaba ejecutando cuando se detectó el error.

Notas

Esta función puede intervenir en expresiones en la forma habitual.

Advertencia

Si se incluye ERL en expresiones de comparación, es necesario ponerla al lado izquierdo de la expresión si se quiere que BASIC reconozca el lado derecho como número de línea cuando ejecuta una orden RENUM. Si se escribe la expresión al revés, **RENUM no encontrará el número de línea y no podrá modificarlo.**

Palabras clave asociadas

ON ERROR, ERR, ERROR, OSERR

Número de error. Función

Aplicación

Esta función puede ser utilizada en las subrutinas de gestión de errores para averiguar el número del último error detectado.

Notas

Esta función puede intervenir en expresiones en la forma habitual.

Palabras clave asociadas

ON ERROR, ERL, ERROR, OSERR

Provocar una acción de error. — Orden

Aplicación

Invocar una acción de error con un número de error dado.

Forma

ERROR *expresión-entera*

donde la *expresión-entera* tiene que tomar un valor del margen de 1 a 255.

Notas

Si el número de error es uno de los reconocidos por BASIC, la acción que provoca esta orden es la misma que se pondría en marcha automáticamente si BASIC hubiera detectado ese error.

El programa puede usar los números de error superiores a los reconocidos por BASIC para señalar sus propios errores. Los números del 200 en adelante no son utilizados por BASIC ni lo serán en versiones futuras.

Si BASIC intenta imprimir un mensaje de error para un número de error no reconocido, el mensaje es Unknown error ('error desconocido').

Palabras clave asociadas

ON ERROR, ERR, ERL

Exponencial.

Función

Aplicación

Calcular una potencia del número *e*. El número *e* es la base de los logaritmos neperianos o naturales, o sea, el número cuyo logaritmo neperiano es 1 (aprox. 2.7182818).

Forma

EXP(*expresión-numérica*)

Notas

EXP produce un error de sobrepasamiento cuando el argumento es mucho mayor que 88.

Si el argumento es mucho menor que −88.7, se produce subdesbordamiento y EXP da resultado cero.

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado de EXP es también un número de precisión sencilla.

Palabras clave asociadas

LOG

Captar el valor de la clave actual. Función de Jetsam

Aplicación

Generar el valor de la clave que está considerada como posición actual.

Forma

FETCHKEY$(*referencia-de-fichero*)

La *referencia-de-fichero* da el número de fichero de un fichero de acceso por claves (abierto).

Notas

Esta función da el valor de la última clave encontrada en el índice. Si no está bloqueado el registro asociado (y tampoco lo está el fichero), no hay ninguna garantía de que esa clave todavía exista.

Palabras clave asociadas

FETCHRANK, FETCHREC

FETCHRANK

Captar el rango de claves actual. Función de Jetsam

Aplicación

Dar el rango en el que se encuentra la clave que está considerada como posición actual.

Forma

FETCHRANK(*referencia-de-fichero*)

La *referencia-de-fichero* da el número de fichero de un fichero de acceso por claves (abierto).

Notas

Esta función da el rango en el que está la última clave encontrada en el índice. Si no está bloqueado el registro asociado (y tampoco lo está el fichero), no hay ninguna garantía de que esa clave todavía exista.

Palabras clave asociadas

FETCHKEY$, FETCHREC

Captar el número de registro actual. Función de Jetsam

Aplicación

Dar el número del registro al que hace referencia la clave que está considerada como posición actual.

Forma

FETCHREC(*referencia-de-fichero*)

La *referencia-de-fichero* da el número de fichero de un fichero de acceso por claves (abierto).

Notas

Si está definida la posición actual, esta función da el número de registro correspondiente a la clave. Si no está bloqueado el registro (y tampoco lo está el fichero), no hay ninguna garantía de que esa clave todavía exista.

Palabras clave asociadas

FETCHRANK, FETCHKEY$

Definir una plantilla para los datos de un fichero. Orden

Aplicación

La orden FIELD define una plantilla que distribuye en campos los datos de los registros de los ficheros de acceso aleatorio y asocia una variable literal a cada campo.

Forma

FIELD *referencia-de-fichero,lista-de: campo*

donde:

La *referencia-de-fichero* da el número del fichero para el que será válida esta plantilla. El fichero tiene que estar abierto para acceso aleatorio.

campo es *tamaño-de-campo* AS *variable-literal*

donde *tamaño-de-campo* es una *expresión-entera* cuyo valor está en el margen de 0 a 255. La suma de todos los tamaños de campo tiene que ser igual o menor que el tamaño del registro.

Notas

Cada campo del registro se define con una longitud determinada y con una variable de campo asociada a él. Los campos se proyectan sobre el registro: el primer campo abarca los primeros bytes; el segundo, los inmediatamente siguientes, etc.

Las variables literales son punteros hacia el tampón de registros. Al ejecutar una orden FIELD no se lee ni escribe ningún dato; solamente se dispone las variables literales «apuntando» hacia el contenido actual del tampón.

Cada orden FIELD que se ejecuta para un determinado fichero establece una plantilla distinta para el tampón de registros. No hay límite en cuanto al número de plantillas que pueden estar asignadas a un fichero; todas son válidas.

El número de bytes disponibles para la orden FIELD en los ficheros de acceso por claves es dos menos que la longitud del registro. Estos bytes están reservados para Jetsam y no son accesibles para el usuario ni pueden ser asignados a ningún campo.

Advertencia

A las variables literales asociadas con los campos no se les puede dar valor más que mediante LSET, RSET y MID$. Cualquier otro método de asignación no producirá interacción con el tampón de registros, sino que creará una variable literal ordinaria que no tendrá ninguna relación con el tampón.

Palabras clave asociadas

OPEN, GET, PUT, LSET, RSET, MID$

Listado del directorio. Orden

Aplicación

Escribir un listado del directorio en la consola. Puede ser el listado total o un listado parcial que dependerá del nombre de fichero especificado, el cual puede contener «símbolos comodín».

Forma

FILES *[expresión-de-nombre-de-fichero]*

La *expresión-de-nombre-de-fichero* especifica qué ficheros deben aparecer en el listado. Si se la omite, el listado que se obtiene es el listado de todos los ficheros que hay en la unidad implícita.

Notas

En la expresión literal se puede incluir los «símbolos comodín» ? y * para especificar un grupo de ficheros de la forma habitual.

Palabras clave asociadas

DIR, FIND$

FIND$

Buscar un fichero dado. Función

Aplicación

FIND$ busca en el directorio el fichero especificado y da su nombre, si lo encuentra.

Forma

FIND$(*expresión-de-nombre-de-fichero[,ordinal]*)

La *expresión-de-nombre-de-fichero* especifica el nombre del fichero que se ha de buscar. Su valor es una cadena literal que puede contener «símbolos comodín», con el significado habitual.

El *ordinal*, si se lo incluye, tiene que se una *expresión-entera* que dé un valor perteneciente al margen de 1 a 255.

Notas

Si se encuentra el fichero buscado, FIND$ genera una cadena de 12 caracteres que contiene el nombre del fichero, incluidos los bits que representan los atributos del fichero (cuando éste tiene alguno). Obsérvese que se puede suprimir (poner a cero) esos bits mediante la función STRIP$.

Si no aparece ningún fichero concordante con la especificación, FIND$ genera la cadena vacía.

El argumento *ordinal* es útil cuando la *expresión-de-nombre-de-fichero* contiene «símbolos comodín». Si el valor de *ordinal* es n, FIND$ busca el n-ésimo fichero que concuerda con la especificación. Si se omite *ordinal*, FIND$ busca el primer fichero concordante.

Palabras clave asociadas

DIR, FILES, UPPER$, STRIP$

Truncar un número a entero. Función

Aplicación

FIX trunca el valor dado para producir un entero. El método no consiste en redondear un número en punto flotante a entero, sino simplemente en suprimir la parte decimal, de modo que el valor siempre se redondea hacia el entero más próximo a cero.

Forma

FIX(*expresión-numérica*)

Palabras clave asociadas

CINT, INT, ROUND

Bucle de tipo FOR. Orden

Aplicación

Ejecutar una sección del programa cierto número de veces, haciendo avanzar una variable de control desde un valor inicial hasta un valor final.

Forma

FOR *variable-sencilla=principio* TO *fin [*STEP *salto]*

variable-sencilla es la variable de control del bucle FOR; tiene que ser entera o de precisión sencilla; no puede ser un elemento de una matriz.

principio es una *expresión-numérica* que da el valor inicial de la variable de control.

fin es una *expresión-numérica* que da el valor final de la variable de control.

salto es una *expresión-numérica* que da el valor que se debe añadir a la variable de control en cada iteración del bucle. Si se omite este parámetro, se le supone el valor 1. Nótese que salto puede ser negativo.

Los valores de las expresiones *principio*, *fin* y *salto* se fuerzan para que sean del mismo tipo que la variable de control.

Notas

La orden FOR inicia un bucle de tipo FOR. El final del bucle está señalado por la orden NEXT.

Cuando se ejecuta la orden FOR, BASIC evalúa las expresiones *principio*, *fin* y *salto* y asigna a la variable de control el valor de *principio*. Si este valor inicial sobrepasa el valor final especificado, el bucle no se ejecuta. Cada vez que se encuentra el NEXT correspondiente al FOR, se suma el valor de *salto* a la variable de control y se vuelve al principio del bucle, a menos que el nuevo valor de la variable de control haya sobrepasado el valor especificado por *fin*.

Si el valor de *salto* es positivo (o si no se ha incluido este parámetro), el bucle termina cuando la variable de control es mayor que el valor final. Si el salto es negativo, el bucle termina cuando la variable de control es menor que el valor final.

La orden NEXT que corresponde a un FOR dado se determina estáticamente cuando se ejecuta el FOR. Esto quiere decir que la determinación de qué NEXT corresponde a FOR depende del orden de las sentencias del programa, no del orden de ejecución. Por consiguiente, es posible que un mismo FOR tenga varios NEXT asociados.

Los bucles FOR pueden ser anidados, bien dentro de otros bucles FOR, bien dentro de bucles WHILE.

El valor de la variable de control queda definido (y es utilizable) cuando termina el bucle. Se puede terminar un bucle FOR eludiendo el NEXT; en tal caso el valor de la variable de control no se modifica.

Se ha de tener cuidado especial cuando se especifique un valor fraccionario para el *salto*; los efectos del redondeo pueden hacer que la variable de control nunca llegue a ser exactamente igual al valor final especificado.

Palabras clave asociadas

NEXT, WHILE

Determinar el espacio de memoria libre. Función

Aplicación

Averiguar cuánta memoria queda libre de la reservada para BASIC. Hay dos versiones de la función FRE; una de ellas realiza una «limpieza de memoria» antes de medir el espacio libre.

Forma

FRE(*expresión-numérica*)
FRE(*expresión-literal*)

Notas

El valor del argumento de FRE es indiferente, de modo que se recomienda usar las formas FRE(0) y *FRE("")*.

BASIC dedica una sección de la memoria al almacenamiento de cadenas literales. Esta sección va creciendo a medida que se crean nuevas cadenas, pero al mismo tiempo pueden ir quedando huecos libres en ella; ese espacio «muerto» no puede ser utilizado por BASIC mientras no se realice una «limpieza de memoria». En general, BASIC sólo reorganiza la memoria cuando ha agotado el espacio libre.

FRE(0) da el espacio de memoria libre en el que la mencionada sección puede crecer.

FRE("") obliga a BASIC a hacer la «limpieza de memoria» para que los huecos no aprovechados en la sección queden liberados y utilizables. El resultado de la función es entonces el espacio de memoria máximo disponible. Obsérvese que, si hay gran número de cadenas, la «limpieza de memoria» puede requerir un tiempo apreciable.

Palabras clave asociadas

MEMORY, CLEAR, HIMEM

Leer un registro de un fichero aleatorio o de acceso por claves. Orden

Ampliada en las versiones Jetsam

Aplicación

Leer un registro de un fichero de acceso aleatorio o de un fichero de acceso por claves y cargarlo en el tampón de registros aleatorios, donde queda a disposición del programa.

Forma

GET *referencia-de-fichero[,número-de-registro]*
GET *referencia-de-fichero,[número-de-registro],bloqueo*

La *referencia-de-fichero* especifica el fichero en el que se va a leer; tiene que ser un fichero aleatorio o de acceso por claves que esté abierto.

Para ficheros aleatorios:

El *número-de-registro* especifica el registro que se va a leer. Si no se especifica ninguno, se supone el siguiente al que intervino en la orden GET o PUT más reciente; si no se ha ejecutado ninguna orden GET o PUT antes que ésta, se lee el registro 1.

El parámetro *bloqueo*, si se lo incluye, especifica el tipo de bloqueo que se ha de aplicar al registro antes de leerlo. Este parámetro sólo puede intervenir si la orden se aplica a un fichero aleatorio.

Para ficheros de acceso por claves:

El *número-de-registro* especifica el número del registro de datos que se va a leer. Si se omite este parámetro, se lee el registro dado por la posición actual.

Nota (para ficheros aleatorios)

El tampón de registros aleatorios tiene asociado un puntero interno que permite leer datos del registro. La orden GET reinicializa ese puntero.

Notas (para ficheros de acceso por claves)

El registro (o, en su defecto, el fichero) tiene que estar bloqueado en lectura o en escritura.

Sólo se debe utilizar números de registro que hayan sido calculados por la función FETCHREC.

Notas (para ficheros aleatorios en versiones multiusuario)

GET libera el bloqueo temporal en lectura que pudiera estar activo en el registro. Véase en el apartado 8.3.4 la descripción del mecanismo de bloqueo temporal de registros en escritura.

El bloqueo de los registros sólo es necesario cuando el fichero ha sido abierto desbloqueado. Si en GET se especifica un *bloqueo*, BASIC intenta establecerlo antes de leer el registro, y genera un error 69 si no lo consigue. El registro puede haber sido bloqueado con LOCK antes de emitirse la orden GET. Si el registro no está bloqueado cuando GET se dispone a leerlo, BASIC intenta bloquearlo temporalmente en escritura.

Palabras clave asociadas

OPEN, FIELD, INPUT #, LINE INPUT #, PUT

Invocar una subrutina. Orden

Aplicación

Invocar la subrutina dada.

Forma

GOSUB *número-de-línea*
GO SUB *número-de-línea*

Notas

Cuando se ejecuta una orden GOSUB, el programa salta a la línea especificada y recuerda cuál es la sentencia inmediatamente posterior a GOSUB para volver a ella cuando la subrutina termine.

El final de la subrutina está indicado por una orden RETURN. Una subrutina puede contener varios RETURN.

Las subrutinas pueden estar anidadas y ser recurrentes, hasta una profundidad que sólo está limitada por el tamaño de la pila.

Palabras clave asociadas

RETURN, ON *expresión* GOSUB

Ir a una línea. **Orden**

Aplicación

Salto incondicional a una línea dada.

Forma

GOTO *número-de-línea*
GO TO *número-de-línea*

Palabras clave asociadas

ON *expresión* GOTO

Cadena hexadecimal. Función

Aplicación

Crear una cadena de dígitos hexadecimales que represente el valor de la expresión dada.

Forma

HEX$(*expresión-entera-sin-signo*[,*tamaño-de-campo*])

El valor de la *expresión-entera-sin-signo* se trata como entero sin signo de 16 bits que ha de ser convertido en una cadena de caracteres hexadecimales. Véase en la sección 2.10 la descripción de los enteros sin signo.

El *tamaño-de-campo* opcional es una *expresión-entera* que especifica la longitud mínima de la cadena que se va a crear; su valor ha de pertenecer al margen de 0 a 16.

Notas

HEX$ siempre genera todos los caracteres necesarios para representar el número (el cero genera como mínimo un dígito). Si se ha especificado el *tamaño-de-campo* opcional y la cadena es demasiado corta, HEX$ la rellena con ceros por la izquierda. Por larga que sea la cadena, HEX$ no la trunca.

Palabras clave asociadas

OCT$, DEC$, STR$

Dar la dirección de la posición de memoria más alta accesible a BASIC. Función

Aplicación

El espacio de memoria utilizable por BASIC se puede cambiar con las órdenes MEMORY y CLEAR. La función intrínseca HIMEM informa sobre cuál es el límite superior actual de la memoria dedicada a BASIC.

Forma

HIMEM

Notas

HIMEM genera un número de precisión sencilla que da la dirección del byte más alto de la memoria utilizada por BASIC.

Palabras clave asociadas

CLEAR, MEMORY, FRE

Si. Orden

Aplicación

Ejecutar sentencias condicionalmente.

Forma

IF *expresión-lógica* THEN *parte-opcional [*ELSE *parte-opcional]*
IF *expresión-lógica* GOTO *número-de-línea [*ELSE *parte-opcional]*

donde *parte-opcional* es *sentencia(s)*
o bien *número-de-línea*

y donde *sentencia(s)* es una o varias sentencias incluidas en la misma línea que IF (separadas entre sí por signos de dos puntos).

Notas

Se evalúa la *expresión-lógica* y, si el resultado es distinto de cero, se ejecuta la opción especificada tras THEN o GOTO. Si el resultado es cero, se ignora la opción, hasta que se llega al final de la línea o al ELSE que corresponda a este IF; si se encuentra ELSE, se ejecuta la opción especificada tras él.

Las instrucciones IF pueden ser anidadas hasta una profundidad que sólo está limitada por la longitud de la línea de programa. La cláusula ELSE se asocia al IF más interno que no tenga ya su propio ELSE.

THEN *número-de-línea* y ELSE *número-de-línea* equivalen a THEN GOTO *número-de-línea* y ELSE GOTO *número-de-línea*, respectivamente.

La orden IF termina con el final de la línea de programa. No es posible incluir en la misma línea sentencias independientes de IF.

Palabras clave asociadas

WHILE

INKEY$

Captar un carácter por el teclado. Función

Aplicación

Leer la siguiente tecla, si hay alguna pulsada (entrada por la consola). El carácter así leído no aparece en la pantalla.

Forma

INKEY$

Notas

Si hay un carácter pendiente de ser leído, INKEY$ lo entrega (en forma de cadena de un sólo carácter). Si no hay ninguno, INKEY$ genera la cadena vacía.

Todos los caracteres son entregados al programa sin modificación, excepto control-C y control-S. Control-C detiene el programa de la forma habitual. Control-S suspende la ejecución de la forma habitual. Por lo tanto, es posible que el programa dé resultados anómalos, dependiendo del momento exacto en que se pulse estas teclas.

OPTION RUN modifica el efecto de control-C y control-S. Si OPTION RUN está activa, todas las versiones de BASIC tratan estos caracteres como caracteres ordinarios y la función INKEY$ los entrega al programa.

Por compatibilidad con las versiones completas, las versiones de 'sólo ejecución' normalmente ignoran los caracteres control-C y control-S, pero los entregan al programa si OPTION RUN está activa.

Palabras clave asociadas

INPUT$, OPTION RUN, OPTION STOP

Leer una puerta de entrada/salida. Función

Aplicación

Captar el valor presente en la puerta de E/S dada.

Forma

INP(*número-de-puerta*)
INPW(*número-de-puerta*)

En Mallard-80 el *número-de-puerta* es una *expresión-entera* que debe dar un valor perteneciente al margen de 0 a 255.

En Mallard-86 el *número-de-puerta* es una *expresión-de-dirección*.

Notas

INP lee un byte en la puerta de E/S especificada.

INPW lee una palabra en la puerta de E/S especificada. INPW es «ilegal» en Mallard-80.

Palabras clave asociadas

OUT, OUTW, WAIT, WAITW

Captar datos por la consola. Orden

Aplicación

Inducir al operador a que escriba los datos y leerlos.

Forma

INPUT *[;][cadena-entrecomillada;]lista-de: variable*
INPUT *[;][cadena-entrecomillada,]lista-de: variable*

donde la *cadena-entrecomillada* es una cadena inductora opcional.

Notas

El inductor implícito es el signo de interrogación. Si se especifica una cadena inductora seguida del signo de punto y coma, el signo de interrogación aparece a la derecha de la cadena; la coma (en lugar del punto y coma) inhibe el signo de interrogación.

Cuando se ejecuta la orden INPUT, se emite el mensaje inductor y se lee una línea en la consola. La línea leída se trata como lista de elementos; BASIC trata de asignar sucesivamente el valor de cada elemento a una variable, comprobando antes la compatibilidad del elemento con la variable. La línea tiene la forma:

lista-de: elemento

donde cada *elemento* puede ser alguno de los siguientes:

un *valor-numérico*, que puede estar rodeado de *espacios*: los espacios se ignoran;

una *cadena-entrecomillada*, que puede estar rodeada de *espacios*; los espacios se ignoran;

una *cadena-sin-comillas*, que puede estar rodeada de *espacios*; los espacios se ignoran; una *cadena-sin-comillas* es un grupo de entre 0 y 255 caracteres que terminan con una coma o con el retorno del carro.

Si el número de elementos que hay en la línea es el correcto y sus tipos son compatibles con las variables correspondientes, la orden asigna los valores de los elementos a sus variables respectivas.

Si los elementos son demasiados, o demasiado pocos, o si alguno es incompatible con su variable, se emite el mensaje ?Redo from start y se vuelve a ejecutar la orden INPUT desde el principio (incluido el mensaje inductor). En tal caso ninguna de las *variables* habrá sido afectada.

El signo de punto y coma opcional que sigue a la palabra clave INPUT impide que BASIC escriba en la pantalla el retorno del carro con el que termina la introducción de la línea; el cursor se queda, pues, junto al texto escrito.

Los valores se asignan a las variables en el orden en que éstas aparecen en la orden INPUT. Una *variable* puede ser un elemento de una matriz. Puede ocurrir que una de las variables a las que asigna valor la orden INPUT intervenga más tarde, en la misma orden INPUT, en una expresión de índice (es decir, una expresión cuyo valor especifica un elemento de una matriz). En tal caso, el valor del índice se calcula con el nuevo valor de la variable. (Si se va a aprovechar esta característica, es necesario que la expresión de índice dé un índice válido tanto antes como después de que las variables en cuestión hayan sido afectadas por INPUT.)

Palabras clave asociadas

LINE INPUT, DATA, READ, INPUT #, LINE INPUT #, INKEY$, INPUT$

INPUT

Captar datos de un fichero. Orden

Aplicación

Leer datos en un fichero secuencial o en el tampón de registros aleatorios.

Forma

INPUT # *expresión-de-número-de-fichero,lista-de: variable*

La *expresión-de-número-de-fichero* especifica el fichero que se va a leer. Si el fichero está abierto para acceso aleatorio, los datos se leen en el tampón de registros aleatorios.

La *lista-de: variables* especifica a qué variables se va a asignar los datos y de qué tipo son éstos.

Notas

BASIC trata de leer en el fichero un elemento para cada variable de la lista. Cada elemento tiene que ser compatible con su correspondiente variable. Los elementos pueden ser:

- Un *valor-numérico*, que puede ir precedido de un *espacio*; el espacio se ignora. Un elemento numérico termina con un *espacio*, con una coma, con un retorno del carro o con el final del fichero. Se ignora el *espacio* que pueda haber a continuación del elemento, así como la coma o el retorno del carro que suceda al elemento.
- Una *cadena-entrecomillada*, que puede ir precedida de un *espacio*; el espacio se ignora. La cadena está formada por todos los caracteres que haya después de las comillas de abrir, incluidos los retornos del carro y los avances de línea, hasta las comillas de cerrar. La cadena termina con las comillas o con el final del fichero. Si hay algún *espacio* después de las comillas de cerrar, se lo ignora; también se ignora la coma o el retorno del carro que pueda haber a continuación de la cadena.
- Una *cadena-sin-comillas*, que puede ir precedida de un *espacio*; el espacio se ignora; una *cadena-sin-comillas* termina con una coma o con el retorno del carro.

(En todo caso se ignoran los caracteres 'null', código 0; el retorno del carro seguido inmediatamente del avance de línea se trata como un retorno del carro, y vice versa. Las cadenas están limitadas a 255 caracteres y terminan automáticamente cuando se ha completado este número.)

Si el fichero está abierto para acceso aleatorio, la orden INPUT # lee el tampón de registros aleatorios y fracasa (error 50) si se intenta sobrepasar el final del tampón. El tampón tiene asociado un puntero interno, gracias al cual se puede leer el contenido del tampón por tramos, con diferentes órdenes INPUT #. El puntero se reinicializa al principio del tampón cada vez que se lee el registro (con GET).

Palabras clave asociadas

INPUT, LINE INPUT, LINE INPUT #, INKEY$, INPUT$, DATA, READ, GET.

Captar una cadena de longitud fija. Función

Aplicación

Leer un número fijo de caracteres, en el teclado o en un fichero.

Forma

INPUT$(*expresión-entera[,[#]expresión-de-número-de-fichero]*)

La *expresión-entera* da el número de caracteres que van a ser leídos; su valor tiene que estar en el margen de 1 a 255.

Notas

Cuando se lee en el teclado, todos los caracteres se incluyen en la cadena sin modificación y sin reproducirlos en la pantalla, excepto control-C y control-S. Control-C detiene el programa de la forma habitual. Control-S suspende la ejecución de la forma habitual. Por lo tanto, no es posible que el programa dé resultados anómalos, dependiendo del momento exacto en que se pulse estas teclas.

OPTION RUN modifica el efecto de control-C y control-S. Si OPTION RUN está activa, todas las versiones de BASIC tratan estos caracteres como caracteres ordinarios y la función INPUT$ los entrega al programa.

Cuando se lee un fichero de acceso aleatorio, se produce un error si se intenta leer más allá del final del registro actual.

Cuando se lee un fichero de acceso aleatorio o de acceso por claves, no se atribuye significado especial al carácter control-Z.

Por compatibilidad con las versiones completas, las versiones de 'sólo ejecución' normalmente ignoran los caracteres control-C y control-S, pero los entregan al programa si OPTION RUN está activa.

Palabras clave asociadas

INKEY$, INPUT #, OPTION RUN, OPTION STOP

Buscar una subcadena en una cadena. Función

Aplicación

Buscar la primera ocurrencia de una cadena dentro de otra, a partir de una posición dada.

Forma

INSTR([*expresión-entera*,]*cadena*,*subcadena*)

La *expresión-entera* da la posición dentro de la *cadena* a partir de la cual comienza la búsqueda; su valor tiene que estar en el margen de 1 a 255. Si se omite este parámetro, la búsqueda empieza en el primer carácter de la *cadena*.

La *cadena* es una *expresión-literal* cuyo valor es la cadena en la que se busca.

La *subcadena* es una *expresión-literal* cuyo valor es la cadena buscada.

Notas

Si se encuentra la *subcadena*, INSTR da la posición, dentro de la *cadena*, en la que empieza la *subcadena*. Si no, INSTR da el valor cero.

Si en la *cadena* no hay caracteres en los que buscar, a partir de la posición inicial dada (o implícita), INSTR da el valor cero.

Si la *subcadena* es vacía, se la encuentra inmediatamente, a no ser que se esté en el caso de la regla anterior.

Convertir a entero. Función

Aplicación

Redondear el valor dado al menor entero más próximo (redondear hacia $-\infty$). El método no consiste en redondear un número en punto flotante a entero, sino simplemente en suprimir la parte decimal.

(Aplicada a números positivos, INT equivale a FIX. En el caso de los números negativos, el valor producido por INT es igual al que produciría FIX menos 1, a menos que el argumento ya sea entero.)

Forma

FIX(*expresión-numérica*)

Palabras clave asociadas

CINT, FIX, ROUND

Borrar fichero. Orden

Aplicación

Borrar un fichero.

Forma

KILL *expresión-de-nombre-de-fichero*

La *expresión-de-nombre-de-fichero* da el nombre del fichero que va a ser borrado.

Notas

Si la *expresión-de-nombre-de-fichero* contiene «símbolos comodín», se borran todos los ficheros concordantes.

No se recomienda borrar un fichero que esté abierto. En los sistemas multiusuario esta orden puede fallar cuando el fichero está siendo utilizado por otros usuarios o está bloqueado.

Palabras clave asociadas

ERA

Extraer parte de una cadena por la izquierda. Función

Aplicación

Copiar un número de caracteres dado del extremo izquierdo de una cadena dada.

Forma

LEFT$(*expresión-literal*,*expresión-entera*)

La *expresión-literal* da la cadena de la cual se va a tomar los caracteres.

La *expresión-entera* da el número de caracteres que se va a copiar; su valor tiene que estar en el margen de 0 a 255.

Notas

Si la longitud de la cadena dada es menor que el número de caracteres solicitados, LEFT$ genera la cadena completa. Si no, se toma la parte izquierda de la cadena hasta formar una cadena de la longitud especificada.

Palabras clave asociadas

MID$, RIGHT$

Averiguar la longitud de una cadena. Función

Aplicación

Medir la longitud de una cadena dada.

Forma

LEN(*expresión-literal*)

Notas

LEN da un valor comprendido entre 0 y 255. El 0 indica que la cadena es vacía.

Esta función cuenta todos los caracteres de la cadena, incluso los no imprimibles.

Operación de asignación. Orden

Aplicación

La orden LET no tiene ninguna utilidad; en realidad, es una reliquia de las primeras versiones de BASIC y puede ser ignorada.

Forma

LET *variable = expresión*

Notas

Se evalúa la *expresión* y se asigna el resultado a la *variable*, salvo que sean incompatibles.

LET es *redundante* (excepto en el 'modo directo especial' de las versiones de 'sólo ejecución').

LINE INPUT

Captar una línea completa por la consola. **Orden**

Aplicación

Induce al operador a que escriba una línea de texto en la consola, la lee y la asigna a una variable literal.

Forma

LINE INPUT *[;][inductor;]variable-literal*
LINE INPUT *[;][inductor,]variable-literal*

donde *inductor* es, si se lo incluye, una *cadena-entrecomillada*.

La *variable-literal* da el nombre de la variable a la que se asignará la cadena leída.

Notas

Las dos formas son equivalentes.

Cuando se ejecuta la orden LINE INPUT, se emite el mensaje *inductor*, se lee una línea en la consola y se la asigna a la variable literal especificada. La línea termina con el retorno del carro o cuando se ha escrito 255 caracteres, lo que antes ocurra.

El signo de punto y coma opcional que sigue a la palabra clave LINE INPUT impide que BASIC escriba en la pantalla el retorno del carro con el que termina la introducción de la línea; el cursor se queda, pues, junto al texto escrito.

Palabras clave asociadas

DATA, READ, INPUT, INPUT #, LINE INPUT #, INKEY$, INPUT$

LINE INPUT

Leer una línea completa en un fichero. **Orden**

Aplicación

Leer una línea completa en un fichero y asignarla a una variable dada.

Forma

LINE INPUT *#expresión-de-número-de-fichero,variable-literal*

Notas

El fichero tiene que estar abierto para salida secuencial o para acceso aleatorio.

Si está abierto para acceso aleatorio, la orden LINE INPUT # lee el tampón de registros aleatorios y fracasa (error 50) si se intenta sobrepasar el final del tampón. El tampón tiene asociado un puntero interno, gracias al cual se puede leer el contenido del tampón por tramos, con diferentes órdenes LINE INPUT #. El puntero se reinicializa al principio del tampón cada vez que se lee el registro (con GET).

Palabras clave asociadas

DATA, READ, INPUT, INPUT #, LINE INPUT, INKEY$, INPUT$

Listar el programa.

Orden

No disponible en las versiones de 'sólo ejecución'

Aplicación

Listar el programa actual. El listado puede ser total o parcial, y puede ser enviado a la consola (LIST) o a la impresora (LLIST).

Forma

LIST *[margen-de-números-de-línea]*
LLIST *[margen-de-números-de-línea]*

El margen de números de línea especifica las líneas que han de ser listadas. Si se lo omite, el listado es total.

Notas

BASIC vuelve al modo directo cuando termina de obedecer cualquiera de estas dos órdenes.

Pulsando control-C se abandona el listado y se entra en el modo directo.

Cargar un programa en la memoria. Orden

Aplicación

Lee un programa en el disco y lo carga en la memoria, sustituyendo al programa actual. Ofrece la opción de poner en marcha el programa inmediatamente.

Forma

LOAD *expresión-literal*[,R]

La *expresión-literal* da el nombre del fichero en el que está el programa. Si no se indica distintivo de tipo, se supone ".BAS".

Notas

La cláusula opcional ,R indica que se debe ejecutar el programa inmediatamente después de cargarlo.

Esta orden borra de la memoria el programa actual, las funciones de usuario y las variables. Se restablecen a su situación inicial las características establecidas con DEFINT, DEFSNG, DEFDBL, DEFSTR y OPTION BASE. Si no se incluye ,R se cierran todos los ficheros.

En las versiones de 'sólo ejecución' la cláusula ,R es obligatoria. Se se la omite, BASIC vuelve al nivel del sistema operativo en cuanto termina de cargar el programa.

Palabras clave asociadas

SAVE, RUN, MERGE, CHAIN, CHAIN MERGE

Posición actual dentro del fichero. Función

Aplicación

Averiguar el número de registro actual en un fichero dado.

Forma

LOC (*expresión-de-número-de-fichero*)

donde la *expresión-de-número-de-fichero* da el número de un fichero abierto (para cualquier forma de acceso).

Notas

Si el fichero está abierto para acceso aleatorio, LOC da el número de registro correspondiente al registro al que afectó la orden GET o PUT más reciente. Si no se ha ejecutado ninguna de estas órdenes desde que se abrió el fichero, LOC da el valor cero.

Si el fichero está abierto para entrada o salida secuencial, LOC da el número de bloques de 128 bytes que se han leído o escrito hasta el momento.

LOC no puede ser utilizada con ficheros de acceso por claves.

Palabras clave asociadas

LOF, EOF

Cambiar el bloqueo de un registro. Función de Jetsam

Aplicación

Cambiar la situación de bloqueo de un registro.

Forma

LOCK(*referencia-de-fichero,bloqueo,número-de-registro*)

La *referencia-de-fichero* da el número de un fichero de acceso aleatorio o de acceso por claves que está abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de dar al registro.

El *número-de-registro* especifica a qué registro del fichero de datos se le va a cambiar la situación de bloqueo.

Códigos generados

0	Éxito	Sin calificativos.
130	Fracaso	El registro ya ha sido bloqueado en lectura por otro usuario.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado.

Notas

Con los ficheros de acceso por claves sólo se debe utilizar números de registro que hayan sido proporcionados por la función FETCHREC. Cuando se encontró el registro (con alguna de las diversas funciones SEEK) se le aplicaría un bloqueo de registro. La orden LOCK permite modificar o liberar ese bloqueo. El registro tiene que haber sido bloqueado en lectura o en escritura previamente.

Con los ficheros de acceso aleatorio la función LOCK puede ser utilizada en cualquier momento para restablecer, cambiar o liberar bloqueos en cualquier registro del fichero. No es necesario que el fichero esté bloqueado. Los bloqueos de registro dependen también de las órdenes GET y PUT. La función LOCK y el mecanismo de bloqueo temporal de registros en escritura se afectan mutuamente; véase el apartado 8.3.4.

Palabras clave asociadas

GET, PUT, SEEKRANK, SEEKKEY, SEEKREC, SEEKSET, SEEKNEXT, SEEKPREV

Longitud de fichero. Función

Aplicación

LOF da una indicación de la longitud del fichero especificado.

Forma

LOF(*expresión-de-número-de-fichero*)

donde la *expresión-de-número-de-fichero* da el número de un fichero abierto (para cualquier modo de acceso).

Notas

El valor generado da una indicación de la longitud del fichero; el valor exacto depende del sistema operativo subyacente. LOF sólo da el valor cero cuando el fichero está vacío.

LOF no puede ser aplicada a ficheros de acceso por claves.

Palabras clave asociadas

LOC, EOF

Logaritmo natural. Función

Aplicación

Calcular el logaritmo natural o neperiano del número dado.

Forma

LOG(*expresión-numérica*)

donde el valor de *expresión-numérica* tiene que ser mayor que cero.

Notas

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado de LOG es también un número de precisión sencilla.

Palabras clave asociadas

EXP, LOG10

Logaritmo decimal. Función

Aplicación

Calcular el logaritmo decimal (en base 10) del número dado.

Forma

LOG10(*expresión-numérica*)

donde el valor de *expresión-numérica* tiene que ser mayor que cero.

Notas

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado de LOG10 es también un número de precisión sencilla.

Palabras clave asociadas

EXP, LOG

Convertir cadena a minúsculas. Función

Aplicación

Crea una cadena que es copia de otra, con todas las letras mayúsculas convertidas a minúsculas.

Forma

LOWER$(*expresión-literal*)

Notas

La cadena generada es la *expresión-literal*, con todas las letras del margen de A a Z convertidas a sus correspondientes del margen de a a z.

Palabras clave asociadas

UPPER$

Posición en la impresora. Función

Aplicación

Averiguar la posición actual de escritura en la impresora.

Forma

LPOS(*expresión-numérica*)

donde el valor de la *expresión-numérica* es indiferente, por lo que se recomienda la forma LPOS(0).

Notas

LPOS da la posición lógica en la impresora, que puede no coincidir con la posición física de la cabeza impresora.

En el cálculo de la posición lógica no se incluyen los caracteres no imprimibles (los de valor menor que 32), a excepción de los siguientes:

- Retroceso (valor 8), que cuenta como −1, salvo cuando la posición es 1.
- Tab (valor 9), que se expande a espacios, cada uno de los cuales cuenta como 1. (Los topes de tabulación están dispuestos de forma que hay uno cada ocho columnas, si no se los define de otra manera.)
- Retorno del carro (valor 13), que restablece la posición lógica a 1.

La posición también depende de los retornos del carro insertados por BASIC cuando la posición lógica excede de la anchura de la impresora (establecida con WIDTH LPRINT).

Si se ha hecho la anchura infinita (con WIDTH LPRINT 255), la posición lógica puede llegar a 255, pero no puede superar este valor.

Palabras clave asociadas

WIDTH LPRINT, POS

Escribir en la impresora. **Orden**

Véase PRINT, leyendo 'impresora' donde diga 'consola'.

Palabras clave asociadas

LPOS, PRINT

LSET

Asignar una cadena a otra, justificando a la izquierda. **Orden**

Aplicación

Sustituir el contenido de una cadena por otra, rellenando con espacios por la derecha hasta completar la longitud original.

Forma

LSET *variable-literal = expresión literal*

Notas

Se evalúa la *expresión-literal* y luego se la fuerza a tener la misma longitud que el valor de la *variable-literal* dada; para ello puede ser necesario rellenar con espacios por la derecha o ignorar los caracteres sobrantes por la derecha. El resultado de esta operación sustituye el contenido original de la *variable-literal*.

LSET se utiliza, en particular, para dar nuevos valores a los campos de un registro aleatorio.

Palabras clave asociadas

FIELD, RSET, MID$, MKI$, MKS$, MKD$

Determinar el valor máximo. Función

Aplicación

Determinar cuál de los valores dados es el máximo.

Forma

MAX(*lista-de:expresión-numérica*)

Nota

MAX da el valor de la *expresión-numérica* mayor.

Palabras clave asociadas

MIN

MEMORY

Restablecer los parámetros de memoria de BASIC. **Orden**

Aplicación

Cambiar la memoria utilizable por BASIC, el número máximo de ficheros y el tamaño máximo de los registros aleatorios.

Formas

MEMORY
MEMORY,*memoria-alta*
MEMORY,*[memoria-alta],tamaño-de-pila*
MEMORY,*[memoria-alta],[tamaño-de-pila],número-de-ficheros*
MEMORY,*[memoria-de-alta],[tamaño-pila],[número-de-ficheros],*
longitud-de-registro-máxima

memoria-alta es una *expresión-de-dirección* que da la dirección del byte más alto de la zona de memoria que puede ser utilizada por BASIC. Si se omite este parámetro, se mantiene la situación actual.

tamaño-de-pila es una *expresión-de-dirección* que da el número de bytes que puede ocupar la pila de BASIC. Si se omite este parámetro, se mantiene la situación actual. Si se lo incluye, la expresión tiene que dar un valor de, como mínimo, 256.

número-de-ficheros es una *expresión-de-dirección* que da el número máximo de ficheros que pueden ser manejados simultáneamente. Si se omite este parámetro, se mantiene la situación actual.

longitud-de-registro-máxima es una *expresión-de-dirección* que especifica el tamaño máximo de los registros. Si se omite este parámetro, se mantiene la situación actual.

Notas

Sólo es necesario especificar valores para las características que se desea cambiar.

Los parámetros opcionales permiten establecer el número máximo de ficheros, el tamaño máximo de los registros y la memoria total de forma muy similar a como se hace en la orden del sistema operativo con la que se carga BASIC (véase el apéndice I).

Al modificar el tamaño de la pila, BASIC olvida las órdenes FOR, WHILE y GOSUB que puedan estar activas.

Al reducir el número máximo de ficheros se cierran automáticamente todos los ficheros cuyo número sea mayor que el nuevo máximo.

Al modificar el tamaño máximo de los registros se cierran automáticamente todos los ficheros que estén activos. Los ficheros de acceso por claves cerrados de esta forma quedan marcados como inconsistentes; así pues, se los debe cerrar explícitamente (con CLOSE) antes de dejar que esto ocurra.

La pila de BASIC se utiliza durante la evaluación de expresiones y para almacenar la información necesaria para los bucles FOR y WHILE y para los retornos de las subrutinas. El tamaño implícito es 512 bytes, que debe de ser adecuado para la mayor parte de los programas, salvo los que tengan un grado extremo de anidamiento de bucles o muchos GOSUB.

Palabras clave asociadas

CLEAR, HIMEM, OPEN, FRE

MERGE

Leer un programa en el disco y mezclarlo con el actual. **Orden**

No disponible en las versiones de 'sólo ejecución'

Aplicación

Mezclar las líneas de un fichero de programa con el programa actualmente residente en la memoria.

Forma

MERGE *expresión-literal*

La *expresión-literal* da el nombre del fichero que contiene el programa requerido. Si no se especifica distintivo de tipo, se supone ".BAS".

En la *expresión-literal* se puede incluir la especificación de la unidad de disco (por ejemplo, B:) como prefijo.

Notas

Las líneas del nuevo programa sustituyen a las que pudiera haber en el programa actual con el mismo número de línea.

MERGE ejerce los siguientes efectos secundarios:

- Se desechan todas las variables.
- Se olvidan todas las funciones de usuario.
- Se desactiva ON ERROR GOTO.
- Se conservan todos los ficheros que estén abiertos.
- Se ejecuta implícitamente una orden RESTORE.
- Se restablece a sus situación inicial las características establecidas por OPTION BASE, DEFINT, DEFSNG, DEFDBL y DEFSTR.

BASIC entra en el modo directo en cuanto termina de ejecutar una orden MERGE.

Si se mezcla con MERGE un programa protegido con uno desprotegido, el programa resultante estará protegido.

Palabras clave asociadas

LOAD, CHAIN MERGE, SAVE, CHAIN, RUN

MID$

Sustituir parte de una cadena. **Orden**
Extraer parte de una cadena. **Función**

Aplicación

MID$ especifica una parte de una cadena (subcadena) que puede ser utilizada como destino en una operación de asignación (MID$ como orden) o bien como argumento en una expresión literal (MID$ como función).

Forma

MID$(*cadena,posición-inicial[,longitud-de-cadena]*)

Para MID$ como orden, la *cadena* tiene que ser una *variable-literal*, parte de la cual se va a modificar.

Para MID$ como función, la *cadena* tiene que ser una *expresión-literal*, parte de la cual se va a generar como valor de la función.

La *posición-inicial* es una *expresión-entera* que especifica qué carácter de la *cadena* va a ser el primer carácter de la subcadena. El valor de la *expresión-entera* tiene que estar en el margen de 1 a 255.

La *longitud-de-subcadena* es una *expresión-entera* que especifica la longitud de la subcadena. Si se omite este parámetro, o si su valor es mayor que el número de caracteres que quedan a partir de la *posición-inicial*, la subcadena se extiende hasta el final de la cadena original. El valor de la *expresión-entera* tiene que estar en el margen de 0 a 255.

Notas

La subcadena especificada en MID$ está definida por la posición del carácter inicial y la longitud. El primer carácter de la cadena original está en la posición 1. Si no se especifica longitud para la subcadena, ésta incluye todos los caracteres restantes, desde la posición especificada hasta el final de la cadena original. Si la posición inicial está más allá del final de la cadena original, la subcadena es vacía. La subcadena no puede sobrepasar el final de la cadena original, de modo que su longitud puede ser menor que la especificada.

Cuando se usa MID$ como orden, tiene que figurar en el primer miembro (o sea, a la izquierda) de una asignación. El segundo miembro tiene que ser una *expresión-literal*. La subcadena especificada por MID$ será reemplazada por el valor de la *expresión-literal*. Si la *expresión-literal* da un valor cuya longitud es menor que la de la subcadena, los caracteres sobrantes en ésta no resultan afectados. Si la longitud del valor de la *expresión-literal* es menor que la de la subcadena, se ignoran los caracteres sobrantes en aquélla.

Palabras clave asociadas

LSET, RSET, LEFT$, RIGHT$, FIELD

Determinar el valor mínimo. Función

Aplicación

Determinar cuál de los valores dados es el mínimo.

Forma

MIN(*lista-de: expresión-numérica*)

Nota

MIN da el valor de la *expresión-numérica* menor.

Palabras clave asociadas

MAX

Convertir doble precisión a cadena. Función

Aplicación

Convertir un número de doble precisión en una cadena de ocho bytes, de modo que el valor pueda ser almacenado en esta versión. Esto es particularmente útil para el almacenamiento de valores de doble precisión en registros aleatorios.

Forma

MKD$(*expresión-numérica*)

Notas

Se convierte el resultado de la expresión a doble precisión si no estuviera ya en esta representación.

No se recomienda procesar la cadena, salvo para asignarla a otra. La función CVD realiza la conversión inversa, de cadena de ocho bytes a número en doble precisión.

Esta función no realiza ninguna conversión de binario a decimal, de modo que la cadena representa exactamente el número.

Palabras clave asociadas

CVD, MKI$, MKS$

Convertir entero a cadena. Función

Aplicación

Convertir un número entero en una cadena de dos bytes, de modo que el valor pueda ser almacenado en esta versión. Esto es particularmente útil para el almacenamiento de valores enteros en registros aleatorios.

Forma

MKI$(*expresión-numérica*)

Notas

Si el resultado de la expresión no es entero, se lo redondea a entero.

No se recomienda procesar la cadena, salvo para asignarla a otra. La función CVI realiza la conversión inversa, de cadena de dos bytes a entero.

Esta función no realiza ninguna conversión de binario a decimal.

MKIK$ y MKUK$ son funciones similares que producen cadenas de dos bytes utilizables como claves (o como parte de claves) de Jetsam. La cadena producida por MKI$ no es adecuada para los algoritmos de ordenación.

Palabras clave asociadas

CVI, MKS$, MKD$

Convertir entero a cadena para clave. Función

Aplicación

Convertir un número entero en una cadena de dos bytes, adecuada para ser utilizada como clave (o como parte de una clave) de Jetsam.

Forma

MKIK$(*expresión-numérica*)

Notas

Si el resultado de la expresión no es entero, se lo redondea a entero.

No se recomienda procesar la cadena, salvo para asignarla a otra. La función CVIK realiza la conversión inversa, de cadena de dos bytes a entero.

Esta función no realiza ninguna conversión de binario a decimal.

MKIK$ es la función que se debe emplear cuando se desea convertir valores enteros del margen de −32768 a +32767 en cadenas utilizables como claves de Jetsam. Interesa que las claves de Jetsam sean lo más cortas posible; esta función produce una representación compacta del número. Para enteros del margen de 0 a 65535 se debe usar la función MKUK$.

La función MKI$ produce una cadena similar, pero que no es adecuada para los algoritmos de ordenación cuando se la utiliza como clave de Jetsam.

Palabras clave asociadas

CVIK, MKI$, MKUK$

Convertir precisión sencilla a cadena. Función

Aplicación

Convertir un número de precisión sencilla en una cadena de cuatro bytes, de modo que el valor pueda ser almacenado en esta versión. Esto es particularmente útil para el almacenamiento de valores de precisión sencilla en registros aleatorios.

Forma

MKS$(*expresión-numérica*)

Notas

Se convierte el resultado de la expresión a precisión sencilla, si no estuviera ya en esta representación.

No se recomienda procesar la cadena, salvo para asignarla a otra. La función CVS realiza la conversión inversa, de cadena de cuatro bytes a número en precisión sencilla.

Esta función no realiza ninguna conversión de binario a decimal, de modo que la cadena representa exactamente el número.

Palabras clave asociadas

CVS, MKI$, MKD$

Convertir un número entero sin signo a cadena para clave. Función

Aplicación

Convertir un número entero sin signo en una cadena de dos bytes, adecuada para ser utilizada como clave (o como parte de una clave) de Jetsam.

Forma

MKUK$(*expresión-numérica*)

Notas

Si el resultado de la expresión no es entero, se lo redondea a entero.

No se recomienda procesar la cadena, salvo para asignarla a otra. La función CVUK realiza la conversión inversa, de cadena de dos bytes a entero sin signo.

Esta función no realiza ninguna conversión de binario a decimal.

MKUK$ es la función que se debe emplear cuando se desea convertir valores enteros del margen de 0 a 65535 en cadenas utilizables como claves de Jetsam. Interesa que las claves de Jetsam sean lo más cortas posible; esta función produce una representación compacta del número. Para enteros del margen de −32768 a +32767 se debe usar la función MKIK$.

La función MKI$ produce una cadena similar, pero que no es adecuada para los algoritmos de ordenación cuando se la utiliza como clave de Jetsam.

Palabras clave asociadas

CVUK, MKI$, MKIK$

Cambiar el nombre de un fichero. Orden

Forma

NAME *expresión-de-nombre-de-fichero* AS *expresión-de-nombre-de-fichero*

La primera *expresión-de-nombre-de-fichero* da el nombre antiguo del fichero.

La segunda *expresión-de-nombre-de-fichero* da el nombre nuevo.

Notas

El fichero cuyo nombre se va a cambiar tiene que existir. No puede existir otro fichero con el nombre nuevo.

No se recomienda cambiar el nombre de un fichero que esté abierto. En los sistemas multiusuario esta orden puede fracasar si el fichero está siendo utilizado por otros usuarios o está bloqueado.

Palabras clave asociadas

REN

Preparar la introducción de un nuevo programa. **Orden**

No disponible en las versiones de 'sólo ejecución'

Aplicación

NEW se utiliza para borrar completamente la memoria antes de empezar a introducir un programa nuevo.

Forma

NEW

Notas

Además de borrar el programa actual, produce los siguientes efectos:

- Se desechan todas las variables.
- Se olvidan todas las funciones de usuario.
- Se desactiva ON ERROR GOTO.
- Se cierran todos los ficheros.
- Se restablece a su situación inicial todas las características establecidas por OPTION BASE, DEFINT, DEFSNG, DEFDBL y DEFSTR.
- Se desactiva TRON.

BASIC entra en el modo directo cuando termina de ejecutar una orden NEW.

Hacer avanzar la variable de control de un bucle FOR hasta el valor siguiente.

Orden

Aplicación

La orden NEXT marca el final de un bucle de tipo FOR. La orden NEXT puede ser anónima o mencionar explícitamente el nombre de la variable de control del bucle.

Forma

NEXT *[lista-de: variable]*

donde NEXT *variable,lista-de: variable*
equivale a NEXT *variable:* NEXT *lista-de: variable*

Notas

Una orden NEXT indica el final de un bucle FOR. La forma en que FOR y NEXT están conectadas se explica en la descripción de FOR. Cuando ejecuta una orden NEXT, BASIC sabe a qué FOR corresponde. Si tras NEXT se menciona el nombre de una variable, ese nombre tiene que ser el de la variable de control del bucle.

Se produce un error cuando se ejecuta un NEXT que no tenga ningún FOR asociado.

Palabras clave asociadas

FOR

Cadena octal. Función

Aplicación

Crear una cadena de dígitos octales que represente el valor de la expresión dada.

Forma

OCT$(*expresión-entera-sin-signo[,tamaño-de-campo]*)

El valor de la *expresión-entera-sin-signo* se trata como entero sin signo de 16 bits que ha de ser convertido en una cadena de caracteres octales. Véase en la sección 2.10 la descripción de los enteros sin signo.

El *tamaño-de-campo* opcional es una *expresión-entera* que especifica la longitud mínima de la cadena que se va a crear; su valor ha de pertenecer al margen de 0 a 16.

Nota

OCT$ siempre genera todos los caracteres necesarios para representar el número (el cero genera como mínimo un dígito). Si se ha especificado el *tamaño-de-campo* opcional y la cadena es demasiado corta, OCT$ la rellena con ceros por la izquierda. Por larga que sea la cadena, OCT$ no la trunca.

Palabras clave asociadas

HEX$, DEC$, STR$

GOSUB y GOTO en función de un valor calculado. **Orden**

Aplicación

Elegir, dependiendo del resultado de una expresión, qué subrutina se debe invocar o a qué línea del programa se debe transferir el control, de entre las mencionadas en una lista.

Forma

ON *expresión-entera* GOSUB *lista-de: número*
ON *expresión-entera* GOTO *lista-de: número*

La *expresión-entera* tiene que dar un valor perteneciente al margen de 0 a 255.

Notas

Con el resultado de la *expresión-entera* se selecciona un número de línea de los incluidos en la lista. Si el resultado es 1, se elige la línea que figura en primer lugar; si es 2, la que está en segundo lugar; y así sucesivamente. Según de qué orden se trate, se transfiere el control a la línea elegida o se invoca la subrutina que empieza en ella. Si el resultado de la *expresión-entera* es cero o mayor que el número de líneas citadas en la lista, la orden no tiene ningún efecto.

Es «legal» que en la lista figuren elementos «nulos» (o sea, dos comas seguidas). No lo es, en cambio, seleccionar uno de esos elementos nulos (error 2).

Palabras clave asociadas

GOTO, GOSUB

Establecer intercepción para errores. Orden

Aplicación

Cuando BASIC detecta un error durante la ejecución de una orden, puede, o bien realizar la acción por defecto, que es emitir un mensaje de error y volver al modo directo, o bien invocar una subrutina de gestión de errores preparada en el programa. ON ERROR GOTO ordena a BASIC que, en caso de error, vaya a la línea especificada.

Forma

ON ERROR GOTO *número-de-línea*
ON ERROR GOTO 0

El *número-de-línea* especifica la línea a la que se debe transferir el control en caso de error.

Notas

La forma ON ERROR GOTO *número-de-línea* permite la intercepción y gestión de errores. ON ERROR GOTO 0 inhibe la intercepción y además ejerce otros efectos en las rutinas de gestión de errores; véase más abajo.

Si se produce un error en modo de programa cuando está habilitada la intercepción de errores, el control se transfiere a la línea especificada en la orden ON ERROR GOTO. BASIC entra entonces en el 'modo de gestión de errores', en el que se dispone de las funciones ERR y ERL para averiguar qué error ha ocurrido y en qué línea ha sido detectado.

Dentro del modo de gestión de errores, ON ERROR GOTO 0 no sólo inhibe la intercepción de errores, sino que hace que BASIC lleve a cabo la acción implícita correspondiente al error detectado. Por consiguiente, las rutinas de gestión de errores pueden tratar los errores para los que son competentes y dejar que BASIC se encargue directamente de los restantes.

Si se produce un nuevo error cuando se está en el modo de gestión de errores, BASIC lleva a cabo inmediatamente la acción implícita.

La orden RESUME es «legal» solamente en el modo de gestión de errores; sirve para reanudar la ejecución del programa de alguna de las siguientes maneras:

— en la sentencia en la que se detectó el error
— en la sentencia siguiente a aquélla en la que se detectó el error
— en un número de línea especificado

Palabras clave asociadas

ERR, ERL, RESUME

Abrir un fichero.

Orden
Ampliada en las versiones Jetsam

Aplicación

En los ficheros aleatorios y secuenciales, abrir un fichero en el disco y asociarle un número de fichero a través del cual se accederá al fichero.

En los ficheros de acceso por claves, abrir un fichero de índices y uno de datos y asociarles un número de fichero mediante el cual se identificará el fichero de acceso por claves.

Forma (para ficheros aleatorios y secuenciales)

OPEN *modo,referencia-de-fichero,expresión-de-nombre-de-fichero[,longitud-de-registro]*
OPEN *modo,referencia-de-fichero,expresión-de-nombre-de-fichero*
[,longitud-de-registro],bloqueo

modo es una *expresión-literal* que especifica el modo de acceso al fichero. El primer carácter de su valor tiene que ser alguno de los siguientes:

I (entrada secuencial)
O (salida secuencial)
R (acceso aleatorio)

La *referencia-de-fichero* da el número de fichero al que se asociará el fichero y por el que se lo identificará en el futuro. Se produce el error 55 si se intenta usar un número de fichero que ya está asignado.

La *expresión-de-nombre-de-fichero* da el nombre del fichero que se va a abrir.

En la *expresión-literal* se puede incluir la especificación de la unidad de disco (por ejemplo, B:) como prefijo.

La *longitud-de-registro* sólo se puede especificar si el modo es "R"; es una expresión entera que tiene que dar un valor perteneciente al margen de 1 a *máx*, donde *máx* se establece mediante las órdenes CLEAR y MEMORY, o bien cuando se carga BASIC (valor implícito: 128). Si no se especifica *longitud-de-registro*, se supone 128.

Se puede incluir el parámetro *bloqueo* para especificar el bloqueo que se debe aplicar al fichero una vez abierto. En los sistemas uniusuario este parámetro no ejerce ningún efecto. En los sistemas multiusuario, predomina sobre el tipo de bloqueo de fichero implícito.

Notas (para ficheros aleatorios y secuenciales)

Cuando se abre un fichero para entrada, es necesario que el fichero exista.

Cuando se abre un fichero para salida, el fichero que haya en el disco con el mismo nombre se borra inmediatamente y se crea uno nuevo.

Cuando se abre un fichero para acceso aleatorio, se crea un fichero nuevo si no existe el especificado.

Notas (sobre bloqueo de ficheros aleatorios y secuenciales en sistemas multiusuario)

En los sistemas multiusuario, si no se especifica ningún *bloqueo*, BASIC supone un bloqueo implícito, establecido por el parámetro *modo* en la orden OPEN. Si se incluye un *bloqueo*, el tipo de bloqueo posible está restringido por *modo*. Así,

- Los ficheros de entrada sólo pueden ser bloqueados en lectura o en escritura; por defecto, en escritura. (El bloqueo en escritura da acceso en exclusiva al fichero e impide que se escriba en él.)
- Los ficheros de salida sólo pueden ser bloqueados en escritura; el bloqueo implícito es en escritura.
- Los ficheros aleatorios pueden tener cualquier tipo de bloqueo; el bloqueo implícito es en escritura.

Si BASIC no consigue establecer el bloqueo solicitado, la orden OPEN fracasa y genera el error 73.

Forma (para ficheros de acceso por claves)

OPEN "K",*referencia-de-fichero,nombre-de-fichero-de-datos,nombre-de-fichero-de-índices,bloqueo[,variable-de-longitud-de-registro]*

OPEN "K",*referencia-de-fichero,nombre-de-fichero-de-datos,nombre-de-fichero-de-índices,bloqueo[,variable-de-longitud-de-registro],variable-literal-de-usuario*

La *referencia-de-fichero* da el número de fichero al que se asociará el fichero de claves.

El *nombre-de-fichero-de-datos* es una *expresión-de-nombre-de-fichero* que da el nombre del fichero de datos que se va a abrir.

El *nombre-de-fichero-de-índices* es una *expresión-de-nombre-de-fichero* que da el nombre del fichero de índices que se va a abrir.

El *bloqueo* es el que se aplicará al fichero de claves después de abrirlo.

La *variable-de-longitud-de-registro*, si se la incluye, es una *variable-numérica* cuyo valor da la longitud de registro (especificada cuando se creó el fichero).

La *variable-literal-de-usuario*, si se la incluye, es una *variable-literal* cuyo valor es la cadena de usuario que se especificó cuando se creó el fichero.

Notas (para ficheros de acceso por claves)

La orden OPEN puede fracasar por diversas razones:

- Error 33: no existe uno de los dos ficheros.
- Error 73: no se ha podido aplicar el bloqueo solicitado.
- Error 113: uno de los dos ficheros no fue creado por una orden CREATE (es decir, el fichero no es reconocible por Jetsam).
- Error 115: los ficheros no son consistentes el uno con el otro (es decir, el fichero de acceso por claves no fue cerrado adecuadamente después de modificarlo).

La posición actual queda indefinida.

Palabras clave asociadas

CLOSE, CLEAR, MEMORY, CREATE (Jetsam)

OPTION BASE

Establecer el valor base para los índices de las matrices. **Orden**

Aplicación

Los índices de las matrices pueden partir de 0 o de 1. OPTION BASE elige entre estas dos alternativas.

Forma

OPTION BASE *expresión-entera*

donde la *expresión-entera* tiene que tomar el valor 0 o el 1.

Notas

Si se intenta ejecutar más de una orden OPTION BASE se provoca un error (error 10).

Si se intenta ejecutar una orden OPTION BASE cuando ya existe alguna matriz se provoca un error (error 10).

El valor base implícito es 0.

Palabras clave asociadas

DIM, ERASE, CLEAR

OPTION FIELD

Hacer accesibles al usuario los dos bytes reservados en el registro. **Orden de Jetsam**

Aplicación

Permitir que los dos bytes reservados para Jetsam en cada registro de un fichero de acceso por claves sean accesibles al usuario. Los bytes tienen que ser definidos más tarde en una orden FIELD.

Forma

OPTION FIELD *expresión-numérica*

Notas

Si *expresión-numérica* = 2, los dos bytes quedan ocultos al usuario. Ésta es la situación implícita.

Si *expresión-numérica* = 0, las órdenes FIELD afectarán a los dos bytes del principio del registro reservados para Jetsam. Esta situación es compatible con versiones anteriores de BASIC.

Advertencia

Esta orden se incluye sólo por razones de compatibilidad con versiones anteriores de BASIC; no se la debe utilizar normalmente.

Palabras clave asociadas

FIELD

Establecer la unidad de disco y el número de usuario implícitos.

Orden de CP/M

Aplicación

Cambiar la unidad implícita y el número de usuario implícito.

Forma

OPTION FILES *expresión-literal*

donde la *expresión-literal* es un número de usuario válido o una letra que representa la unidad. Se puede omitir cualquiera de los dos.

Notas

Esta orden es válida solamente en los sistemas operativos de tipo CP/M.

La selección de número de usuario y de unidad así establecida queda en vigor hasta que se la vuelva a cambiar desde BASIC o hasta que se retorne al sistema operativo. Cuando se devuelve el control al sistema operativo, la unidad y el número de usuario implícitos vuelven a ser los mismos que cuando se cargó BASIC.

OPTION INPUT

Establecer intercepción para la entrada por la consola. **Orden**

Aplicación

Designar una subrutina en código de máquina que será la encargada de leer los caracteres introducidos por el teclado. Esa subrutina será invocada en lugar de la del sistema operativo.

Forma

OPTION INPUT = *expresión-de-dirección,expresión-de-dirección*
OPTION INPUT

La primera *expresión-de-dirección* da la dirección de una subrutina en código de máquina cuya misión sea 'examinar el estado del teclado'.

La segunda *expresión-de-dirección* da la dirección de una subrutina en código de máquina cuya misión sea 'captar carácter por el teclado'.

Notas

La primera forma de esta orden establece las direcciones de estas dos subrutinas. La segunda forma (y la orden RUN) cancela la anterior definición y hace que BASIC vuelva a utilizar las subrutinas del sistema operativo.

Las dos subrutinas son:

'Examinar el estado del teclado'

BASIC invoca esta subrutina con regularidad para averiguar si se ha pulsado alguna tecla, con objeto de detectar si se ha introducido control-C o control-S. Esta operación afecta a la velocidad de BASIC, así que conviene que la subrutina sea lo más rápida posible.

'Captar carácter por el teclado'

BASIC invoca esta subrutina cuando necesita leer un carácter en el teclado. La subrutina tiene que esperar hasta que se haya pulsado algún carácter y entregar su valor.

Los requisitos de interfaz de ambas subrutinas son muy estrictos. Cualquier desviación con respecto a esos requisitos causará el fracaso de BASIC. Las subrutinas pueden utilizar una pequeña porción de la pila.

En Mallard-80 las subrutinas deben cumplir las siguientes especificaciones:

'Examinar el estado del teclado'

Entrada: Sin condiciones

Salida: Si hay un carácter disponible:
Indicador de arrastre a 1
A=carácter pulsado

Si no hay carácter disponible:
Indicador de arrastre a 0
A puede quedar alterado
BC, DE, HL y otros indicadores pueden quedar alterados
Todos los demás registros han de ser preservados

'Captar carácter por el teclado'

Entrada: Sin condiciones

Salida: A=carácter pulsado
BC, DE, HL y los indicadores pueden quedar alterados
Todos los demás registros han de ser preservados

En Mallard-86:

Las *expresiones-de-dirección* dan la componente de segmento de las direcciones de las subrutinas. Esta componente es la que esté en vigor en el momento de ejecutarse la orden OPTION INPUT y habrá sido establecida con DEF SEG. Los interfaces para las subrutinas son:

'Examinar el estado del teclado'

Entrada: CS contiene la componente de segmento de la dirección de la subrutina
DS, ES y SS contienen la base del segmento de datos de BASIC

Salida: Si hay un carácter disponible:
Indicador de arrastre a 1
AL=carácter leído

Si no hay carácter disponible:
Indicador de arrastre a 0
AL puede quedar alterado

Siempre:
AH y otros indicadores pueden quedar alterados
Todos los demás registros deben ser preservados

'Captar carácter por el teclado'

Entrada: CS contiene la componente de segmento de la dirección de la subrutina
DS, ES y SS contienen la base del segmento de datos de BASIC

Salida: AL=carácter pulsado
AH y los indicadores pueden quedar alterados
Todos los demás registros han de ser preservados.

Estas rutinas se invocan con una 'llamada lejana', luego deben terminar con un 'retorno lejano'.

Palabras clave asociadas:

OPTION LPRINT, OPTION PRINT, DEF SEG

OPTION LPRINT

Establecer intercepción para la salida por la impresora. **Orden**

Aplicación

Designar una subrutina en código de máquina que será la encargada de enviar los caracteres a la impresora. Esa subrutina será invocada en lugar de la del sistema operativo.

Forma

OPTION LPRINT = *expresión-de-dirección,expresión-de-dirección*
OPTION LPRINT

La *expresión-de-dirección* da la dirección de la subrutina en código de máquina.

Notas

La primera forma de esta orden establece la dirección de la nueva subrutina. La segunda forma (y la orden RUN) cancela la anterior definición y hace que BASIC vuelva a utilizar la subrutina del sistema operativo.

Los requisitos de interfaz de la subrutina son muy estrictos. Cualquier desviación con respecto a esos requisitos causará el fracaso de BASIC. La subrutina puede utilizar una pequeña porción de la pila.

En Mallard-80 la subrutina debe cumplir las siguientes especificaciones:

Entrada: C contiene el carácter que se va a enviar a la impresora
Salida: A, BC, DE, HL y todos los indicadores pueden quedar alterados.

En Mallard-86:

La *expresión-de-dirección* da la componente de segmento de la dirección de la subrutina. Esta componente es la que esté en vigor en el momento de ejecutarse la orden OPTION LPRINT y habrá sido establecida con DEF SEG. El interfaz para la subrutina es:

Entrada: CL contiene el carácter que se va a enviar a la impresora
CS contiene la componente de segmento de la dirección de la subrutina
DS, ES y SS contienen la base del segmento de datos de BASIC
Salida: AX y los indicadores pueden quedar alterados
Todos los demás registros deben ser preservados

Esta rutina se invoca con una 'llamada lejana', luego debe terminar con un 'retorno lejano'.

Palabras clave asociadas

OPTION INPUT, OPTION PRINT, LPRINT, DEF SEG

OPTION NOT TAB

Desactivar la expansión de 'tab'. **Orden**

Aplicación

Impedir que BASIC transforme los caracteres 'tab' (&H09) en cadenas de espacios cuando los escribe en la pantalla o la impresora.

Forma

OPTION NOT TAB

Notas

BASIC normalmente transforma el código 'tab' (&H09) en espacios, en número suficiente como para avanzar hasta el siguiente tope de tabulación (hay uno cada ocho columnas). La orden OPTION NOT TAB impide tal expansión, de modo que los caracteres 'tab' son enviados a la consola o la impresora sin modificación.

OPTION NOT TAB no afecta a la función de escritura TAB ni a las zonas de escritura de BASIC.

OPTION NOT TAB queda en vigor hasta la siguiente orden OPTION TAB o RUN.

Palabras clave asociadas

PRINT, LPRINT, WRITE, OPTION TAB, RUN

Establecer intercepción para la salida por la consola. Orden

Aplicación

Designar una subrutina en código de máquina que será la encargada de enviar los caracteres a la consola. Esa subrutina será invocada en lugar de la del sistema operativo.

Forma

OPTION PRINT = *expresión-de-dirección,expresión-de-dirección*
OPTION PRINT

La *expresión-de-dirección* da la dirección de la subrutina en código de máquina.

Notas

La primera forma de esta orden establece la dirección de la nueva subrutina. La segunda forma (y la orden RUN) cancela la anterior definición y hace que BASIC vuelva a utilizar la subrutina del sistema operativo.

Los requisitos de interfaz de la subrutina son muy estrictos. Cualquier desviación con respecto a esos requisitos causará el fracaso de BASIC. La subrutina puede utilizar una pequeña porción de la pila.

En Mallard-80 la subrutina debe cumplir las siguientes especificaciones:

Entrada:	C contiene el carácter que se va a enviar a la consola
Salida:	A, BC, DE, HL y todos los indicadores pueden quedar alterados.

En Mallard-86:

La *expresión-de-dirección* da la componente de segmento de la dirección de la subrutina. Esta componente es la que esté en vigor en el momento de ejecutarse la orden OPTION PRINT y habrá sido establecida con DEF SEG. El interfaz para la subrutina es:

Entrada:	CL contiene el carácter que se va a enviar a la consola CS contiene la componente de segmento de la dirección de la subrutina DS, ES y SS contienen la base del segmento de datos de BASIC
Salida:	AX y los indicadores pueden quedar alterados Todos los demás registros deben ser preservados

Esta rutina se invoca con una 'llamada lejana', luego debe terminar con un 'retorno lejano'.

Palabras clave asociadas

OPTION INPUT, OPTION LPRINT, PRINT, DEF SEG

OPTION RUN

Impedir que un programa de BASIC pueda ser interrumpido o suspendido. **Orden**

Aplicación

OPTION RUN inhibe el efecto de introducir control-C y control-S por el teclado.

Forma

OPTION RUN

Notas

Normalmente, control-C interrumpe el programa actualmente en curso y hace que BASIC vuelva al modo directo. Además, normalmente, control-S suspende la ejecución del programa hasta que se pulse otra tecla. OPTION RUN inhibe estas dos acciones. Por otra parte, dado que entonces BASIC no tiene que estar examinando periódicamente el teclado, la velocidad de ejecución aumenta considerablemente. (En las versiones de 'sólo ejecución', control-C y control-S no ejercen efecto en ningún caso.)

OPTION RUN interfiere con INKEY$. Normalmente, INKEY$ no puede detectar ni control-C ni control-S, porque BASIC reacciona ante estos códigos aunque los reciba cuando está ejecutando INKEY$; si no fuera así, el efecto de estos códigos sería impredecible en los programas.

Después de una orden OPTION RUN, INKEY$ sí puede captar control-C y control-S, ya que entonces no hay posibilidad de confusión.

Por razones de compatibilidad, las versiones de 'sólo ejecución' ignoran control-C y control-S. Así pues, el único efecto de OPTION RUN y OPTION STOP es modificar la forma en que INKEY$ reacciona ante control-C y control-S. Cuando está en vigor OPTION STOP esos caracteres no son detectados. En el caso de OPTION RUN, INKEY$ los detecta y los entrega.

La situación de OPTION RUN/OPTION STOP no afecta a las órdenes STOP, END y SYSTEM, ni tampoco a la forma en que se esté gestionando los errores.

Palabras clave asociadas

INKEY$, INPUT$, OPTION STOP

OPTION STOP

Invierte el efecto de OPTION RUN. **Orden**

Aplicación

OPTION STOP rehabilita los efectos habituales de introducir control-C y control-S por el teclado cuando éstos han sido inhibidos por una orden OPTION RUN.

Forma

OPTION STOP

Notas

Cuando está en vigor OPTION STOP, INKEY$ no detecta los caracteres control-C y control-S.

La situación de OPTION RUN/OPTION STOP no afecta a las órdenes STOP, END y SYSTEM, ni tampoco a la forma en que se esté gestionando los errores.

Palabras clave asociadas

INKEY$, OPTION RUN

OPTION TAB

Activar la expansión de 'tab'. **Orden**

Aplicación

Volver a permitir que BASIC transforme los caracteres 'tab' (&H09) en cadenas de espacios cuando los escribe en la pantalla o la impresora.

Forma

OPTION TAB

Notas

BASIC normalmente transforma el código 'tab' (&H09) en espacios, en número suficiente como para avanzar hasta el siguiente tope de tabulación (hay uno cada ocho columnas). La orden OPTION NOT TAB impide tal expansión, de modo que los caracteres 'tab' son enviados a la consola o la impresora sin modificación.

OPTION TAB vuelve a activar la expansión de 'tab' cuando ésta ha sido inhibida por OPTION NOT TAB después de la última orden RUN.

Palabras clave asociadas

PRINT, LPRINT, WRITE, OPTION NOT TAB, RUN

Dar una indicación de error generada por el sistema operativo. Función

Aplicación

Identificar con mayor precisión un error 21 de BASIC.

Forma

OSERR

Notas

Algunos sistemas operativos pueden informar a BASIC de diversos errores. BASIC en lugar de manejar una tabla de errores distinta para cada sistema operativo, genera el error 21 y proporciona la función OSERR. Cuando se ha producido un error 21, la función OSERR da un número que depende del sistema operativo y que identifica el error.

El valor generado por OSERR está en el margen de 0 a 65535.

Palabras clave asociadas

ERR, ERL, ON ERROR GOTO

Escribir en una puerta de entrada/salida. Función

Aplicación

Escribir un valor es una puerta de E/S del microprocesador.

Forma

OUT *número-de-puerta,expresión-entera*
OUTW *número-de-puerta,expresión-de-dirección*

En Mallard-80 el *número-de-puerta* es una *expresión-entera* que debe dar un valor perteneciente al margen de 0 a 255.

En Mallard-86 el *número-de-puerta* es una *expresión-de-dirección*

En OUT, la *expresión-entera* da el byte de datos que se ha de enviar a la puerta especificada. La expresión debe tomar un valor perteneciente al margen de 0 a 255.

En OUTW, la *expresión-de-dirección* da la palabra de datos que se ha de enviar a la puerta especificada.

Notas

OUTW es «ilegal» en Mallard-80.

Palabras clave asociadas

INP, INPW, WAIT, WAITW

Examinar una posición de memoria. Función

Aplicación

Leer un byte en la memoria de la máquina.

Forma

PEEK(*expresión-de-dirección*)

Notas

PEEK genera un valor comprendido entre 0 y 255.

En Mallard-86 la *expresión-de-dirección* da la componente de desplazamiento de la dirección que se va a leer. La componente de segmento es la base del segmento de usuario establecida por la DEF SEG más reciente.

Palabras clave asociadas

POKE, DEF SEG

POKE

Escribir un valor en la memoria de la máquina. **Orden**

Aplicación

Escribir un dato en una posición de memoria especificada.

Forma

POKE *expresión-de-dirección,expresión-entera*

La *expresión-de-dirección* da la dirección de memoria en la que se va a escribir el dato.

La *expresión-entera* da el byte de datos que se va a escribir en la dirección especificada. Esta expresión tiene que dar un valor comprendido entre 0 y 255.

Notas

En Mallard-86 la *expresión-de-dirección* da la componente de desplazamiento de la dirección en que va a escribir. La componente de segmento es la base del segmento de usuario establecida por la DEF SEG más reciente.

Palabras clave asociadas

PEEK, DEF SEG

Posición en la consola. Función

Aplicación

Averiguar la posición actual de escritura en la consola.

Forma

POS(*expresión-numérica*)

donde el valor de la *expresión-numérica* es indiferente, por lo que se recomienda la forma POS(0).

Notas

POS da la posición lógica en la consola, que puede no coincidir con la posición física del cursor.

En el cálculo de la posición lógica no se incluyen los caracteres no imprimibles (los de valor menor que 32), a excepción de los siguientes:

- Retroceso (valor 8), que cuenta como −1, salvo cuando la posición es 1.
- Tab (valor 9), que se expande a espacios, cada uno de los cuales cuenta como 1. (Los topes de tabulación están dispuestos de forma que hay uno cada ocho columnas, si no se los define de otra manera.)
- Retorno del carro (valor 13), que restablece la posición lógica a 1.

La posición también depende de los retornos del carro insertados por BASIC cuando la posición lógica excede de la anchura de la consola (establecida con WIDTH).

Si se ha hecho la anchura infinita (con WIDTH 255), la posición lógica puede llegar a 255, pero no puede superar este valor.

Palabras clave asociadas

WIDTH LPRINT, LPOS

PRINT

Escribir en la consola. — Orden

Aplicación

Escribir datos, tanto numéricos como literales, en la consola.

Forma

PRINT *[lista-de-escritura][cláusula-using][separador]*

lista-de-escritura		es	*elemento[separador elemento]**
donde *elemento*	es		*expresión*
	o bien		SPC(*expresión-entera*)
	o bien		TAB(*expresión-entera*)
cláusula-using	es		USING *expresión-literal;lista-using*
donde *lista-using*	es		*expresión[separador expresión]**

separador es el signo de coma o de punto y coma.

(Obsérvese que la construcción *[. . .]** significa que el objeto es opcional y que puede ser incluido un número cualquiera de veces, o ninguna.)

Notas

En el capítulo 6 están descritas todas las funciones de escritura.

Los *elementos*, si los hay, son evaluados y escritos en formato libre. Una coma puesta a la derecha de un *elemento* hace que BASIC salte hasta el principio de la siguiente zona de escritura en cuanto termine de escribir el elemento. El signo de punto y coma sirve para separar los elementos, pero no ejerce ningún efecto sobre la salida.

Si se incluye una *cláusula-using*, la *expresión-literal* define una plantilla que controla el formato con el que escriben los valores de las expresiones mencionadas en la *lista-using*. Los elementos de esta lista pueden ir separados por comas o signos de punto y coma, ninguno de los cuales ejerce efecto sobre la salida.

Cuando ha terminado de escribir todos los argumentos, BASIC salta al principio de la línea siguiente, a menos que la orden PRINT haya terminado con un *separador*, un TAB o un SPC.

Palabras clave asociadas

LPRINT, POS, PRINT #, WIDTH, ZONE

Escribir en un fichero. Orden

Aplicación

Enviar datos a un fichero, con formatos elaborados igual que para escribir en la consola.

Forma

PRINT *#expresión-de-número-de-fichero[,argumentos]*

La *expresión-de-número-de-fichero* especifica a qué fichero se va a enviar los datos. El fichero tiene que estar abierto para salida secuencial o para acceso aleatorio.

Los *argumentos* tienen la misma forma que en la orden PRINT. Si se los omite, BASIC escribe un par retorno del carro/avance de línea.

Notas

En el capítulo 6 están descritas todas las funciones de escritura. PRINT # es equivalente a PRINT, con la única diferencia de que escribe en un fichero en vez de hacerlo en la consola; además, se considera que en el fichero las líneas son de anchura infinita. Los *argumentos* son tratados de la misma forma, si bien PRINT # no expande los caracteres 'tab' (&H09), los cuales no afectan a la posición lógica.

También es posible escribir en los ficheros de acceso aleatorio con PRINT #. Los caracteres se envían al tampón de registros aleatorios. BASIC mantiene un puntero interno asociado al tampón, gracias al cual sabe dónde debe escribir el siguiente dato. Este puntero se restablece a la posición inicial (principio del tampón) cuando se abre el fichero y, más tarde, cada vez que se ejecuta una orden GET o PUT. Se produce un error si se trata de sobrepasar el final del tampón cuando se está escribiendo en él.

Palabras clave asociadas

PRINT, WRITE #

Escribir datos en un registro de un fichero aleatorio o de acceso por claves.

Orden

Ampliada en las versiones Jetsam

Aplicación

Escribir el contenido del tampón de registros aleatorios en un registro dado de un fichero aleatorio o de un fichero de acceso por claves.

Forma

PUT *referencia-de-fichero,[número-de-registro]*
PUT *referencia-de-fichero,[número-de-registro],bloqueo*

La *referencia-de-fichero* especifica el fichero en el que se va a escribir; tiene que ser un fichero aleatorio o de acceso por claves que esté abierto.

Para ficheros aleatorios:

El *número-de-registro* especifica el registro que se va a escribir. Si no se especifica ninguno, se supone el siguiente al que intervino en la orden GET o PUT más reciente; si no se ha ejecutado ninguna orden GET o PUT antes que ésta, se escribe el registro 1.

El parámetro *bloqueo* si se lo incluye, especifica el tipo de bloqueo que se ha de aplicar al registro antes de escribirlo. Este parámetro sólo puede intervenir si la orden se aplica a un fichero aleatorio.

Para ficheros de acceso por claves:

El *número-de-registro* especifica el número del registro de datos que se va a escribir. Si se omite este parámetro, se utiliza el registro dado por la posición actual.

Nota (para ficheros aleatorios)

El tampón de registros aleatorios tiene asociado un puntero interno que permite escribir datos en el registro. La orden PUT reinicializa ese puntero.

Nota (para ficheros de acceso por claves)

El registro (o, en su defecto, el fichero) tiene que estar bloqueado en escritura.

Sólo se debe utilizar números de registro que hayan sido calculados por la función FETCHREC.

Nota (para ficheros aleatorios en versiones multiusuario)

PUT libera el bloqueo temporal en escritura que pudiera estar activo en el registro. Véase en el apartado 8.3.4 la descripción del mecanismo de bloqueo temporal de registros en escritura.

Palabras clave asociadas

PRINT #, WRITE #, GET

Cambiar el punto de partida de la sucesión de números pseudoaleatorios. **Orden**

Aplicación

El generador de números aleatorios de BASIC produce una sucesión de números pseudoaleatorios, cada uno de los cuales depende del anterior. Partiendo de una posición dada, la sucesión es siempre la misma. RANDOMIZE establece la posición inicial dentro de la sucesión, bien a un valor especificado, bien a un valor suministrado por el operador.

Forma

RANDOMIZE *[expresión-entera]*

Si se omite la *expresión-entera*, BASIC emite el mensaje:

```
Random Number Seed?
```

para pedir al operador que escriba el valor de la «semilla» y éste puede introducir un dato de cualquier tipo numérico.

Notas

El valor de la «semilla» suministrada a RANDOMIZE caracteriza plenamente la sucesión de números pseudoaleatorios. Por consiguiente, dos sucesiones iniciadas con la misma semilla serán idénticas.

Palabras clave asociadas

RND

RANKSPEC

Especificar si un rango puede contener claves repetidas. Función

Aplicación

Especificar si a cierto rango se le puede añadir claves cuyo valor sea idéntico al de alguna otra clave del rango.

Forma

RANKSPEC(*referencia-de-fichero,rango,únicas*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *rango* da el número del rango al que va a afectar la orden.

únicas es una *expresión-numérica*; si vale 0, está permitida la duplicidad de las claves; si vale 1, está prohibida.

Códigos generados

0	Éxito	Se ha asignado al *rango* la nueva propiedad *únicas*.
130	Fracaso	El fichero está bloqueado en lectura por otro usuario

Notas

La situación establecida por RANKSPEC queda almacenada en la cabecera del fichero, y permanece así hasta que se ejecute otra función RANKSPEC. Cuando se crea un fichero, está permitida la duplicidad de las claves mientras no se especifique lo contrario.

Si está prohibida la repetición de claves, ADDKEY y ADDREC fracasan y generan el código 116. La nueva clave no llega a ser añadida y la posición actual queda establecida en la clave que existía con el mismo valor.

RANKSPEC no comprueba la posible existencia de claves repetidas, sino que sólo afecta a la futura incorporación de claves.

La función RANKSPEC marca el fichero como inconsistente. Éste vuelve a ser consistente si se lo cierra normalmente o si se lo marca como tal mediante la función CONSOLIDATE.

Palabras clave asociadas

ADDKEY, ADDREC.

Leer datos en sentencias DATA. **Orden**

Aplicación

READ toma datos de las sentencias DATA y los asigna a variables.

Forma

READ *lista-de: variable*

Notas

Todas las sentencias DATA de un programa, consideradas en orden de números de línea ascendentes, forman una lista de constantes que pueden ser asignadas a variables mediante la orden READ. Cada vez que se lee un dato, READ avanza hasta el dato siguiente y lo asigna a la variable siguiente. La constante tiene que ser compatible con la variable a la cual se la asigna. Se produce un error (error 4) si se intenta leer más datos que los incluidos en líneas DATA.

Palabras clave asociadas

DATA, RESTORE

Observación. Orden

Aplicación

Insertar comentarios en los programas de BASIC.

Forma

REM *resto-de-la-línea*

BASIC ignora todo lo que hay entre la palabra clave REM y el final de la línea de programa (incluidos los posibles signos de dos puntos).

Notas

Las «órdenes» REM no tienen más efecto que hacer que el programa salte a la primera instrucción de la línea siguiente. Es permisible transferir el control a una orden REM (mediante GOTO o GOSUB).

En lugar de REM: se puede poner un apóstrofo (') para preceder al comentario, salvo en:

- Sentencias DATA, donde el apóstrofo sería considerado como parte de una cadena sin comillas.
- Las órdenes DEL, ERA, DIR y TYPE, en las que se considera que todo el resto de la línea constituye el parámetro de la orden.

En algunos lenguajes la función que calcula el resto de la división entera se llama 'REM' (en BASIC es MOD). Esto da lugar a confusión y produce numerosos errores de sintaxis en los programas de BASIC.

Cambiar el nombre de un fichero. Orden

Forma

En sistemas operativos de tipo CP/M:

REN *nombre-nuevo=nombre-antiguo*

En sistemas operativos de tipo MS-DOS (PC-DOS):

REN *nombre-antiguo nombre-nuevo*

El *nombre-antiguo* es un *nombre-de-fichero* que especifica el fichero cuyo nombre se va a cambiar.

El *nombre-nuevo* es un *nombre-de-fichero* que especifica el nombre que se va a dar al fichero.

Notas

El fichero objeto de la orden tiene que existir. No puede haber otro fichero con el nombre nuevo.

REN considera que todo el resto de la línea es su argumento, a pesar de los posibles signos de dos puntos y apóstrofos.

No se recomienda cambiar el nombre de un fichero que esté abierto. En los sistemas multiusuario esta orden puede fracasar si el fichero está siendo utilizado por otros usuarios o está bloqueado.

Palabras clave asociadas

NAME

RENUM

Renumerar el programa actual. **Orden**

No disponible en versiones de 'sólo ejecución'

Aplicación

Cambiar sistemáticamente los números de todas las líneas del programa o de parte de ellas.

Forma

RENUM
RENUM *número-de-línea-nuevo*
RENUM *[número-de-línea-nuevo],número-de-línea-antiguo*
RENUM *[número-de-línea-nuevo],[número-de-línea-antiguo],incremento*

RENUM sin parámetros equivale a RENUM 10,,10.

número-de-línea-nuevo es el *número-de-línea* que se va a dar a la primera línea renumerada. Si se omite este parámetro se le supone el valor 10.

número-de-línea-antiguo es el *número-de-línea* de la primera línea que va a ser renumerada. Si se omite este parámetro la renumeración empieza en la primera línea del programa.

incremento es un *número-de-línea* que especifica cómo se va a espaciar los nuevos números de línea.

Notas

RENUM modifica también las referencias a números de línea concretos que se hayan dentro de las instrucciones del programa. En concreto, las mencionadas en las siguientes órdenes:

> ELSE, GOSUB, GOTO, LIST, LLIST, RESTORE, RESUME, RUN, THEN, DELETE y la cláusula DELETE en CHAIN MERGE

(véase también la descripción de ERL, donde se explica su relación con RENUM).

Cuando se renumera parte del programa, RENUM rechaza los intentos de invertir el orden de las líneas y de utilizar como números de línea otros ya utilizados en la parte del programa que no se va a renumerar.

Si RENUM descubre referencias a líneas no existentes, emite un mensaje del tipo:

> Undefined line *99999* in *88888* (Línea indefinida ... en ...)

donde *99999* es el número de línea al que se hace referencia y que no existe
y *88888* el número (nuevo) de la línea donde aparece la referencia.

Cuando termina de ejecutar una orden RENUM, BASIC entra en el modo directo.

Reinicializar el sistema de disco. Orden

Aplicación

Reinicializar el sistema de disco, de forma que se pueda cambiar unos discos por otros.

Forma

RESET *[expresión-literal]*

Notas

La *expresión-literal* opcional indica qué unidad debe ser reinicializada; se cierran todos sus ficheros. El valor de la *expresión-literal* tiene que ser una letra del margen de A a P; no puede ser la cadena vacía.

En ausencia de parámetros, RESET reinicializa todas las unidades y cierra todos los ficheros.

Si hay algún fichero de acceso por claves abierto, queda marcado como inconsistente. Por consiguiente, antes de ejecutar una orden RESET se debe cerrar explícitamente los ficheros de acceso por claves que estén abiertos en la unidad afectada.

Esta orden puede fracasar en los sistemas multiusuario si algún otro usuario está utilizando los ficheros, pues este hecho hace imposible la reinicialización del sistema de disco.

Palabras clave asociadas

CLOSE

Modificar la posición del puntero de DATA. Orden

Aplicación

Cambia la posición del puntero de datos para que señale un dato especificado (en vez del dato siguiente al último leído).

Forma

RESTORE *[número-de-línea]*

El *número-de-línea* especifica la sentencia DATA a la que debe apuntar el puntero. Si se omite el parámetro, se supone la primera sentencia DATA del programa.

Notas

READ hace avanzar el puntero dato a dato según los va leyendo. RESTORE lleva el puntero a una línea especificada. El READ siguiente buscará, a partir de esa línea, una sentencia DATA y empezará a leer en ella.

Palabras clave asociadas

READ, DATA

Reanudar la ejecución después de procesar un error. **Orden**

Legal solamente en el modo de gestión de errores

Aplicación

Cuando un error ha sido interceptado por ON ERROR GOTO y procesado por una rutina de gestión de errores, se puede reanudar la ejecución del programa de diversas formas mediante la orden RESUME.

Formas

RESUME
RESUME *número-de-línea*
RESUME NEXT

Notas

La orden RESUME sólo es «legal» dentro del modo de gestión de errores (es decir, dentro de una rutina invocada por ON ERROR GOTO).

RESUME sin parámetro devuelve el control al principio de la sentencia en la que se detectó el error.

RESUME *número-de-línea* devuelve el control a la línea especificada.

RESUME NEXT devuelve el control a la sentencia inmediatamente posterior a aquélla en la que se detectó el error.

Palabras clave asociadas

ON ERROR GOTO

RETURN

Retornar de una subrutina. **Orden**

Aplicación

Señala el final de una subrutina. Hace que la ejecución del programa continúe en la instrucción siguiente al GOSUB que invocó la subrutina.

Forma

RETURN

Notas

Es permisible retornar de una subrutina desde el interior de un bucle WHILE o FOR. RETURN abandona los bucles que se haya establecido después del GOSUB que invocó la subrutina.

Palabras clave asociadas

GOSUB, ON x GOSUB

Extraer parte de una cadena por la derecha. Función

Aplicación

Copiar un número de caracteres dado del extremo derecho de una cadena dada.

Forma

RIGHT$(*expresión-literal,expresión-entera*)

La *expresión-literal* da la cadena de la cual se va a tomar los caracteres.

La *expresión-entera* da el número de caracteres que se va a copiar; su valor tiene que estar en el margen de 0 a 255.

Notas

Si la longitud de la cadena dada es menor que el número de caracteres solicitados, RIGTH$ genera la cadena completa. Si no, se toma la parte derecha de la cadena hasta formar una cadena de la longitud especificada.

Palabras clave asociadas

MID$, LEFT$

Número aleatorio. Función

Aplicación

Obtener el siguiente número aleatorio de la sucesión actual, repetir el último número generado u obtener el primero de una nueva sucesión.

Forma

RND*[(expresión-numérica)]*

Notas

RND da un valor de precisión sencilla, tal que $0 \leqslant$ valor < 1. El generador de números pseudoaleatorios genera cada número basándose en el anterior, de modo que cualquier sucesión es reproducible si se parte del mismo valor inicial.

Si se omite la *expresión-numérica*, o si ésta da un valor mayor que 0, se obtiene el siguiente número de la sucesión actual.

Si el valor de la *expresión-numérica* es 0, se repite el último número generado.

Si el valor de la *expresión-numérica* es menor que 0, se genera el primer número de una nueva sucesión. La nueva sucesión, aunque aleatoria a todos los efectos prácticos, es predecible a través de ese valor.

Palabras clave asociadas

RANDOMIZE

Redondear un valor. Función

Aplicación

Redondear un valor a un número de cifras decimales especificado o a una potencia de 10 especificada.

Forma

ROUND(*expresión-numérica[,decimales]*)

La *expresión-numérica* da el valor que ha de ser redondeado; puede ser de cualquier tipo (numérico).

El parámetro opcional *decimales* especifica el número de cifras decimales con que se expresará el resultado del redondeo. Si se lo incluye, *decimales* es una *expresión-entera* que da un valor perteneciente al margen de −39 a +39.

Notas

El valor de *expresión-numérica* se redondea al número de cifras decimales especificado por *decimales* de la siguiente forma:

- *decimales* mayor que 0

  el valor se redondea de modo que a la derecha del punto decimal queden las cifras especificadas.

- *decimales* igual a 0 o ausente

  el valor se redondea a un entero.

- *decimales* menor que 0

  el valor se redondea de modo que tenga ABS(*decimales*) ceros a la izquierda del punto decimal.

(Redondear, en este contexto, es «redondear al más próximo»; véase el apartado 2.9.1.)

Palabras clave asociadas

INT, FIX, CINT

Asignar una cadena a otra, justificando a la derecha. Orden

Aplicación

Sustituir el contenido de una cadena por otra, rellenando con espacio por la izquierda hasta completar la longitud original.

Forma

RSET *variable-literal=expresión-literal*

Notas

Se evalúa la *expresión-literal* y luego se la fuerza a tener la misma longitud que el valor de la *variable-literal* dada; para ello puede ser necesario rellenar con espacios por la izquierda o ignorar los caracteres sobrantes por la derecha. El resultado de esta operación sustituye el contenido original de la *variable-literal*.

RSET se utiliza, en particular, para dar nuevos valores a los campos de un registro aleatorio.

Palabras clave asociadas

FIELD, LSET, MID$, MKI$, MKS$, MKD$

RUN nombre-de-fichero

Cargar y ejecutar un programa. Orden

Aplicación

Cargar un programa desde un fichero del disco e iniciar su ejecución.

Forma

RUN *expresión-literal*[,R]

La *expresión-literal* da el nombre del fichero que contiene el programa. Si no se especifica distintivo de tipo, se supone ".BAS".

En la *expresión-literal* se puede incluir la especificación de la unidad de disco (por ejemplo, B:) como prefijo.

Notas

Esta orden borra de la memoria el programa actual, las funciones de usuario, las variables y las matrices. Restablece a su situación inicial las caractrísticas establecidas con DEFINT, DEFSNG, DEFDBL, DEFSTR, OPTION BASE, OPTION TAB, OPTION PRINT, OPTION LPRINT, OPTION INPUT y OPTION STOP.

RUN *nombre-de-fichero* es equivalente a LOAD *nombre-de-fichero* seguida de RUN.

RUN *nombre-de-fichero*,R es equivalente a LOAD *nombre-de-fichero*,R.

Palabras clave asociadas

LOAD, CHAIN, SAVE, CHAIN MERGE, MERGE

RUN [número-de-línea]

Ejecutar el programa actual. **Orden**

Aplicación

Iniciar la ejecución del programa actual, desde el principio o desde una línea especificada.

Forma

RUN *[número-de-línea]*

Si se omite el *número-de-línea*, la ejecución comienza por la primera línea del programa.

Notas

Esta orden borra de la memoria el programa actual, las funciones de usuario, las variables y las matrices. Restablece a su situación inicial las características establecidas con DEFINT, DEFSNG, DEFDBL, DEFSTR, OPTION BASE, OPTION TAB, OPTION PRINT, OPTION LPRINT, OPTION INPUT y OPTION STOP.

Grabar el programa en el disco. Orden

Aplicación

Grabar en el disco el programa que está actualmente en la memoria. Hay tres formas posibles de ficheros de programa.

Forma

SAVE *expresión-de-nombre-de-fichero[,especificador-de-forma]*

La *expresión-de-nombre-de-fichero* da el nombre del fichero en que se va a escribir el programa. Si no se incluye el distintivo de tipo, se supone ".BAS". Si ya existe un fichero con ese nombre, se lo borra.

En la *expresión-literal* se puede incluir la especificación de la unidad de disco (por ejemplo, B:) como prefijo.

El *especificador-de-forma* puede ser uno de los siguientes:

A, que especifica 'forma ASCII'
P, que especifica 'forma protegida'

Si no se incluye ningún *especificador-de-forma*, el fichero se graba con 'forma estándar'.

Notas

Los ficheros grabados en 'forma estándar' contiene el programa de BASIC codificado de tal manera que sólo sirven para volver a cargar el programa en la memoria desde BASIC.

Los ficheros grabados en 'forma ASCII' contienen el programa de BASIC con la misma forma que en los listados. Estos ficheros pueden ser procesados por otros programas; por ejemplo, editores de texto.

Los ficheros grabados en 'forma protegida' contienen el programa de BASIC en una versión similar a la de la 'forma estándar', pero cifrada de tal manera que BASIC rechaza toda orden que trate de acceder al texto del programa una vez que éste ha sido cargado.

Palabras clave asociados

LOAD, RUN, CHAIN, CHAIN MERGE, MERGE

Buscar una clave dada. **Función de Jetsam**

Aplicación

Llevar la posición actual a la primera clave que tiene el valor dado y está en el rango especificado.

Forma

SEEKKEY(*referencia-de-fichero,bloqueo,rango,valor-de-clave*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al registro correspondiente a la la primera clave que tenga el valor dado y esté en el rango dado.

Códigos generados

0	Éxito	Se ha establecido la posición actual en la primera clave que tiene el valor requerido y está en el rango especificado; se ha bloqueado el registro correspondiente según se había solicitado.
103	Fracaso	No existe ninguna clave con el valor especificado en el rango especificado, ni tampoco ninguna clave con valor superior en ese rango ni en ningún otro de número mayor. La posición actual ha quedado indefinida.
105	Fracaso	No existe ninguna clave con el valor especificado en el rango especificado. La posición actual ha quedado establecida en la clave que sería la siguiente a la buscada si ésta hubiera existido.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Notas

La función SEEKKEY permite el acceso aleatorio a los registros de datos a través de sus claves.

Palabras clave asociadas

SEEKRANK, SEEKREC, SEEKNEXT, SEEKSET, SEEKPREV

SEEKNEXT

Buscar la siguiente clave. **Función de Jetsam**

Aplicación

A partir de la posición actual, avanzar hasta la siguiente clave.

Forma

SEEKNEXT(*referencia-de-fichero,bloqueo[,posición-en-índice]*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al siguiente registro.

La *posición-en-índice* especifica la clave a partir de la cual se va a avanzar. Si se omite este parámetro, se supone la posición actual.

Códigos generados

0	Éxito	La nueva posición actual está en el mismo conjunto de claves (igual valor y rango) que la anterior.
101	Éxito	La nueva posición no está en el mismo conjunto de claves (tiene distinto valor) pero sí en el mismo rango que la anterior.
102	Éxito	La nueva posición está en el siguiente rango que existe.
103	Fracaso	La posición actual o la *posición-en-índice* especificada es la última del último rango. La posición actual ha quedado indefinida.
105	Fracaso	No existe la posición actual (o la *posición-en-índice*). La posición actual ha quedado establecida en la primera clave del conjunto de claves que estaría inmediatamente a continuación de la posición especificada si ésta hubiera existido. Pero si no existe tal posición nueva, la posición actual ha quedado indefinida.
115	Fracaso	Se ha omitido el parámetro *posición-en-índice* y la posición actual ha quedado indefinida.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Nota

La función SEEKNEXT permite el acceso secuencial a los registros de datos a través de sus claves.

Palabras clave asociadas

SEEKRANK, SEEKKEY, SEEKREC, SEEKSET, SEEKPREV

SEEKPREV

Buscar la clave anterior. **Función de Jetsam**

Aplicación

A partir de la posición actual, retroceder hasta la clave anterior.

Forma

SEEKPREV(*referencia-de-fichero,bloqueo[,posición-en-índice]*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al registro anterior.

La *posición-en-índice* especifica la clave a partir de la cual se va a retroceder. Si se omite este parámetro, se supone la posición actual.

Códigos generados

0	Éxito	La nueva posición actual está en el mismo conjunto de claves (igual valor y rango) que la anterior.
101	Éxito	La nueva posición no está en el mismo conjunto de claves (tiene distinto valor) pero sí en el mismo rango que la anterior.
102	Éxito	La nueva posición está en el rango anterior que existe.
103	Fracaso	La posición actual o la *posición-en-índice* especificada es la primera del primer rango. La posición actual ha quedado indefinida.
105	Fracaso	No existe la posición actual (o la *posición-en-índice*). La posición actual ha quedado establecida en la primera clave del conjunto de claves que estaría inmediatamente a continuación de la posición especificada si ésta hubiera existido. Pero si no existe tal posición nueva, la posición actual ha quedado indefinida.
115	Fracaso	Se ha omitido el parámetro *posición-en-índice* y la posición actual ha quedado indefinida.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Nota

La función SEEKPREV permite el acceso secuencial a los registros de datos a través de sus claves.

Palabras clave asociadas

SEEKRANK, SEEKKEY, SEEKREC, SEEKNEXT, SEEKSET

Buscar la primera clave del rango dado. Función de Jetsam

Aplicación

Llevar la posición actual a la primera del rango especificado.

Forma

SEEKRANK(*referencia-de-fichero,bloqueo,rango*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al registro correspondiente a la primera clave que esté en el rango dado.

Códigos generados

0	Éxito	Se ha establecido la posición actual en la primera clave que está en el rango especificado; se ha bloqueado el registro correspondiente según se había solicitado.
102	Fracaso	No existe ninguna clave en el rango especificado. La posición actual ha quedado establecida en la primera clave del siguiente rango que existe.
103	Fracaso	No existe ninguna clave en el rango especificado, ni tampoco en ningún otro rango de número mayor. La posición actual ha quedado indefinida.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Nota

La función SEEKRANK deja a Jetsam preparado para procesar secuencialmente los datos cuyas claves están en el rango.

Palabras clave asociadas

SEEKKEY, SEEKREC, SEEKNEXT, SEEKSET, SEEKPREV

Establecer la posición actual. — Función de Jetsam

Aplicación

Establecer la posición actual a partir de datos recordados por el programa.

Forma

SEEKREC(*referencia-de-fichero,bloqueo[,posición-en-índice]*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se ha de aplicar al registro.

La *posición-en-índice* especifica la clave en la que se va a situar la posición actual. Si se omite este parámetro, se supone la posición actual.

Códigos generados

0	Éxito	Se ha establecido la posición actual según se había especificado; se ha aplicado al registro el bloqueo solicitado.
103	Fracaso	No existe ninguna clave con el valor y número de registro especificados en el rango especificado, ni tampoco ninguna clave con valor superior en ese rango ni en ningún otro de número mayor. La posición actual ha quedado indefinida.
105	Fracaso	No existe ninguna clave con el valor y número de registro especificados en el rango especificado. La posición actual ha quedado establecida en la primera clave del conjunto de claves que estaría inmediatamente a continuación de la posición especificada si ésta hubiera existido.
115	Fracaso	Se ha omitido el parámetro *posición-en-índice* y la posición actual ha quedado indefinida.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Nota

La función SEEKREC sirve para restablecer la posición actual en una posición recordada por el programa.

Palabras clave asociadas

SEEKRANK, SEEKKEY, SEEKNEXT, SEEKSET, SEEKPREV

Buscar la siguiente clave que sea distinta de la actual. Función de Jetsam

Aplicación

A partir de la posición actual, avanzar hasta la primera clave del siguiente conjunto (es decir, hasta la primera clave que tenga valor distinto del de la actual).

Forma

SEEKSET(*referencia-de-fichero,bloqueo[,posición-en-índice]*)

La *referencia-de-fichero* da el número de un fichero de acceso por claves abierto.

El parámetro *bloqueo* especifica el tipo de bloqueo que se hace aplicar al nuevo resgistro actual.

La *posición-en-índice* especifica la clave a partir de la cual se va a avanzar. Si se omite este parámetro, se supone la posición actual.

Códigos generados

101	Éxito	La nueva posición actual está en el mismo rango que la anterior.
102	Éxito	La nueva posición está en el siguiente rango que existe.
103	Fracaso	La posición actual o la *posición-en-índice* especificada está en el último rango del índice. La posición actual ha quedado indefinida.
105	Fracaso	No existe la posición actual (o la *posición-en-índice*). La posición actual ha quedado establecida en la primera clave del conjunto de claves que estaría inmediatamente a continuación de la posición especificada si ésta hubiera existido. Pero si no existe tal posición nueva, la posición actual ha quedado indefinida.
115	Fracaso	Se ha omitido el parámetro *posición-en-índice* y la posición actual ha quedado indefinida.
132	Fracaso	No se ha podido aplicar el bloqueo en lectura solicitado para el registro.
133	Fracaso	No se ha podido aplicar el bloqueo en escritura solicitado para el registro.

Notas

La función SEEKSET es similar a SEEKNEXT, solo que ésta no se salta las claves de igual valor.

Palabras clave asociadas

SEEKRANK, SEEKKEY, SEEKREC, SEEKSET, SEEKPREV

Signo de un número. Función

Aplicación

Averiguar el signo del número dado.

Forma

SGN(*expresión-numérica*)

Notas

SGN da alguno de los siguientes valores enteros:

−1, si la *expresión-numérica* es menor que 0
0, si la *expresión-numérica* es igual a 0
+1, si la *expresión-numérica* es mayor que 0

Palabras clave asociadas

ABS

Seno. Función

Aplicación

Calcular el seno de un ángulo dado en radianes.

Forma

SIN(*expresión-numérica*)

La *expresión-numérica* da el ángulo, expresado en radianes; su valor debe estar en el margen de (aprox.) −200000 a +200000.

Notas

Cuando los ángulos son mucho mayores que 2*PI, la precisión de esta función se deteriora como consecuencia de la necesaria transformación del ángulo para que quede en el margen de −PI a +PI. En vez de dar un resultado inexacto, BASIC rehusa calcular el seno cuando el argumento está fuera del margen antes citado.

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado es también un número de precisión sencilla.

Palabras clave asociadas

COS, TAN, ATN

Cadena de espacios. Función

Aplicación

Crear una cadena de espacios de longitud especificada.

Forma

SPACE$(*expresión-entera*)

donde la *expresión-entera* debe dar un valor del margen de 0 a 255 y especifica el número de espacios de que ha de constar la cadena. (Si ese número es cero se obtiene la cadena vacía.)

Palabras clave asociadas

SPC, TAB, STRING$

Escribir el número de espacios especificado. Función de escritura

Aplicación

En una orden de escritura, SPC escribe el número de espacios dado.

Forma

SPC(*número-de-espacios*)

número-de-espacios es una *expresión-entera*. Si el *número-de-espacios* es negativo, se supone el valor cero. Si es mayor que la anchura del dispositivo, se lo hace encajar en el margen de 1 a *anchura-de-dispositivo* restándole la anchura las veces que sea necesario. El valor resultante da el número de espacios que se debe escribir.

Notas

Las funciones SPC sólo pueden ser utilizadas como elementos de una instrucción de escritura.

Se escribe el número de espacios especificado, empezando en todos los casos en la posición actual. Esto no es lo mismo que escribir una cadena SPACE$, pues BASIC comprueba si la cadena SPACE$ cabe en lo que queda de la línea y, si no cabe, escribe un retorno del carro antes de los espacios.

SPC no necesita ir seguida de una coma ni de un signo de punto y coma; siempre se supone el punto y coma, aun en el caso de que SPC sea el último elemento de la orden de escritura.

Palabras clave asociadas

LPRINT, PRINT, PRINT #, SPACE$, TAB

Raíz cuadrada. Función

Aplicación

Calcular la raíz cuadrada de un número dado.

Forma

SQR(*expresión-numérica*)

donde la *expresión-numérica* debe dar un número igual o mayor que cero.

Nota

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado es también un número de precisión sencilla.

STOP

Detener la ejecución. **Orden**

Aplicación

Interrumpir la ejecución del programa, pero de forma tal que puede ser reanudada más tarde. Esta posibilidad se puede aprovechar para detener el programa en algún punto de interés durante el proceso de depuración.

Forma

STOP

Notas

Cuando ejecuta la orden STOP, BASIC detiene el programa y emite el mensaje "Break". (En las versiones de 'sólo ejecución' STOP equivale a END.)

La ejecución se puede reanudar con la orden CONT, siempre que no se haya modificado el programa después de la interrupción.

Palabras clave asociadas

CONT, END

Representación literal de un valor numérico. Función

Aplicación

Convertir un valor numérico dado en una representación en forma de cadena literal.

Forma

STR$(*expresión-numérica*)

Notas

El valor de la *expresión-numérica* se convierte en una cadena decimal de la misma forma que las utilizadas por la orden PRINT. Los valores positivos producen cadenas que llevan un espacio por la izquierda, mientras que en las producidas por los valores negativos ese espacio está ocupado por un signo menos.

Palabras clave asociadas

VAL, PRINT, DEC$, HEX$, OCT$

Cadena formada por un carácter repetido. Función

Aplicación

Construir una cadena repitiendo un carácter dado las veces especificadas. El carácter se puede especificar por su número.

Forma

STRING$(*expresión-entera,especificador-de-carácter*)

La *expresión-entera* da la longitud que ha de tener la cadena; su valor tiene que estar en el margen de 0 a 255.

El *especificador-de-carácter* puede ser alguno de los siguientes:

expresión-entera, que especifica CHR$(*expresión-entera*)
expresión-literal, que especifica el carácter que se repite para formar la cadena.

Palabras clave asociadas

SPACE$

Suprimir el bit 7 de todos los caracteres de una cadena. Función

Aplicación

Crear una cadena que es copia de la cadena dada pero poniendo a 0 el bit 7 de todos sus caracteres.

Forma

STRIP$(*expresión-literal*)

Notas

El resultado es el valor de la *expresión-literal* con todos los caracteres forzados al margen normal ASCII (valores del 0 al 127). Esta función puede servir para suprimir los marcadores que utilicen el bit más significativo.

Palabras clave asociadas

LOWER$, UPPER$

Intercambiar el valor de dos variables. **Orden**

Aplicación

Intercambiar el contenido de dos variables sin necesidad de usar una variable intermedia.

Forma

SWAP *variable,variable*

Las dos variables tienen que ser del mismo tipo.

Volver al nivel del sistema. Orden

Aplicación

Abandonar BASIC y volver al nivel del sistema.

Forma

SYSTEM

Notas

Esta orden cierra todos los ficheros.

Escribir espacios hasta alcanzar la posición especificada. **Función de escritura**

Aplicación

En una orden de escritura, TAB escribe los espacios necesarios para llegar a la posición dada.

Forma

TAB(*posición-de-escritura*)

posición-de-escritura es una *expresión-entera*. Si la *posición-de-escritura* es menor que 1, se le supone el valor 1. Si es mayor que la anchura del dispositivo, se lo hace encajar en el margen de 1 a *anchura-de-dispositivo* restándole la anchura las veces que sea necesario. El valor resultante da la posición a la que se debe llegar.

Notas

Las funciones TAB sólo pueden ser utilizadas como elementos de una instrucción de escritura.

Si la posición solicitada es igual o mayor que la posición actual, se escribe el número de espacios necesario (puede no requerirse ninguno).

Si la posición especificada es menor que la actual, BASIC salta a la línea siguiente y luego escribe los espacios necesarios para llegar a la posición solicitada.

TAB no necesita ir seguida de una coma ni de un signo de punto y coma; siempre se supone el punto y coma, aun en el caso de que TAB sea el último elemento de la orden de escritura.

Palabras clave asociadas

LPRINT, PRINT, PRINT #, SPC

Tangente. Función

Aplicación

Calcular la tangente de un ángulo dado en radianes.

Forma

TAN(*expresión-numérica*)

La *expresión-numérica* da el ángulo, expresado en radianes; su valor debe estar en el margen de (aprox.) −200000 a +200000.

Notas

Cuando los ángulos son mucho mayores que 2*PI, la precisión de esta función se deteriora como consecuencia de la necesaria transformación del ángulo para que quede en el margen de −PI a +PI. En vez de dar un resultado inexacto, BASIC rehusa calcular la tangente cuando el argumento está fuera del margen antes citado.

La *expresión-numérica* puede ser de cualquier tipo numérico, pero BASIC fuerza su valor a representación en precisión sencilla antes de procesarlo. El resultado es también un número de precisión sencilla.

Palabras clave asociadas

COS, SIN, ATN

Activar y desactivar el seguimiento de la ejecución del programa. Orden

No disponible en las versiones de 'sólo ejecución'

Aplicación

BASIC permite ejecutar los programas en un modo de seguimiento que consiste en que escribe los números de las líneas inmediatamente antes de ejecutarlas.

Forma

TRON
TROFF

Notas

TRON activa el modo de seguimiento. TROFF lo desactiva.

Cuando está activado este modo, BASIC escribe el número de cada línea entre corchetes antes de ejecutarla.

El modo de seguimiento se desactiva automáticamente en cuanto se carga otro programa, es decir, cuando se ejecuta una orden NEW, LOAD, CHAIN o RUN"fichero".

TYPE

Mostrar en la consola el contenido de un fichero. Orden

Aplicación

Escribir en la consola el contenido del fichero especificado.

Forma

TYPE *nombre-de-fichero*

El nombre de fichero no puede contener «símbolos comodín».

Notas

Esta orden lee el fichero y lo envía directamente a la consola. Pulsando control-C se abandona la operación y se vuelve al modo directo (salvo en las versiones de 'sólo ejecución', en las que control-C se ignora).

Cuando el fichero está siendo enviado a la consola, pulsando control-S se suspende la salida (salvo en las versiones de 'sólo ejecución', en las que control-C se ignora). La salida se puede reanudar de la forma habitual (véase la sección 3.3).

TYPE considera que todo el resto de la línea es su argumento, a pesar de los posibles signos de dos puntos (:) o apóstrofos (').

Los tabuladores y retornos del carro se procesan como de costumbre; se respeta la anchura lógica de la consola. Todos los demás caracteres se envían a la consola sin modificación, tal como figuran en el fichero. Por consiguiente, en el fichero se puede incluir caracteres de control y caracteres ajenos al margen normal ASCII para producir efectos especiales en la consola.

Palabras clave asociadas

DISPLAY

Convertir cadena a mayúsculas. Función

Aplicación

Crea una cadena que es copia de otra, con todas las letras minúsculas convertidas a mayúsculas.

Forma

UPPER$(*expresión-literal*)

Notas

La cadena generada es la *expresión-literal*, con todas las letras del margen de a a z convertidas a sus correspondientes del margen de A a Z.

Palabras clave asociadas

LOWER$

Convertir entero sin signo. Función

Aplicación

Convertir en entero un valor interpretado como entero sin signo de 16 bits.

Forma

UNT(*expresión-de-dirección*)

Notas

UNT da un entero del margen de −32768 a +32767 que es el equivalente en complemento a dos del entero sin signo que se le entrega como argumento. Véase en la sección 2.10 una descripción de los enteros sin signo.

Palabras clave asociadas

INT, FIX, CINT, ROUND

Invocar una función externa. Función

Aplicación

Invoca una de las diez funciones externas posibles. Una función externa se define mediante una orden DEF USR, en la que se especifica la dirección de la correspondiente rutina en código de máquina.

Forma

USR *[dígito](expresión)*

El *dígito* puede ser cualquiera del margen de 0 a 9. Si se lo omite, se supone el 0. La función USR*dígito* tiene que haber sido definida antes en una orden DEF USR.

La *expresión* puede ser una expresión de cualquier forma.

Nota

Véase en el apéndice III un estudio general de las funciones y subrutinas externas.

Palabras clave asociadas

DEF USR, CALL, DEF SEG

Convertir una cadena en un valor numérico. Función

Aplicación

Convertir en valor numérico la representación literal de un número.

Forma

VAL(*expresión-literal*)

Notas

VAL realiza la conversión de cadena literal a número de la misma manera que INPUT. El resultado puede ser entero o en doble precisión (de modo que si la cadena contiene parte decimal, VAL da la máxima precisión posible).

Palabras clave asociadas

STR$

Puntero de variable. Función

Aplicación

Averiguar la dirección de una variable o de un tampón de fichero, para entregarla a una rutina o función externa.

Forma

VARPTR (*variable*)
VARPTR (*#expresión-de-número-de-fichero*)

La primera forma de la dirección de la variable. Si la variable es indexada, la dirección que se obtiene es la del elemento especificado.

La segunda forma da la dirección del tampón asignado al fichero especificado. El fichero tiene que estar abierto (de lo contrario, no tendrá tampón).

Notas

VARPTR da un número en precicisión sencilla del margen de 0 a 65535.

Véase en el apéndice III un estudio general de las funciones y subrutinas externas.

Cuando se la aplica a un fichero de acceso por claves, VARPTR da la dirección del primer byte reservado. Para obtener la dirección del primer byte de datos del usuario, se le debe sumar 2.

Palabras clave asociadas

CALL, USR, DEF USR, UNT

Averiguar qué versión de BASIC se está usando. Función

Aplicación

BASIC se esfuerza por ocultar las diferencias que hay entre los diversos procesadores y sistemas operativos con los que puede trabajar. En ocasiones un programa puede necesitar saber en qué versión de BASIC está funcionando. La función VERSION proporciona esa información.

Forma

VERSION(*expresión-entera*)

La *expresión-entera* tiene que dar un número entre 0 y 4, el cual especifica qué información se desea.

Nota

La función VERSION genera un número entero. Su significado depende del valor del parámetro:

Parámetro	Valor Generado	Significado
0	7	Versión para 8080/8085
	8	Versión para Z80
	16	Versión para 8088/8086
1	0	Intérprete de BASIC completo
	1	Intérprete de BASIC 'sólo ejecución'
2	1	Versión uniusuario
	2	Versión multiusuario para CP/M-86 concurrente (con o sin DR/Net)
3	1	BASIC de tipo CP/M
	2	BASIC de tipo MS-DOS (PC-DOS)
4	*	Información dependiente del sistema operativo

Esperar por un valor en una puerta de entrada/salida. Función

Aplicación

Esperar hasta que en la puerta de E/S dada aparezca un grupo de bits especificado.

Forma

WAIT *número-de-puerta*,máscara[,*inversión*]
WAITW *número-de-puerta*,máscara[,*inversión*]

En Mallard-80 el *número-de-puerta* es una *expresión-entera* que debe dar un valor perteneciente al margen de 0 a 255.

En Mallard-86 el *número-de-puerta* es una *expresión-de-dirección*.

En WAIT, tanto la *máscara* como la *inversión* son *expresiones-enteras*, con valores entre 0 y 255.

En WAITW, tanto la *máscara* como la *inversión* son *expresiones-de-dirección*.

Notas

BASIC entra en bucle de espera en el que lee incesantemente la puerta de E/S especificada. Con el valor leído y el parámetro *inversión* realiza la operación lógica 'OR exclusivo'; con ese resultado y el parámetro *máscara* realiza la operación lógica 'AND'; el proceso se repite hasta que el resultado final sea distinto de cero. (Si se omite el parámetro *inversión*, no se realiza la operación 'OR exclusivo'.) WAIT lee bytes en la puerta de E/S; WAITW lee palabras.

No hay forma de interrumpir desde fuera ese bucle, de modo que BASIC se queda en él indefinidamente si no se cumple la condición especificada.

WAITW es «ilegal» en Mallard-80.

Palabras clave asociadas

INP, INPW, OUT, OUTW

Señalar el final de un bucle WHILE. — Orden

Aplicación

Un bucle de tipo WHILE ejecuta reiteradamente una sección del programa hasta que se cumpla una condición dada. La orden WEND indica dónde termina esa sección del programa.

Forma

WEND

Notas

La forma en que WHILE y WEND están conectados se explica en la descripción de WHILE. Cuando BASIC encuentra una orden WEND, sabe a qué WHILE corresponde.

Se produce un error si se ejecuta un WEND que no tenga asociado ningún WHILE.

Palabras clave asociadas

WHILE

Iniciar un bucle WHILE. — Orden

Aplicación

Un bucle de tipo WHILE ejecuta reiteradamente una sección del programa hasta que se cumpla una condición dada. La orden WHILE señala el comienzo del bucle y especifica la condición. La orden WEND indica dónde termina el bucle.

Forma

WHILE *expresión-lógica*

Notas

Cuando BASIC ejecuta una orden WHILE, lo primero que hace es evaluar la *expresión-lógica*. Si el resultado es cero, salta hasta el WEND correspondiente. Si no es cero, ejecuta las instrucciones comprendidas entre WHILE y WEND; cuando llega a WEND salta hasta el WHILE y repite el proceso.

El bucle WHILE no termina inmediatamente por el hecho de que la *expresión-lógica* se haga falsa, sino la próxima vez que BASIC la comprueba cuando llega a WEND.

La orden WEND que corresponde a un WHILE dado se determina estáticamente cuando se ejecuta WHILE por primera vez. Esto quiere decir que la determinación de qué WEND corresponde a WHILE depende del orden de las sentencias del programa, no del orden de ejecución. Por consiguiente, es posible que un mismo WHILE tenga varios WEND asociados.

Los bucles WHILE pueden ser anidados, bien dentro de otros bucles WHILE, bien dentro de bucles FOR.

Se puede abandonar un bucle WHILE evitando el paso por WEND.

Palabras clave asociadas

WEND, FOR

WIDTH

Establecer la anchura de la consola. **Orden**

Aplicación

Informar a BASIC de cuál es la anchura de la consola (en número de caracteres). BASIC necesita esta información para generar nuevas líneas en el momento adecuado cuando está escribiendo en la consola.

Dado que en muchos sistemas la propia consola se encarga de generar las líneas, WIDTH puede servir para informar a BASIC de este hecho.

Forma

WIDTH *anchura[,columna-de-autoCRLF]*

anchura es una *expresión-entera* que da la anchura de la consola en caracteres; su valor tiene que estar entre 1 y 255 (los valores pequeños producen unos efectos curiosos).

El parámetro opcional *columna-de-autoCRLF* es una *expresión-entera* que da la posición de 'auto CRLF' de la consola (véase más abajo); su valor tiene que estar entre 1 y 255. Si se omite el parámetro, se conserva la situación anterior.

Notas

La situación actual de estos valores ha sido establecida en el programa de instalación (véase el apéndice V).

El valor 255 como *anchura* tiene un significado especial. BASIC trata la consola como infinitamente ancha, de modo que nunca genera líneas nuevas. BASIC mantiene un contador que da la posición lógica en la consola. Cuando la posición lógica alcanza 255, el contador ya no sigue incrementándose, de forma que las posiciones mayores que 255 son tratadas como 255. El editor de líneas no se deja afectar por el valor 255, sino que continúa operando con el último valor «real» (es decir, distinto de 255).

Si se hace la *columna-de-autoCRLF* igual a 255, BASIC supone que la consola no genera líneas nuevas. Cualquier otro valor indica la columna en la que la consola salta a la línea siguiente. BASIC no genera líneas nuevas cuando los valores de los dos parámetros son iguales. No es recomendable definir una posición de 'autoCRLF' menor que la anchura de la pantalla (a menos que ésta sea 255).

El valor de la *anchura* afecta a toda la salida por la consola.

Palabras clave asociadas

WIDTH LPRINT, PRINT, POS, EDIT

WIDTH LPRINT

Establecer la anchura de la impresora. **Orden**

Aplicación

Informar a BASIC de cuál es la anchura de la impresora (en número de caracteres). BASIC necesita esta información para generar nuevas líneas en el momento adecuado cuando está escribiendo en la impresora.

Forma

WIDTH LPRINT *expresión-entera*

La *expresión-entera* tiene que dar un valor comprendido entre 1 y 255 (los valores pequeños producen unos efectos curiosos).

Notas

El valor inicial de la anchura de la impresora es 132.

El valor 255 como anchura de la impresora tiene un significado especial. BASIC trata la impresora como infinitamente ancha, de modo que nunca genera líneas nuevas. BASIC mantiene un contador que da la posición lógica en la impresora. Cuando la posición lógica alcanza 255, el contador ya no sigue incrementándose, de forma que las posiciónes mayores que 255 son tratadas como 255.

El valor de la anchura afecta a toda la salida por la impresora.

Palabras clave asociadas

WIDTH, LPRINT, LPOS

WRITE

Escribir en la consola. **Orden**

Aplicación

Escribir en la consola los valores de varias expresiones, separándolos por comas y poniendo las cadenas literales entre comillas.

Forma

WRITE *[lista-de-escritura]*

lista-de-escritura es *expresión[separador-expresión]**

donde *separador* puede ser una coma o un signo de punto y coma, indistintamente.

(La construcción */...]** significa que el objeto es opcional y que puede ser incluido un número cualquiera de veces, o ninguna.)

Notas

WRITE es similar a PRINT pero:

- ignora las zonas de escritura
- escribe las cadenas literales entre comillas
- escribe una coma entre cada dos elementos
- en WRITE el separador final de la orden no tiene ningún efecto

Palabras clave asociadas

WRITE #, PRINT

WRITE#

Escribir en un fichero. **Orden**

Aplicación

Escribir en un fichero los valores de varias expresiones, separándolos por comas y poniendo las cadenas literales entre comillas. WRITE # escribe en los ficheros en una forma que es directamente legible por INPUT #.

Forma

WRITE *#expresión-de-número-de-fichero,[lista-de-escritura]*

La *expresión-de-número-de-fichero* especifica en qué fichero se va a escribir. El fichero tiene que estar abierto para salida secuencial o para acceso aleatorio.

lista-de-escritura es *expresión[separador-expresión]**

donde *separador* puede ser una coma o un signo de punto y coma, indistintamente.

(La construcción */...]** significa que el objeto es opcional y que puede ser incluido un número cualquiera de veces, o ninguna.)

Notas

WRITE # es similar a PRINT # pero:

- ignora las zonas de escritura
- escribe las cadenas literales entre comillas
- escribe una coma entre cada dos elementos
- en WRITE # el separador final de la orden no tiene ningún efecto
- si el fichero está abierto para acceso aleatorio, el registro se rellena con espacios (en número igual a la longitud del registro menos 1) antes de insertar el retorno del carro

Palabras clave asociadas

WRITE, PRINT #

ZONE

Establecer la anchura de las zonas de escritura. **Orden**

Aplicación

Cambiar la anchura de las zonas de escritura (utilizadas por las órdens PRINT, LPRINT y PRINT #).

Forma

ZONE *expresión-entera*

La *expresión-entera* da la nueva anchura de las zonas de escritura; su valor tiene que estar entre 1 y 255.

Notas

En ausencia de otra especificación, la anchura de las zonas de escritura es 15. La anchura se restablece a este valor implícito cada vez que se carga un programa, esto es, cada vez que se ejecuta una orden NEW, LOAD, CHAIN O RUN"fichero".

Palabras clave asociadas

PRINT, LPRINT, PRINT#, WIDTH

Inicialización de BASIC

Cuando se carga BASIC desde el sistema operativo, la línea de órdenes puede incluir varios parámetros:

[nombre-de-fichero][/F:*número-de-ficheros*][/M:*dirección*][/S:*tamaño*]

Si no se especifica ningún parámetro, BASIC se carga con los valores implícitos y entra en el modo directo.

Si se incluye el parámetro *nombre-de-fichero*, tiene que figurar en primer lugar. Cuando ha concluido el proceso de inicialización de BASIC, éste carga el programa que hay en el fichero mencionado e inicia su ejecución.

Los restantes parámetros pueden aparecer en cualquier orden:

/F: BASIC no puede manejar simultáneamente más que cierto número de ficheros; /F establece ese número máximo. El *número-de-ficheros* ha de estar en el margen de 0 a 255 (si bien puede no haber memoria suficiente para más de un centenar de ficheros). Si no se especifica /F, se establece ese máximo en 3.

Cada fichero requiere aproximadamente 176 bytes de memoria, más el tampón especificado por /S, a menos que el tampón sea de menos de 128, pues entonces se requiere un total de 304 bytes.

/M: Establece el límite de la memoria utilizable por BASIC. La *dirección* da la dirección del último byte (el más alto) que puede utilizar BASIC. Si no se especifica /M:, BASIC usa toda la memoria que puede conseguir:

Mallard-80 toma el valor que hay en las posiciones 6 y 7 como límite de su memoria.

Mallard-86 intenta conseguir del sistema operativo un banco de 64K entero para datos. Parte de él se dedicará a las variables de BASIC. Si se especifica /M:, la zona del segmento de datos posterior a la *dirección* no será utilizada por BASIC.

BASIC necesita como mínimo 1024 bytes para poder funcionar.

/S: Establece la longitud máxima de los registros aleatorios. El *tamaño* se expresa en bytes. Si no se especifica /S:, se supone una longitud máxima de registro de 128 bytes.

Los diversos números pueden estar expresados como enteros decimales sin signo (valor máximo: 65535) o como números octales o hexadecimales (con los prefijos &O y &H, respectivamente).

Todos estos valores pueden ser modificados desde BASIC mediante las órdenes MEMORY y CLEAR.

Números y mensajes de error

A continuación damos la lista, en orden numérico, de los errores generados por BASIC, junto con una somera descripción de las posibles causas.

1 Errores de BASIC ordinario

1 Unexpected NEXT

Se ha encontrado un NEXT que no corresponde a ningún FOR, o bien la variable mencionada tras NEXT no coincice con la de FOR.

2 Syntax Error

BASIC no puede entender una línea porque algo en ella no se ajusta a las reglas de sintaxis del lenguaje.

3 Unexpected RETURN

Se ha encontrado una orden RETURN fuera de la subrutina.

4 DATA exhausted

Una orden READ ha intentado leer más datos que los incluidos en líneas DATA.

5 Improper argument

Éste es un error no específico. Puede haber algo incorrecto en el argumento de una función o en un parámetro de una orden.

6 Overflow

El resultado de una operación aritmética ha sobrepasado los márgenes aceptables. Si ha ocurrido en un número en punto flotante, el resultado ha sido mayor que 1.7E+38 (aprox.). También puede haber ocurrido al intentar convertir un número en punto flotante en un entero con signo de 16 bits.

7 Memory full

El programa actual o sus variables ocupan demasiado espacio en la memoria, o bien el anidamiento de las estructuras de control es demasiado profundo (demasiados GOSUB, WHILE o FOR anidados); puede ser necesario asignar más espacio a la pila con las órdenes CLEAR o MEMORY.

Si se presenta este error cuando se está editando el programa, se puede recuperar espacio de memoria borrando las variables con CLEAR.

Si se presenta cuando se está tratando de bloquear un registro, se puede usar BUFFERS para aumentar el número máximo de bloqueos de registro por fichero. (El máximo implícito es 0.)

8 Line does not exist

Se ha mencionado un número de línea que no existe en el programa.

9 Subscript out of range

Uno de los subíndices mencionados en una lista es demasiado pequeño o demasiado grande.

10 Array already dimensioned

Una de las listas declaradas en una orden DIM ya ha sido dimensionada antes.

11 Division by zero

División por cero, que puede ocurrir en la división de números reales, en la división entera, en la operación MOD y en la exponenciación.

12 Invalid direct command

Se ha intentado dar como orden directa una orden que sólo es válida si va precedida de un número de línea.

13 Type mismatch

Incongruencia de tipos. Un dato es literal cuando tenía que ser numérico, o viceversa. O se ha introducido un dato numérico incorrectamente formado en READ o INPUT #.

14 String space full

Se han creado demasiadas cadenas literales y ya no hay espacio disponible para más, incluso después de reorganizar la memoria.

15 String too long

Una cadena literal consta de más de 255 caracteres. Puede haber sido generada por concatenación de otras.

16 String expression too complex

Expresión literal demasiado compleja. Las expresiones literales pueden generar valores intermedios; cuando el número de estos valores excede de un límite razonable, se produce este error.

17 Cannot CONTinue

No se puede reanudar la ejecución del programa con CONT. Obsérvese que esta orden reanuda el programa después de una orden STOP, control-C o un error, pero a condición de que no se modifique el programa durante la interrupción.

18 Unknown user function

Se ha invocado una FN sin la correspondiente DEF FN.

19 RESUME missing

Se ha llegado al final del programa mientras se ejecutaba una rutina de gestión de errores (esto es, una rutina invocada por ON ERROR GOTO).

20 Unexpected RESUME

RESUME sólo es válida en una rutina de gestión de errores (esto es, en una rutina invocada por ON ERROR GOTO).

21 O/S dependent error

Error dependiente del sistema operativo. La función OSERR da información más explícita.

22 Operand missing

BASIC ha encontrado una expresión incompleta.

23 Line too long

BASIC ha convertido a su forma interna una línea y ésta ha resultado ser demasiado larga.

26 NEXT missing

BASIC ha encontrado un FOR al que no corresponde ningún NEXT.

29 WEND missing

BASIC ha encontrado un WHILE al que no corresponde ningún WEND.

30 Unexpected WEND

Se ha encontrado un WEND que no corresponde a ningún WHILE, o bien el WEND no concuerda con el WHILE del bucle actual.

2 Errores relacionados con los discos y los ficheros

50 Record overflow

Se ha sobrepasado el final del registro en una orden FIELD, o en cualquier otra que lea o escriba en el tampón de registros aleatorios.

51 Internal error

Error interno. No debería aparecer nunca.

52 File number error

Se ha especificado un número de fichero que está fuera del margen permisible, o se ha intentado leer o escribir en un fichero que no está abierto.

53 File not found

No ha sido posible abrir el fichero especificado.

54 File type error

Se ha intentado realizar con un fichero una operación que es incompatible con la forma en que se abrió el fichero. Este error se presenta cuando Mallard BASIC está leyendo lo que supone que debe ser un fichero de programa y no reconoce su contenido.

55 File already open

El número del fichero ya está asignado a otro fichero.

57 Disc I/O error

Error de entrada/salida en el disco.

58 File already exists

Ya existe un fichero con el nombre nuevo especificado en NAME o REN.

La orden CREATE de Jetsam genera este error si ya existe el fichero de datos o el de índices.

61 Disc full

Disco lleno.

62 EOF met

Se ha intentado sobrepasar el final de un fichero secuencial al leerlo.

63 Record number error

Se ha especificado un número de registro que está fuera del margen permisible (1 a 32767), o bien se ha usado un número de registro imposible al tratar de acceder a un fichero de datos de Jetsam.

64 File name invalid

Nombre de fichero incorrecto.

66 Direct command found

Al cargar un fichero de programa, BASIC ha encontrado una orden directa (sin número de línea). Esto puede indicar que el fichero es un fichero de tipo ASCII generado desde fuera de BASIC y que no está en la forma adecuada para ser cargado como programa.

67 Directory full

Directorio lleno.

68 Read past EOF

Puede ocurrir en algunos sistemas operativos cuando se intenta bloquear un registro en el que nunca se ha escrito.

69 Record is locked

Una operación GET o PUT sobre un registro de un fichero aleatorio ha fracasado porque el registro ya está bloqueado y el bloqueo actual es incompatible con el solicitado.

70 Read-only disc

El disco es de 'sólo lectura'.

71 Read-only file

El fichero es de 'sólo lectura'.

72 Invalid drive

No existe la unidad de disco especificada.

73 File is locked

Una operación de bloqueo de un fichero ha fracasado porque el fichero ya está bloqueado y el bloqueo actual es incompatible con el solicitado.

74 RESET denied

El sistema operativo ha rechazado la orden RESET.

3 Errores específicos de Jetsam

113 Not a keyed file

Al abrir un fichero de acceso por claves, Jetsam no ha reconocido los registros de cabecera del fichero de datos o del de índices.

114 Record is not locked

Se ha intentado realizar una operación GET o PUT con un registro que no está adecuadamente bloqueado.

115 Inconsistent files

Al abrir un fichero de acceso por claves, Jetsam ha descubierto que los dos ficheros, de datos y de índices, no están marcados como consistentes. Esto indica que no se cerró (CLOSE) ni consolidó (CONSOLIDATE) el fichero después de escribir en él. El fichero es inservible; el usuario debe utilizar la copia de seguridad más moderna.

Rutinas externas

BASIC puede manejar dos tipos de rutinas externas: las invocadas por la orden CALL y las funciones USR.

Las rutinas han de estar cargadas en una zona de memoria distinta de la utilizada por BASIC. En Mallard-80 esa zona suele ser la inmediatamente posterior a la dirección más alta usada por BASIC. Mallard-86 permite que las rutinas estén en su propio segmento, o bien en el segmento de datos de BASIC por encima de la dirección más alta usada por BASIC.

La dirección más alta usada por BASIC puede ser establecida mediante la opción /M: de la orden de carga del lenguaje (véase el apéndice I), o bien, desde BASIC, mediante las órdenes CLEAR y MEMORY.

1 Formato de los datos

A continuación vamos a describir muy brevemente el formato de los cuatro tipos de datos. El método de almacenamiento es siempre con el byte menos significativo en la dirección más baja.

Entero. Valores de 16 bits en complemento a 2.

Literal. Las cadenas se almacenan en dos partes: el 'descriptor de la cadena' y el 'cuerpo de la cadena'. El descriptor es un vector de 3 bytes. El primero de ellos da la longitud de la cadena; si su valor es cero, la cadena es la cadena vacía y se debe ignorar el resto del descriptor. Los otros dos bytes dan la dirección en la que empieza el cuerpo de la cadena. (En Mallard-86 el cuerpo de la cadena está en el segmento de datos de BASIC.)

Precisión sencilla. Mantisa de 3 bytes seguida de un exponente de 1 byte. Los números se guardan siempre normalizados; el bit más significativo de la mantisa es reemplazado por el signo. El exponente está desplazado en 128. Si el exponente es cero, indica que el número es cero y que se debe ignorar la mantisa. La mantisa está en forma signo/valor absoluto, con el punto binario a la izquierda del bit más significativo. El primer byte de la mantisa es el menos significativo.

Doble precisión. Igual que en precisión sencilla, pero con cuatro bytes más para la mantisa.

2 Funciones USR

BASIC reserva diez nombres para otras tantas funciones externas: USR0 a USR9. La orden DEF USR declara la dirección de una función externa y la asigna a un nombre USR dado. Las funciones USR se invocan como las demás; llevan un parámetro, que puede ser de cualquier tipo. El valor generado por la función puede ser del mismo tipo que el parámetro o bien un entero.

Las condiciones de entrada de la función son las siguientes:

Mallard-80. El registro A indica el tipo del parámetro:

2 ⇒ Parámetro entero
El par HL contiene la dirección de un valor de 16 bits en complemento a 2.

3 ⇒ Parámetro literal
El par DE contiene la dirección del descriptor de la cadena (véase más arriba).

4 ⇒ Parámetro de precisión sencilla
El par HL contiene la dirección del número de 4 bytes.

8 ⇒ Parámetro de doble precisión.
El par HL contiene la dirección del bit 4 del número de 8 bytes (siendo el byte 0 el primero de ellos).

Mallard-86. El registro AL indica el tipo del parámetro:

2 ⇒ Parámetro entero
El registro BX contiene la dirección de un valor de 16 bits en complemento a 2.

3 ⇒ Parámetro literal
El registro DX contiene la dirección del descriptor de la cadena (véase más arriba).

4 ⇒ Parámetro de precisión sencilla
El registro BX contiene la dirección del número de 4 bytes.

8 ⇒ Parámetro de doble precisión
El registro BX contiene la dirección del bit 4 del número de 8 bytes (siendo el byte 0 el primero de ellos).

Todas las direcciones hacen referencia a posiciones dentro del segmento de datos de BASIC. Véase más abajo la descripción de los registros de segmento.

El estado de los demás registros (salvo los de segmento) es indefinido.

Se puede usar una porción limitada de la pila. Las rutinas que requieran una pila grande deben gestionar su propia pila.

Las condiciones de salida son:

Todos los registros e indicadores, salvo los de segmento, pueden quedar alterados.

La rutina genera un valor del mismo tipo que el original y lo entrega en el área de memoria que ocupaba el parámetro,

o bien

entrega un valor entero a través de la subrutina RETURN_INTEGER (véase más abajo).

Si bien las funciones USR pueden tomar un parámetro de tipo literal y generar un resultado literal, **no pueden** alterar el descriptor de la cadena en modo alguno.

Es de esperar que las funciones USR lleven generalmente un parámetro entero y generen un valor entero. BASIC ofrece dos rutinas adecuadas a este método de utilización de las USR. Son las siguientes:

GET_INTEGER

Fuerza el parámetro a entero, redondeando los números en representación en punto decimal. Fracasa si el parámetro es una cadena literal. (Equivalente a CINT.)

Condiciones de entrada:
El parámetro no ha sido alterado.

Condiciones de salida:
Mallard-80: HL contiene el valor entero del parámetro.
Mallard-86: BX contiene el valor entero del parámetro.
Todos los demás registros (excepto los de segmento) e indicadores: alterados.

RETURN_INTEGER

Hacer que el valor generado por la función sea el entero dado.

Condiciones de entrada:
Mallard-80: HL contiene el valor entero que va a ser generado.
Mallard-86: BX contiene el valor entero que va a ser generado.

Condiciones de salida:
Mallard-80: Registro A e indicadores: alterados.
Mallard-86: Registro AL e indicadores: alterados.
Todos los demás registros: preservados.

En Mallard-80 las direcciones de estas dos rutinas se encuentran en las siguientes posiciones en el intérprete de BASIC:

GET_INTEGER:	259 decimal	103 hexadecimal
RETURN_INTEGER:	261 decimal	105 hexadecimal

En Mallard-86 las dos rutinas se invocan mediante una 'llamada lejana' a las siguientes posiciones en el intérprete de BASIC:

GET_INTEGER:	259 decimal	103 hexadecimal
RETURN_INTEGER:	263 decimal	107 hexadecimal

3 Subrutinas invocadas con CALL

La orden CALL especifica la dirección de la subrutina y puede incluir una lista de parámetros opcionales. La dirección sólo puede estar en forma de valor de una variable; es decir, no puede ser una expresión. Los parámetros sólo pueden ser variables (no expresiones). Tales variables pueden ser elementos de una matriz.

Los parámetros se entregan en forma de referencia; esto es, lo que se entrega es la dirección del valor del parámetro (en el caso de una cadena, el valor es el descriptor de la cadena). El número y tipo de los parámetros tiene que estar acordado entre el programa de BASIC y la subrutina; BASIC no realiza ninguna comprobación. Si es posible, los parámetros se entregan a través de los registros; si no, en un área de la memoria.

Para Mallard-80 las condiciones de entrada son:

Si hay 3 parámetros o menos:

HL contiene la dirección del parámetro 1 (si lo hay)
DE contiene la dirección del parámetro 2 (si lo hay)
BC contiene la dirección del parámetro 3 (si lo hay)

Si hay 4 parámetros o más:

HL contiene la dirección del parámetro 1
DE contiene la dirección del parámetro 2
BC contiene la dirección de un área de la memoria en la que se encuentran las direcciones de los restantes parámetros, organizadas de la siguiente manera:

2 bytes	dirección del parámetro n	dirección alta
	...	
	dirección del parámetro 4	
	dirección del parámetro 3	dirección baja

BC contiene la dirección del byte menos significativo de la dirección del parámetro 3.

Para Mallard-86 las condiciones de entrada son:

Pila:	2 bytes	dirección del parámetro 1	dirección alta
		dirección del parámetro 2	
		...	
		dirección del parámetro n	
		desplazamiento de retorno del segmento de código de BASIC	dirección baja

El registro SP apunta al byte menos significativo del desplazamiento de retorno. Los parámetros se pueden referenciar mediante direccionamiento relativo al BP. Todas las direcciones se refieren a posiciones dentro del segmento de datos de BASIC.

Véase más abajo la descripción del estado de los registros de segmento.

El estado de los demás registros (salvo los de segmento) es indefinido.

Las condiciones de salida son:

Todos los registros (salvo los de segmento) e indicadores: alterados.

Dado que los parámetros se entregan en forma de referencia, la subrutina puede generar valores. Los valores pueden ser también cadenas literales, a condición de que la subrutina no modifique en modo alguno el descriptor de la cadena.

4 Mallard-86 y registros de segmento

Mallard-86 utiliza una 'llamada lejana' siempre que tiene que invocar una rutina externa, tanto si es a través de CALL como si se trata de una USR. El desplazamiento de la dirección de la rutina es el mencionado en la correspondiente DEF USR o en la orden CALL. La base de segmento es la establecida en la orden DEF SEG más reciente (o, en su defecto, la base del segmento de datos de BASIC). Cuando se entra en la rutina, el valor de la cima de la pila es el 'retorno lejano' a BASIC. Los registros de segmento están como sigue:

CS: valor establecido por la DEF SEG más reciente (o, en su defecto, la base del segmento de datos de BASIC)
SS: base del segmento de datos de BASIC
DS: base del segmento de datos de BASIC
ES: base del segmento de datos de BASIC

Todas las direcciones de los parámetros son desplazamientos dentro del segmento de datos de BASIC. Para usar las rutinas GET_INTEGER y RETURN_INTEGER, la subrutina debe crear una 'dirección lejana' leyendo el segmento de código de BASIC y utilizando el desplazamiento dado más arriba.

Si una subrutina externa modifica algún registro de segmento, tiene que restaurarlo al valor de entrada antes de retornar.

Palabras clave de BASIC

Damos a continuación la lista de las palabras clave de BASIC. Son palabras reservadas y, por consiguiente, no pueden ser utilizadas como nombres de variables. (Algunas sólo están reservadas en las versiones Jetsam.)

ABS, ADDKEY, ADDREC, ALL, AND, AS, ASC, ATN, AUTO

BASE, BUFFERS

CALL, CDBL, CHAIN, CHR$, CINT, CLEAR, CLOSE, COMMON, CONSOLIDATE, CONT, COS, CREATE, CSNG, CVD, CVI, CVIK, CVS, CVUK

DATA, DEC$, DEF, DEFDBL, DEFINT, DEF SEG, DEFSNG, DEFSTR, DELETE, DELKEY, DIM, DIR, DISPLAY

EDIT, ELSE, END, EOF, EQV, ERA, ERASE, ERL, ERR, ERROR, EXP

FETCHKEY$, FETCHRANK, FETCHREC, FIELD, FILES, FIND$, FIX, FN, FOR, FRE

GET, GOSUB, GOTO

HEX$, HIMEM

IF, IMP, INKEY$, INP, INPUT, INPUT #, INPUT$, INPW, INSTR, INT

KILL

LEFT$, LEN, LET, LINE, LIST, LLIST, LOAD, LOC, LOCK, LOF, LOG, LOG10, LOWER$, LPOS, LPRINT, LSET

MAX, MEMORY, MERGE, MID$, MIN, MKD$, MKI$, MKIK$, MKS$, MKUK$, MOD

NAME, NEXT, NEW, NOT

OCT$, ON, ON ERROR GOTO 0, OPEN, OPTION, OR, OSERR, OUT, OUTW

PEEK, POKE, POS, PRINT, PRINT #, PUT

RANDOMIZE, RANKSPEC, READ, REM, REN, RENUM, RESET, RESTORE, RESUME, RESUME 0, RETURN, RIGHT$, RND, ROUND, RSET, RUN

SAVE, SEEKKEY, SEEKNEXT, SEEKPREV, SEEKRANK, SEEKREC, SEEKSET, SGN, SIN, SPACE$, SPC, SQR, STEP, STOP, STR$, STRING$, STRIP$, SWAP, SYSTEM

TAB, TAN, THEN, TO, TROFF, TRON, TYPE

UNT, UPPER$, USING, USR

VAL, VARPTR, VERSION

WAIT, WAITW, WEND, WHILE, WIDTH, WRITE, WRITE #

XOR

ZONE

Apéndice V

Instalación de BASIC

En el proceso de instalación de BASIC se decide lo siguiente:

a. Si se va a usar el editor de líneas basado en la pantalla o el gestionado por órdenes.

b. Los caracteres y secuencias de control de la pantalla para el editor de líneas basado en la pantalla.

c. Los caracteres y secuencias de control del teclado para el editor de líneas basado en la pantalla.

d. La anchura de la consola y la posición en la que se emite el CR/LF automático.

e. La 'compresión de espacios' (véase la sección 2.2).

En ausencia de otra especificación, se selecciona el editor de líneas gestionado por órdenes, se establece la anchura de la consola y la posición de 'auto CR/LF' en 80 y se inhibe la 'compresión de espacios'.

Las versiones 'completas' de BASIC incluyen un 'editor de líneas basado en la pantalla'. El usuario puede mover el cursor por la línea y añadir o suprimir caracteres. El editor actualiza la representación de la línea en la pantalla, de modo que siempre exhibe la versión más moderna.

Las versiones de 'sólo ejecución' de BASIC no incluyen ningún editor de líneas y no requieren más instalación que la especificación de la anchura de la consola y la posición de 'auto CR/LF', en el supuesto de que no sean ambas 80.

El editor de líneas basado en la pantalla necesita un reducido número de órdenes de movimiento del cursor y alguna información sobre la pantalla con la que está operando. BASIC contiene una tabla en la que está almacenada esta información.

Las secuencias de pulsaciones para las órdenes del editor no son fijas, sino que también están almacenadas en una tabla en BASIC.

El programa de instalación permite definir la información contenida en estas tablas para adaptar las correspondientes características a la pantalla y al teclado en los que se va a utilizar BASIC. Algunas versiones de BASIC se suministran completamente instaladas para un ordenador concreto; tales versiones no incluyen el programa de instalación.

1 Requisitos de la pantalla

El editor necesita que la pantalla sea capaz de realizar las siguientes operaciones:

- Desplazar el cursor una posición hacia la izquierda (sin borrar el carácter).
- Desplazar el cursor una posición hacia la derecha.
- Desplazar el cursor una línea hacia abajo.
- Desplazar el cursor una línea hacia arriba.
- Llevar el cursor al principio de la línea de pantalla actual.

En general, BASIC espera poder invocar los movimientos del cursor enviando a la pantalla un carácter de control seguido de cuatro parámetros como máximo. La forma de estas secuencias de caracteres de control se especifica en las tablas de BASIC a través del programa de instalación. En el IBM PC, el sistema PC-DOS normalmente no funciona a base de códigos de control de este tipo, de manera que la pantalla ha de ser controlada directamente. El programa de instalación ofrece como opción el controlador de pantalla para el IBM PC.

El editor también necesita saber cuál es la anchura de la pantalla y cuál es el efecto de escribir un carácter en la última columna. Muchas pantallas realizan automáticamente un retorno del carro/avance de línea ('auto CR/LF') cuando se escribe un carácter en la última columna. La orden WIDTH de BASIC permite definir la anchura de la pantalla y declarar en qué columna se produce el 'auto CR/LF'.

2 Órdenes de teclado para el editor

En el capítulo 4 se describe el juego de órdenes que obedece el editor. Cada orden se invoca mediante una secuencia de hasta tres caracteres. El primero de ellos debe ser un carácter de control (el valor &H7F o alguno del margen de &H00 a &H1F). Las órdenes y una posible asignación de caracteres son como sigue:

Cursor a la izquierda	Control-S
Cursor a la derecha	Control-D
Cursor arriba	Control-E
Cursor abajo	Control-X
Buscar carácter	Control-L
Inserción/Sobreescritura	Báscula con control-V
Borrar hacia delante	Control-G
Borrar hacia atrás	&H7F
Borrar hasta un carácter	Control-K
Terminar la edición	Retorno del carro (control-M)
Abandonar la edición	Control-C

3 El programa de instalación

El programa de instalación está escrito en BASIC y puede ser ejecutado por el Mallard BASIC con el que se lo suministra. Está diseñado para que sea fácil de usar y contiene cierta información de ayuda. Cualquier máquina dotada de Mallard BASIC puede instalar el lenguaje para cualquier otra máquina. El programa de instalación escribe en las tablas de BASIC lo siguiente:

- Los valores implícitos de la anchura de la pantalla y de la posición de 'auto CR/LF'.
- Información sobre si el editor está activado o inhibido.
- Las secuencias de control para las órdenes de movimiento del cursor, o el controlador de pantalla.
- Pulsaciones que han de ser reconocidas como órdenes del editor.

Esta información puede ser introducida a mano o leída en un fichero. El programa dispone de un repertorio de parámetros para los tipos de pantalla más comunes que el usuario puede ampliar.

El programa ofrece además las siguientes funciones:

- Grabar los datos de instalación en un fichero en una forma que pueda ser leída cuando se necesite.
- Cargar los datos de instalación desde un fichero.
- Enviar los datos de instalación a la impresora en forma legible.
- Examinar los datos de instalación para comprobar que son consistentes.
- Probar las pulsaciones y las órdenes de pantalla.

Sugerencias para el trabajo con Jetsam y en sistemas multiusuario

1 Claves numéricas

Las claves de Jetsam son cadenas literales. Los números que hayan de ser utilizados como claves se pueden convertir en cadena ASCII mediante la función DEC$. Jetsam trabaja tanto más deprisa cuanto más cortas sean las claves. Se debe evitar, siempre que sea posible, la expansión de las claves numéricas a cadenas ASCII. Para valores enteros se recomienda hacer lo siguiente:

i. Valores del margen de 0 a 255:
   usar CHR$(x%) para generar una cadena de 1 byte.

ii. Valores del margen de −32768 a +32767 (es decir, enteros):
   usar MKIK$(x%) para generar una cadena de 2 bytes.

iii. Valores del margen de 0 a 65535 (es decir, enteros sin signo):
   usar MKUK$(x) para producir una cadena de 2 bytes.

A fin de reducir en lo posible la longitud de las cadenas, vale la pena tomarse la molestia de convertir los valores de las claves numéricas a alguno de los márgenes anteriores.

Si las claves han de contener parte decimal, la única solución práctica es usar DEC$ (pero obsérvese que los números negativos requieren un tratamiento especial). Las funciones MKD$ y MKS$ no producen cadenas adecuadas a su utilización como claves de Jetsam.

2 Proceso secuencial de ficheros de acceso por claves en sistemas multiusuario

El siguiente fragmento de un programa para el proceso secuencial de un fichero avanza hasta el siguiente registro y lo bloquea:

```
10 registro.anterior% = FETCHREC(#núm.fichero%)
20 código.generado% = SEEKNEXT(#núm.fichero%,1)
30 IF registro.anterior%<>0 THEN IF LOCK(#núm.fichero%,0,registro.anterior%)
                                                        THEN STOP
40 IF código.generado%
```

El procedimiento consiste en: hallar el número de registro actual; avanzar hasta el siguiente registro y bloquearlo (en lectura); y luego desbloquear el anterior registro actual (si la posición actual estaba definida). Esta operación tiene que concluir con éxito, luego STOP es la acción indicada para el caso de que fracase.

En el supuesto de que el registro actual esté bloqueado cuando se entra en esta sección del programa, este procedimiento evita los problemas que daría el hecho de que se borrara la posición actual, ya que el avance al siguiente registro y su bloqueo se realizan antes de desbloquear la posición anterior.

Una alternativa sería la siguiente:

```
10 IF FETCHREC(#núm.fichero%)<>0 THEN IF LOCK(#núm.fichero%,0,
                                    FETCHREC(#núm.fichero%)) THEN STOP
20 código.generado% = SEEKNEXT(#núm.fichero%,1)
30 IF código.generado%
```

Pero en este caso, si se borra la posición antigua entre el momento en que se la desbloquea y el avance al registro siguiente, SEEKNEXT genera el código 105.

3 Subclaves

Algunos sistemas de ordenación trabajan con una clave primaria y un número variable de subclaves. Si se encuentran varios registros con la misma clave primaria, el orden se establece a base de la primera subclave. Si también ésta es común, se los ordena por la segunda subclave, y así sucesivamente. Jetsam no ofrece este mecanismo; los rangos de Jetsam no constituyen subclaves; de hecho, las claves de un rango son totalmente independientes de las de los restantes rangos.

Para simular ese método de ordenación por claves y subclaves, el programa tiene que crear una clave de Jetsam concatenando las claves de los distintos niveles. Esto es sencillo si todas las claves de un nivel dado son de la misma longitud; por ejemplo,

```
10 IF ADDREC(#núm.fichero%,0,rango%,MKIK$(fecha%)+MKIK$(factura%)) THEN
                                        PRINT"ADDREC ha fracasado"
```

inserta un nuevo registro en el índice en el lugar que le corresponde por orden de fecha y número de factura. Si las claves no son de longitud fija, se las separa con un CHR$(0); por ejemplo,

```
10 IF ADDREC(#núm.fichero,0,rango%,apellido$+CHR$(0)+iniciale$) THEN PRINT
                                        "ADDREC ha fracasado"
```

inserta un nuevo registro en el índice por orden alfabético de apellido e iniciales.

4 Gestión sencilla de ficheros aleatorios en sistemas multiusuario

El sistema de 'bloqueo de registros en escritura' descrito en el apartado 8.3.4 está diseñado para facilitar la conversión de programas de gestión de ficheros aleatorios. Los programas que se limitan a leer un registro a la vez, procesarlo y posiblemente reescribirlo modificado, no necesitan bloquear explícitamente los registros, ya que BASIC les aplica automáticamente un bloqueo en escritura. Lo único que hay que modificar es la orden OPEN para abrir el fichero sin bloqueo, pues la acción implícita es abrirlo bloqueado en escritura.

Si se adopta esta solución, hay que tener en cuenta cuáles son sus limitaciones: sólo se bloquea un registro, el último leído. Los programas que necesiten conservar inalterados varios registros mientras están procesando uno o varios de ellos, tendrán que bloquear explícitamente todos los registros en cuestión.

5 Registro de salvaguardia

El programa de aplicación necesitará instrumentar un registro auxiliar (registro de salvaguardia) cuando pretenda controlar el acceso a los ficheros por métodos distintos de los de bloqueo de registros y ficheros proporcionados por el sistema. El siguiente ejemplo está referido a un fichero de acceso por claves, pero la técnica es igualmente aplicable a los ficheros aleatorios. Para controlar el acceso a un rango completo de claves, el programa puede crear un registro auxiliar y utilizar las siguientes instrucciones:

```
10 REM Para acceder sin exclusiva a un rango
20 código.generado% = SEEKKEY(#núm.fichero%,1,rango%,'')
30 IF código.generado% <>0 THEN
...
90 IF LOCK(#núm.fichero%,0,FETCHREC(#núm.fichero%)) THEN STOP
```

y

```
10 REM Para acceder en exclusiva a un rango
20 código.generado% = SEEKKEY(#núm.fichero%,2,rango%,'')
30 IF código.generado% <>0 THEN
...
90 IF LOCK(#núm.fichero%,0,FETCHREC(#núm.fichero%)) THEN STOP
```

La aplicación con éxito del bloqueo en lectura del primer fragmento permite a otros usuarios el acceso compartido al rango para el que se ha establecido el registro de salvaguardia. Análogamente, la aplicación del bloqueo en escritura del segundo fragmento permite el acceso en exclusiva al rango.

Con instrucciones similares se puede programar el control del acceso a los ficheros aleatorios y a los de acceso por claves, tanto a los ficheros completos como a parte de ellos.

Ejemplo de utilización de los ficheros de Jetsam

El siguiente programa es un ejemplo de la aplicación del sistema Jetsam en el desarrollo de una sencilla base de datos.

```
1000 REM ******************************************************
1010 REM EJEMPLO DE APLICACION DE JETSAM
1020 REM Lista de direcciones y telefonos
1030 REM ******************************************************
1040 OPTION RUN:WIDTH 255:DEFINT A-Z
1050 BS$=STRING$(80,127):DIM A$(50):ESC$=CHR$(27):pitido$=CHR$
     (7)
1060 DEF FN raya$(b$,a$,c$,d$)=b$+STRING$(30,d$)+a$+STRING$(40
     ,d$)+a$+STRING$(15,d$)+c$
1070 DEF FN locate$(v,h)=esc$+"Y"+CHR$(32+v)+CHR$(32+h)
1080 CURSORarriba=31:CURSORabajo=30:CURSORizquierda=1:CURSORde
     recha=6:retroceso=8:DEL=127
1090 aa$=CHR$(224):ee$=CHR$(225):ii$=CHR$(226):oo$=CHR$(227):u
     u$=CHR$(228)
1100 PRINT esc$;"0":GOTO 2220
1110 REM
1120 REM
1130 REM ******************************************************
1140 REM rutinas especiales de pantalla
1150 REM ******************************************************
1160 PRINT FN locate$(v,h);:RETURN: REM LOCATE
1170 PRINT esc$;"E";esc$;"H";:v=0:h=0:RETURN: REM CLS
1180 PRINT esc$"K";:RETURN: REM BORRA HASTA EL FINAL DE LINEA
1190 PRINT esc$"J";:RETURN: REM BORRA HASTA EL FINAL DE PAG.
1200 PRINT esc$"p";:RETURN: REM ACTIVA VIDEO INVERSO
1210 PRINT esc$"q";:RETURN: REM CANCELA VIDEO INVERSO
1220 PRINT esc$"f";:RETURN: REM APAGA CURSOR
1230 PRINT esc$"e";:RETURN: REM ENCIENDE CURSOR
1240 REM
1250 REM ******************************************************
1260 REM lectura de teclado
1270 REM ******************************************************
1280 A$=INKEY$:IF A$="" THEN 1270 ELSE RETURN
```

```
1290 REM
1300 REM ************************************************
1310 REM rutina de introduccion de caracteres
1320 REM ************************************************
1330 IN$="":PRINT STRING$(LO,"_");STRING$(LO,8);:GOSUB 1230
1340 EN$=INPUT$(1)
1350 EN=ASC(EN$)
1360 IF EN=13 THEN GOSUB 1220:RETURN
1370 IF EN=RETROCESO OR EN=DEL THEN 1410
1380 IF EN=ASC(ESC$) THEN IN$=ESC$:RETURN
1390 IF LEN(IN$)=LO OR EN$<" " THEN 1340
1400 IN$=IN$+EN$:PRINT EN$;:GOTO 1340
1410 IF LEN(IN$)=0 THEN 1340 ELSE IN$=LEFT$(IN$,LEN(IN$)-1):PR
     INT CHR$(8);"_";CHR$(8);:GOTO 1340
1420 REM
1430 REM ************************************************
1440 REM rutina de presentacion de cabezeras
1450 REM ************************************************
1460 GOSUB 1220:GOSUB 1170:PRINT CHR$(134);STRING$(87,138);CHR
     $(140)
1470 V=V+1:GOSUB 1180:GOSUB 1210:GOSUB 1160:PRINT CHR$(133);:H
     =44-LEN(A$)/2:GOSUB 1160:PRINT A$:H=88:GOSUB 1160:PRINT C
     HR$(133)
1480 V=V+1:H=0:GOSUB 1160:PRINT CHR$(131);STRING$(87,138);CHR$
     (137):GOSUB 1210:RETURN
1490 REM
1500 REM ************************************************
1510 REM rutina de presentacion de pies
1520 REM ************************************************
1530 V=27:H=0:GOSUB 1160:PRINT CHR$(134);STRING$(87,138);CHR$(
     140)
1540 V=V+1:GOSUB 1160:GOSUB 1180:PRINT CHR$(133);STRING$(44-LE
     N(A$)/2,32);A$:H=88:GOSUB 1160:PRINT CHR$(133)
1550 h=0:v=v+1:GOSUB 1160:PRINT CHR$(131);STRING$(87,138);CHR$
     (137);:RETURN
1560 A$="ELIJA LA OPCION DESEADA CON LAS TECLAS DEL CURSOR O P
     ULSANDO LA INICIAL":GOSUB 1530:RETURN
1570 REM
1580 REM ************************************************
1590 REM rutina de presentacion de menus y seleccion
1600 REM ************************************************
1610 puntero=1:GOSUB 1220:MAY$="":LMAX=0:FOR I=1 TO NU:IF LEN(
     A$(I)) > LMAX THEN LMAX=LEN(A$(I))
1620 MAY$=MAY$+LEFT$(A$(I),1)
1630 NEXT I
```

```
1640 IF MEN$="" THEN 1660 ELSE IF LMAX<LEN(MEN$) THEN LMAX=LEN
     (MEN$)
1650 MEN$=MEN$+STRING$(LMAX-LEN(MEN$)," ")
1660 FOR I=1 TO NU:A$(I)=A$(I)+STRING$(LMAX-LEN(A$(I))," "):NE
     XT I
1670 VI=15-INT(NU/2):V=VI:H=40-INT(LMAX/2):VERTICAL=VI
1680 GOSUB 1200:IF MEN$<>"" THEN V=VI-2:VERTICAL=V:GOSUB 1160:
     PRINT CHR$(134);STRING$(LMAX+2,138);CHR$(140):V=V+1:GOSUB
      1160:PRINT CHR$(133);" ";MEN$;" ";CHR$(133):V=V+1:GOSUB
     1160:PRINT CHR$(135);STRING$(LMAX+2,138);CHR$(141):GOTO 1
     700
1690 GOSUB 1160:PRINT CHR$(134);STRING$(LMAX+2,138);CHR$(140)
1700 FOR I=1 TO NU:V=VI+I:GOSUB 1160:PRINT CHR$(133);" ";A$(I)
     ;" ";CHR$(133)
1710 NEXT I
1720 V=V+1:GOSUB 1160:PRINT CHR$(131);STRING$(LMAX+2,138);CHR$
     (137)
1730 IF XXXX=1 THEN GOSUB 1210:RETURN
1740 I=1:V=VI+1:H=H+1:A$=A$(1)
1750 GOSUB 1210:GOSUB 1160:PRINT " ";A$(I);" "
1760 A$=INPUT$(1):A=ASC(A$):IF A=27 OR a=8 THEN I=NU+1:GOSUB 1
     210:in$=esc$:RETURN
1770 IF A>=97 THEN A=A-32:A$=CHR$(A)
1780 IF A=CURSORabajo THEN IF I=NU THEN PRINT pitido$;:GOTO 17
     60 ELSE GOSUB 1200:GOSUB 1160:PRINT " ";A$(I);" ":I=I+1:V
     =V+1:GOTO 1750
1790 IF A=CURSORarriba THEN IF I=1 THEN PRINT pitido$;:GOTO 17
     60 ELSE GOSUB 1200:GOSUB 1160:PRINT " ";A$(I);" ":I=I-1:V
     =V-1:GOTO 1750
1800 GT=INSTR(MAY$,A$):IF GT<>0 THEN GOSUB 1820:IF ii=0 THEN G
     OSUB 1200:GOSUB 1160:PRINT " ";A$(I);" ":I=GT:V=VI+I:GOSU
     B 1210:GOSUB 1160:PRINT " ";A$(I);" ":GOTO 1760
1810 IF A=13 THEN GOSUB 1210:RETURN ELSE PRINT pitido$;:GOTO 1
     760
1820 IF gt<>gta OR puntero>LEN(may$)+1 THEN gta=gt:puntero=gta
     +1:ii=0:RETURN
1830 FOR ii=puntero TO LEN(may$)
1840 IF MID$(may$,ii,1)=a$ THEN puntero=ii+1:gta=ii:gt=gta:ii=
     0:RETURN
1850 NEXT ii:RETURN
1860 REM
1870 REM ********************************************************
1880 REM rutina de presentacion de listados
1890 REM ********************************************************
1900 REM
```

```
1910 PRINT"":PRINT FN raya$(CHR$(134),CHR$(142),CHR$(140),CHR$
     (138))
1920 PRINT FN raya$(CHR$(133),CHR$(133),CHR$(133),CHR$(32))
1930 PRINT FN raya$(CHR$(135),CHR$(143),CHR$(141),CHR$(138))
1940 v=7:FOR i=1 TO 19:GOSUB 1970:v=v+1:NEXT i
1950 PRINT FN raya$(CHR$(131),CHR$(139),CHR$(137),CHR$(138))
1960 h=2:v=5:GOSUB 1160:PRINT"Nombre":h=33:GOSUB 1160:PRINT"Di
     recci"oo$"n":h=74:GOSUB 1160:PRINT"Tel"ee$"fono":a$="":GO
     SUB 1530:RETURN
1970 PRINT FN raya$(CHR$(133),CHR$(133),CHR$(133),CHR$(32))
1980 RETURN
1990 REM ******************************************************
2000 REM rutina de correccion de datos
2010 REM ******************************************************
2020 REM
2030 GOSUB 1160:L=0:B$="":GOSUB 1230
2040 B$=INPUT$(1)
2050 A=ASC(B$):IF A=13 OR A=27 OR a=CURSORabajo THEN GOSUB 122
     0:RETURN
2060 IF A=CURSORderecha AND L<LEN(A$) THEN PRINT MID$(A$,L+1,1
     );:L=L+1:GOTO 2040
2070 IF A=CURSORizquierda AND L>=1 THEN PRINT esc$"D";:L=L-1:G
     OTO 2040
2080 IF B$<" " THEN 2040
2090 L=L+1:IF L>LEN(A$) THEN L=L-1:GOTO 2040 ELSE MID$(A$,L,1)
     =B$:PRINT B$;:GOTO 2040
2100 REM
2110 REM ******************************************************
2120 REM rutina de apertura de los ficheros
2130 REM ******************************************************
2140 BUFFERS 10:OPEN "K",2,"direccio","indices",2
2150 FIELD #2,30 AS nom$,40 AS direc$,15 AS telefono$
2160 RETURN
2170 REM
2180 REM ******************************************************
2190 REM PROGRAMA PRINCIPAL
2200 REM ******************************************************
2210 REM
2220 a$="Programa de demostraci"+oo$+"n de ficheros JETSAM. "+
     CHR$(164)+" INDESCOMP 1985.":GOSUB 1460
2230 men$="Men"+uu$+" principal : ":a$(1)="Introducir nuevas f
     ichas"
2240 a$(2)="B"+uu$+"squeda de fichas"
2250 a$(3)="Listados"
2260 a$(4)="Correcci"+oo$+"n de fichas"
```

```
2270 a$(5)="Eliminaci"+oo$+"n de fichas"
2280 a$(6)="Retornar a BASIC":a$(7)="Destruir los ficheros de
     datos"
2290 nu=7:GOSUB 1560:GOSUB 1610
2300 ON i GOTO 2310,2510,2710,2860,3040,3190,3200,2290
2310 IF FIND$("direccio")<>"" THEN 2390
2320 REM
2330 REM
2340 REM
2350 REM
2360 CREATE #1,"direccio","indices",2,87
2370 REM 30 nombre, 40 direccion, 15 telefono +2 =87
2380 CLOSE 1
2390 GOSUB 2140
2400 REM
2410 REM
2420 GOSUB 1170:a$="Introduzca nuevas fichas, o pulse SAL para
      volver al men"+uu$+" principal.":GOSUB 1460:GOSUB 1910
2430 v=6
2440 v=v+1:IF v>25 THEN v=3:GOSUB 1160:GOSUB 1190:GOSUB 1910:v
     =7
2450 h=1:GOSUB 1160:lo=30:GOSUB 1330:IF in$=esc$ THEN CLOSE 2:
     GOTO 2220 ELSE LSET nom$=in$:GOSUB 1160:PRINT nom$
2460 h=32:GOSUB 1160:lo=40:GOSUB 1330:IF in$=esc$ THEN GOSUB 1
     160:PRINT SPACE$(lo):GOTO 2450 ELSE LSET direc$=in$:GOSUB
      1160:PRINT direc$
2470 h=73:GOSUB 1160:lo=15:GOSUB 1330:IF in$=esc$ THEN GOSUB 1
     160:PRINT SPACE$(lo):GOTO 2460 ELSE RSET telefono$=in$:GO
     SUB 1160:PRINT telefono$
2480 result=ADDREC(2,2,1,nom$):IF result<>0 THEN STOP
2490 result=ADDKEY(2,2,2,telefono$,FETCHREC(2)):IF result<>0 T
     HEN STOP
2500 GOTO 2440
2510 GOSUB 1170:a$="Para volver al men"+uu$+" principal pulse
     SAL.":GOSUB 1460
2520 men$="B"+uu$+"squeda por: "
2530 a$(1)="Nombre"
2540 a$(2)="Tel"+ee$+"fono"
2550 nu=2:GOSUB 1560:GOSUB 1610:ON i GOTO 2560,2570,2220
2560 rank=1:long=30:GOTO 2580
2570 rank=2:long=15
2580 GOSUB 1170:a$="Escriba ":IF rank=1 THEN a$=a$+"el nombre,
     " ELSE a$=a$+"el tel"+ee$+"fono,"
2590 a$=a$+" o pulse SAL para volver al men"+uu$+" principal."
2600 GOSUB 2140
```

```
2610 GOSUB 1460:GOSUB 1910
2620 v=6
2630 v=v+1
2640 IF rank=1 THEN h=1:GOSUB 1160:lo=30:GOSUB 1330:IF in$=esc
     $ THEN CLOSE 2:GOTO 2220 ELSE LSET nom$=in$:clave$=nom$:G
     OTO 2660
2650 h=73:GOSUB 1160:lo=15:GOSUB 1330:IF in$=esc$ THEN CLOSE 2
     :GOTO 2220 ELSE RSET telefono$=in$:clave$=telefono$
2660 result=SEEKKEY(2,0,rank,clave$):IF result<>0 THEN a$="No
     existe esa clave. Pulse una tecla para continuar.":PRINT
     pitido$;:vv=v:GOSUB 1530:GOSUB 1260:a$="":GOSUB 1530:v=vv
     :GOTO 2640
2670 GOSUB 2690
2680 result=SEEKNEXT(2,0):IF result<>0 THEN 2640 ELSE GOSUB 26
     90:GOTO 2680
2690 GET #2:h=1:GOSUB 1160:PRINT nom$:h=32:GOSUB 1160:PRINT di
     rec$:h=73:GOSUB 1160:PRINT telefono$:v=v+1:IF v>25 THEN 2
     700 ELSE RETURN
2700 PRINT pitido$;:a$="Pulse cualquier tecla para continuar."
     :GOSUB 1530:GOSUB 1260:v=3:GOSUB 1160:GOSUB 1910:v=7:RETU
     RN
2710 GOSUB 1170:a$="Para volver al men"+uu$+" principal pulse
     SAL.":GOSUB 1460
2720 men$="Listados: "
2730 a$(1)="Todo el fichero"
2740 a$(2)="A partir de un dato"
2750 nu=2:GOSUB 1560:GOSUB 1610:ON i GOTO 2810,2760,2220
2760 a$="Introduzca el nombre a partir del cual quiera listar.
      SAL para salir.":GOSUB 1460:GOSUB 1190:GOSUB 1910
2770 GOSUB 2140
2780 v=7:h=1:GOSUB 1160:lo=30:GOSUB 1330:IF in$=esc$ THEN CLOS
     E 2:GOTO 2710 ELSE LSET nom$=in$
2790 result=SEEKKEY(2,0,1,nom$):IF result<>0 THEN a$="No exist
     e esa clave. Pulse cualquier tecla para continuar.":GOSUB
      1530:GOSUB 1260:a$="":GOSUB 1530:GOTO 2780
2800 GOTO 2830
2810 a$="Listado de todo el fichero de direcciones."
2820 GOSUB 1460:GOSUB 1190:GOSUB 1910:v=7:GOSUB 2140:result=SE
     EKRANK(2,0,1):IF result<>0 THEN PRINT pitido$;:a$="No se
     ha encontrado ninguna clave. Pulse cualquier tecla para c
     ontinuar.":GOSUB 1530:GOSUB 1260:CLOSE 2:GOTO 2220
2830 GOSUB 2690
2840 result=SEEKNEXT(2,0):IF result<>0 AND result<>101 THEN 28
     50 ELSE 2830
```

```
2850 CLOSE 2:a$="No hay m"+aa$+"s datos. Pulse cualquier tecla
     para continuar.":GOSUB 1530:GOSUB 1260:GOTO 2220
2860 GOSUB 1170:a$="Correci"+oo$+"n de fichas. Introduzca el n
     ombre o pulse SAL para volver al men"+uu$+".":GOSUB 1460:
     GOSUB 1910:v=6
2870 GOSUB 2140
2880 v=v+1:IF v>25 THEN GOSUB 2700
2890 h=1:GOSUB 1160:lo=30:GOSUB 1330:IF in$=esc$ THEN CLOSE 2:
     GOTO 2220
2900 LSET nom$=in$:clave$=nom$
2910 result=SEEKKEY(2,0,1,clave$):IF result<>0 THEN vv=v:a$="N
     o existe esa clave. Pulse cualquier tecla para continuar.
     ":PRINT pitido$:GOSUB 1530:GOSUB 1260:a$="":GOSUB 1530:v=
     vv:GOTO 2890
2920 GET #2:h=1:GOSUB 1160:PRINT nom$:h=32:GOSUB 1160:PRINT di
     rec$:h=73:GOSUB 1160:PRINT telefono$
2930 a$="Introduzca los datos correctos o pulse 'cursor abajo'
     para corregir otra ficha.":vv=v:GOSUB 1530:v=vv
2940 a$=nom$:h=1:GOSUB 2030:IF a=CURSORabajo THEN 3030 ELSE LS
     ET nom$=a$
2950 a$=direc$:h=32:GOSUB 2030:LSET direc$=a$
2960 a$=telefono$:telefonoa$=telefono$:h=73:GOSUB 2030:RSET te
     lefono$=a$
2970 PUT #2
2980 result=ADDKEY(2,0,1,nom$,FETCHREC(2))
2990 result=DELKEY(2,0,1,clave$,FETCHREC(2))
3000 result=ADDKEY(2,0,2,telefono$,FETCHREC(2))
3010 result=DELKEY(2,0,2,telefonoa$,FETCHREC(2))
3020 GOTO 2880
3030 result=SEEKNEXT(2,0):IF result<>0 THEN 2880 ELSE v=v+1:IF
     v>25 THEN GOSUB 2700:GOTO 2920 ELSE GOTO 2920
3040 a$="Introduzca el nombre y el tel"+ee$+"fono de la ficha
     que desea borrar, o pulse SAL.":GOSUB 1170:GOSUB 1460
3050 GOSUB 1910:GOSUB 2140
3060 v=6
3070 v=v+1:IF v>25 THEN GOSUB 2700
3080 h=1:lo=30:GOSUB 1160:GOSUB 1330:IF in$=esc$ THEN CLOSE 2:
     GOTO 2220 ELSE LSET nom$=in$:GOSUB 1160:PRINT nom$
3090 h=73:lo=15:GOSUB 1160:GOSUB 1330:IF in$=esc$ THEN GOSUB 1
     160:PRINT SPACE$(15):GOTO 3080 ELSE RSET telefono$=in$:cl
     ave$=telefono$:GOSUB 1160:PRINT telefono$
3100 result=SEEKKEY(2,0,1,nom$):IF result<>0 THEN a$="No se ha
     encontrado ese nombre. Pulse cualquier tecla para contin
     uar.":PRINT pitido$:vv=v:GOSUB 1530:GOSUB 1260:a$="":GOSU
     B 1530:v=vv:GOTO 3080
```

```
3110 GET #2:IF telefono$=clave$ THEN 3140
3120 result=SEEKNEXT(2,0):IF result<>0 THEN a$="No se ha encon
     trado esa ficha. Pulse cualquier tecla para continuar.":P
     RINT pitido$;:vv=v:GOSUB 1530:GOSUB 1260:a$="":GOSUB 1530
     :v=vv:GOTO 3070
3130 GOTO 3110
3140 h=1:GOSUB 1160:PRINT nom$:h=32:GOSUB 1160:PRINT direc$:h=
     73:GOSUB 1160:PRINT telefono$
3150 a$="Pulse 'B' para borrar la ficha, o cualquier otra tecl
     a para no borrar.":vv=v:GOSUB 1530:GOSUB 1260:IF UPPER$(a
     $)<>"B" THEN a$="":GOSUB 1530:v=vv:GOTO 3070
3160 result=DELKEY(2,0,1,nom$,FETCHREC(2))
3170 result=DELKEY(2,0,2,telefono$,FETCHREC(2))
3180 v=vv:GOTO 3070
3190 GOSUB 1170:GOSUB 1230:END
3200 GOSUB 1170:a$="Destruir los ficheros de datos.":GOSUB 146
     0
3210 men$="":a$(1)=""
3220 a$(2)="ATENCION !!!"
3230 a$(3)=""
3240 a$(4)="Al destruir los ficheros perder"+aa$+" "
3250 a$(5)="irremisiblemente todos los datos."
3260 a$(6)=""
3270 a$(7)=""
3280 a$(8)="PULSE LA TECLA 'D' PARA DESTRUIR LOS FICHEROS."
3290 a$(9)="Cualquier otra tecla para volver al men"+uu$+" pri
     ncipal."
3300 a$(10)=""
3310 nu=10:a$="":GOSUB 1530:xxxx=1:GOSUB 1610
3320 GOSUB 1260:IF UPPER$(a$)<>"D" THEN 3340
3330 KILL "direccio":KILL "indices"
3340 xxxx=0:GOTO 2220
```

Índice

Acuerdo de licencia para el usuario final del programa

ADVERTENCIA AL USUARIO: POR FAVOR, LEA ATENTAMENTE ESTA NOTA. NO ABRA LAS CAJAS DE LOS DISCOS MIENTRAS NO HAYA LEIDO ESTE CONTRATO.

POR EL HECHO DE ABRIR LA CAJA USTED RECONOCE QUE ACEPTA LAS CLAUSULAS Y CONDICIONES SIGUIENTES.

1. Definiciones

En este contrato:

1. 'DRI' significa DIGITAL RESEARCH (CALIFORNIA) INC., P. O. Box 579, Pacific Grove, California 93950, EE.UU., propietario de los derechos de reproducción o licenciatario autorizado del programa.
2. 'Máquina' significa un único microordenador en que se utiliza el programa. Para los ordenadores con microprocesadores múltiples se requieren licencias adicionales.
3. 'Programa' es el conjunto de programas, documentación y anejos incluidos en esta aplicación, así como las actualizaciones y modificaciones que DRI pueda suministrarle, cualquiera que sea la forma en que usted los utilice, con o sin sus propias modificaciones.
4. 'AMSTRAD' significa AMSTRAD CONSUMER ELECTRONICS PLC., Brentwood House, 169 Kings Road, Brentwood, Essex CM14 4EF, Reino Unido.

   Usted asume toda la responsabilidad derivada de la selección del programa y de su instalación, utilización y resultados.

2. Licencia

Usted puede:

1. Utilizar el programa en una sola máquina.
2. Copiar el programa en forma electrónica o impresa, bien sea como medida de seguridad o para modificarlo, siempre condicionado a su utilización en una sola máquina. Para estos propósitos puede hacer 3 (tres) copias como máximo. (No obstante, algunos pro-

gramas pueden estar dotados de mecanismos que limiten las posibilidades de copia o la impidan; son los marcados con 'copy protected'). Está prohibido copiar la documentación y los demás impresos. También está prohibido desensamblar el código de máquina.

3. Modificar el programa o mezclarlo con otro para utilizar el programa resultante en una sola máquina. (Toda porción de este programa que se mezcle con otro seguirá estando sujeta a las cláusulas de este contrato.)

4. Transferir el programa y la licencia a otro usuario a condición de que comunique a DRI el nombre y las señas de ese otro usuario, y de que éste acepte (a) las condiciones establecidas en este contrato, (b) fimar y enviar a DRI una copia de la tarjeta de registro y (c) pagar los derechos de transferencia vigentes en el momento. Si usted transfiere el programa, debe al mismo tiempo: o bien transferir todas las copias, incluido el original, tanto en versión electrónica como impresa, al mismo usuario, o bien destruir todas las copias no transferidas; esto es de aplicación también a todas las modificaciones y a todas las porciones del programa que haya mezclado con otros programas.

Usted está obligado a incluir el mensaje de copyright en todas las copias, modificaciones y porciones mezcladas con otros programas.

CADA DISCO TIENE GRABADO UN NUMERO DE SERIE. USTED NO PUEDE UTILIZAR, COPIAR, MODIFICAR, TRANSFERIR NI DE NINGUNA OTRA FORMA PERMITIR QUE OTRA PERSONA LLEGUE A ESTAR EN POSESION DE ESTE PROGRAMA, NI DE NINGUNA COPIA, MODIFICACION O PORCION MEZCLADA CON OTRO PROGRAMA, SALVO EN LAS CONDICIONES PREVISTAS EN ESTE CONTRATO.

SI USTED TRANSFIERE A OTRA PERSONA UNA COPIA, MODIFICACIÓN O PORCIÓN MEZCLADA CON OTRO PROGRAMA, ESTA LICENCIA QUEDA ANULADA INMEDIATAMENTE.

3. Vigencia del contrato

Esta licencia estará en vigor mientras no sea cancelada. Usted puede cancelarla en cualquier momento destruyendo el programa junto con sus copias, modificaciones y porciones mezcladas con otros programas. También quedará cancelada en las condiciones mencionadas en otro lugar de este contrato, o por el hecho de que usted no cumpla alguna de las claúsulas del contrato. Al aceptar este acuerdo, usted se compromete a destruir todas las copias, modificaciones y porciones con otros programas en cuanto la licencia quede cancelada.

4. Garantía limitada

EL PROGRAMA SE SUMINISTRA 'TAL CUAL'. NI DRI NI AMSTRAD OFRECEN GARANTIA ALGUNA, NI EXPRESA NI IMPLICITA, INCLUYENDO LAS GARANTIAS IMPLICITAS DE COMERCIABILIDAD Y ADECUACION A UN PROPOSITO CONCRETO, AUNQUE SIN LIMITARSE A ESTAS. USTED ASUME TODOS LOS RIESGOS EN CUANTO A LA CALIDAD Y BUEN FUNCIONAMIENTO DEL PROGRAMA. SI ESTE FUERA DEFECTUOSO, USTED (Y DRI Y AMSTRAD) ASUME EL COSTO TOTAL DE LA REPARACION O CORRECCION NECESARIAS.

Ni AMSTRAD ni DRI garantizan que las funciones del programa vayan a satisfacer sus necesidades, ni que el funcionamiento del programa sea ininterrumpido y libre de error.

No obstante, AMSTRAD garantiza que el disco soporte del programa está libre de defectos, si se le da un trato normal, por un periodo de 90 días contados a partir de la fecha de la compra, la cual se acreditará mediante la factura.

5. Limitación de la responsabilidad

La responsabilidad de AMSTRAD se limita a su compromiso de sustituir el disco defectuoso, que habrá de ser enviado a AMSOFT junto con un copia de la factura.

DRI Y AMSTRAD NO SERAN EN NINGUN CASO RESPONSABLES DE NINGUN DAÑO, INCLUIDAS PERDIDAS DE BENEFICIOS, PERDIDAS DE AHORROS Y CUALESQUIERA OTROS PERJUICIOS DIRECTOS O INDIRECTOS, AUN CUANDO DRI O AMSTRAD HAYAN SIDO AVISADOS DE LA POSIBILIDAD DE TALES DAÑOS, NI DE NINGUNA RECLAMACION DE TERCEROS.

6. Tarjeta de registro

DRI actualiza de vez en cuando sus programas. DRI sólo enviará tales actualizaciones si ha recibido de usted la tarjeta de registro firmada, o si la ha recibido un receptor autorizado por DRI. DRI no está obligada a actualizar el programa ni a enviarle a usted las actualizaciones.

7. Condiciones generales

Usted no puede ceder, asignar o transferir ni la licencia ni el programa a otras personas, salvo en la forma expresamente autorizada en este contrato. Todo intento de ceder, asignar o transferir los derechos y obligaciones descritos en este contrato será nulo.

Este acuerdo será regido e interpretado de acuerdo con las leyes británicas.

Si tiene usted alguna pregunta que hacer en relación con este contrato, puede escribir a Digital Research Inc., P. O. Box 579, Pacific Grove, California 93950, EE. UU.

ESTE CONTRATO NO PUEDE SER MODIFICADO POR PROPUESTAS DE PEDIDO, POR ANUNCIOS PUBLICITARIOS NI DE NINGUNA OTRA FORMA. SOLAMENTE PUEDE SER MODIFICADO MEDIANTE CONTRATO SUSCRITO ENTRE USTED Y UN APODERADO DE DRI Y OTRO DE AMSTRAD.

USTED RECONOCE QUE HA LEIDO ESTE CONTRATO, QUE LO ENTIENDE Y QUE ESTA DE ACUERDO CON LAS CONDICIONES EN EL EXPUESTAS. USTED ACEPTA ADEMAS QUE ESTE CONTRATO ES EL UNICO ESTABLECIDO ENTRE USTED, DRI Y AMSTRAD, Y QUE ANULA CUALQUIER OTRA PROPUESTA O CONTRATO ANTERIOR, VERBAL O ESCRITO, Y CUALESQUIERA OTRAS COMUNICACIONES ENTRE USTED Y DRI O AMSTRAD EN RELACION CON ESTE ASUNTO.

LAS CONDICIONES DE ESTE ACUERDO NO AFECTAN A LOS DERECHOS LEGALES DEL USUARIO.

Acuerdo de licencia

LOS PROGRAMAS GRABADOS EN EL JUEGO DE DISCOS DEL SISTEMA SE SUMINISTRAN EN LAS CONDICIONES MÁS ABAJO INDICADAS. EL HECHO DE QUE USTED ABRA LAS CAJAS DE LOS DISCOS IMPLICA QUE ACEPTA TALES CONDICIONES. SI USTED NO CONSIDERA ACEPTABLES ESAS CONDICIONES, DEBE DEVOLVER LOS DISCOS SIN ABRIRLOS AL COMERCIO EN QUE LOS ADQUIRIÓ, DONDE LE SERÁ DEVUELTO SU DINERO. SI LAS CAJAS ESTÁN ABIERTAS, NO PODRÁ RECLAMAR LA DEVOLUCIÓN DEL DINERO SALVO CUANDO LOS DISCOS SEAN DEFECTUOSOS Y TAL DEVOLUCIÓN SEA EXIGIBLE EN CUMPLIMIENTO DE LA CLÁUSULA 7.

En lo que sigue,

«Locomotive» significa 'Locomotive Software Limited'
«Amstrad» significa 'Amstrad Consumer Electronics plc'
«El programa» significa 'los programas grabados en la cara 1 del juego de discos suministrado con el ordenador.'

1. Copyright

El programa es propiedad de Locomotive. Locomotive concede al comprador de este paquete el derecho no exclusivo de utilizar el programa en las condiciones pactadas en este Acuerdo. Esta licencia puede ser transferida solamente en los términos descritos en la Cláusula 3. Queda estrictamente prohibida cualquier utilización o transferencia que no esté expresamente autorizada en este acuerdo.

2. Utilización

El programa puede ser utilizado en una sola máquina o terminal; no obstante, se lo puede copiar y mezclar con otros programas, pero siempre con la condición de usarlo en una sola máquina. Tal copia o mezcla está además sujeta a la limitación de que no se modifique el programa y de que se conserve en la copia o programa resultante el mensaje de copyright de Locomotive. A excepción de estas copias y mezclas, cualquier otra manipulación del programa (tal como su modificación o desensamblado) está estrictamente prohibida.

3. Transferencia

El programa puede ser transferido a otro usuario a condición de que se transfiera tanto el original como las copias (o que se destruyan las copias no transferidas) y de que ese otro usuario conozca y acepte cumplir las cláusulas de este Acuerdo. Sin tal transferencia y compromiso, toda utilización del programa y sus copias por parte de cualquier persona distinta del comprador original se considerará no autorizada por Locomotive, y violará por lo tanto los derechos de propiedad de Locomotive.

4. Documentación

La documentación que se suministra con el programa está sujeta también a copyright de Locomotive. Locomotive no concede el derecho de reproducir tal documentación, ni en todo ni en parte. Si el usuario necesitase otros ejemplares de esa documentación, debe solicitarlos por escrito; Locomotive decidirá si debe facilitárselos o no.

5. Incumplimiento

Si el usuario, ya sea el comprador original u otro a quien haya sido transferido el programa, viola cualquiera de las condiciones pactadas en este Acuerdo, indemnizará a Locomotive por todos los daños ocasionados (incluida la pérdida de beneficios) y la licencia aquí concedida será considerada nula. En ese momento, el usuario tendrá la obligación de devolver a Locomotive todas las copias del programa, legales o no.

6. Exclusión de garantías

Ni Locomotive ni sus apoderados garantizan, ni implícita ni explícitamente, que el programa esté libre de errores y vaya a satisfacer las necesidades del usuario. Incumbe al usuario cerciorarse, antes de aceptar las condiciones de este Acuerdo, de que el programa cumple sus requisitos. El programa se suministra «tal cual»; a excepción de lo expresamente previsto en este acuerdo, se excluye todo tipo de garantías, implícitas o explícitas.

7. Limitación de la responsabilidad

Amstrad garantiza que el disco en el que se suministra grabado el programa está libre de defectos físicos y que, si se lo somete a un uso normal, lo seguirá estando durante un periodo de 90 días contados a partir de la fecha de la compra. En caso de que el disco tenga algún defecto (así como en satisfación de los derechos legales que asistan al usuario y no puedan ser excluidos por este Acuerdo) la responsabilidad de Amstrad se limita a la sustitución del programa o a la devolución del importe al comprador, a criterio de Amstrad.

La única excepción a lo antedicho es que Locomotive acepta la responsabilidad derivada de la muerte o las lesiones de las personas cuando sean imputables a la negligencia de Locomotive. Sin embargo, Locomotive no será en ningún caso responsable de ningún daño, incluidas pérdidas de beneficios, pérdidas de tiempo y cualesquiera otros perjuicios directos o indirectos resultantes del uso o de la imposibilidad de usar el programa o el disco, aun cuando Locomotive o sus distribuidores hayan sido avisados de la posibilidad de tales daños.

Las condiciones de este Acuerdo no afectan a los derechos legales del usuario.

AMSTRAD
ESPAÑA
AVDA. DEL MEDITERRANEO, 9 - 28007 MADRID