TK85

Programación Basic

programación basic

TK85

manual de operación

Usando al máximo su TK85

Ud. puede usar su TK 85 para varias actividades. El TK 85 puede entretenerlo, enseñarle y ejecutar todos los tipos de cálculos. Él puede ayudarlo a familiarizarse con computadores. Ud. puede aprender "BASIC" y comenzar a escribir sus propios programas.

En el hogar, Ud. puede usar su TK 85 para controlar su libreta de cheques, mantener su presupuesto doméstico, mantener una lista de domicílios o catalogar una colección. Ud. puede formar un diario de jardinería, una lista de tareas domésticas o una lista de organizaciones. El TK 85 puede ser usado como una calculadora de bolsillo, teniendo como exhibidor toda una pantalla de televisión.

Existe una variedad ilimitada de juegos que pueden ser efectuados en su computador. Muchos programas en cassette imitan los famosos juegos de video, incluyendo "alienígenas del espacio", "códigos secretos", "laberinto" y "magia"; y los juegos favoritos, de carta y tablero, como "Back Gammon", "Ajedrez" y "Paciencia".

El TK 85 puede ser su protector para ayudarlo en cualquier asunto. Ud. puede hacer programas para práctica y aprendizaje de conceptos matemáticos. Ud. puede escribir un programa que calcule el volumen del cubo o convierta pies y millas en metros y kilómetros. Ud. puede hacer su propio "arte" usando los símbolos gráficos de su computador.

En el trabajo Ud. puede mantener un archivo con informaciones de los empleados o mantener registros sobre el stock. Usando el "software" — T-KALC — Ud. puede controlar parte del presupuesto, mudar proyecciones y ventas de cada mes, y proyectar las ganancias. Ud. puede visualizar cambios que ocurren en sus negocios a través de gráficos y mapas.

Muchos programas han sido escritos para Ud. y están dispuestos en cassettes pre-programados.

Acuérdese de leer el catálogo de "software" del TK 85 donde se hace una descripción de lo que está disponible.

Con el uso de programas ya preparados, Ud. está a un paso para desarrollar sus propios programas. Trabaje cuidadosamente con el Manual "TK 85 programación BASIC". A medida que Ud. avanza, piense en diferentes situaciones, para testar sus conocimientos sobre cada concepto. Aprender a programar le puede dar práctica en el pensamiento lógico. Ud. se va a descubrir pensando sobre las actividades diarias, en términos de pasos de programación.

A medida que sus habilidades para programar se van tornando más sofisticadas, Ud. va a querer aumentar la capacidad de memoria, adicionando el módulo de 16K ó 64K.

RECURSOS: Existen varios recursos que pueden ser usados para ayudarlo a entender y usar su TK 85. Incluya a sus amigos en sus experiencias con el TK 85. Ya fueron formados varios grupos de usuarios en el país. Puede haber uno en su ciudad o bien, Ud. puede decidir formar uno. Muchos de esos grupos producen sus propias publicaciones, que contienen programas, sugerencias y novedades de interés para los usuarios del TK 85.

Muchos libros que se encuentran en el mercado, están usando el computador TK 85, desde libros para principiantes hasta algunos que describen el funcionamiento técnico del computador. Las informaciones sobre grupos de usuarios y libros, pueden obtenerse a través de los distribuidores locales y del propio fabricante.

Especificaciones del TK85

DIMENSIONES.

Ancho	230 mm.
Profundidad	140 mm.
Altura	30 mm.
Peso	500 gr.

MICROPROCESADOR/MEMORIA.

Microprocesador: Z80A con reloj de 3,25 MHZ;
ROM: 10K con interpretador "BASIC";
RAM: 16K bytes o 48K bytes de memoria interna.

TECLADO.

Tipo máquina de escribir, con 40 teclas. El uso en el modo "función", modo "gráfico" y teclas simples, dan el equivalente a 91 teclas, además de permitir usar 22 símbolos gráficos y 54 de video invertido.

VIDEO.

Necesita de un aparato de televisión normal, en blanco y negro o a colores. El cable conector del Video que acompaña al conjunto, hace la conexión del TK 85 a las terminales VHF de la antena del aparato de TV. La pantalla cuenta con 24 líneas de 32 caracteres negros en fondo blanco.

MODOS "RÁPIDO" Y "LENTO".

El TK 85 puede operar de dos modos a elección, de "software": RÁPIDO ("FAST") y LENTO ("SLOW"). El "FAST" es cuatro veces más veloz que el "SLOW". Sin embargo, en el modo "SLOW", el TK 85 puede simultáneamente procesar mientras usa el video. Esto permite un movimiento contínuo de la imagen que es útil para juegos animados en la pantalla.

PROGRAMACIÓN.

Los programas pueden ser cargados en la memoria, a través del teclado o de una cinta cassette. Los programas y datos pueden ser guardados en cintas cassette, para no perderlos cuando el TK 85 fuera desconectado.

PRUEBA DE SINTÁXIS.

La sintáxis de una línea es probada en cada entrada. Un cursor de error de sintáxis, muestra el lugar donde está el error.

El cursor de error de sintáxis desaparece, cuando éste es corregido. Sólo son aceptadas en el programa líneas sin errores de sintáxis.

GRÁFICOS.

Además de los 22 caracteres gráficos, espacios y

sus inversos, la pantalla también puede ser dividida en una resolución de 44 x 64 puntos gráficos, pudiendo aclarar u oscurecer cada uno, bajo control del programa.

EDICIÓN.

El modo de edición, le permite a Ud. editar cualquier línea del programa o de entrada, incluyendo nuevas líneas del programa. Las líneas pueden ser canceladas, aumentadas o disminuidas de tamaño.

ARITMÉTICA.

Operadores aritméticos: +, —, x, ÷, exponencial. Operadores relacionales: =, <>, >, <, <=, >=, pueden comparar "string" y variables aritméticas para campos Ø (Falso) ó 1 (Verdadero). Operadores lógicos: AND, OR, NOT dan resultados Booleanos.

NÚMEROS DE PUNTO FLUCTUANTE.

Los números son guardados en 5 "bytes" en forma binária de punto fluctuante, permitiendo una variación de $\pm$ 3 x 10^{-39} a $\pm$ 7 x 10^{38} con precisión de 9 1/2 dígitos decimales.

VARIABLES.

Numéricas: cualquier letra seguida de caracteres alfanuméricos.

"String": A$ hasta Z$.

FOR-NEXT ("loops"): A hasta Z (permitiendo cualquier "ninho" de "loops").

"Arrays" Numéricos: A hasta Z.

"Arrays" de "Strings": A$ hasta Z$.

"ARRAYS".

Los "Arrays" pueden ser multidimensionados, con la variable suscrita iniciándose en el 1.

EVALUACIÓN DE EXPRESIÓN.

Un "evaluador de expresión" es llamado siempre que una expresión, constante o variable fuera encontrada durante la ejecución de un programa. Esa poderosa característica del TK 85, permite el uso de expresiones en lugar de constantes, lo que es especialmente útil a los comandos GOTO, GOSUB, etc.

MODO DE COMANDO.

El TK 85 puede ejecutar sentencias inmediatamente, permitiendo su uso como una calculadora.

GRABADOR CASSETTE.

Use un grabador normal. El límite de transferencia es 250 "bauds", siendo usado un formato único de grabación, incompatible con otros sistemas. El TK 85 guarda tanto sea datos como programas, para evitar así la necesidad de volver a entrar con esos datos cuando el programa fuera nuevamente cargado. (Son definidos nombres a los programas para que el TK 85 pueda ubicarlos en la cinta cuando fueran solicitados).

PUERTA DE EXPANSIÓN.

En la parte posterior se encuentra la barra de datos, de direcciones y control de la CPU Z80A, como así también una tensión de alimentación de ØV, + 5V, + 9V y las líneas de selección de memoria.

REQUERIMIENTO DE ENERGÍA.

El TK 85 requiere aproximadamente de 420 mA a 7-11 VDC. El tiene un regulador interno de 5 V y viene acompañado de una Fuente de Alimentación.

JOYSTICK.

El Joystick es un periférico para ser usado con los computadores TK.

Se trata de un módulo de comando, para ser utilizado en juegos animados.

Es conectado en la parte posterior del TK 85, y sustituye a los comandos de las teclas 5, 6, 7 y 8, en las direcciones indicadas por las flechas del teclado, y está provisto también, de un botón disparador que sustituye al comando de la tecla Ø.

La utilización de él, traerá al usuario del TK, mayor comodidad y mucha más agilidad en sus decisiones durante un partido.

Economiza el uso del teclado, además de protegerlo contra golpes violentos, lo que es normal en el transcurso de una emocionante partida de video.

Con su uso, Ud. tendrá mayor habilidad de comando, debido a que su concentración está puesta sólo en el video, y no en el teclado.

Como operar el TK85

El TK 85 es un computador personal. Es un microcomputador de uso general, que puede procesar información de acuerdo con las sentencias que le son dadas, para cualquier número de infinitos propósitos: haciendo cálculos simples, jugando, aprendiendo fracciones y procesando presupuestos. Él es tan capaz e inteligente como el usuario que sigue las instrucciones. Cuando el usuario entra con las sentencias, el computador entonces puede ejecutarlas automáticamente con rapidez electrónica.

De la misma manera que el conocimiento profundo de mecánica no es necesario para manejar un auto, el funcionamiento interno de los computadores, no es necesario para operar el TK 85.

El TK 85 es el primer microcomputador a operar con tan pocos "CHIPS". Los principales circuitos que él contiene son:

1 - La Unidad de Procesamiento Central (CPU) que dirige todo el funcionamiento del computador y controla las operaciones de máquina. Ella también ejecuta el procesamiento aritmético y lógico de los datos;

2 - El sistema operacional está situado em la memoria "ROM" (READ ONLY MEMORY) que es preprogramada y sólo puede ser leída. La "ROM" actúa como un depósito de informaciones, donde la CPU va a buscar las operaciones de la máquina;

3 - La Memoria de Acesso Randómico o Memoria de Lectura y Grabación "RAM" (RANDOM ACCESS MEMORY) almacena informaciones para uso inmediato y alimenta la CPU. Esa memoria, al contrario de la "ROM", nos es permanente. Los programas que UD. hace son almacenados en la "RAM".

El TK 85 viene con 16 K o 48 K de memória "RAM", esto significa que tiene capacidad para mantener aproximadamente 16.000 ou 48.000 "bytes" en la Memoria de Acesso Rondómico y 10 K de memoria "ROM".

Un "byte" en el TK 85 como en otros computadores pequeños, consiste de 8 "Bits". "Bits" son Ø ó 1's, que son comprensibles al computador y que constituyen el lenguaje de la máquina o código de la máquina.

De manera general, Ud. puede considerar un "byte" como si fuera igual a un caracter en el programa (y en el TK 85 una tecla única de palavra-llave es guardada en un "byte"). Ud. va a aprender más sobre palabras-llave en el Manual "TK 85 BASIC" de la MICRODIGITAL.

Capacidades adicionales de memoria "RAM" están disponibles con el uso de módulos de memoria, que son conectados directamente en la parte posterior del TK 85 expandiendo así la memoria interna.

Para tener accesso permanente a la memoria, el usuario del TK 85 puede transferir información de la memoria "RAM" para cintas cassette. Esa información puede ser recargada en el computador, para uso posterior.

4 Los "CHIP's" lógicos del computador TK 85 auxilian a la CPU en el control de las operaciones de teclado, uso del video de TV y grabador cassette. Todos esos componentes son conectados a través de cables. Esos componentes y piezas, forman lo que es llamado "Hardware" del computador.

El usuario es capaz de interaccionar con el computador por medio de la digitación de mensajes por el teclado. El mensaje a ser insertado es preparado por el usuario, en forma de un programa en el lenguaje "BASIC". Un programa es un conjunto de sentencias que auxilian al computador a través de procedimientos, paso a paso, para resolver un problema. Tales programas son conocidos como "Software" del computador.

El TK 85 usa un lenguaje de programación "BASIC" (Beginner's Allpurpose Symbolic Instruction Code).

"BASIC" es el lenguaje más usado en microcomputadores. Es un lenguaje de alto nível, que usa la lengua inglesa de una manera tal que el computador pueda entender o por lo menos traducir a través de la "ROM".

Cuando el usuario digita un programa, la información es transferida del teclado para la CPU, para ser transformada en código de máquina, bajo el control de las instrucciones de la "ROM". Ahí las operaciones lógicas y matemáticas son ejecutadas, y los resultados transmitidos de vuelta a la memoria "RAM", a través del "CHIP" lógico, mostrados en una pantalla de TV.

Almacenando y recuperando programas

Siga estas instrucciones cuidadosamente para almacenar y recuperar programas en cinta para su TK 85.

ALMACENANDO PROGRAMAS:

- Coloque una cinta borrada y vuélvala al inicio.
- Deje pasar la marca de inicio.
- Coloque los controles de tonalidad al máximo (Graves en el mínimo y Agudos en el máximo, si ellos fueran separados).
- Conecte el plug del "MIC", pero no conecte el del "EAR".
- Presione la tecla "RECORD", conjuntamente con la tecla "PLAY" del grabador.
- Presione la tecla "SAVE" del computador y digite el "nombre del programa" y después de 1 segundo, presione la tecla "NEW LINE".
- Cuando aparezca un "Ø/Ø" en la parte inferior de la pantalla, pare el grabador. En este punto, su programa ya fué almacenado.

RECUPERANDO PROGRAMAS:

- Coloque la cinta y vuélvala al inicio.
- Desconecte el plug del "EAR" y presione la tecla "PLAY" del grabador.
- Un sonido será oído seguido de silencio y después, un zumbido agudo (el programa). Vuélva la cinta y pare en el inicio de la zona silenciosa.
- Coloque los controles de tonalidad en el máximo (Graves en el mínimo y Agudos en el máximo si fueran separados).
- Coloque el control del volúmen aproximadamente en el nivel 8 en una escala de 1 (mínimo) a 10 (máximo). El volúmen varia para distintos grabadores. Si Ud. no consigue cargar la primera vez, intente controlar el volúmen en diferentes niveles, usando medio nivel cada vez, hasta que encuentre el nivel correcto para su grabador.
- Conecte el cable del "EAR" pero no el del "MIC".
- Presione la tecla "LOAD" del computador seguido del "nombre del programa" y entonces presione la tecla "PLAY" del grabador y después de 1 segundo presione la tecla "NEW LINE".
- Cuando aparezca un "Ø/Ø" en la parte inferior de la pantalla, pare el grabador y presione la tecla "RUN".

Yo seguí las instrucciones para almacenar y recuperar programas correctamente, pero todavia tengo algunos problemas. ¿Cómo puedo resolverlos?

- *Cuando cargue un programa en su computador, use el siguiente procedimento como guía:*
  - A - La pantalla debe mostrar líneas finas con inclinación de 30° antes del inicio de la función de carga (LOAD).
  - B - La imagen del paso anterior A, debe ser seguida de fuertes barras negras, moviéndose mientras el programa se carga (LOAD).
  - C - La imagen entonces debe quedar clara, mostrando el símbolo "Ø/Ø" en la orilla inferior izquierda de la pantalla.

IDENTIFICACIÓN DE PROBLEMAS.

Si ocurriera A; B; A; entonces intente LOAD" " en vez de LOAD *"nombre del programa".*

Si ocurriera solamente A, eso indicará que el programa no está siendo leído, entonces aumente el volumen y verifique si oye el ruido del programa en la cinta.

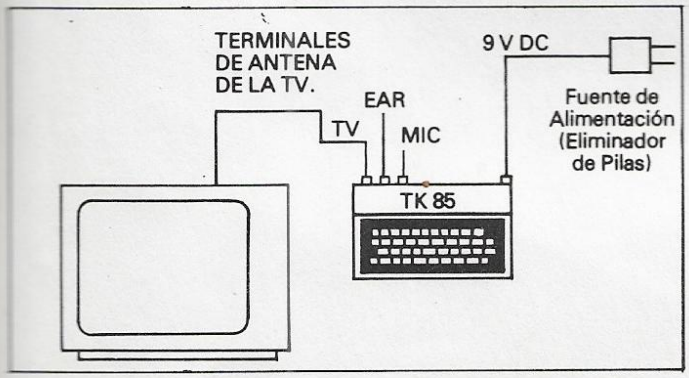
Si ocurriera solamente B, eso indicará que el programa comenzó su carga (LOAD) a partir del medio, preocúpese de volver la cinta al inicio de la zona silenciosa.

Índice

capítulo 1

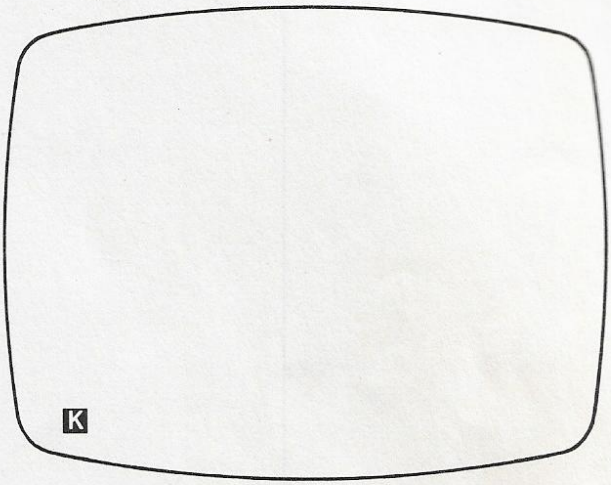
Colocando el TK85 en operación

1. Desconecte la antena VHF de su (aparato de) televisión.
   (*) Conecte el Cable Conector de Video del computador a los tornillos de la antena de VHF, en la parte posterior de su aparato de televisión.
2. Una vez conectado el Cable Conector de Video de su computador, vuelva a conectar la antena. (Esta conexión es opcional, porque la recepción normal de TV puede ser reestablecida convenientemente cambiando el interruptor de "ON" de la Fuente de Alimentación para la posición "OF" cuando la antena estuviera conectada.
3. Conecte el Cable Conector de Video, de aproximadamente 120 cm. de la TV. al zócalo de TV. del TK 85. Ud. no deberá tener problemas en la decisión sobre donde conectar este cable, pués él se conecta sólo en los lugares antes mencionados.
4. Prenda la Fuente de Alimentación del TK 85 y conecte el plug del cable al zócalo de 9V DC del computador. Ud. no va a dañar al computador si conecta inadvertidamente en los lócalos EAR o MIC. Conecte el transformador al tomacorriente.
Verifique si el interruptor de la Fuente de Alimentación está en la posición "ON".



5. Conecte su aparato de TV. y sintonice el Canal 2.
6. En ese momento, la pantalla de la TV se va a aclarar y el cursor aparecerá en la orilla inferior izquierda de la misma.

El cursor es una letra "K" invertida (K). Ella indica su posición en la pantalla. Ud. también deberá oir un zumbido bajo y probablemente tendrá deseos de bajar el volumen. Para sintonización fina, ajuste el control vertical y horizontal de su aparato de TV. Así Ud. obtendrá una imagen más nítida.



7. Ahora Ud. puede comenzar a programar. Pruebe un poco y trate de sentir el teclado. Nada que Ud. digite podrá causar daño. Presione alguna de las teclas en el teclado del TK 85. Al contrario de la máquina de escribir, Ud. no sentirá ningún movimiento cuando presione; la presión de sus dedos será suficiente para activar las teclas. Al presionar las teclas, Ud. verá palabras y letras apareciendo en la pantalla. Éste y cualquier otro símbolo que el TK 85 mostrara, son conocidos como caracteres.
8. Una palabra sobre el teclado: Sin duda Ud. quedará fascinado con la cantidad de caracteres presentes en el teclado del TK 85. Cuando Ud. lea el Manual del "TK 85 Programación BASIC" de la Microdigital, Ud. va a aprender como tener acceso a cada uno de esos caracteres. Al contrario de la máquina de escribir, que generalmente opera con dos modos de impresión, el TK 85 puede operar de cuatro modos diferentes, permitiendo que ciertas teclas produzcan hasta seis caracteres diferentes. (Esta capacidad del TK 85 permite al usuario obtener el equivalente a 91 teclas, 22 caracteres gráficos y 54 caracteres de video invertido). La habilidad de digitar comandos con sólo un toque, es otra capacidad del TK 85.

capítulo 2

Diciendo al computador lo que debe hacer

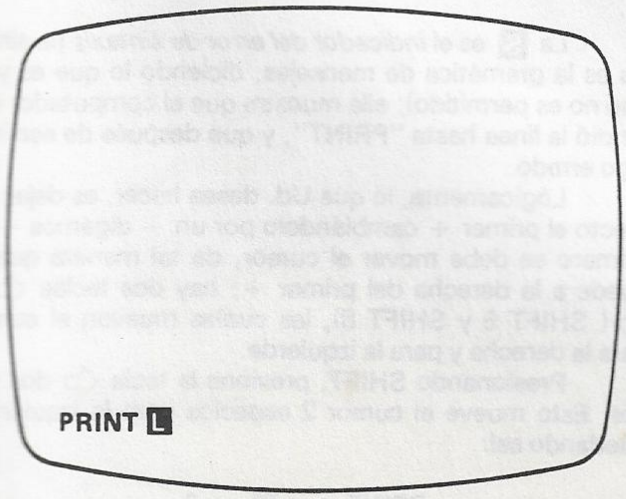
Ahora que Ud. tiene en la pantalla la imagen presentada en el Capítulo 1, Ud. puede digitar sentencias especiales de computadores. Por ejemplo:

PRINT 2 + 2

hace procesar al computador la suma 2 + 2 y exhibir (PRINT) el resultado en la pantalla. Una sentencia como ésta, la cual el computador ejecuta de inmediato, es técnicamente llamada *comando* BASIC. (Hay otras instrucciones BASIC usadas en los programas — ellas no son ejecutadas de inmediato. Serán vistas en los Capítulos del 8 al 13.)

Para digitar un comando:

1. Primeiro digite PRINT. Pero, como Ud. puede ver, a pesar que el teclado tiene una tecla para cada letra, Ud. no necesita digitar la palabra P,R,I,N,T. Tan pronto como Ud. presione P, toda la palabra aparecerá en la pantalla, seguida de un espacio, y la pantalla quedará de la siguiente manera:



La razón de esto, es que en el comienzo de cada comando el computador está esperando una *palabra-llave* — una palabra que especifique cual es el tipo de comando. Las palabras llaves están escritas encima de la teclas, y Ud. verá que "PRINT" aparecerá sobre la tecla P, de tal manera, que para conseguir el comando "PRINT", basta presionar P.

El computador informa que está esperando una palabra llave a través de la letra K que aparecerá en el video. Casi siempre existe una letra blanca sobre fondo negro (video invertido) que puede ser una K o una L (o también podremos ver la letra F o G), llamado cursor. La K significa que cualquier tecla a ser presionada, debe ser considerada como la palabra — llave. Como Ud. vió, después de haber presionado la P para PRINT, la K cambió para una L.

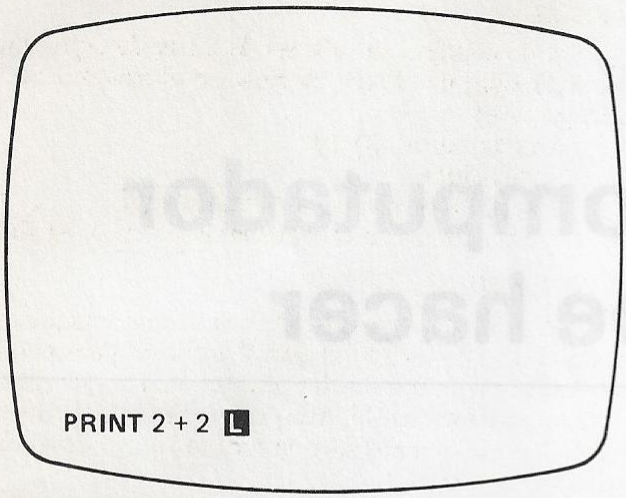
Este sistema de presionar sólo una tecla para obtener un símbolo más, es muy usado en el TK 85. En el resto de este manual, las palabras que tengan sus propias teclas son impresas en negro. Las palabras-llaves no pueden ser digitadas letra por letra.

2. Ahora digite 2. Esto no debe causar ningún problema. Nuevamente Ud. deberá ver aparecer el 2 en su pantalla, y la letra L avanzar un espacio.

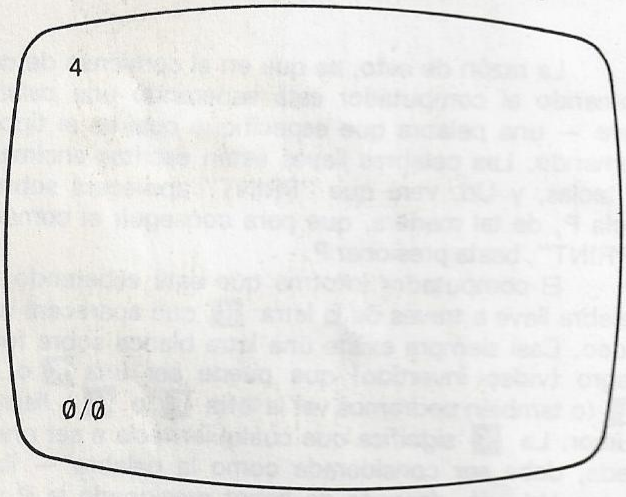
Fíjese también, que un espacio es automáticamente colocado entre el PRINT y el 2, para que sea más nítido. Esto es hecho en lo posible, de tal manera que Ud. no tenga que digitar el espacio. Si Ud. digita un espacio, este aparecerá en la pantalla, pero no afectará al comando.

3. Ahora digite +. Este es un caracter *mayúscula* (tales caracteres están marcado en amarillo — el color de la tecla SHIFT en la orilla superior de la tecla), y para conseguir "+" Ud. debe presionar la tecla SHIFT y al mismo tiempo, presionar la tecla que contiene los símbolos LIST, K, + y LEN.

4. Ahora nuevamente digite 2. La pantalla quedará de ésta manera:



5. Ahora presione NEWLINE. Ud. presiona esta tecla cuando finaliza la línea de instrucción. El computador entonces procesará. En nuestro caso, la pantalla cambiará para:



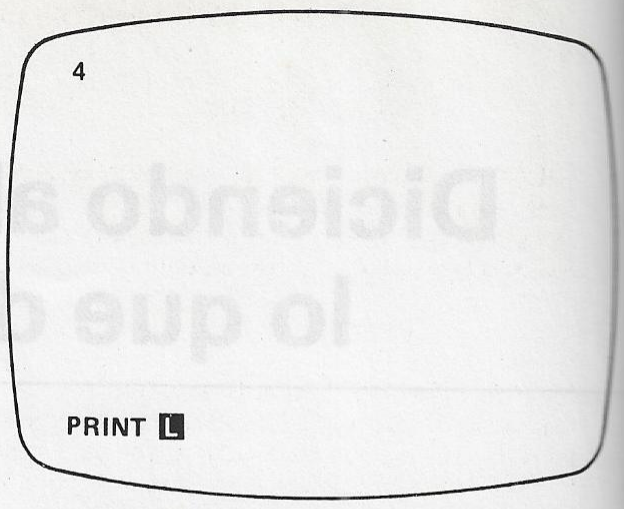
4 es la repuesta — pero está claro que Ud. no precisa comprar un computador para descubrir eso.

Ø/Ø es la *indicación* a través de la cual él informa como el resultado fué conseguido (note que el cero está escrito cruzado por una barra, para que sea diferenciado de la 0 mayúscula. Esto es muy común en la área de computación).

El primer Ø significa "OK, sin problemas". (En el apéndice B hay una lista de otros códigos de *indicaciones* que pueden aparecer; por ejemplo, si alguna cosa estuviese errada.)

El segundo Ø significa que "la última cosa hecha fué la línea Ø". Ud. verá más tarde, cuando comience a escribir programas, que para una sentencia puede ser dado un número, para que podamos guardarla para ejecución posterior: ella pasa a ser entonces una *línea de programa.* En realidad, los comandos no tienen números, pero por conveniencia de las indicaciones el computador los trata como línea Ø.

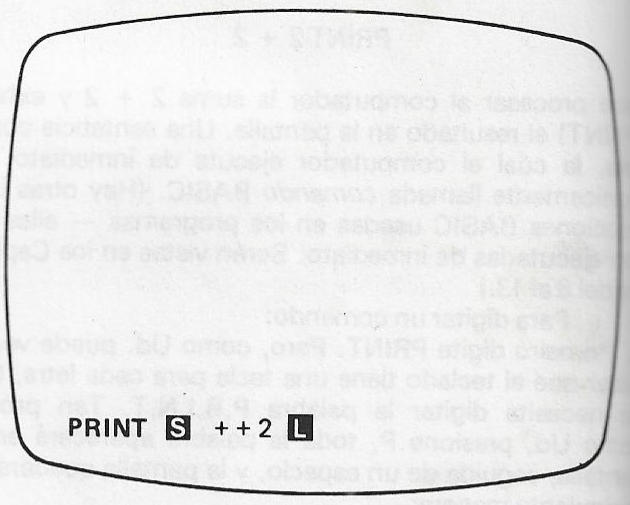
Ud. debe considerar la indicación, como si ella estuviese escondiendo el cursor K — si Ud. presiona P para PRINT, la indicación desaparecerá y la pantalla cambiará para



El cursor también puede ser usado para corregir errores: digite + + 2, para obtener

PRINT + + 2 L

en la línea inferior, cuando Ud. presione NEWLINE, obtendrá:



La S es el *indicador del error de sintaxis* (la sintaxis es la gramática de mensajes, diciendo lo que es y lo que no es permitido); ella muestra que el computador entendió la línea hasta "PRINT", y que después de eso hay algo errado.

Lógicamente, lo que Ud. desea hacer, es dejar sin efecto el primer + cambiándolo por un — digamos — 3. Primero se debe mover el cursor, de tal manera que él quede a la derecha del primer +; hay dos teclas ⇦ y ⇨ (SHIFT 5 y SHIFT 8), las cuales mueven el cursor para la derecha y para la izquierda.

Presionando SHIFT, presione la tecla ⇦ dos veces. Esto mueve el cursor 2 espacios para la izquierda, quedando así:

PRINT + L + 2

Ahora presione la tecla RUBOUT (SHIFT Ø), y Ud. obtendrá:

PRINT L + 2

RUBOUT anula el caracter o palabra-llave inmediatamente para la izquierda del cursor.

Si ahora Ud. presiona el 3, será insertado "3", nuevamente a la izquierda del cursor, dando

PRINT 3 L + 2

y presionando NEWLINE da la respuesta (5).

La tecla ⇨ (SHIFT 8) funciona exactamente igual a la tecla ⇦, siendo que mueve el cursor para la derecha al contrario de la anterior, que lo mueve para la izquierda.

RESUMEN

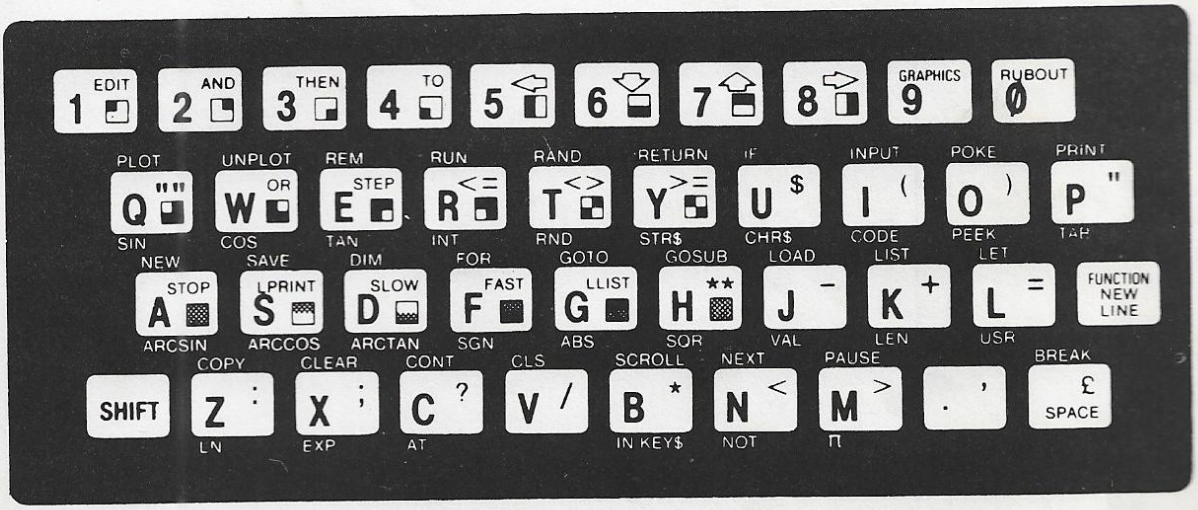
Este capítulo se refiere a la parte de como digitar mensajes para el TK 85, explicando el sistema de digitación simple:

Los cursores K L
Indicaciones,
Indicador de errores de sintaxis, S
Como corregir errores usando ⇦, ⇨ y RUBOUT.

EL TECLADO.

Si Ud. estuviera familiarizado con máquinas de escribir. Ud. puede utilizar esta ilustración para aprender donde se encuentran las letras. Acuérdese que para caracteres mayúsculas Ud. debe presionar la tecla SHIFT al mismo tiempo que presiona la otra tecla. No confunda el dígito Ø con la letra O.

Aquí está una reproducción del teclado.



capítulo 3

Comentarios sobre el lenguaje Basic

Las sentencias que Ud. digita en el computador están en un lenguaje de computador llamado BASIC (Beginner's ALL-PURPOSE Symbolic Instruction Code). El computador, en realidad, hace muchos esfuerzos para transformar las sentencias BASIC en sus propias operaciones rudimentarias. El lenguaje BASIC contiene bastantes palabras en inglés (como PRINT) para hacerlas fácil de memorizar.

El BASIC fué desarrollado en el Dartmouth College, en New Hampshire, en el año 1964, y desde entonces, él se tornó el lenguaje de computador más usado por principiantes y personas aficionadas a ese hobby. Esto se debe en gran parte, a que el BASIC se ha adaptado bien al uso *on-line,* donde el usuario digita alguna cosa y el computador responde inmediatamente.

Hay otros lenguajes de programación — tales como el PASCAL — con estructura más regular y mucho más poderosa que el BASIC, pero sólo algunos, como APL, son fáciles de usar on-line. Otros que deben ser mencionados son FORTRAN, COBOL, RPG y PL1.

Muchas revistas de computadores personales publican programas en BASIC, y vale la pena dar una mirada para buscar algunas ideas. Por supuesto, Ud. tendrá que alterarlas un poco, pués todo computador que usa el lenguaje BASIC, tiene su propia versión, que es diferente a todas las otras.

El computador como calculadora

capítulo

4

El computador como calculadora

Conecte el computador. Ahora Ud. puede usarlo como una calculadora de bolsillo, como fué descripto en el Capítulo 2: digite PRINT, y enseguida, cualquier cosa que desee resolver, y por último NEWLINE.

El TK 85 no sólo adiciona, también sustrae, multiplica (usando el asterisco en lugar del signo usual de veces — esto es muy común en computadores) y divide (usando / en vez de ÷). Pruebe.

+, —, *, y / *son operaciones* y los números por ellos operados son llamados *operandos.*

El computador también puede elevar un número a la potencia de otro, usando el signo ** (SHIFT H). Digite:

```
PRINT 2**3
```

y Ud. tendrá la respuesta 8 (2^3, ó 2 al cubo).

El TK 85 también procesará combinaciones de las operaciones. Por ejemplo:

```
PRINT 20 — 2*3**2+4/2*3
```

da la respuesta 8. Él hace eso de manera indirecta, porque primero él hace la exponenciación (**) en el orden, de izquierda para derecha; después todas las multiplicaciones y divisiones (* y /), nuevamente de izquierda para derecha, y entonces ejecuta las adiciones y sustracciones (+ y —), aún de izquierda para derecha.

Así, nuestro ejemplo es ejecutado en las siguientes etapas:

```
20 — 2*3**2 + 4/2*3
```
Primero las potencias,
```
20 — 2*9 + 4/2*3
```
enseguida multiplicaciones y divisiones,
```
20 — 18 + 4/2*3
20 — 18 + 2 *3
20 — 18 + 6
```
por último tenemos las adiciones y sustraciones.
```
2 + 6
8
```

Nosotros formalizamos esto dando para cada operación una *prioridad,* un número entre 1 y 16. Las operaciones con las más altas prioridades son ejecutadas primero, y las de igual propriedad son ejecutadas en el orden en que aparecen, de izquierda a derecha.

** tiene prioridad 10
* y / tienen prioridad 8
+ y — tienen prioridad 6

Cuando — es usado para obtener algo negativo, como por ejemplo cuando Ud. escribe — 1, él tiene prioridad 9. Este es el menos *unário,* el que difiere del menos *binário* en 3-1. (Una operación unária tiene sólo un operando, mientras que una operación binária tiene dos). Fíjese que en el TK 85 no puede ser usado + como una operación unitária.

Este órden es absolutamente rígido, pero Ud. puede alterarlo a través del uso de paréntesis: cualquier cosa entre paréntesis es ejecutada primero y después pasa a ser considerada como un número simple, de tal manera que:

PRINT 3*2 + 2

da como respuesta 6 — 2 = 8, pero

PRINT 3*(2 + 2)

da como respuesta 3*4 = 12

Una combinación como ésta as llamada *expresión* — en este caso, una expresión *aritmética* o *numérica,* porque el resultado es un número. Generalmente, cuando el computador está esperando la entrada de un número, Ud. puede digitar una expressión y él encontrará el resultado.

Ud. puede escribir números con punto decimal (el punto equivale a la coma) y también puede usar una notación científica, lo que es muy común en calculadoras de bolsillo. De esta manera, después de un número común (con o sin punto decimal), Ud. puede escribir el *exponente,* el cual consiste en la letra E, y el signo + ó —, y un número sin punto decimal. La E significa aquí "**10**", ("veces 10 elevado a la potencia de"). Así tenemos:

2.34E0 = 2.34 * 10 ** 0 = 2.34
2.34E3 = 2.34 * 10 ** 3 = 2340
2.34E−2 = 2.34 * 10 ** −2 = 0.0234 y así en adelante.
(Trate de hacer eso en el TK 85).

Ud. entenderá mejor esto, si se imagina el exponente como un cambio del punto decimal hacia la derecha (si el exponente fuera positivo) o para la izquierda (si el exponente fuera negativo).

Ud. también puede hacer un PRINT de más de un número al mismo tiempo, separándolos con comas (,) o punto y coma (; es SHIFT X). Si Ud. usa coma, el próximo número será impreso en el margen izquierdo o en el medio de la línea 16. Si Ud. usa un punto y coma, el próximo número será impreso inmediatamente después del último. Pruebe:

PRINT 1;2;3;4;5;6;7;8;9;10
PRINT 1,2,3,4,5,6,7,8,9,10

para ver las diferencias. Ud. puede mezclar comas y puntos y comas en la misma sentencia PRINT.

RESUMEN

Sentencias: PRINT, con comas o puntos y comas.
Funciones: +, —, *, /, **
Expressiones: notación científica.

EJERCICIOS:

1. Pruebe realizar:

PRINT 2.34E0
PRINT 2.34E1
PRINT 2.34E2

hasta

PRINT 2.34E15

Ud. podrá notar que el TK 85 también utiliza connotación científica. Esto es porque él nunca usa más de 14 espacios para escribir un número. De la misma forma pruebe.

PRINT 2.34E-1
PRINT 2.34E-2

y así en adelante.

2. Pruebe:

PRINT 1,,2,,3,,,4,,,,5

Una coma siempre ocupará el lugar del próximo número. Ahora pruebe

PRINT 1;;2;;;3;;;4;;;;5

¿Por qué una secuencia de punto y coma no es diferente de un único punto y coma?

3. PRINT proporciona sólo 8 dígitos significativos. Pruebe.

PRINT 4294967295, 4294967295 — 429E7

Esto prueba que el computador puede trabajar con todos estos dígitos, a pesar de no estar preparado para imprimirlos de una vez.

4. Si Ud. tiene una tabla de logaritmos, pruebe esta regla:

Elevar 10 a la potencia de un número es lo mismo que sacar el antilogaritmo de aquel número.

Por ejemplo, digite

PRINT 10**0.3010

y ve el *antilogaritmo* de 0.3010. ¿ Por qué el resultado no es exactamente igual?

5. El TK 85 utiliza aritmética de *punto fluctuante,* lo que significa que él mantiene separados los dígitos de un número (la *mantisa*) la posición del punto (el *exponente*). El resultado no siempre es exacto, ni siquiera para los números enteros.

Digite

PRINT 1E10 + 1 — 1E10, 1E10 — 1E10 + 1

Los números son mantenidos en una precisión de 9 1/2 dígitos, así 1E10 es muy grande para ser mantenido exactamente correcto. El error (falta de precisión) (en verdad 2) es más que 1, así los números 1E10 y 1E10 + 1 parecen ser iguales para el computador.

Para un ejemplo más detallado, digite:

PRINT 5E9 + 1 — 5E9

Aquí, el error en 5E9 es sólo = 1, y el 1 al ser adicionado, en realidad, se torna *redondeando* para 2. Aquí los números 5E9 + 1 y 5E9 + 2 parecen ser iguales para el computador.

El entero más grande que puede ser tratado con precisión absoluta es $2^{32} - 1$ (4.294.967.295).

Este órden es absolutamente rígido, pero Ud. puede alterarlo a través del uso de paréntesis: cualquier cosa entre paréntesis es ejecutada primero y después pasa a ser considerada como un número simple, de tal manera que:

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

da como respuesta 6 — 2 = 8, pero

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

da como respuesta 3*4 = 12

Una combinación como ésta as llamada *expresión* — en este caso, una expresión *aritmética* o *numérica,* porque el resultado es un número. Generalmente, cuando el computador está esperando la entrada de un número, Ud. puede digitar una expressión y él encontrará el resultado.

Ud. puede escribir números con punto decimal (el punto equivale a la coma) y también puede usar una notación científica, lo que es muy común en calculadoras de bolsillo. De esta manera, después de un número común (con o sin punto decimal), Ud. puede escribir el *exponente,* el cual consiste en la letra E, y el signo + ó —, y un número sin punto decimal. La E significa aquí "**10**", ("veces 10 elevado a la potencia de"). Así tenemos:

2.34E0 = 2.34 * 10 ** 0 = 2.34
2.34E3 = 2.34 * 10 ** 3 = 2340
2.34E−2 = 2.34 * 10 ** −2 = 0.0234 y así en adelante.
(Trate de hacer eso en el TK 85).

Ud. entenderá mejor esto, si se imagina el exponente como un cambio del punto decimal hacia la derecha (si el exponente fuera positivo) o para la izquierda (si el exponente fuera negativo).

Ud. también puede hacer un PRINT de más de un número al mismo tiempo, separándolos con comas (,) o punto y coma (; es SHIFT X). Si Ud. usa coma, el próximo número será impreso en el margen izquierdo o en el medio de la línea 16. Si Ud. usa un punto y coma, el próximo número será impreso inmediatamente después del último. Pruebe:

```
PRINT 1;2;3;4;5;6;7;8;9;10
PRINT 1,2,3,4,5,6,7,8,9,10
```

para ver las diferencias. Ud. puede mezclar comas y puntos y comas en la misma sentencia PRINT.

RESUMEN

Sentencias: PRINT, con comas o puntos y comas.
Funciones: +, —, *, /, **
Expressiones: notación científica.

EJERCICIOS:

1. Pruebe realizar:

```
PRINT 2.34E0
PRINT 2.34E1
PRINT 2.34E2
```

hasta

```
PRINT 2.34E15
```

Ud. podrá notar que el TK 85 también utiliza la connotación científica. Esto es porque él nunca usa más de 14 espacios para escribir un número. De la misma forma pruebe.

```
PRINT 2.34E-1
PRINT 2.34E-2
```

y así en adelante.

2. Pruebe:

```
PRINT 1,,2,,3,,,4,,,,5
```

Una coma siempre ocupará el lugar del próximo número. Ahora pruebe

```
PRINT 1;;2;;;3;;;4;;;;5
```

¿Por qué una secuencia de punto y coma no es diferente de un único punto y coma?

3. PRINT proporciona sólo 8 dígitos significativos. Pruebe.

```
PRINT 4294967295, 4294967295 — 429E7
```

Esto prueba que el computador puede trabajar con todos estos dígitos, a pesar de no estar preparado para imprimirlos de una vez.

4. Si Ud. tiene una tabla de logaritmos, pruebe esta regla:

Elevar 10 a la potencia de un número es lo mismo que sacar el antilogaritmo de aquel número.

Por ejemplo, digite

```
PRINT 10**0.3010
```

y ve el *antilogaritmo* de 0.3010. ¿ Por qué el resultado no es exactamente igual?

5. El TK 85 utiliza aritmética de *punto fluctuante,* lo que significa que él mantiene separados los dígitos de un número (la *mantisa*) la posición del punto (el *exponente*). El resultado no siempre es exacto, ni siquiera para los números enteros.

Digite

```
PRINT 1E10 + 1 — 1E10, 1E10 — 1E10 + 1
```

Los números son mantenidos en una precisión de 9 1/2 dígitos, así 1E10 es muy grande para ser mantenido exactamente correcto. El error (falta de precisión) (en verdad 2) es más que 1, así los números 1E10 y 1E10 + 1 parecen ser iguales para el computador.

Para un ejemplo más detallado, digite:

```
PRINT 5E9 + 1 — 5E9
```

Aquí, el error en 5E9 es sólo = 1, y el 1 al ser adicionado, en realidad, se torna *redondeando* para 2. Aquí los números 5E9 + 1 y 5E9 + 2 parecen ser iguales para el computador.

El entero más grande que puede ser tratado con precisión absoluta es $2^{32} - 1$ (4.294.967.295).

capítulo 5

Funciones

Matemáticamente, una función es una regla para proporcionar un número (el *resultado*) en cambio de otro (el *argumento* u *operando*) y así es realmente una operación unária. El TK 85 tiene algunas de esas funciones ya definidas íntegramente. Sus nombres son palabras escritas abajo de las teclas; SQR por ejemplo, es la conocida función raíz cuadrada, y

PRINT SQR 9

tiene como resultado 3, la raiz cuadrada de 9. Para obtener SQR, Ud. debe presionar la tecla FUNCTION (SHIFT NEWLINE). Eso cambia el cursor para **F** . Ahora presione la tecla SQR (H): SQR aparece en la pantalla y el cursor vuelve nuevamente para **L** . El mismo método funciona para todas las palabras que están escritas abajo de las teclas, las cuales casi todas son nombres de funciones.

Pruebe:

PRINT SQR 2

Ud. puede probar la precisión de la respuesta de la siguiente manera:

PRINT SQR 2*SQR 2

la cual tiene que ser 2. Note que ambos SQR's son ejecutados antes de la *, y en verdad todas las funciones (excepto una — NOT) son ejecutadas antes de las cinco operaciones +, —, *, / y **. Esta regla puede ser alterada a través del uso de paréntesis:

PRINT SQR (2*2)

el resultado es 2.

Aquí están algunas funciones más (hay una lista completa en el apéndice C).

SGN La función señal (algunas veces llamada *signum* para no confundir con SIN). El resultado es — 1, Ø ó + 1, dependiendo si el argumento es negativo, cero o positivo, respectivamente.

ABS El valor absoluto, o módulo. El resultado es siempre el argumento positivo, de tal manera que:

ABS — 3.2 = ABS 3.2 = 3.2

SIN seno
COS coseno
TAN tangente
ASN arcoseno
ATN arcotangente

Las funciones trigonométricas trabajan em radianes, no en grados.

LN logaritmo natural (base 2,718281828 . . .; llamado "E")

EXP función exponencial (base e)

SQR raíz cuadrada

INT parte entera. Esta función siempre redondea para menos; así, INT 3.9 = 3 y INT — 3.9 = — 4 (Un entero es un número no fraccionario, positivo o negativo.)

PI π = 3.1415927 (PI no tiene argumento).

RND RND tampoco tiene argumento. Él genera un número randómico entre Ø (valor que puede ser asumido) y 1 (el cual no puede asumir).

Usando la terminología del último capítulo, todas las funciones, excepto PI y RND son unárias (función con un único operando) con prioridad 11. (PI y RDN son operaciones nulas, porque no tienen operandos.)

Las funciones trigonométricas, como también EXP, LN y SQR, son generalmente calculadas con precisión de 8 dígitos.

RND y RAND están en la misma tecla, pero mientras que RND es una función, RAND es una palabra-llave, como PRINT. RAND es usado para controlar la *randomicidad* de RND.

RND no es verdaderamente randómico, pués sigue una secuencia de 65536 números tan mezclados, que parece ser randómico (RND es un pseudo-randómico). Ud. puede usar RAND para inicializar RND en diferentes lugares de la secuencia, digitando RAND y un número entre 1 y 65535, y luego NEWLINE.

No es importante saber donde el número dado, inicia RND, pero sí, que el mismo número después de RAND iniciará RND siempre en el mismo lugar. Por ejemplo, digite

RAND 1 (y NEWLINE)

y entonces

```
PRINT RND
```

ponga eso en "looping" varias veces (Acuérdese de usar FUNCTION para obtener RND). El resultado de RND será siempre 0.0022735596, no una secuencia randómica.

```
RAND 0
```

(el cero no necesita ser digitado) actúa ligeramente diferente: determina donde se inicia RND a lo largo del tiempo que el aparato de TV. esté conectado, y éste es genuínamente randómico.

RESUMEN

Sentencia: RAND
Funciones: SGN, ABS, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND.

EJERCICIOS

1. Para obtener los logaritmos (base 10), que Ud. encontrará en las tablas de log. divida el logaritmo natural por LN 10. Por ejemplo, para encontrar logaritmo 2.

```
PRINT LN 2/LN 10
```

la respuesta es 0.30103

Pruebe realizar multiplicaciones y divisiones usando el TK 85 como un conjunto de tablas de logaritmos de esta manera (para antilogaritmos, vea el capítulo 2 — ejercicio 3). Confiera las respuestas usando * y /. La manera directa es más precisa.

2. EXP y LN son funciones *inversas* en el sentido, que si Ud. aplica una y luego la otra, Ud. consigue de vuelta su número original. Por ejemplo,

```
LN EXP 2 = EXP LN 2 = 2
```

lo mismo acontece para SIN y ASN, para COS y ACS, y para TAN y ATN. Ud. puede usar esto para probar la precisión con la cual el computador trabaja estas funciones.

3. π radianes es igual a 180°. Para convertir grados en radianes, divida por 180 y multiplique por π . Así

```
PRINT TAN (45/180*PI)
```

que da TAN 45° (1).

Para convertir radianes en grados, divida por π y multiplique por 180.

4. Pruebe:

```
PRINT RND
```

algunas veces para verificar como varía el resultado. ¿Ud. detectó algún padrón? (Probablemente no.)

¿ Cómo usaría Ud. RND e INT para obtener un número entero entre 1 y 6, si representara un riesgo de vida? (Respuesta: INT ((RND*6) + 1)

5. Pruebe esta regla:

Suponga que Ud. escoge un número entre 1 y 872 y digite

```
RAND y su número (y NEWLINE)
```

El próximo valor de RND será

```
(75* (su número + 1) — 1) / 65536
```

6. (Sólo para matemáticos.)

p es el (mayor) número primo, y *a* es la raiz primitiva módulo p.

Luego, si b_i es el residuo de *a* módulo *p* ($1 \leqslant b_i < p - 1$), la secuencia

$$\frac{b_i - 1}{p - 1}$$

es una secuencia cíclica de un número distinto de P — 1 en la escala de 0 a 1 (excluyendo 1). Escogiendo convenientemente, puede Ud. hacer parecer esto bastante randómico.

65535 es un número primo Mersenne, $2^{16} - 1$. Use esto, con la Ley de Gauss de la reciprocidad cuadrática, para demostrar que 75 es la raíz primitiva módulo 65537.

El TK 85 usa $p = 65535$ y $a = 75$, y almacena $b_i - 1$ en la memoria. La función RND incluye la reposición de $b_i - 1$ en la memoria por $b_{i+1} - 1$, y la generación del resultado $(b_{i+1} - 1)/(p - 1)$. RAND n (con $1 \leqslant n \leqslant 65535$) hace b_i igual a N + 1.

7. INT siempre redondea para menos. Para redondear al entero más próximo, agrega 0.5 primeramente. Por ejemplo:

```
INT (2.9 + 0.5) = 3
INT (—2.9 + 0.5) = —3
INT (2.4 + 0.5) = 2
INT (—2.4 + 0.5) = —2
```

Compare estos resultados con los que Ud. obtiene cuando no agrega 0.5.

8. Pruebe:

```
PRINT PI, PI — 3, PI — 3.1, PI — 3.14, PI — 3.141
```

Esto muestra la precisión con la cual el computador almacena π .

capítulo 6

Variables

Mi calculadora de bolsillo puede almacenar un número y recordarlo más tarde. ¿ Hace eso el TK 85?

Sí, en realidad, hablando literalmente puede almacenar centenares de ellos, utilizando la sentencia LET. Suponga que la docena de huevos cueste $ 58.00, y Ud. desea almacenar eso. Digite

LET HUEVOS = 58 (y lógicamente NEWLINE)

Ahora, eso provocó una serie de acontecimientos. Primero, el computador reservó un lugar para el almacenamiento de un número; segundo, dió a ese lugar el nombre de "HUEVOS", así Ud. puede tener acceso a él posteriormente; esta combinación de lugar de almacenamiento y nombre es llamada *variable*. Tercero, almacenó el número 58 en ese lugar: nosotros decimos que él *relacionó* el valor 58 a la variable (cuyo nombre es) HUEVOS. HUEVOS es una variable numérica, porque su valor es un número.

Ud. quiere saber ¿ cuánto cuestan los huevos? Digite:

PRINT HUEVOS

Si Ud. quiere saber cuanto cuesta media docena de HUEVOS, digite:

PRINT HUEVOS/2

Suponga que el precio de los huevos subió para $ 61.00 la docena. Digite:

LET HUEVOS = 61

Esto actualiza el valor de 58 para 61. Confiera digitando:

PRINT HUEVOS

Ahora digite:

PRINT LECHE

Ud. obtendrá la indicación 2/Ø. Según el apéndice B, código 2, significa "variable no localizada" — no fué dado al computador el valor de la variable "LECHE". Digite:

LET LECHE = 18.5

Y digite:

PRINT LECHE

nuevamente.

Ud. puede usar cualquier letra o dígito en nombre de una variable, desde que la primera sea una letra. (Ud. también puede colocar espacios, sin embargo, ellos no serán considerados parte del nombre).

Por ejemplo, hay algunos conjuntos admitidos como nombres de variables:

LJQBP
X
A3
ÁREA UNO (esto es AREA UNO y son tratados como iguales)

pero estos no son admitidos:

3 B (comienza con un dígito)
TAL? (? no es una letra ni un dígito)
K (caracter invertido de video no es permitido)
ALPHA-1 (- no es letra ni dígito)

Ahora digite:

CLEAR

La función del CLEAR, es liberar todo el espacio de almacenamiento que fue reservado para variables. Pruebe digitando:

PRINT HUEVOS

nuevamente. Ud. obtendrá indicación 2 (variable no encontrada).

NOTA: en algunas versiones del BASIC es permitido omitir el LET y simplemente digitar:

HUEVOS = 58

Eso no está permitido en el TK 85. En cualquier caso; Ud. vería que es imposible digitar.

También en algunas versiones, sólo los dos primeros caracteres de un nombre son conferidos, de manera que RADIO 3 y RADIO 33 sería considerado el mismo nombre; y en algunos otros, un nombre de variable debe ser una letra seguida de un dígito. Ninguna de esas restricciones se refieren al TK 85.

Nuevamente, en algunas versiones del BASIC, si una variable no apareció en el lado izquierdo de una ilustración LET, es considerada con valor Ø. Como Ud. vió anteriormente con PRINT LECHE, eso no acontece en el TK 85.

RESUMEN.

Variables
Sentencias: LET, CLEAR.

EJERCICIOS.

1. Si Ud. no está familiarizado con elevación a la potencia (**, SHIFT H), entonces haga este ejercicio.

En su nivel más elemental, "A**B" significa "A multiplicado por si mismo B veces"; pero eso, obviamente sólo tiene sentido si B es un número entero y positivo. Para encontrar una manera que funcione para otros valores de B, nosotros consideramos la regla

A**(B + C) = A**B*A**C

Ud. no necesitará de mucho para convencerse que no funciona, cuando B y C son positivos, mas si decidiéramos que nosotros queremos que funcione aunque ellos no sean, nos inclinaremos a aceptar que

A ** Ø = 1
A ** (– B) = 1/A ** B
A ** (1/B) = B raíz de A
A ** (B * C) = (A ** B) ** C

Si Ud. nunca ha visto nada de esto, no trate de asimilarlo todo enseguida; acuérdese sólo que

A ** – 1 = 1/A
A ** (1/2) = a la raíz cuadrada de A

y tal vez, cuando Ud. se familiarice con esto, el resto comenzará a tener sentido para Ud.

Pruebe con todo esto, decir al computador que imprima varias expresiones conteniendo **, o sea,

PRINT 3 ** (2 – Ø), 3**2*3**Ø
PRINT 4 ** – 1,1/4

2. Digite:

LET E = EXP 1
PRINT E

Ahora E tiene el valor 2,718281828 . . ., la base del logaritmo natural. Pruebe la regla

EXP de un número = E ** el número para varios números.

capítulo 7

Strings

Manipulear con textos es una cosa que el TK 85 hace, y que ninguna calculadora de bolsillo puede hacer. Digite:

PRINT "OLA. YO SOY SU TK 85"
("y SHIFT P)

El texto entre comillas es llamado STRING (lo que significa una secuencia de caracteres o cadena) él puede contener cualquier caracter que Ud. quiera, excepto las comillas (") de la cadena. (Pero Ud. puede usar las llamadas comillas revertidas, " " (SHIFT Q), y ellas serán impresas como".)

Un error común de digitación de STRINGS es olvidar una de las comillas — eso hará que **S** , indicador de errores de sintaxis, entre en acción.

Si Ud. estuviera imprimiendo números, puede usar STRINGS para explicar lo que significan los números. Por ejemplo, digite:

LET HUEVOS = 61

y luego

PRINT "EL PRECIO DE LOS HUEVOS ES"; HUEVOS; "PESOS LA DOCENA."

(No se preocupe si la digitación excede la línea.)

Esa sentencia imprime tres cosas (ítem de impresión), que es: la STRING "EL PRECIO DE LOS HUEVOS ES", el número 61 (el valor de la variable HUEVOS), y la STRING "Pesos la docena". En realidad, Ud. puede hacer un PRINT de cualquier número de ítem y cualquier mezcla de strings y números (o expresiones). Note que en una string los espacios forman parte de ella, del mismo modo que las letras. Ellos no son ignorados, ni siquiera al final.

Hay varias cosas que Ud. puede hacer con las strings:

1. Ud. puede asociarlas a variables. Sin embargo el nombre de la variable debe ser especial, para demostrar que su valor es un string y no un número. El nombre puede ser sólo una letra seguida de $ (SHIFT U). Por ejemplo, digite

LET A$ = "TORTILLA DE QUESO"

y

PRINT A$

2. Ud. también puede juntarlos. Eso es llamado de *concatenación,* significando "unión", que es exactamente lo que hace. Pruebe:

PRINT "TOCINO" + "Y HUEVOS"

Ud. no puede restar, multiplicar ni dividir strings, ni elevarlas a potencias.

3. Ud. puede aplicar algunas funciones en strings para obtener números, y vice-versa.

LEN es aplicado a una string y el resultado es su largo. Por ejemplo LEN "QUESO" = 6.

VAL es aplicado a una string formada por números o expresiones numéricas. Lo que el VAL hace, es permitir que estos valores sean tratados como números y no como string.

Por ejemplo (si A = 9): VAL "1/2 + SQRA" = 3,5. Si en la string en la cual VAL es aplicada contiene variables, entonces dos reglas deben ser obedecidas.

(I) Si la función VAL fuera parte de una expresión más larga, ella debe ser el primer ítem, esto es, 10 LET X = 7 + VAL "Y" debe ser cambiado para 10 LET X = VAL "Y" + 7.

(II) VAL sólo puede aparecer en la primera coordenada de una sentencia PRINT AT, PLOT o UNPLOT

(Vea capítulos 17 y 18), esto es:

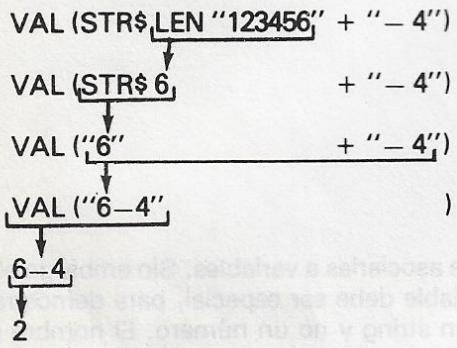
10 PLOT 5, VAL "X" debe cambiarse a:
10 LET Y = VAL "X"
15 PLOT 5, Y

STR$ se aplica a un número y lo transforma en una string, permitiendo que estos valores sean tratados por el computador como string. Por ejemplo STR$ 3,5 = "3,5".

4. De la misma forma que con números, Ud. puede hacer combinaciones creando *expresiones strings,* como:

VAL (STR$ LEN "123456" + "– 4")

la cual es procesada como:



RESUMEN.

Cadena (Strings)
Operación: + (para strings)
Funciones: LEN, VAL, STR$

EJERCICIOS.

1. Digite:

LET A$ = "2 + 2"

y entonces

PRINT A$; " = " ; VAL A$

Trate de cambiar A$ por un ítem más complicado, esto es,

LET A$ = "ATN 1 * 4"

(La respuesta aquí debe ser π).

2. La string " " sin caracteres es llamada de string *vacía* o *nula.* Es la única string con tamaño 0. Acuérdese que los espacios son significativos y una string vacía no es lo mismo que una que contiene espacios.

No la confunda con las comillas revertidas, "" (un toque único, SHIFT O). Éste es sólo un dispositivo para permitir que sean colocadas comillas en el medio de una string, porque si se escribieran comillas en el medio de una string, ocurrirá un error de sintáxis. Cuando una comilla revertida aparece en una string que tiene su comilla en el final (por ejemplo, una lista de programa), se representan como dos símbolos de comillas, para distinguirla de las comillas normales; sin embargo; cuando ella es impresa por la sentencia PRINT, sólo aparecerá una comilla.

Pruebe

PRINT "X"; " "; "X"; "" ""; " "; "" ""

3. Digite

PRINT "2 + 2 = "; 2 + 1

capítulo

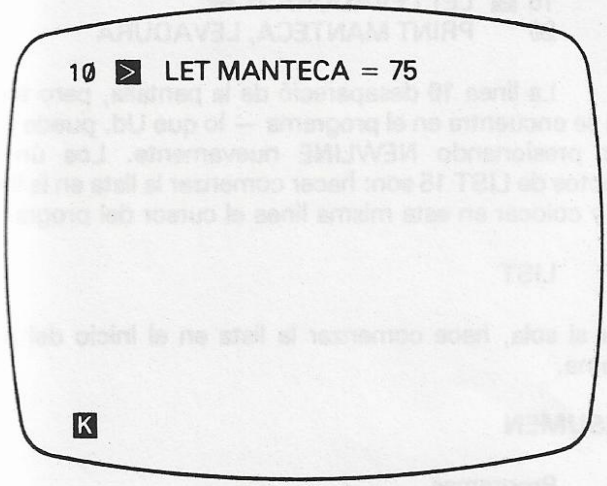
8

Programación

Y ahora finalmente Ud. puede escribir un programa de computador. Desconecte y conecte el computador para verificar que él está limpio. Ahora digite:

10 LET MANTECA = 75 (y NEWLINE)

y la pantalla quedará así:



Es diferente de lo que sucedió con HUEVOS en el capítulo 6; si Ud. digita

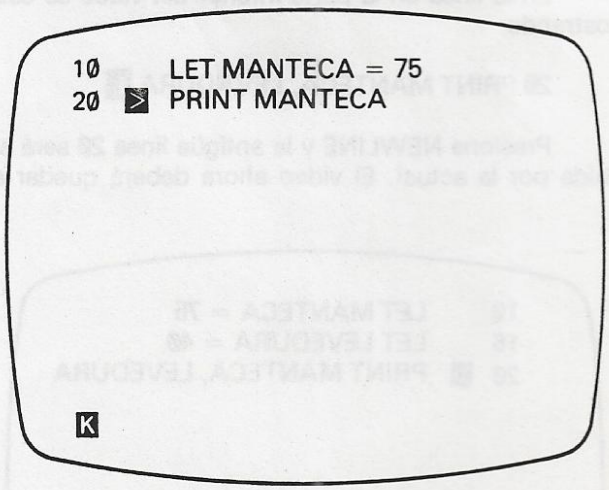
PRINT MANTECA

Ud. verá (indicación 2) que la variable MANTECA no fué definida. (presione NEWLINE nuevamente y la pantalla volverá a quedar como en la figura anterior.)

Porque la sentencia LET tenía un número 10 antes de ella, el computador no ejecutó inmediatamente, sino que la guardó para un uso posterior. 10 es el *número de línea* usado para referirse a ella, de la misma manera que los nombres son usados para referirse a variables. Un conjunto de esas sentencias almacenadas es llamado de *programa.* Ahora digite:

20 PRINT MANTECA

La pantalla debe quedar como sigue:



Esta es una *lista* de su programa. Para hacer que el programa sea *ejecutado,* digite:

RUN (y NEWLINE, por supuesto)

la respuesta 75 aparecerá en la orilla superior izquierda de la pantalla.

En la orilla inferior izquierda Ud. verá la indicación 0/20.0 como Ud. sabe, significa "OK, sin problemas", y 20 es el número de línea donde terminó la ejecución. Presione NEWLINE y la lista reaparecerá.

Observe que las sentencias fueron ejecutadas en el orden de los números de sus líneas.

Ahora, suponga que Ud. se acuerda que también necesita grabar el precio de la levadura. Digite:

15 LET LEVADURA = 40

Esto sería mucho más difícil si las dos primeras líneas estuvieran enumeradas 1 y 2 en vez de 10 y 20 (los números de las líneas deben ser enteros y estar entre 1 y 9999). Eso es porque cuando se está digitando un programa, es bueno dejar espacios en los números de línea.

Ahora Ud. necesita cambiar la línea 20 a

20 PRINT MANTECA, LEVADURA

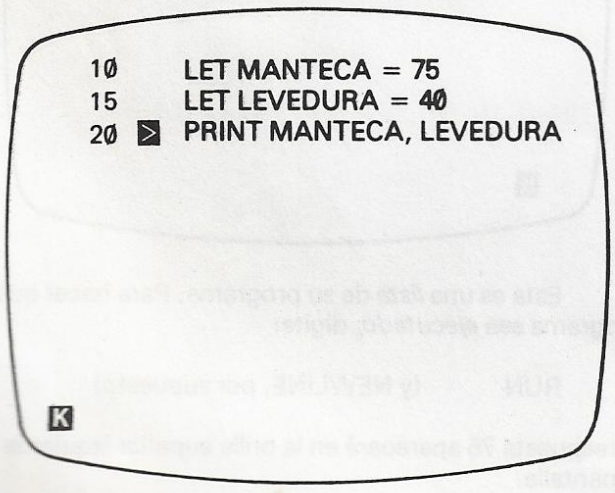
Ud. podría hacerlo digitando toda la línea, pero hay una manera de utilizar lo que ya tiene. ¿Ud. ve aquel pequeño ▶ próximo a la línea 15? Este es el *cursor de programa*, es la línea para la cual él apunta y es la *línea corriente*. Presione la tecla (SHIFT 6), y él se moverá para abajo, para la línea 20 (↑ mueve el cursor para arriba nuevamente.) Ahora presione la tecla EDIT (SHIFT 1) y una copia de la línea 20 aparecerá en la parte inferior de la pantalla. Presione la tecla → 9 veces, de tal manera que el cursor se mueva para el final de la línea y entonces digite:

LEVEDURA (sin NEWLINE)

En la línea en la parte inferior del video se estará mostrando

20 PRINT MANTECA, LEVADURA L

Presione NEWLINE y la antigüa línea 20 será sustituída por la actual. El video ahora deberá quedar así:



Cuando Ud. ejecute el programa (RUN), ambos precios serán impresos.

(Aquí hay un truco útil empleando EDIT, para ser usado cuando Ud. quiera borrar la parte inferior de la pantalla.

Presione EDIT y la línea corriente será llevada para la parte inferior, sustituyendo lo que Ud. quiera dejar sin efecto. Si presiona NEWLINE, la línea será recolocada en el programa sin alteraciones, y la parte inferior del video quedará limpia, dejando sólo el cursor.)

Ahora digite:

12 LET LEVADURA = 40

Eso quedará en el programa. Para dejar sin efecto esa línea innecesaria, digite

12 (con NEWLINE, por supuesto)

Ud. notará que el cursor desapareció. Debe imaginarlo como si él estuviera entre las líneas 10 y 15, entonces se presiona ↑ y él irá para la línea 10, mientras que si presiona ↓ él se irá para la línea 15.

Por último, digite:

LIST 15

y Ud. verá en el video

15 ▶ LET LEVADURA = 40
20 PRINT MANTECA, LEVADURA

La línea 10 desapareció de la pantalla, pero todavía se encuentra en el programa — lo que Ud. puede probar presionando NEWLINE nuevamente. Los únicos efectos de LIST 15 son: hacer comenzar la lista en la línea 15 y colocar en esta misma línea el cursor del programa.

LIST

Por si sola, hace comenzar la lista en el inicio del programa.

RESUMEN

Programas
Editando programas usando ↑ ↓ y EDIT.
Sentencias: RUN, LIST

EJERCICIOS.

1. *Modifique el programa de tal manera, que él no sólo* muestre los dos precios, sino también cual es cual.

2. Use la tecla EDIT para modificar el precio de la manteca.

3. Ejecute el programa y luego digite:

PRINT MANTECA, LEVADURA

Las variables aún permanecen, a pesar de haber terminado el programa.

4. Digite

12 (y NEWLINE)

Nuevamente el cursor del programa será escondido entre las líneas 10 y 15. Ahora presione EDIT y la línea 15 irá para la parte inferior de la pantalla: cuando el cursor del programa está escondido entre dos líneas, EDIT comienza en la siguiente. Digite NEWLINE para limpiar la parte inferior de la pantalla.
Ahora, digite

30

En esta ocasión, el cursor del programa está escondido después del fín del programa; y si Ud. presiona EDIT, entonces la línea 20 irá para abajo.

5. Introduzca una sentencia LIST en el programa, de tal manera que, cuando Ud. lo ejecute, él haga el listado por sí mismo.

capítulo

9

Más programación

Digite:

NEW

Esto borrará cualquier programa y variables del TK 85, (Esto es casi como CLEAR, sin embargo el CLEAR sólo borra las variables.)

Ahora digite este programa cuidadosamente:

```
10 REM ESTE PROGRAMA EXTRAE RAÍZ QUADRADA
20 IMPUT A
30 PRINT A, SQR A
40 GOTO 20
```

REM en la línea 10 es usado para comentarios; está ahí solamente para recordarle lo que hace el programa; el computador ignorará esta instrucción.

Ahora ejecute el programa. Aparentemente la pantalla quedó blanca y no aconteció nada, pero mire en la orilla inferior izquierda: donde debería estar una K hay una L — la máquina está esperando que sea digitado un número (o inclusive una expresión), y ella no proseguirá hasta que Ud. haya dado una entrada de datos. Después de eso, tendrá el mismo efecto de

20 LET A = . . . lo que Ud. digitó.

Digite un número — digamos 2 — y lugo presione NEWLINE. El 2 y su raíz cuadrada aparecerán en la pantalla, y UD. puede pensar que sólo fué eso. Pero, nó. La máquina quiere otro número. Eso es debido a la línea 40, GOTO 20, que significa "vaya para la línea 20". en lugar de ejecutar y parar, el computador vuelve a la línea 20 y comienza nuevamente. Así, digite otro número (es aconsejable que sea positivo).

Después de algunas veces, digite STOP (SHIFT A) en lugar del número.

Para reiniciar el programa digite:

CONT

(abreviatura de CONTINÚE) y el computador borrará la pantalla y pedirá otro número.

Para CONT el computador toma el número de línea en la última indicación, que tenga un código diferente de 0 y salta para esa línea. Siempre que la última indicación sea D/20 (y D no es 0), en nuestro caso CONT es idéntico a GOTO 20.

Ahora digite números hasta que la pantalla quede llena. Cuando esté llena, el computador parará con la connotación 5/30.5, significa "pantalla llena", y 30 es el número de la sentencia PRINT que estaba tratando de ejecutar, cuando él descubrió que no había espacio. Nuevamente,

CONT

borrará la pantalla y continuará — esta vez, CONT significa GOTO 30.

Fíjese que la pantalla quedo limpia, no porque CONT sea una sentencia, pero sí porque es un comando. Todos los comandos (excepto COPY, el cual aparece en el capítulo 20) borran en primer lugar la pantalla.

Cuando Ud. esté cansado de eso, use STOP para parar el programa y obtenga un listado apretando NEWLINE.

Vea la sentencia PRINT en la línea 30. Cada vez que el par de números A y SQR A son impresos, en una nueva línea, eso es porque la sentencia PRINT no termina con coma o punto y coma. Siempre que esto ocurre, la próxima sentencia PRINT comenzará a imprimirse en una nueva línea. (Así, para imprimir una línea en blanco, use una sentencia PRINT en la cual no haya nada para imprimir — solamente un PRINT).

Sin embargo, una sentencia PRINT puede termi-

nar en una coma o punto y coma, y entonces el próximo PRINT se imprimirá, como si los dos fuesen una sentencia larga.

Por ejemplo, con coma cambie la línea 30 por

```
30 PRINT A,
```

y ejecute el programa para ver como sentencias sucesivas, PRINT puede imprimir en la misma línea, pero, separadas en dos columnas.

Por otro lado, con punto y coma, con

```
30 PRINT A;
```

quedan todas juntas.

Pruebe también:

```
30 PRINT A
```

Ahora digite estas líneas extras:

```
100 REM ESE PROGRAMA MIDE STRING
110 INPUT A$
110 INPUT A$, LEN A$
130 GOTO 110
```

Este programa es diferente del último, sin embargo Ud. puede mantener a ambos en la memoria de la máquina al mismo tiempo. Para ejecutar el nuevo programa, digite

```
RUN 100
```

En este programa es dada la entrada de una STRING en lugar de números, imprimiéndolas, como sus tamaños.
Digite

```
GATO                (y NEWLINE como siempre)
```

Debido a que el computador está esperando unas string, él imprime dos comillas; esto es una alerta para Ud. y, normalmente, evita también que UD. las digite. Pero, UD. no tiene que limitarse a constantes de tipo string (string explícita con comillas de apertura y cierre y caracteres intermedios); el computador procesará cualquiera expresión string, tal como si fuera una variable string. En ese caso, Ud. tendría que borrar las comillas impresas por el computador. Pruebe. Bórrelas (con ⇨ y RUBOUT dos veces) y digite:

```
A$
```

Puesto que A$ todavía tiene el valor "GATO", el resultado es 4 nuevamente.

Ahora introduzca

```
A$
```

nuevamente, esta vez sin borrar las comillas. Ahora A$ tiene el valor "A$", y la respuesta es 2.

Si desea usar STOP para la entrada de strings, primero debe mover el cursor para el inicio de la línea, usando SHIFT EDIT.

Ahora mire el comando RUN 100 que dimos anteriormente. Él salta para la línea 100, ¿ no podríamos entonces haber hecho GOTO 100? En ese caso la respuesta es sí, pero, hay una diferencia. RUN 100, primero borra las variables (como el CLEAR en el capítulo 6), y después de eso funciona de la misma forma que el GOTO 100. El GOTO 100 no borra nada. Hay ciertas ocasiones en que se desea ejecutar el programa sin borrar las variables. Aquí GOTO es necesario y RUN podría ser desastroso; así es mejor no adquirir el hábito de digitar RUN automáticamente para ejecutar un programa.

Otra diferencia es que Ud. puede digitar RUN sin número de línea, y él iniciará la ejecución en la primera línea del programa. GOTO siempre debe tener un número de línea.

Ambos programas pararán porque Ud. digitó STOP en la línea INPUT; sin embargo, algunas veces — debido a un error —, Ud. escribe un programa que no consigue parar y que no parará por si solo. Digite

```
200 GOTO 200
RUN 200
```

Éste parece que quedará eternamente en ejecución, a menos que Ud. desconecte; sin embargo hay una solución menos drástica. Presione la tecla SPACE, la cual, si UD. mira bien, tiene "BREAK" escrito encima de las letras.

El programa se detendrá, apareciendo D/200.

Al final de cada línea del programa, el computador verifica si la tecla fué presionada; si fué, entonces se detiene. El BREAK también puede ser usado, cuando están siendo usados el grabador y la impressora.

Ahora UD. ya vió las sentencias PRINT, LET, INPUT, RUN, LIST, GOTO, CONT, CLEAR, NEW y REM y la mayoría de ellas puede ser usada tanto como comandos, como también, como líneas de programa — lo que es verdad para casi todas las sentencias en BASIC. La única excepción, en realidad, es INPUT, la cual no puede ser usada como un comando (UD. obtendrá una indicación 8 si lo intenta; la razón es que la misma área interna del computador es usada para comandos y para área de datos, y un comando INPUT ocasionaría confusión.) RUN, LIST, CONT, CLEAR y NEW no son de mucha utilidad en el programa, pero pueden ser utilizados.

RESUMEN

Sentencias: GOTO, CONT, INPUT, NEW, REM, PRINT, STOP, BREAK.

EJERCICIOS.

1. En el programa de la raíz cuadrada, trate de sustituir la línea 40 por GOTO 5, GOTO 10 ó GOTO 15 se produce una diferencia perceptible en la ejecución del programa. Si el número de la línea de una sentencia GOTO se refiere a una línea inexistente, el salto será hecho para la línea inmediatamente posterior. Lo mismo ocurrirá para RUN; de hecho RUN por sí sola significa RUN 0.

2. Ejecute el programa de longitud de string; cuando él pida una entrada de datos, digite

X$ (después de borrar las comillas)

Desde luego que X$ es una variable indefinida y Ud. obtendrá una indicación 2/110.

Si ahora Ud. digita

LET X$ = "ALGUNA COSA DEFINIDA"
(la cual tiene su propia indicación 0/0) y
CONT

Ud. verá que puede usar X$ como un dato de entrada sin ningún problema.

El punto llave en ese ejercicio es que CONT tiene el mismo efecto que GOTO 110. Eso anula la connotación 0/0, porque él tiene código 0, y por lo tanto, toma el número de la línea de la indicación anterior, 2/110. Eso es muy útil. Si un programa para, debido a algún error, Ud. puede hacer cualquier cosa para corregirlo, y CONT todavía funcionará.

3. Plantee éste programa:

```
10 INPUT A$
20 PRINT A$ ; "=" ; VAL A$
30 GOTO 10
```

(Vea el capítulo 7, ejercicio 1).

Aumente otra sentecia PRINT, de tal manera, que el computador informe lo que va a hacer y entonces pida los datos.

4. Escriba un programa para dar entrada de precios y hacer una remarcación de esos precios en un límite de (15%). Nuevamente, ponga una sentencia PRINT, de tal forma, que el computador explique lo que está haciendo. Modifique el programa para que pueda introducir el valor del I.V.A. (para permitir presupuestos futuros).

5. Escriba un programa para imprimir el total de los números que digita. (Sugerencia: trabaje con dos variables: TOTAL — que debe contener 0 en el inicio — e ÍTEM. De entrada a la variable ÍTEM, y agregue TOTAL, imprima ambos, y comience de nuevo).

6. Los listados automáticos (los que no son resultado de una sentencia LIST) pueden llegar a confundirlo. Si Ud. digita un programa con 50 líneas, todas con sentencia REM,

```
1 REM
2 REM
3 REM
.  .
.  .
.  .
49 REM
50 REM
```

Ud. estará habilitado para probar.

La primera cosa que Ud. debe recordar es que la línea actual (con ▶) aparecerá en la pantalla, preferiblemente próximo al medio.

Digite:

LIST (y NEWLINE, por supuesto)

luego presione NEWLINE nuevamente. Ud. debería obtener las líneas de 1 hasta 22 en la pantalla. Ahora digite:

23 REM

Ud. debería obtener las líneas del 2 hasta 23 en la pantalla. Digite:

48 REM

y Ud. obtiene las líneas 27 hasta 48. (En ambos casos, digitando una nueva línea, Ud. mueve el cursor del programa de tal manera, que un nuevo listado debe ser confeccionado).

El computador mantiene un registro no sólo de línea actual — la que tiene que aparecer en la pantalla — sino, también la línea más alta de la pantalla.

Cuando él trata de hacer un listado, la primera cosa a hacer, es comparar la línea superior con la línea actual. Si la línea superior viene después, entonces no es el lugar donde se deba comenzar, así él usa la línea actual como una nueva línea superior y hace su listado.

Por otro lado, primero trata de hacer el listado comenzando en la línea superior. Si la línea actual aparece en la pantalla, todo bién. Si la línea actual está sólo un poco abajo de la pantalla, traslada la línea superior un lugar e intenta nuevamente; y si la línea actual está bastante abajo de la pantalla, cambia la línea superior de tal manera, que ella se torne la línea anterior a la línea actual.

Pruebe moviendo la línea actual en el programa, digitando

número de línea REM

LIST mueve la línea actual, pero no la línea superior; así los listados siguientes deben ser diferentes. Por ejemplo, digite

LIST

para obtener el listado, entonces presione NEWLINE nuevamente para hacer que la línea 0 pase a ser la línea superior. UD. Debería tener en la pantalla las líneas del 1 al 22. Digite:

LIST 22

que proporciona las líneas desde el 22 hasta el 43; cuando Ud. presiona LIST y NEWLINE nuevamente, Ud. obtiene de nuevo las listas del 1 al 22. Eso tiende a ser más útil para programas cortos que para programas largos.

7. ¿ Qué hacen CONT, CLEAR y NEW en un programa?
   ¿ UD. puede pensar en algún uso para esas sentencias?

capítulo 10

If . . .

Todos los programas vistos hasta ahora han sido bastante simples — ellos seguían todas las sentencias y luego volvían al inicio nuevamente. Eso no es muy útil. En la práctica, el computador está apto para distinguir entre diferentes casos y actuar de acuerdo a cada uno; lo hace empleando la sentencia IF.

Borre el computador (usando NEW) y digite y ejecute este ridículo programita:

```
10 PRINT "PUEDO CONTARLE UN CHISTE?
20 INPUT A$
30 IF A$ = "DE NINGUNA MANERA" THEN
   GO TO 200
40 PRINT "CUÁNTAS PIERNAS TIENE UN CA-
   BALLO MONTADO?
50 INPUT PIERNAS
60 IF PIERNAS = 6 THEN GOTO 100
70 PIRNT "NO, SEIS. CUATRO DEL CABALLO
   Y DOS DEL CABALLERO"
80 STOP
100 PRINT "SI",   "QUIERE QUE YO LO CUEN-
   TE DE NUEVO?"
110 GOTO 20
200 PRINT "ESTÁ BIÉN. NO LO CONTARÉ."
```

Antes que discutamos la sentencia IF, Ud. primero debe mirar la sentencia en la línea 80: una sentencia STOP detiene la ejecución del programa, dando una indicación 9.

Ahora, como Ud. puede ver, una sentencia IF tiene el siguiente formato:

IF condición THEN sentencia

La sentencia aquí es GOTO, pero podría ser de cualquier tipo, inclusive otro IF. La condición es alguna cosa que debe ser probada, como verdadera o falsa. Si la condición fuera verdadera, la sentencia después del THEN es ejecutada, en caso contrario es ignorada.

La condición más usada es comparar dos números o dos strings: ella puede probar si dos números son iguales; o si uno es mayor que el otro: y puede probar si dos strings son iguales, o si una viene antes que la otra en el orden alfabético.

Para eso son usadas las relaciones = , < , > , < = , > = y < >.

= , el cual hemos usados dos veces en el programa (una vez para números y otra para strings), significa "igual". No es lo mismo que el = en una sentencia LET.

< significa "es menor que", así

```
      1 <  2
    - 2 <- 1
y   - 3 <- 1
```

todos son verdaderos, pero,

```
      1 <  0
y     0 <- 2
```

no (ellos son falsos).

Para ver como funciona eso, vamos a escribir un programa que pide para que digitemos una serie de números e imprimir el mayor de ellos hasta el momento.

```
10 PRINT "NÚMERO", "MAYOR HASTA
   AHORA"
20 INPUT A
30 LET MAYOR = A
40 PRINT A, MAYOR
50 INPUT A
60 IF MAYOR < A THEN LET MAYOR = A
70 GOTO 40
```

La parte crucial está en la línea 60, la cual actualiza el MAYOR, si su valor antiguo fuera menor que el nuevo valor digitado en A.

> (SHIFT M) significa "es mayor que". Es semejante a <, sólo que es el inverso. Ud. puede distinguirlos acordándose que la parte más aguda apunta para el número supuestamente menor.

<= (SHIFT R — no lo digite como < seguido de =) significa "es menor que o igual a", así es igual a < excepto que la decisión es tomada si los dos números son iguales: así 2 < = 2 ejecuta, sin embargo 2 < 2 no ejecuta.

> = (SHIFT Y) significa "es mayor que o igual a" y es similar a >.

< >(SHIFT T) significa "no es igual a", opuesto en significado a =.

Los matemáticos normalmente escriben < =, > = y < > como $\leq$, $\geq$ y $\neq$. Ellos también escriben secuencias como "2 < 3 < 4" para significar "2 < 3 y 3 < 4", pero eso no es posible en BASIC.

Esas relaciones pueden ser combinadas a través del uso de *operaciones lógicas* AND, OR y NOT.

Una relación AND otra relación
ejecuta siempre que ambas sean verdaderas.

Una relación OR otra relación
ejecuta siempre que una de las dos relaciones sea verdadera.

NOT relación
ejecuta siempre que la relación sea falsa.

Las expressiones lógicas pueden ser construídas con relaciones y AND, OR y NOT de la misma forma que las expresiones numéricas pueden realizarse con números y +, − y así en adelante; incluso Ud. puede hasta colocar paréntesis, si fuera necesario, en las expressiones lógicas. NOT tiene prioridad 4, AND 3 y OR 2.

Para ilustrar ésto borre el computador y plantee este programa:

```
10 INPUT F$
20 INPUT A
30 IF F$ = "X" AND A < 10 THEN
PRINT "SI"
50 GOTO 10
```

Finalmente, no sólo podemos comparar números, sino que también strings. Nosotros vimos " =" usado en F$ = "X". ud. puede usar cualquier otro, como por ejemplo, .

¿ Qué significado tiene para una strings "menor que"? Una cosa que no significa es "más corto", así que no cometa este error. Nosotros hacemos la distinción que una string es menor que otra, si ella viene primero en el orden alfabético, así:

```
"SMITH"     < "SMYTHE"
"SMYTHE     > "SMITH
"BILLION"   < "MILLION"
"DOLLAR"    < "POUND".
```

todas son verdadeiras < = significa "es menor que o igual a", y así sucesivamente, de la misma forma que para los números.

Nota: en algunas versiones del BASIC — pero, no en el TK 85 — la sentencia IF puede tener el formato:

IF condición THEN número de línea.

Eso es lo mismo que:

IF condición THEN GOTO número de línea.

RESUMEN

Sentencias: IF, STOP
Operaciones: =, < , > , < =, > =, < > ,
AND, OR.
Fucnión: NOT

Ejercicios

1. < > y = son opuestos en el sentido que NOT A = B significa lo mismo que A < > B y

NOT A <|> B es lo mismo que A = B

Convénzase que > = es opuesto a <, y < = es opuesto a >, y que Ud. puede usar la sentencia NOT frente a una de las relaciones del par, para transformarla en la otra, su opuesta.

De la misma forma,

NOT (una primera expresión lógica, AND una segunda expresión)

es lo mismo que

NOT (la primera) OR NOT (la segunda),
NOT (una primera expresión lógica OR una segunda)

y es lo mismo que

NOT (la primera) AND NOT (la segunda).

Utilizando esto, Ud. puede trabajar con NOT entre paréntesis, cuando fuera eventualmente aplicado en una relación. Así, Ud. puede usarlo a voluntad. Sin embargo, lógicamente hablando, NOT no es necesario, pero UD. todavía puede hallar que utilizándolo hace que el programa sea más claro.

2. En Basic, muchas veces pueden aparecer frases. Considere por ejemplo, la cláusula "Si A no es igual a B o C". ¿ Cómo Ud. escribiría eso en BASIC? [La respuesta no es

'IF A < > B OR C' ni 'IF A < > B OR A < > C']

No se preocupe si Ud. no entiende los ejercicios 3, 4 y 5, los puntos que son vistos en ellos, son bastantes refinados.

3. (A menos que UD. ya conozca BASIC perfectamente, saltee éste ejercicio.)

Pruebe:

PRINT 1 = 2,1 2

lo que UD. espera es que le de un error de sintáxis. en realidad hasta donde el computador puede comprender, no existe nada más allá de los valores lógicos.

(i) =, <, >, < =, > = y < > son todos operaciones binárias con prioridad 5. El resultado es 1 (para verdadero), y Ø (para falso), cuando el resultado de una relación fuera Ø las sentencias no son ejecutadas.

(ii) en

IF condición THEM sentencia

La condición puede, en realidad, ser cualquier expresión numérica. Si su valor fuera Ø, entonces es considerada como falsa y cualquier otro valor como verdadero. Así, la sentencia IF significa lo mismo que

IF condición < > Ø THEN sentencia

(iii) AND, OR y NOT son también operaciones binárias

X AND Y tiene valor
- X si Y no es cero (es considerada como verdadera)
- Ø si Y es cero (es considerada como falsa)

X OR Y tiene valor
- 1 si Y no es cero
- X si Y es cero

y

NOT X tiene valor
- Ø si X no es cero
- 1 si X es cero

Con eso en mente, lea el capítulo nuevamente, asegurándose que todo eso funciona.

En las expresiones X AND Y, X OR Y y NOT X, X e Y van nuevamente a asumir el valor Ø ó 1, para falso o verdadero. Prepare 1Ø condiciones diferentes y vea si en ellas AND, OR y NOT, hacen lo que Ud. espera que hagan.

4. Intente realizar este programa:

```
1Ø INPUT A
2Ø INPUT B
3Ø PRINT (A AND A > = B) + (B AND A < B)
4Ø GOTO 1Ø
```

Cada vez, él imprime el mayor número entre A y B. ¿ Por qué? Convénzase que Ud. puede pensar.

X AND Y

significa
'X si Y (o entonces el resultado es Ø)'
y

X OR Y

significa
'X a menos que Y (en el caso en que el resultado es 1)'

Una expresión AND y OR utilizada de esta manera es llamada expresión *condicional.*

Un ejemplo usando OR podría ser

```
LET PRECIO POR UNIDAD = PRECIO ACTUAL*
(1.15 OR U$ = "ARTICULO SIN IMPUESTO
I.V.A.").
```

Ahora note como OR tiende a la adición (porque su valor neutro es Ø), y AND tiende a la multiplicación (porque su valor neutro es 1).

5 Ud. también puede realizar strings asumiendo valores en expresiones condicionales, pero solamente usando AND.

X$ AND Y tienen valores
- X$ si Y no fuera Ø
- " " si Y fuera Ø

entonces significa "X$ si Y (o entonces la string vacía)" Plantee éste programa, que tiene como entrada dos strings y colóquelas en orden alfabético.

```
1Ø INPUT A$
2Ø INPUT B$
3Ø IF A$ < = B$ THEN GOTO 7Ø
4Ø LET C$ = A$
5Ø LET A$ = B$
6Ø LET B$ = C$
7Ø PRINT A$; " "; ("<" AND A$ < B$) +
("=" AND A$ = B$); " "; B$.
8Ø GOTO 1Ø
```

6. Experimente este programa:

```
1Ø PRINT "X"
2Ø STOP
3Ø PRINT "Y"
```

Cuando Ud. lo ejecute, irá imprimiendo "X" y se detendrá con la indicación 9/2Ø. Ahora digite:

CONT

Ud. debe suponer que esto funciona como "GOTO 2Ø".

Pero de esta manera, el computador sólo iría a pararse nuevamente, sin imprimir "Y". Esto no es muy útil, así el programa es preparado de tal manera, que de indicaciones con código 9 (sentencia STOP ejecutada), el número de línea es aumentado en 1, para que la sentencia CONT funcione correctamente. Así, en nuestro ejemplo, "CONT" funciona como "GOTO 21" (el cual una vez que no tiene líneas entre 2Ø y 3Ø, funciona como "GOTO 3Ø").

7. Muchas versiones del BASIC (pero, no el BASIC del TK 85) tienen una sentencia ON, la cual tiene el formato:

ON expresión numérica GOTO número de línea, número de línea, . . ., número de línea. Cuando la expresión numérica es resuelta; suponga que el resultado obtenido sea 3, entonces el efecto sería:

GOTO enésimo número de la línea

Por ejemplo,

ON A GOTO 100, 200, 300, 400, 500

Aquí, si el valor fuera 3, entonces 'GOTO 300' es ejecutado. En BASIC del TK 85, eso puede ser substituído por:

GOTO 100* A

En los casos en que los números de líneas no fueran progresando de 100 en 100, crée una manera de utilizar este recurso

GOTO una expresión condicional

en vez del anterior.

capítulo 11

El conjunto de caracteres

Las letras, dígitos, signos de puntuación y demás signos gráficos que pueden aparecer en *strings* son llamados caracteres y formam el alfabeto, o el conjunto de caracteres, que el TK 85 usa. La mayoría de estos caracteres son símbolos simples, pero hay algunos llamados *señales,* que representan una palabra entera, como en el caso de PRINT, STOP, ** y otros.

Hay 256 caracteres en total y cada uno de ellos tiene un código entre 0 y 255. Una lista completa de ellos, aparece en el apéndice A. Para convertir códigos y caracteres existen dos funciones: CODE y CHR$.

CODE es aplicada a una string y proporciona el código del primer caracter de la string (ó 0, si la string fuera vacía).

CHR$ es aplicada a un número Y proporciona el caracter, cuyo código es un número.

Ese programa imprime todo el conjunto de caracteres.

```
10 LET A = 0
20 PRINT CHR$ A;
25 PAUSE 20
30 LET A = A + 1
40 IF A < 256 THEN GO TO 20
```

Al principio Ud. puede ver los símbolos ", £, $ y así en adelante, hasta Z; todos aparecen en el teclado y pueden ser digitados cuando tenemos el cursor [L] . Más adelante, Ud. puede ver los mismos caracteres en blanco sobre fondo negro (imagen en negativo). Si Ud. presiona GRAPHICS (SHIFT 9), el cursor quedará [G] significa *modo gráfiço.* Si Ud. digita un símbolo, él aparecerá en su imagem en negativo, y eso continuará hasta que Ud. presione la tecla GRAPHICS nuevamente. RUBOUT tendrá su significado normal. Tenga cuidado de no perder el cursor [G] entre los caracteres invertidos que UD. digitó.

Cuando Ud. haya experimentado un poco, deberá tener todavía el conjunto de caracteres al principio de la pantalla de lo contrario, ejecute el programa nuevamente. Los primeros son un espacio en blanco y 10 símbolos gráficos en negro, blanco y bloques grises. Además hay otros 11. Todos ellos son llamados de *símbolos gráficos* y son utilizados para formar dibujos y figuras. Ud, puede digitarlos, por el teclado, usando el modo gráfico. (Excepto para el espacio en blanco, que es un símbolo común, usando el cursor [G] ; el cuadro negro es la inversa del espacio). Use las 20 teclas que tienen símbolos gráficos escritos. Por ejemplo: Suponga que Ud. deseara el símbolo ▘ , que está en la tecla T. Presione GRAPHICS (SHIFT 9) para obtener el cursor [G] ; entonces presione SHIFT T. De la descripción anterior, Ud. puede estar esperando obtener un símbolo en negativo, pero SHIFT T es normalmnte < >, una marca, y las marcas no tienen inversas. Entonces, Ud. consigue en su lugar el símbolo gráfico ▘ .

Aquí están los 22 símbolos

Símbolo	Código	Como obtenerlo	Símbolo	Código	Como obtenerlo
	0	K or L SPACE		128	G SPACE
	1	G shift 1		129	G shift Q
	2	G shift 2		130	G shift W
	3	G shift 7		131	G shift 6
	4	G shift 4		132	G shift R
	5	G shift 5		133	G shift 8
	6	G shift T		134	G shift Y
	7	G shift E		135	G shift 3
	8	G shift A		136	G shift H
	9	G shift D		137	G shift G
	10	G shift S		138	G shift F

Ahora vea el conjunto de caracteres nuevamente. Las marcas aparecen claramente separadas en dos bloques. Hay un pequeño grupo de 3 (RND, INKEY$ y PI) después de Z y un gran grupo, comenzando con las comillas después de Z , y continuando de AT hasta COPY.

El resto de los caracteres parecen ser "?", todos ellos. Esta es en realidad la manera en que se obtienen impresos; el signo de interrogación real está entre: y (. Fuera de los paréntesis, algunos son caracteres de control, como ⇦ , EDIT y NEWLINE, y el resto no tiene ningún significado especial para el TK 85.

RESUMEN.

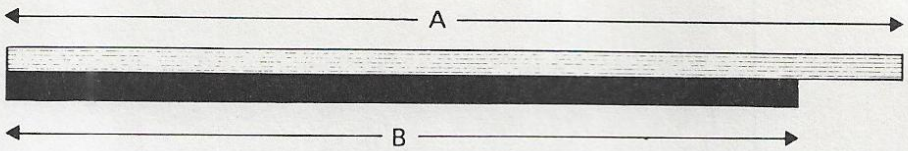
Funciones: CODE, CHR$.

EJERCICIOS.

1. Imagine el espacio entre un símbolo dividido en cuatro cuartos: ⊞ Si cada cuarto puede ser tanto negro como blanco, hay 2*2*2*2 = 16 posibilidades. Encuentre todas en el conjunto de caracteres.

3. Imagine el espacio para un símbolo dividido en 2 horitalmente: ⊟ Si cada mitad puede ser blanca, negra o gris, hay 3*3 = 9 posibilidades. Búsquelas.

3. Los caracteres del ejercicio anterior son dibujados para ser usados en histogramas horizontales, usando dos colores, gris y negro. Escriba un programa que dibuje un histograma de dos valores A y B (ambos entre 0 y 32).



Ud. deberá iniciar imprimiendo " ▀ " y entonces cambiar para " ▀ " o para " ▄ " dependiendo si A fuera mayor o menor que B.

¿ Qué es lo que su programa hace si A y B no son números enteros? O si ellos no están en el grupo de 0 a 32? Un buen programa hará alguna cosa sensible y útil.

4. En el teclado, en la A y en la H, hay 2 caracteres distintos totalmente grises si Ud. los mira bien de cerca, verá que la H es como un tablero de ajedrez en miniatura, en cuanto que la A es un tablero de ajadrez invertido. Si Ud los imprime uno al lado del otro, verá que ellos no se unen correctamente. El del A es usado para unir correctamente con ▀ y ▄ (en S y D), mientras que el del H une perfectamente con ▀ y ▄ (en F y G).

5. Ejecute este programa

```
10 INPUT A
20 PRINT CHR$ A;
30 GO TO 10
```

Si Ud. prueba, verá que CHR$ es redondeado para el entero más próximo. Si A no estuviera en al grupo de 00 hasta 255, el programa se detiene dando la indicación B.

6. Usando los códigos para caracteres, podemos expandir el concepto de orden alfabético, para ser aplicable para strings conteniendo cualquier caracter, no sólo letras. Si, en lugar de pensar en términos de alfabeto común de 28 letras, nosotros utilizaremos el alfabeto expandido de 256 caracteres, en el mismo orden de sus códigos, el principio es exactamente el mismo. Por ejemplo, las siguientes strings están en orden alfabético.

```
"ZACHARY"
" ▄ "
"(ASIDE)"
"123 TAXI SERVICE"
"AASVOGEL"
"AA"
"ZACHARY"
" AA RDVARK"
```

Esta es la regla: primero, compare el 1º caracter en las dos strings, si ellos fueran diferentes, el código de uno es menor que el código del otro, y la string que tiene el primer caracter con código menor, es la que está primero en el orden alfabético. Si ellos son iguales, prosiga y compare el segundo caracter. Si en ese proceso, una de las strings acaba primero que la otra, ésta es la anterior, en caso contrario, ellas son obviamente iguales.

Digite nuevamente el programa del ejercicio 5 del capítulo 10, aquel en que digitamos dos strings y las colocó en orden utilícelo para experimentar.

7. Este programa imprime toda la pantalla con caracteres gráficos randómicamente.

```
10 LET A = INT (16*RND)
20 IF A >= 8 THEN LET A = A + 120
30 PRINT CHR$ A;
35 PAUSE 20
40 GO TO 10
```

¿ Cómo funciona?

capítulo

12

Loops (lazos)

Suponga que Ud. quiere sumar cinco números. Una manera es escribir

```
10 LET TOTAL = 0
20 INPUT A
30 LET TOTAL = TOTAL + A
40 INPUT A
50 LET TOTAL = TOTAL + A
60 INPUT A
70 LET TOTAL = TOTAL + A
80 INPUT A
90 LET TOTAL = TOTAL + A
100 INPUT A
110 LET TOTAL = TOTAL + A
120 PRINT TOTAL
```

Este método no es bueno para programar. Puede hasta ser controlable para cinco variables, pero imagine que monótono sería sumar diez números de esa manera y cien sería prácticamente imposible. Es mucho mejor definir una variable para contar hasta cinco y luego detener el programa por ejemplo:

```
10 LET TOTAL = 0
20 LET COUNT = 1
30 INPUT A
40 REM COUNT = NÚMERO DE VECES QUE FUÉ DIGITADO
50 LET TOTAL = TOTAL + A
60 LET COUNT = COUNT + 1
70   IF COUNT < = 5 THEN GO TO 30
80 PRINT TOTAL
```

Fíjese que fácil es cambiar la línea 70 para que el programa adicione 10 o hasta 100 números.

Esta manera de calcular es tan útil, que hay dos sentencias para harcela más fácil:

La sentencia FOR y la sentencia NEXT. Ellas siempre se usan juntas. Si Ud. usa la sentencia FOR y la sentencia NEXT en el programa anterior, él quedará así:

```
10 LET TOTAL = 0
20 FOR C = 1 TO 5
30 INPUT A
40 REM C = NÚMERO DE VECES QUE FUÉ DIGITADO
50 LET TOTAL = TOTAL + A
60 NEXT C
80 PRINT TOTAL
```

Para obtener este programa del anterior, UD. sólo tiene que editar las líneas 20, 40, 60 y 70. TO y SHIFT 4.

Fíjese que cambiamos COUNT para C. La variable de cálculo — o variable de control de un *loop* FOR — NEXT — debe tener sólo una letra como nombre.

El efecto de éste programa es que C asume los valores 1 (el valor inicial), 2, 3, 4, y 5 (el límite); y para cada valor las líneas 30, 40 y 50 son ejecutadas. Luego, cuando C terminó sus 5 valores, la línea 80 es ejecutada.

Un recurso extra es, que la variable de control no necesita ser iniciada con el valor 1: Ud. puede cambiar ese valor para cualquier otro, a través del uso de la parte STEP de la sentencia FOR. La forma más general para una sentencia FOR es:

FOR variable de control = valor inicial TO límite STEP incremento, donde la variable de control es sólo

una letra y el valor inicial, el límite y el incrementa son expresiones numéricas. Así si UD. cambia la línea 20 del programa por:

```
20 FOR C = 1 TO 5 STEP 3/2
```

C asumirá los valores 1,2.5 y 4.

Fíjese que Ud. no queda limitado a números enteros, y también que el valor del control no necesitará alcanzar exactamente el límite — él permanecerá en looping siempre que su valor sea menor o igual al límite pero, vea el ejercicio 4).

Ud. debe tener cuidado si está ejecutando 2 loop FOR-NEXT, uno dentro de otro. Pruebe éste programa, el cual incluye un conjunto completo de piezas de dominó:

```
10 FOR M = 0 TO 6
20 FOR N = 0 TO M
30 PRINT M;":";N;" ";      } Loop N   } M
40 NEXT N
50 PRINT
60 NEXT M
```

Ud. puede notar que el loop N está enteramente contenido en el loop M — en otras palabras, ellos están en nido. Lo que se debe evitar, es que dos loops FOR — NEXT sean sobrepuestos sin que estén enteramente el uno dentro del otro, como este:

```
 5 REM PROGRAMA ERRADO
10 FOR M = 0 TO 6
20 FOR N = 0 TO M
30 PRINT M;":";N;" ";      } Loop M   } N
40 NEXT M
50 PRINT
60 NEXT N
```

RESUMEN.

Sentencias: FOR-NEXT

Dos loops FOR — NEXT deben estar, o contenidos uno dentro de el otro, o completamente separados.

Otra cosa que se debe evitar, es saltar de afuera hacia adentro de un loop FOR-NEXT. La variable de control es establecida correctamente sólo cuando su sentencia FOR es ejecutada, y si Ud. omite eso, la sentencia NEXT causará problemas.

Ud. debe conseguir una indicación de error 1 ó 2 (significando que una sentencia NEXT no contiene una variable de control) si tuviera suerte.

EJERCICIOS.

1. Escriba el programa del capítulo 11, que imprime el conjunto de caracteres, usando un loop FOR-NEXT.

2. Una variable de control no tiene sólo un nombre y un valor, tiene también un límite, un incremento y un número de línea para retorno de un looping (la línea después de la sentencia FOR; cerciórese que, primero, cuando una sentencia FOR es ejecutada, todas esas informaciones están disponibles (empleando el valor inicial como primer valor); y después, que (utilizando como un ejemplo nuestro segundo y tercer programa) ésta información es suficiente para convertir la línea.

```
NEXT C
en dos líneas
LET C = C + 1
IF C < = 5 THEN GOTO 30
```

3. Ejecute el tercer programa y digite

```
PRINT C
```

¿Por qué la respuesta es 6 y no 5?

4. Altere el programa para que en lugar de sumar automáticamente 5 números, él pregunte la cantidad de números que Ud. quiere sumar. Cuando Ud. ejecuta este programa ¿qué acontece si Ud. digita 0, significando que Ud. no quiere sumar ningún número? ¿Por qué Ud. esperaría que eso causase problemas para el computador a pesar de estar claro lo que Ud. desea? (El computador tiene que buscar la sentencia NEXT C, lo que no siempre es necesario).

5. Plantee éste programa para imprimir números de 1 al 10 en orden invertido.

```
10 FOR M = 10 TO 1 STEP — 1
20 PRINT M
30 NEXT M.
```

Convierta este programa en uno que no use el loop FOR-NEXT de la misma manera que Ud. convertiría el programa 3 en programa 2 (vea ejercicio 2). ¿Por qué el incremento negativo lo hace ligeramente diferente?

capítulo 13

Slow y Fast

El TK 85 puede operar en dos velocidades: SLOW (lento) y FAST (rápido). Cuando es inicialmente conectado, el computador trabaja en modo SLOW y puede computar y mostrar información en la pantalla simultáneamente. Este modo es ideal para mostrar imágenes animadas en la pantalla.

El TK 85 puede ejecutar un programa 4 veces más rápido y esto acontece cuando el computador no usa la pantalla, excepto, cuando no tiene otra tarea para ejecutar.

Para observar esto digite FAST(SHIFT y F).

Ahora cuando Ud. presione una tecla, la pantalla parpadeará y esto ocurre porque el computador deja de mostrar la pantalla, mientras ejecuta la operación indicada por la tecla pulsada.

Escriba ahora el seguiente programa:

```
10 FOR N = 0 TO 255
20 PRINT CHR$ N;
30 NEXT N
```

Cuando ejecute esto, la pantalla quedará oscura hasta el fín del programa. La pantalla será mostrada durante la sentencia INPUT, mientras el computador espera que Ud. digite INPUT.

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

Para volver al modo normal, digite SLOW (SHIFT y D).

El modo FAST es más adecuado cuando:

(I) Su programa contiene muchos cálculos numéricos.

(II) Ud. está digitando un programa largo.

Se puede usar la sentencia SLOW y FAST en un programa sin ningún problema.

Por ejemplo:

```
10 SLOW
20 FOR N = 1 TO 64
30 PRINT "A";
40 IF N = 32 THEN FAST
50 NEXT N
60 GOTO 10
```

RESUMEN:

FAST, SLOW.

capítulo 14

Sub-rutinas

Algunas veces, partes diferentes de su programa tendrán trabajos similares para hacer, y Ud. se encontrará digitando las mismas líneas dos veces. Sin embargo, esto no es necesario. Ud. puede digitar las líneas una sóla vez, en la forma conocida como *sub-rutina* y entonces, usarlas o llamarlas en cualquier otro lugar del programa, sin tener que digitarlas nuevamente.

Para hacer esto, use las sentencias GOSUB y RETURN.

GOSUB n

Donde n es el número de la primera línea de la sub-rutina, como GOTO n, excepto, que el computador recuerda el número de línea de la sentencia GOSUB, de manera que pueda regresar después de hacer la sub-rutina. Hace esto colocando el número de línea (la dirección de *retorno*) en el tope de una pila de sentencias (el STACK GOSUB).

RETURN

Toma el número de línea del tope del stack GOSUB y va a la línea siguiente.

Como un primer ejemplo:

```
10 PRINT "ESE ES EL PROGRAMA PRINCIPAL"
20 GOSUB 1000
30 PRINT "Y NUEVAMENTE"
40 GOSUB 1000
50 PRINT "Y ESO ES TODO"
60 STOP
1000 REM LA SUB-RUTINA COMIENZA AQUÍ
1010 PRINT "ESA ES LA SUB-RUTINA"
1020 RETURN
```

Sin la sentencia STOP en la línea 60 el programa continuaría con la sub-rutina y daría lugar al error 7 cuando encontrase la sentencia RETURN.

Como otro ejemplo, suponga que Ud. desea aumentar un programa para trabajar con metro (M), centímetro (CM) y milímetro (MM). Ud. tendrá 3 variables: M, CM y MM (tal vez otras — M1, CM1 y MM1 y así en adelante). La aritmética es fácil. Primero Ud. separa las cantidades en metros, centímetros y milímetros y trabaja con cada una independientemente de la otra — por ejemplo, si Ud. suma dos distancias, suma los metros, adiciona los centímetros y milímetros separadamente. Para aumentar al doble la distancia, Ud. dobla los metros, los centímetros y los milímetros y luego ajusta las cantidades para la forma correcta, de modo que los milímetros estén entre 0 y 10 y los centímetros entre 0 y 100. La última etapa es común a todas las operaciones, de forma que podemos hacerlo en sub-rutina.

Dejando de lado la noción de sub-rutina por un momento, intente UD. mismo de hacer el programa, pués vale la pena. Dando valores arbitrarios para M, CM, y MM, ¿ cómo convertirlos en los números correctos de metros, centímetros y milímetros?

Lo primero que vendrá a la mente as algo del tipo 1M..220CM..415MM, que Ud. debe convertir en 3M.. 61CM..5MM. Eso no es muy difícil.

Pero suponga que Ud. tiene números negativos. Volvamos a nuestros valores iniciales — 1M..— 220CM.. — 415MM, que deben ser convertidos para — 3M.. — 61CM..— 5MM. ¿ Y que tal fracciones? Si Ud. divide 3M..15CM..75MM por dos, Ud. obtiene 1.5M..7.5CM.. 37.5MM, lo cual, ciertamente nos es tan bueno como 1M..61CM..2.5MM.

A continuación una posible solución:

```
1000 REM SUB-RUTINA PARA CONVERTIR METROS, CENTÍMETROS Y MILÍMETROS.
1010 LET MM = 1000*M + 10*CM + MM
1020 REM AHORA TODO ESTÁ EN MM
1030 LET E = SGN MM
1040 LET MM = ABS MM
1050 REM TRABAJAMOS CON MM POSITIVOS MANTENIENDO SU SIGNO EN E
1060 LET CM = INT (MM/10)
1070 LET MM = (MM-10*CM)*E
1080 LET M = INT (CM/100)*E
1090 LET CM = CM*E-100*M
1100 RETURN
```

Por sí sólo esto no es de mucha utilidad, porque no hay un programa para hacer con ellos alguna cosa posteriormente.

Digite el programa principal y también otra sub-rutina, para imprimir M, CM, y MM.

```
10 INPUT M
20 INPUT CM
30 INPUT MM
40 GOSUB 2000
50 REM IMPRIME VALORES
55 PRINT TAB 8: "=";
60 GOSUB 1000
65 REM LA CONVERSION
70 GOSUB 2000
75 REM IMPRIME LOS VALORES
80 PRINT
90 GOTO 10
2000 REM SUB-RUTINA PARA IMPRIMIR M, CM y MM
2010 PIRNT " "; M; "M.."; CM; "CM.."; MM; "MM.."
2020 RETURN
```

Evidentemente, nosotros preservamos los valores usando la rutina de impresión en la línea 2000, pero la sub-rutina de conversión, hace el programa más largo — a través de un GOSUB y un RETURN — por lo que la longitual del programa no es la única consideración. Bien usadas, las sub-rutinas pueden hacer los programas más fáciles de comprender.

El programa principal se hace más simple por el hecho de usar sentencias más poderosas: cada GOSUB representa algunas sentencias del BASIC complicadas. Pero Ud. puede olvidar eso; lo que importa, es sólo el resultado final. Gracias a eso, es mucho más fácil seguir la estructura principal del programa.

Por otro lado, las sub-rutinas son simplificadas por una razón muy diferente, específicamente porque son más cortas. Todavía ellas usan las mismas viejas y laboriosas sentencias LET y PRINT, pero tienen que hacer sólo parte del trabajo total, y así son más fáciles de escribir.

El secreto consiste en escoger el nivel — o niveles — en el cual escribir las sub-rutinas. Ellas deben ser lo suficientemente grandes, para tener un impacto significativo en el programa principal y aún lo bastante pequeñas para ser significativamente más fáciles de escribir que el programa completo, sin sub-rutinas. Estos ejemplos (*no* recomendados) ilustran bien. Primero:

```
10 GOSUB 1000
20 GOTO 10
1000 INPUT M
1010 INPUT CM
1020 INPUT MM
1030 PRINT " "; M; " "; CM; " "; MM; TAB 8; "=";
1040 LET MM = 1000*M + 10*CM
  .   .
  .   .
  .   .
  .   .
2000 RETURN
```

y segundo:

```
10 GOSUB 1010
20 GOSUB 1020
30 GOSUB 1030
40 GOSUB 1040
50 GOSUB 1050
  .   .
  .   .
  .   .
  .   .
300 GOTO 10
1010 INPUT M
1015 RETURN
1020 INPUT CM
1025 RETURN
1030 INPUT MM
1035 RETURN
  .   .
  .   .
  .   .
  .   .
```

El primero, con su única y poderosa sub-rutina y el segundo, con sus muchas sub-rutinas, demuestran extremos casi opuestos, pero, de igual inutilidad.

Una sub-rutina puede llamar otra, o incluso a sí misma (una sub-rutina que se llama asimisma es llamada de *recursiva*) por lo tanto, no tenga miedo de tener varios niveles.

RESUMEN.

Sentencias: GOSUB, RETURN.

EJERCICIOS.

1. El programa ejemplo es virtualmente una calculadora de distancia universal. ¿Cómo la utilizaría:

(I) Para convertir Yardas y Pulgadas en Yardas, Pies en Pulgadas?

(II) Para convertir metros em pulgadas y pies?

(III) Para encontrar fracciones de una Yarda? (o sea, un tercio de una Yarda o un pie)

Incluya una línea para redondear pulgadas para la pulgada más próxima.

2. Adicione dos sentencias al programa:

```
4 LET CONVERSION = 1000
7 LET MCMMMIMP = 2000
```

Y cambie

GOSUB 1000 para GOSUB CONVERSION

GOSUB 2000 para GOSUB MCMMMIMP

Esto funciona exactamente como Ud. esperaba. En realidad, el número de una línea en un GOSUB (o GOTO o RUN) puede ser cualquier expresión numérica.

Este tipo de cosa puede funcionar maravillosamente bien para hacer su programa más claro.

3. Reescriba el programa principal del ejemplo para hacer alguna otra cosa, pero, usando la misma sub-rutina.

4. GOSUB n
RETURN
en líneas consecutivas pueden ser cambiados por
GOTO n
¿Por qué?

5. Una sub-rutina puede tener varios puntos de entrada. Por ejemplo, debido a la forma como él las usa, con GOSUB 1000, seguido inmediatamente por GOSUB 2000; nosotros podemos reemplazar nuestras dos sub-rutinas por una grande, que ajusta M, CM y MM y los imprime. Ella tiene dos puntos de entrada: uno en el inicio, para toda la sub-rutina y uno más adelante, sólo para la parte de impresión.

Haga los arreglos necesarios.

6. Ejecute este programa:

```
10 GOSUB 20
20 GOSUB 10
```

Las direcciones de retorno son colocadas en el STACK de GOSUB, pero ellas nunca son sacadas; y eventualmente no hay más espacio para nada más en el computador. El programa se detiene, con indicación de error 4 (Vea apéndice B).

Ud. debe tener dificultades para borrarlas sin perder todo, pero una de estas soluciones funcionará:

(I) Deje sin efecto las dos sentencias GOSUB.

(II) Inserte dos nuevas líneas.

```
11 RETURN
21 RETURN
```

(III) Presione

```
RETURN
```

Las direcciones de retorno serán listados hasta que Ud. obtenga error 7.

(IV) Altere su programa para que no vuelva a ocurrir lo mismo. ¿Cómo funciona?

capítulo

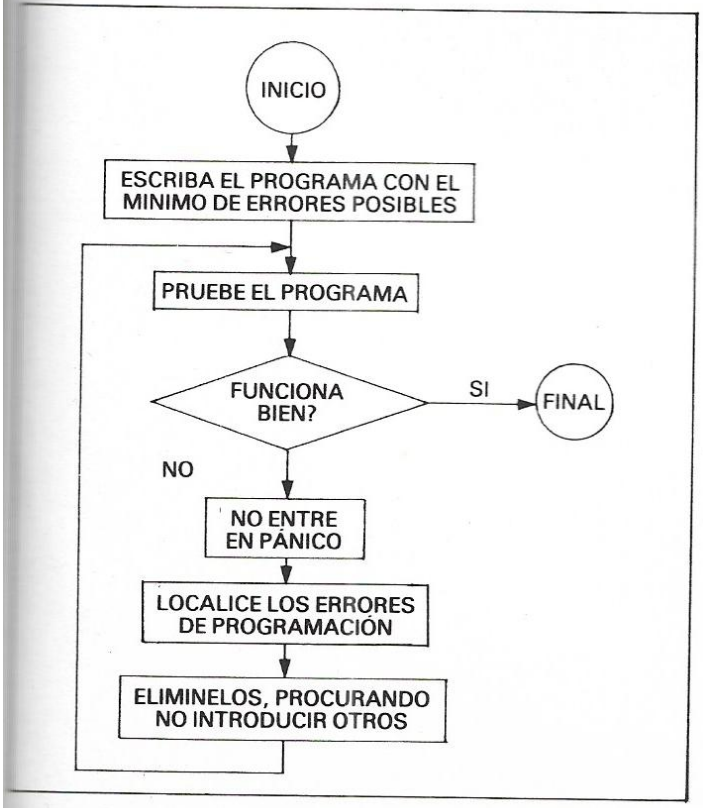
15

Operando los programas

En el arte de programar computadores, hay algo mas que conecer que hacen las sentencias. Ud. ya notó, probablemente, que la mayoría de sus programas presentan lo que técnicamente es conocido como *bug* (error de programación, en inglés), siempre que Ud. vaya a rodarlos por primera vez: la mayoría de las veces son sólo errores de digitación, otras, son errores de sus ideas, con respecto a lo que el computador debería hacer. Eso acontece normalmente debido a la falta de experiencia en el asunto.

Queda entonces explicado ese punto: en un comienzo es muy frecuente que los programas presenten *bugs* o error de programación.

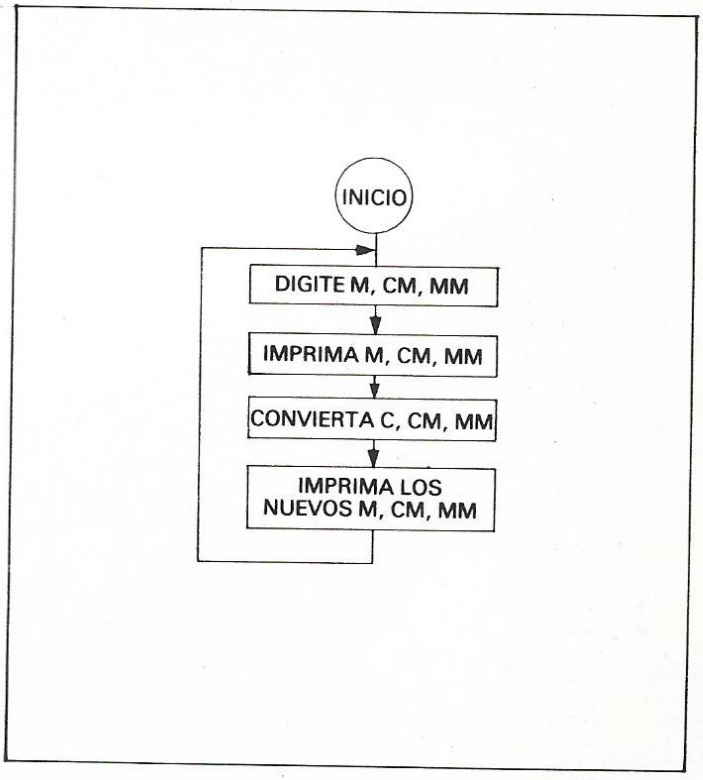
El plan general de un programa puede ser fácilmente ilustrado a través de un *organigrama:*



La idea, es seguir los recuadros de arriba para abajo, de acuerdo con las flechas, ejecutando la sentencia contenida en cada una de ellas. Acostúmbrase a utilizar, como padrón, diferentes formatos de los recuadros para las diversas sentencias existentes. Así:
Un recuadro redondeado ◯ indica inicio o fín. Un recuadro rectangular ▭ indica una sentencia normal cualquiera. Un rombo ◇ pide que se tome una decisión, antes de proseguir.

Estos formatos son ampliamente utilizados, pero no som obligatorios.

Los organigramas, naturalmente sirven para describir la estructura general del programa, con una sub-rutina prácticamente en todos los bloques. Así, el organigrama para nuestro ejemplo de distancias, del capítulo anterior, puede ser el siguiente:



Cualquier cosa — como organigramas, sub-rutinas y también sentencias REM — que hace el programa más claro le proporciona una mejor comprensión del mismo; y de esta forma Ud. puede certificarse de cometer un menor número de errores de programación. Las sub-rutinas también ayudan a encontrar *bugs* ya cometidos, dándole al programa más facilidad para probar cada sub-rutina individualmente y de certificarse de que ellas se ajustan perfectamente en el todo, evitándose así, tener que probar el programa entero no estructurado.

Por lo tanto, las sub-rutinas auxilian por medio del recuadro "localice los errores de programación", donde Ud. podrá encontrar toda la ayuda que necesite.

Otras sugerencias para encontrar los famosos bugs son las siguientes:

1. Certifíquese siempre de que no haya errores de digitación.
2. Trate de determinar todas las variables que debería haber en cada etapa — y si es posible, explíquelas por medio de la sentencia REM. Ud. puede chequear una variable en un determinado punto del programa, insertando una sentencia PRINT en ese punto;
3. Si uno de los objetivos del programa es que se detenga en cuanto aparezca un error, utilice la información al máximo. Observe el código de error, y vea porqué el programa paró en una determinada línea, si es necesario calcule los valores de las variables.
4. Ud. debe estar apto para ejecutar línea por línea del programa, digitando cada una de ellas como comandos.
5. Haga de cuenta que Ud. es el propio computador y ejecute el programa, usando papel y lápiz para anotar los valores de las variables. Una vez localizados los errores, corregirlos, es como escribir el programa original, pero es conveniente probar nuevamente el programa. Es sorprendentemente fácil eliminar un *bug* e introducir otro en su lugar.

EJERCICIOS.

1. Digite un largo programa; después, desconecte el plug del toma-corriente. Esto es el tipo de cosas que acontece espontáneamente, algunas veces; no es un error de programación, si no un problema eléctrico y no hay nada que Ud. pueda hacer al respecto. En caso que esto ocurra con mucha frecuencia, tal vez haya algo errado; de cualquier forma, sería bueno preservar los programas, inclusive incompletos en cintas magnéticas.
2. El organigrama para cálculo de distancia no posee el bloque de "fín"; ¿ eso importa? ¿ Dónde Ud. lo colocaría en el caso que lo deseara?

capítulo 16

Almacenamiento en cinta

Como ya fué mencionado en el capítulo 1 — y Ud. ya no tiene dudas gracias a las experiencias anteriores — cuando el TK 85 es desconectado, se pierde todo el programa y las variables que se encontraban almacenadas en la memoria. La única manera de preservarlos, es instruir al computador para grabarlos en una cinta cassette; de ese modo, Ud. podrá volverlos a cargar en la memoria posteriormente y el computador estará prácticamente en el mismo estado en que estaba, cuando fué efectuada la grabación.

Junto con el computador Ud. debe haber recibido un cable, con el cual se puede conectar el TK 85 al grabador. Conviene que Ud. pruebe su propio grabador, pués algunos trabajan mejor que otros en esa función.

En primer lugar, en lo que se refiere al computador, un grabador simple y barato suele ser tan bueno como el más sofisticado, además de dar menos problemas.

En segundo lugar, el grabador debe poseer un enchufe para micrófono y otro para el auricular. Deben ser de preferencia, del tipo jack hembra, adecuados a los plugs y proporcionados conjuntamente con los cables.

Habiendo conseguido un grabador adecuado, conéctelo al TK 85: el cable debe interconectar la entrada del micrófono y el toma-corriente señalado por "MIC", en el computador. Certifíquese de haberlo conectado correctamente.

Digite entonces un programa en el computador — el programa, por ejemplo, del conjunto de caracteres del capítulo 11. Será necesario dar un nombre al programa, cuando vaya a almacenarlo y sería una buena idea, introducir ese nombre, de manera que él aparezca en la lista. La forma más fácil, es utilizar la sentencia REM.

Digite, entonces:

```
5 REM "CARACTERES"
```

Ahora, esto es sólo un ejemplo, para que Ud. pueda visualizar mejor lo que acontece — digite

```
SAVE "CARACTERES"
```

y observe el aparato de TV. Durante 5 segundos él permanecerá con la pantalla de color grisáceo, después, por cerca de 6 segundos, surgirá un dibujo, formado por listas blancas y negras y entonces, la pantalla quedará blanca, con la indicación 0/0. El computador estaba mandando una señal para el toma corriente "MIC", pero la misma señal estaba siendo enviada para el aparato de TV, produciendo la imagen que Ud. vió. El período grisáceo era apenas silencioso, mientras que las listas (líneas) eran el programa.

Pero lo que Ud. quiere hacer, obviamente, es captar la señal en la cinta; así que vamos a hacerlo bién esta vez.

GRABANDO UN PROGRAMA.

1. Posicione la cinta en un punto en que esté en blanco o pueda ser regrabada;
2. Usando un micrófono, grave su voz diciendo "caracteres". Esto no es escencial, pero hará más fácil la localización de los programas posteriormente. Conecte nuevamente el computador al grabador;
3. Digite entonces

```
SAVE "CARACTERES"          (sin NEWLINE)
```

4. Accione el grabador y dé inicio a la grabación;

5. Presione la tecla NEWLINE.
6. Observe el aparato de TV, como anteriormente; cuando la grabación termine (con la indicación Ø/Ø), pare el grabador.

Para certificarse que todo se realizó bien, Ud. ahora debe escuchar la cinta grabada, a través del alto-parlante del propio grabador probablemente Ud. tendrá que desconectar el cable del computador, que está conectado a la salida del auricular). Rebobine la cinta hasta el inicio del programa y reprodúzcala.

Primero, Ud. escuchará su propia voz diciendo "caracteres". Después vendrá un zumbido suave, en realidad esto no es parte de la grabación, sino el fín de la señal para el televisor (antes que NEWLINE sea presionado), que también llegó al grabador.

Enseguida, vendrán 5 segundos de silencio, el inicio de la señal propiamente dicha; corresponde al período en que la pantalla se puso grisácea. Después, vienen los 6 segundos de un zumbido estridente y alto, que a todo volumen sería hasta desagradable; y la grabación del programa, correspondiendo al dibujo en blanco y negro visto en la pantalla.

Por fín, volverá el zumbido suave una vez más.

En caso que Ud. no escuche nada de esto, compruebe que el computador y el grabador estén conectados correctamente. En algunos modelos de grabador, el toma-corriente no hace contacto si el plug fuera totalmente introducido, trate de introducirla más o menos 2 mm para afuera y Ud. percibirá el encaje en una posición más natural.

Supongamos ahora que la grabación parece estar correcta al oído y que Ud. desea cargarla en el computador.

CARGANDO UN PROGRAMA CON NOMBRE.

1. Rebobine la cinta para el inicio del programa;
2. Compruebe que el toma-corriente "EAR" del TK 85 esté perfectamente conectado al auricular del grabador;
3. Gire el control de volumen del grabador hasta cerca de 3/4 del máximo; en el caso que haya un control de tonalidad, ajústelo al modo más agudo posible.
4. Digite

LOAD "CARACTERES"
(nuevamente sin el NEWLINE)

5. Coloque el grabador en funcionamiento;
6. Presione el NEWLINE;

Otra vez Ud. verá el dibujo de la grabación en la pantalla pero esta vez será un poco diferente, en otra combinación de blanco y negro. Será más difícil distinguir entre el silencio y el programa, pero Ud. percibirá que la parte de la programación, presenta líneas más amplias y definidas (pruebe si quiere el ejercicio 1).

Aproximadamente después de 15 segundos, debe estar cargado el programa y cerrado con la indicación Ø/Ø. En caso contrario, use la tecla BREAK.

Es muy probable que algo errado haya ocurrido con el control de volumen. El debería estar:

1. Lo bastante alto para la parte del programa a ser captado por el computador;
2. No tan alto que lleve a distorcionar el programa (lo que es muy raro);
3. Suficientemente bajo, para que la parte silenciosa del computador sea reconocida por éste.

El mejor ajuste es girar el control de volumen, sin que la parte silenciosa se haga ruidosa, esto puede ser recho al escucharse la grabación por el alto-parlante. En caso que el silencio esté realmente ruidoso, surgirán algunos problemas:

— Algunos grabadores podrán formar un lazo de realimentación con el TK 85. Esto puede ser evitado, desconectando el cable "EAR", durante la grabación;
— Algunos grabadores — principalmente los más antiguos y usados — son intrínsicamente ruidosos. Una cinta de mejor calidad puede ayudar en esta parte, a pesar de no ser necesaria normalmente;
— Ciertos grabadores podrán captar el ruido de 60 Hz de la red. Resuelva el problema alimentándolos con baterias o pilas;
— Por último, el problema puede estar en el plug del auricular, que Ud. empujó completamente hasta adentro.

Estas reglas deben englobar los problemas más comunes; en caso que el defecto persista, desista e intente nuevamente en la mañana siguiente.

En caso que Ud. tenga un programa en cinta y no consiga acordárse del título, todavía es posible cargarlo (pruebe hacerlo con el programa "CARACTERES" que usó antes).

CARGANDO UN PROGRAMA SIN NOMBRE.

1. Posicione la cinta en la parte silenciosa;
2. Verifique todo y ajuste los controles como antes Ud. percibirá que el control del volumen deberá recibir mayores cuidados que en el caso anterior;

3. Digite

LOAD " " (sin el NEWLINE)

4. Ponga el grabador a funcionar;

5. Presione NEWLINE;

6. Para el resto, proceda como ya fué explicado.

La idea aquí, es que el nombre del programa que Ud. pide para cargar es una *string vacía,* el computador carga el primer programa que encuentra. Note que cuando Ud. preserva un programa, no puede hacer una string vacía de su nombre. En caso que Ud. lo intente, de todas maneras obtendrá la señal de error F.

LOAD y SAVE también pueden ser usadas en programas.

Con SAVE, el programa se auto-preservará de tal forma que, cuando fuere cargado, se pondrá inmediatamente en ejecución a partir de la línea siguiente de la sentencia SAVE. Digite a título de ejemplo:

```
 5 REM "INÚTIL"
 10 PRINT "ESTO ES TODO LO QUE ÉL HACE"
 20 STOP
100 SAVE "INÚTIL"
110 GOTO 10
```

Conecte el grabador y digite

RUN 100 (sin el NEWLINE)

ponga el grabador a funcionar y presione NEWLINE. Cuando el programa estuviera grabado, continuará rodando normalmente.

Ahora, para cargar ese programa, rebobine la cinta hasta su inicio y digite

LOAD "INÚTIL (sin el NEWLINE)

ponga el grabador a funcionar y presione NEWLINE. En cuanto el programa estuviera cargado, él irá para la línea 110 y proseguirá sin ningún esfuerzo de su parte.

Fíjese como poniendo la sentencia SAVE al final del programa significa que, para ejecutarlo sin el SAVE, Ud. sólo tiene que digitar RUN, pudiendo omitir el propio SAVE.

No utilice el SAVE en una rutina con GOSUB — él no funcionará.

No incluya caracteres de video invertido en el nombre de un programa; todo lo que haya después del caracter se pierde. El nombre no debe contener más de 127 caracteres.

El nombre en un LOAD o SAVE no necesita ser una constante *string;* puede ser cualquier expresión *string,* como A$ o CHR$ 100.

RESUMEN

— Grabando un programa en cinta.
— Cargando un programa con nombre.
— Cargando el primer programa disponible de una cinta.
— Grabando un programa que pueda ser cargado y continúe rodando.

Sentencias: SAVE, LOAD.

EJERCICIOS.

1. Prepare una cinta con varios programas pequeños, comience a cargarlos en el computador y digite

LOAD "XYZWN"

Ud. percibirá fácilmente la diferencia, en la pantalla entre los espacios vacíos de la cinta (con una imagen desestructurada) y los programas (con líneas más definidas). Ambas imágenes son diferentes de la que Ud. vé cuando preserva un programa. En caso que el volumen sea bajado durante el paso de un programa, Ud. podrá ver que la imagen cambia para la forma de espacio vacío, ya que la señal es demasiado baja para ser identificado como programa.

2. Prepare una cinta en la cual el primer programa, al ser cargado, imprima una lista de los demás programas de la cinta, pida que uno de ellos sea escogido, y lo carga.

3. Digite nuevamente el programa "CARACTERES" y luego digite

LET X = 7

de manera que — a pesar de que esta no aparece en el programa — el computador ahora tiene una variable X de valor 7. Grabe entonces el programa, desconecte el computador, conéctelo nuevamente y cargue el programa otra vez. Digite

PRINT X

y Ud. obtendrá la respuesta 7. La sentencia SAVE preservó no sólo el programa, sino también todas las variables, incluyendo X. En el caso que Ud. quiera mantener las variables cuando ejecuta el programa, debe acordarse de usar GOTO y no RUN (como fué mencionado en el capítulo 9). Para no tener que acordarse de eso cada vez que vaya a rodar un programa, haga que los programas se auto ejecuten (empleando SAVE como línea de programación).

4. Digite un largo programa y desconecte momentáneamente la alimentación del computador. Como ya dijimos, ese tipo de problema acostumbra acontecer espontáneamente en la red eléctrica; no es un error, pero sí un accidente. Una vez más repetimos: Si ocurre con mucha frecuencia, algo debe estar errado, por lo que será conveniente ir preservando programas largos, de a poco en cinta.

Funciones especiales de almacenamiento encontrará en el CAPÍTULO 29.

capítulo

17

Imprimiendo

Ud. debe recordar que una sentencia PRINT posee una lista de ítems, siendo cada una de ellas una expresión (o tal vez nada en absoluto), y que son separadas por coma o punto y coma. Existen dos ítem más PRINT, usados para decir al computador *donde* imprimir, pero no, *que* imprimir.

Así, por ejemplo, PRINT AT 11, 16; "*" imprime un asterisco en el centro de la pantalla.

De este modo, AT línea, columna, mueve la posición PRINT (o sea, el lugar donde el próximo ítem debe ser impreso) para la línea y columna especificadas. Las líneas son enumeradas de 0 (arriba) hasta 21, mientras que las columnas, son enumeradas de 0 (para la izquierda) hasta 31.

De la misma forma, TAB columna mueve la posición de PRINT, para la columna especificada. Sin embargo, permanece en la misma línea y en caso que incluya vuelta, se transfiere para la línea de abajo.

Observe que el computador reduce el número de la columna, módulo 32 (divide por 32 y toma el resto); así, TAB 33 significa lo mismo que TAB 1.

Por ejemplo, vamos a imprimir el encabezamiento de una página llamado "Contenido" siendo 1 el número de esa página:

```
PRINT TAB 30; 1; TAB 12; "CONTENIDO"; TAB
24; "PÁGINA"
```

Algunos puntos a considerar:

1. Esos dos nuevos items tienen, cómo terminación mejor, el punto y coma, como lo hicimos arriba. Pero Ud. puede usar coma (o nada) al final de la sentencia; sin embargo, eso va a significar que, después de haber Ud. definido cuidadosamente la posición del PRINT, él se irá a mover nuevamente;
2. A pesar de que AT y TAB no son funciones, es necesario presionar la tecla FUNCTION (SHIFT NEWLINE) para obtenerlas;
3. No es posible imprimir en las dos líneas inferiores (22 y 23) de la pantalla. Cuando nos referimos a la línea más baja, estamos hablando de 1 a 21;
4. Ud. puede usar AT para posicionar el PRINT hasta donde ya existe algo impreso; la inscripción existente será borrada.

Hay dos sentencias más ligadas al PRINT, denominadas CLS Y SCROLL. La primera borra la pantalla, mientras que la segunda mueve toda la imagen una línea para arriba (perdiéndose así la línea de arriba) y mueve la posición PRINT, para el inicio de la línea inferior (línea 21).

Para darnos cuenta como ella funciona, veamos este programa:

```
10 SCROLL
20 INPUT A$
30 PRINT A$
40 GOTO 10
```

RESUMEN.

Ítems tipo PRINT: AT, TAB
Sentencias: CLS, SCROLL

EJERCICIO.

1. Pruebe rodar el siguiente programa:

```
10 FOR I = 0 TO 20
20 PRINT TAB 8*I;I;
30 NEXT I
```

Esto muestra lo que significa, cuando el número TAB es reducido para el módulo 32. Para obtener un ejemplo más interesante, cambie el número 8, en la línea 20, por un 6.

capítulo

18

Gráficos

Aquí está una de las más atrayentes características del TK 85, utilizando los elementos de imagen. La pantalla usada como *display* para el computador, cuenta con 22 líneas y 32 columnas, formando 22*32 = 704 posiciones de caracteres, conteniendo cada una, 4 elementos de imagen.

El elemento de imagen es especificado por dos números, que son sus coordenadas. El primero, su coordenada X, determina a que distancia se encuentra de la columna más a la izquierda; y el segundo, la coordenada Y, dice lo elevado que él está, en relación a la línea más baja. Tales coordenadas, son normalmente escritas, como un par de números entre paréntesis, de manera que (0,0), (0,43) y (63,43) corresponden respectivamente a las esquinas: inferior izquierda, inferior derecha, superior izquierda y superior derecha de la pantalla.

La sentencia

PLOT coordenada X, coordenada Y

hace que el elemento de imagen en negro, coincida con esas coordenadas, mientras

UNPLOT coordenada X, coordenada Y

lo borra.

Pruebe este programa simple:

```
10 PLOT INT (RND*64), INT (RND*44)
20 INPUT A$
30 GOTO 10
```

Este programa traza un punto aleatório cada vez que el NEWLINE es presionado. Hay un programa mucho más útil; él traza el gráfico de la función de seno (o sea, un senoide) para valores entre 0 y 2 π.

```
10 FOR N = 0 TO 63
20 PLOT N,22 + 20*SIN (N/32*PI)
30 NEXT N
```

Este otro, arma el gráfico de la función SQR (parte de una parábola) entre 0 y 4:

```
10 FOR N = 0 TO 63
20 PLOT N,20*SQR (N/16)
30 NEXT N
```

Note que las coordenadas del elemento de imagen, son bastante diferentes de las líneas y columnas en un ítem AT. Ud. va a darse cuenta como es de útil el diagrama final de este capítulo, para trabajar con coordenadas de los elementos de imagen y con los números de columnas y líneas.

EJERCICIOS.

1. Hay tres diferencias entre los números de un item AT y las coordenadas del elemento de imagen. ¿ Cuáles son ellas?

Suponga que la posición del PRINT corresponda a AT L, C (línea y columna). Pruebe para sí, que los 4 elementos de imagen de aquella posición, poseen coordenada X 2*C ó 2*C + 1 y coordenada Y 2*(21 − L) ó 2*(21 − L) + 1 (vea el diagrama).

2. Procure alterar aquel simple programa, de modo que él primero complete la pantalla en negro (un cuadro negro es un espacio de video invertido), y luego utilice la sentencia UNPLOT sobre algunos puntos aleatórios.

3. Modifique el programa del gráfico senoidal, a fín de que él imprima, antes de trazar el gráfico, una línea horizontal de varios " " como eje X y varios "/" para el eje Y.

4. Escriba programas para armar gráficos de otras funciones, tales como: COS, EXP, LN, ATN, INT y así en adelante. En cada gráfico, Ud. debe comprobar que él cabe en la pantalla, considerando:
a. en que límite de valores Ud. tomará la función (correspondiéndole a seno la zona entre 0 y 2 π);
b. Donde colocar el eje X en la pantalla (correspondiéndo a 22 en la línea 20 del programa del seno);
c. Como determinar la escala del eje y del gráfico (correspondiendo a 20 en la línea 20 del programa del seno).

5. Ejecute este programa:

```
10 PLOT 21, 21
20 PRINT "COMILLAS"
30 PLOT 46, 21
```

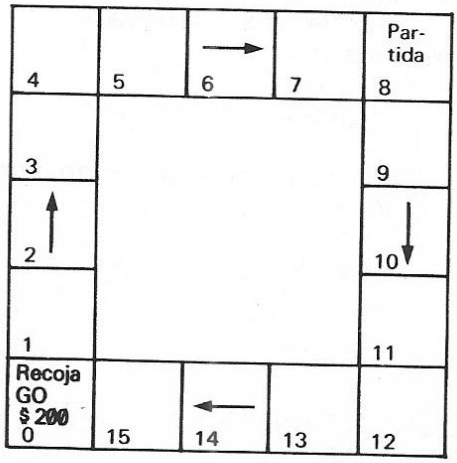
PLOT se mueve sobre la posición PRINT y UNPLOT también.

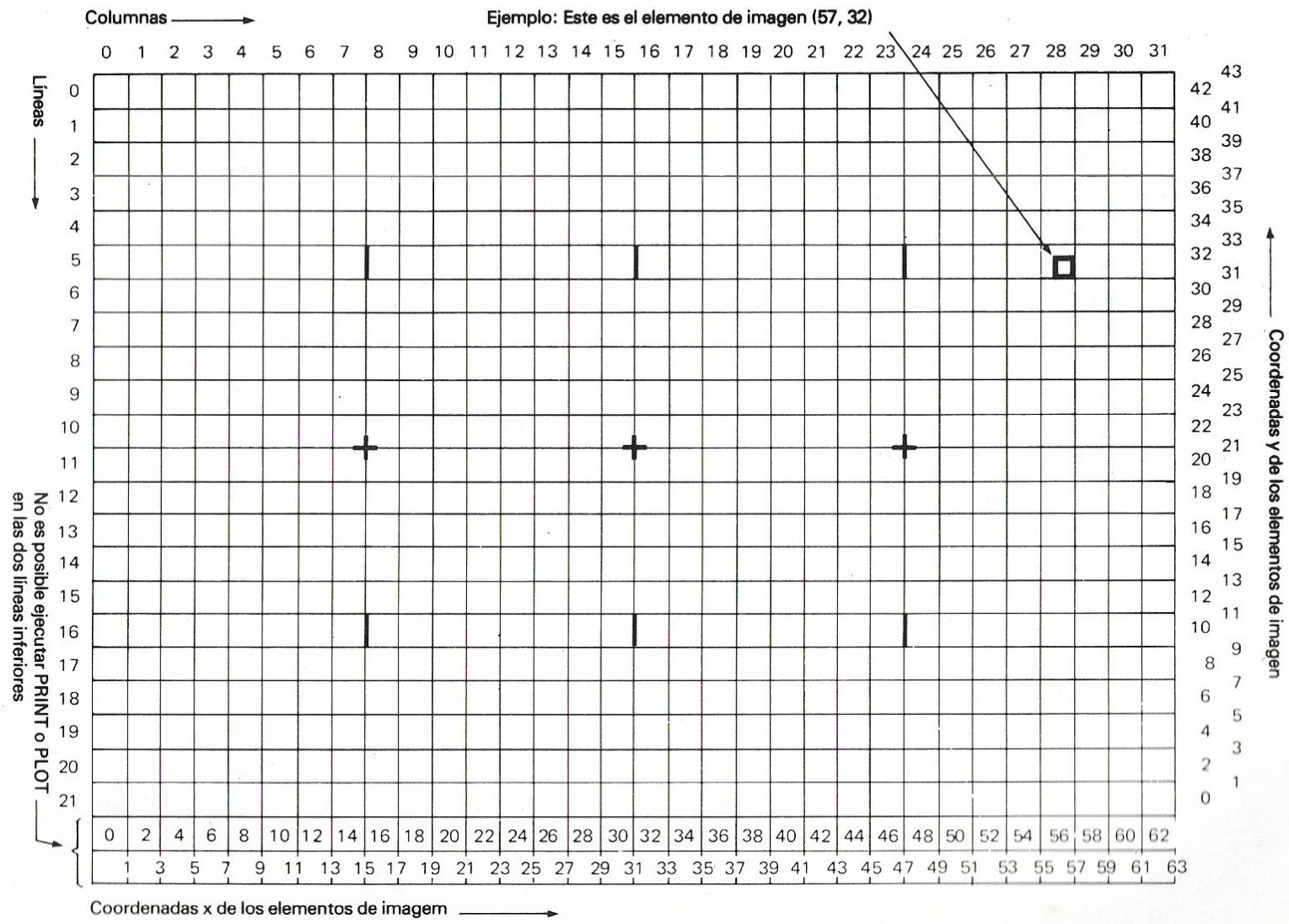
6. Esta sub-rutina traza una línea casi recta del elemento de imagen (A, B) hasta (C, D). Úsela como parte de un programa principal que proporcione los valores A, B, C, y D (en el caso que Ud. no posea la expansión de memoria, tal vez tenga que omitir las sentencias REM).

```
1000 LET U = C-A
1005 REM U MUESTRA CUANTOS PASOS TENEMOS QUE DAR
1010 LET V = D-B
1015 REM V MUESTRA CUANTOS PASOS PARA ARRIBA
1020 LET D1X = SGN U
1030 LET D1Y = SGN V
1035 REM (D1X, D1Y) ES UNO SOLO PASO EN DIAGONAL
1040 LET D2X = SGN U
1050 LET D2Y = 0
1055 REM (D2X, D2Y) ES UN SOLO PASO PARA LA DERECHA O PARA LA IZQUIERDA
1060 LET M = ABS U
1070 LET N = ABS V
1080 IF M > N THEN GOTO 1130
1090 LET D2X = 0
1100 LET D2Y = SGN V
1105 REM AHORA (D2X, D2Y) ES UN SOLO PASO PARA ARRIBA O PARA ABAJO
1110 RET M = ABS V
1120 RET N = ABS U
1130 REM M ES EL MAYOR DE ABS U Y ABS V, N ES EL MENOR
1140 LET S = INT (M/2)
1145 REM NOS QUEREMOS IR DE (A, B) A (C, D) EN M PASOS USANDO N PASOS PARA ARRIBA Ó PARA ABAJO Ó D2 PASOS PARA LA DERECHA Ó IZQUIERDA Y D1 PASOS DIAGONALES M-N, DISTRIBUIDOS LO MÁS UNIFORMEMENTE POSIBLE
1150 FOR I = 0 TO M
1160 PLOT A, B
1170 LET S = S + N
1180 IF S < M THEN GOTO 1230
1190 LET S = S-M
1200 LET A = A + D1X
1210 LET B = B + D1Y
1215 REM UN PASO EN DIAGONAL
1220 GOTO 1250
1230 RET A = A + D2X
1240 LET B = B + D2Y
1245 REM UN PASO PARA ARRIBA/ABAJO Ó PARA DERECHA/IZQUIERDA
1250 NEXT I
1260 RETURN
```

La última parte (de la línea 1150 en adelante) mezcla uniformemente los pasos D1 M-N con los N pasos D2. Imagine un tablero de juego Monopólio con M cuadrados alrededor, enumerados de 0 hasta M-1; el cuadrado en que Ud. está en cualquier momento es el del número S, comenzando en la esquina opuesta a la del GO. Cada movimiento lo lleva N cuadrados a lo largo del tablero y en línea recta; en la pantalla, Ud. hace un movimiento vertical u horizontal (en el caso que Ud. pase por el GO en el tablero), o bien un paso en diagonal. Como su recorrido total en el tablero es de M*N pasos, o dar la vuelta N veces, Ud. pasa por GO N veces también, y uniformemente espaciados en sus M pasos, hay N pasos izquierda/derecha o arriba/abajo.

Ajuste el programa de manera que si otro parámetro — E sea 1, la línea sea trazada en negro (como aquí) y en el caso que sea 0, la línea es trazada en blanco (usándo UNPLOT). Ud. puede entonces, dejar sin efecto una línea que ya había trazado.




Columnas
Ejemplo: Este es el elemento de imagen (57, 32)
Líneas
No es posible ejecutar PRINT o PLOT en las dos líneas inferiores
Coordenadas y de los elementos de imagen
Coordenadas x de los elementos de imagem

capítulo

19

Tiempo y movimiento

Frecuentemente Ud. deseará hacer un programa que dure un tiempo específico en la pantalla, y para ese propósito Ud. hallará la sentencia PAUSE útil.

PAUSE n

para durante n cuadros de la televisión (con 60 cuadros por segundo), n puede ser usado hasta 32767, lo que da cerca de 9 minutos; si n es má grande, significa que debe parar para siempre.

Al final de una pausa, la pantalla brillará más fuerte.

La pausa puede ser abreviada por el accionamiento de una tecla, después de iniciado el periodo de pausa. La sentencia PAUSE debe ser seguida por POKE 16437, 255.

Aparentemente, la PAUSE funciona sin eso, pero a veces podrá perder su programa. Este programa controla el segundero de un reloj:

```
 5 REM PRIMERO DIBUJAMOS EL RELOJ
10 FOR N = 1 TO 12
20 PRINT AT 10-10*COS (N/6*PI), 10 + 10*
SIN (N/6*PI); N
30 NEXT N
35 REM AHORA COMIENZA EL FUNCIONA-
MIENTO DEL RELOJ
40 FOR T = 0 TO 10000
45 REM T ES EL TIEMPO EN SEGUNDOS
50 LET A = T/30*PI
60 LET SX = 21 + 18*SIN A
70 LET SY = 22 + 18*COS A
200 PLOT SX, SY
300 PAUSE 50
305 POKE 16437,255
310 UNPLOT SX, SY
400 NEXT T
```

Note como el tiempo es controlado por la línea 300. Ud. esperaría PAUSE 60 para hacerlo cambiar una vez por segundo, mas el procesamiento también gasta tiempo y ese tiempo tiene que ser considerado. La mejor manera de hacerlo es por tanteo y error, comparando el reloj del computador con uno real y ajustando la línea 300. (Ud. no puede hacer eso con precisión, un ajuste de un cuadro por segundo es 2% o media hora en un día).

La función INKEY$ (la cual no tiene argumento) lee el teclado como si Ud. estuviera presionando una tecla; el resultado es el caracter que aquella tecla de en modo L ; en caso contrario, el resultado es una *string* vacía. Los caracteres de control no tienen el efecto común, pero dan resultado como CHR$ 118 para NEWLINE — ellos son impresos como "?".

Pruebe éste programa, que funciona como una máquina de escribir.

```
10 IF INKEY$ <> "" THEN GOTO 10
20 IF INKEY$ = "" THEN GOTO 20
30 PRINT INKEY$;
35 PAUSE 20
40 GOTO 10
```

La línea 10 espera que Ud. retire su dedo del teclado. La línea 20 espera que Ud. presione una tecla.

Acuérdese que al contrario de INPUT, INKEY$ no solo espera. Así, Ud. no tiene que digitar NEWLINE, pero por otro lado, si Ud. no digita nada, entonces, habrá perdido su oportunidad.

EJERCICIOS

1. ¿ Qué acontece si Ud. saca la línea 10 del programa de la máquina de escribir?
2. ¿ Por qué Ud. no puede digitar espacio o £ en el programa de la máquina de escribir?

Aquí hay una modificación que proporciona un espacio si Ud. digita "CURSOR PARA LA DERECHA" (SHIFT 8).

```
10 IF INKEY$ <|> " " THEN GOTO 10
20 IF INKEY$ = " " THEN GOTO 20
30 LET A$ = INKEY$
40 IF A$ = CRH$ 115 THEN GOTO 110
90 PRINT A$
100 GOTO 10
110 PRINT " ";
120 GOTO 10
```

Note que transportamos INKEY$ para A$ en la línea 30. Sería posible omitir eso y reponer A$ por INKEY$ en las líneas 40 y 90, pero siempre habría una posibilidad que INKEY$ cambiará entre líneas.

Añada alguna cosa más en el programa, de tal manera que, si Ud. digita NEWLINE (CHR$ 118), obtenga una nueva línea.

3. Ud. todavía puede usar INKEY$ en conjunto con PAUSE como en este programa alternativo de la máquina de escribir.

```
10 PAUSE 40000
20 POKE 16437,255
30 PRINT INKEY$;
40 GOTO 10
```

Para hacer funcionar ese programa, ¿ por qué es esencial que una pausa no sea interrumpida cuando va a iniciarse, Ud. ya ha pulsado una tecla?

Este método tiene la desventaja, de hacer parpadear la pantalla. Note que el computador aprovecha la oportunidad de una pausa para imprimir la imagen en el televisor.

4. El programa siguiente hace que el computador imprima un número, el cual Ud. (o una víctima inocente) debería digitar. Para comenzar, Ud. tiene un segundo para hacerlo, pero si lo hiciera errado, tiene más tiempo para la próxima vez El ideal es hacerlo lo más rápido posible. Luego presionar Q para ver su marcación de puntos — cuanto mayor es mejor.

```
10 LET T = 50
15 REM T = NÚMERO DE CUADROS POR JUGADA — INICIALMENTE 50 para 1 segundo
20 SCROLL
30 LET A$ = CHR$ INT (RND*10 + CIDE "0")
35 REM A$ ES UN DÍGITO RANDÓMICO (AL AZAR)
45 PAUSE T
50 POKE 16437,255
60 LET B$ = INKEY$
70 IF B$ = "Q" THEN GOTO 200
80 IF A$ = B$ THEN GOTO 150
85 SCROLL
90 PRINT "NADA BUENO"
100 LET T = T*1.1
110 GOTO 20
115 SCROLL
150 PRINT "OK"
160 LET T = T*0.9
170 GOTO 20
200 SCROLL
210 PRINT "Ud. HIZO"; INT (500 / T); PUNTOS
```

5. (SÓLO PARA AQUELLOS CON EXPANSIÓN DE MEMORIA)

Usando la rutina de la línea recta del capítulo 18, cambie el programa del reloj, para que él también muestre punteros de minutos y horas (haga el puntero de horas más pequeño), arréglelo para que haga algún tipo de señal cada un cuarto de hora.

capítulo

20

La impresora

Este capítulo describe las sentencias BASIC para el uso de la impresora.

Las dos primeras, LPRINT y LLIST, son como PRINT Y LIST, excepto por usar la impresora en lugar de la TV. (la L es un accidente histórico. Cuando fué inventado el BASIC, generalmente usaba una máquina de escribir eléctrica en lugar de un aparato de TV., así PRINT significaba realmente imprimir. Si necesitaba de muchas salidas de datos, debía usar una impresora de línea muy rápida conectada al computador y PRINT significaba "imprimir en la impresora de línea").

Pruebe este programa, como ejemplo:

```
10 LPRINT "ESTE PROGRAMA" . . .
20 LLIST
30 LPRINT "IMPRESIÓN DEL CONJUNTO DE CARACTERES" . . .
40 FOR N = 0 TO 255
50 LPRINT CHR$ N;
60 NEXT N
```

La tercera sentencia — COPY, imprime una copia de la pantalla de la televisión. Por ejemplo, obtenga una lista del programa de arriba en la pantalla y digite:

```
COPY
```

Ud. siempre puede parar la impresora cuando está en funcionamiento, presionando la tecla BREAK (espacio).

RESUMEN.

Sentencias: LPRINT, LLIST, COPY
Nota: Ninguna de estas sentencias es normal en BASIC, a pesar que LPRINT es usada por otros computadores.

EJERCICIOS

1. Pruebe éste:

```
10 FOR N = 31 TO 0 STEP – 1
20 PRINT AT 31 – N, N; CHR$ (CODE "0" + N);
30 NEXT N
```

Ud. verá un dibujo de letras en diagonal, desde la esquina superior derecha, hasta la esquina inferior izquierda de la pantalla y el programa se detiene con indicación de error (5/20).

Ahora, cambie "AT 31 – N,N" en la línea 20 por "TAB N". El programa hará lo mismo que antes. Ahora cambie PRINT en la línea 20 por LPRINT. Esta vez no habrá error 5, lo cual no debería ocurrir con la impresora, y el dibujo continuará 10 líneas más con los dígitos.

Ahora cambie "TAB N" por "AT 21 – N,N" aún usando símbolos. La razón de la diferencia es que la salida para L PRINT no se realiza directamente, sino colocada en un buffer de almacenamiento (memoria intermedia), la imagen de una línea de impresión que el computador imprimirá de una sola vez. La impresión se producirá:

(I) Cuando el buffer estuviera completo,

(II) después de una sentencia que no termine en coma o punto y coma,

(III) cuando una coma o un Ítem TAB requiera una nueva línea, o

(IV) en el fin de un programa, si hubiera algo sin imprimir.

El número (III) explica porque nuestro programa con TAB funciona de esa manera. Como en el caso AT, el número de la línea y la posición del LPRINT son ignorados (como la posición PRINT, pero para la impresora, en lugar de la televisión) se cambia para el número de la columna un Ítem, AT nunca manda una línea para la impresora (en realidad el número de la línea después de AT no es completamente ignorado; tiene que estar entre – 21 y + 21 o dará lugar a un error. Por esa razón es más seguro especificar línea 0.

El Ítem "AT 21 – N,N" en la versión anterior de nuestro programa, sería mucho mejor, a pesar de menos ilustrativa si se substituye por "AT O, N").

2. Haga una impresión del gráfico del seno ejecutando el programa del capítulo 18 y usando COPY.

capítulo

21

Sub-strings

Dada una *string,* una *sub-string* de ella consiste en un número de caracteres consecutivos de la string, recogidos en secuencia.

Hay una escritura llamada SLICING, para describir *sub-strings,* la cual puede ser aplicada para cualquier expresión strings arbitraria. La forma general es:

EXPRESIONES STRINGS (INICIO TO FÍN)

de tal forma que, por ejemplo,

"ABCDEF" (2 TO 5) = "BCDE"

si Ud. omite el inicio, entonces asume 1; si omite el fín, entonces el largo de la string es asumido. Así

"ABCDEF" (TO 5) = "ABCDEF" (1 TO 5) = "ABCDE"
"ABCDEF" (2 TO) = "ABCDEF" (2 TO 6) = "BCDEF"
"ABCDEF" (TO) = "ABCDEF" (1 TO 6) = "ABCDEF"

(Ud. puede escribir esta última como "ABCDEF" ( ).

Una pequeña diferencia toma la omisión TO, y tiene sólo un número.

"ABCDEF" (3) = "ABCDEF" (3 TO 3) = "C"

A pesar que normalmente ambos, el inicio y el fín, se deben referir a partes existentes de la string, esa regla es superada por otra. Si el inicio es mayor que el fín, entonces el resultado es una string vacía. Así

"ABCDEF" (5 TO 7)

Da error 3 (error de suscripto) porque, desde el momento que la string tiene sólo 6 caracteres, 7 es mayor; pero.

"ABCDEF" (8 TO 7) = ""

y

"ABCDE" (1 TO 0) = ""

El inicio y el fín no pueden ser negativos, o Ud. obtiene error 3.

El próximo programa hace B$ igual a A$, pero omite todo espacio en blanco.

```
10 INPUT A$
20 FOR N = LEN A$ TO 1 STEP – 1
30 IF A$ (N) < > " " THEN GOTO 50
40 NEXT N
50 LET B$ = A$ (TO N)
60 PRINT "" "";  A$ "" "";  "" "";  B$;  "" ""
70 GOTO 10
```

Note que si A$ es enteramente espacios, entonces en la línea 50 tenemos N = 0E A$ (TO N) = A$ (1 TO 0) = "".

Para variables *strings,* no sólo podemos extraer sub-strings, sino también atribuir *sub-strings.* Por ejemplo, digite

LET A$ = "ESTÁ MUY AFUERA"

y entonces LET A$ (5 TO 8) = "******"
y PRINT A$

Fíjese que la sub-string A$ (5 TO 8) tiene sólo 4 caracteres, los 4 primeros asteriscos fueron usados. Esa es una característica de atribución para sub-strings: la

sub-string, después de la operación, tiene que tener exactamente el mismo tamaño que anteriormente. Para asegúrarse que esto aconteció, la string que está siendo atribuida, es cortada a la derecha si fuera muy larga llenada con espacios, si fuera muy pequeña. Esto es llamado de atribución PROCRUSTEANA.

Si ahora Ud. intenta:

```
LET A$ (   ) = "ESPLÉNDIDO"
```

y PRINT A$; "."

verá que nuevamente aconteció lo mismo (esta vez con espacio incluido) porque A$ ( ) cuenta con una *sub-string.*

```
LET A$ = "ESPLÉNDIDO"
```

funcionará perfectamente. EL SLICING puede ser considerado como teniendo prioridad 12; así por ejemplo,

LEN "ABCDEF" (2 TO 5) es equivalente a LEN "ABCDEF" (2 TO 5) lo que dará como resultado el valor 4.

Expresiones *string* complicadas necesitan de paréntesis, antes que una operación SLICING sea ejecutada. Por ejemplo:

```
"ABC" + "DEF" (1 TO 2) = "ABCDE"
("ABC" + "DEF") (1 TO 2) = "AB"
```

RESUMEN

Slicing, usando TO; Note que esta notación es exclusiva del TK 85.

EJERCICIOS

1. Algunos tipos de BASIC (no el del TK 85) tienen 3 funciones, llamadas LEFT$, RIGHT$ y MID$.

LEFT$ (A$, N) proporciona la sub-*string* de A$ formada por los primeros N caracteres.

RIGHT$ (A$, N) proporciona la sub-*string* de A$ formada por los caracteres desde n en adelante.

MID$ (A$, N1, N2) proporciona la sub-*string* A$ formada por N2 caracteres iniciando en el caracter N1.

¿ Como escribiría Ud. esto en BASIC de 10K del TK 85?

2. Pruebe ésta secuencia de comandos:

```
LET A$ = "X* — *Y"
LET A$ (2) = CHR$ 11 (el caracter de comillas)
LET A$ (4) = CHR$ 11
PRINT A$
```

A$ es ahora una *string* con las comillas dentro de ella.

Todo esto acontece como si Ud. hubiese digitado:

```
LET A$ = "X" + "Y"
```

La parte de la derecha del signo de igual habría sido tratada como una expresión, dando a A$ el valor "XY".

Ahora digite

```
LET B$ = "X" " +" "Y"
```

Ud. verá a pesar que A$ y B$ parecen ser iguales, cuando se imprimen, ellos no lo son — Pruebe:

```
PRINT A$ = B$
```

Mientras que B$ contiene sólo caracteres de imágen (con código 192), A$ contiene los genuinos caracteres de comillas de string (con código 11).

3. Ejecute este programa:

```
10 LET A$ = "LEN" "ABCD" " "
100 PRINT A$; "="; VAL A$
```

Esto fallará, porque VAL no trata las imágenes de comillas" " como comillas.

Introduzca algunas líneas entre la 10 y la 100 para cambiar las imágenes de comillas por A$ (que Ud. debe llamar CHR$ 11) y trate nuevamente.

Haga el mismo tipo de modificación que para el programa del capítulo 9, ejercicio 3.

4. Digite una sub-rutina que ignora espacios, escribiendo y ejecutando un programa que la use.

5. Este programa deja sin efecto cualquier circunstancia de la string "SUPERMAN" de A$.

```
500 INPUT A$
1000 FOR N = 1 TO LEN A$ — 7
1020 IF A$ (N TO N + 7) = "SUPERMAN"
THEN LET A$ (N TO N + 7) = "********"
1030 NEXT N
```

Escriba un programa que dé a A$ varios valores (esto es, "SUPERMAN ES FORZUDO") y aplique la rutina.

capítulo

22

Arrays

El array es un conjunto de variables, o elementos, todos con el mismo nombre y distinguidos por un número (el subíndice) escrito entre paréntesis después del nombre. Por ejemplo, el nombre podría ser A (como una variable de control de LOOPS FOR-NEXT, el nombre de un array debe ser sólo una letra) y doce variables, serían entonces, A(1), A(2), así en adelante, hasta A(12).

Los elementos de un array son llamados *variables con subíndice,* para distinguirlas de las variables sencillas con las cuales Ud. ya está familiarizado.

Antes de usar un array, Ud. debe reservar espacio para ella en el computador. Para hacer eso use la sentencia DIM (de dimensión)

DIM A (12)

Define un array llamado A con *dimensión 12* [o sea, hay 12 valores suscriptos A(1) . . . . . A(12)], y genera los 12 valores. También deja sin efecto cualquier array llamado A que existiese anteriormente. (Pero no una variable simple. Un array y una variable simple con los mismos nombres pueden coexistir. No habría confusiones entre ellas, porque una variable array, siempre tiene un subíndice).

El subíndice puede ser una expresión numérica arbitraria; ahora Ud. puede escribir

```
10 FOR N = 1 TO 12
20 PRINT A (N)
30 NEXT N
```

Ud. también puede definir arrays con más de una dimensión. En un array bidimensional, Ud. necesita dos números para especificar un elemento — como la línea y la columna que especifican la posición del caracter en la pantalla — y así tiene la forma de una tabla.

Alternativamente, si Ud. imagina los números de la línea y columna (Dos DIMENSIONES) como referencia a una página impresa, podrá usar una dimensión extra para el número de la página. Por supuesto que estamos hablando de arrays numéricos; los elementos no serían caracteres impresos como en un libro, pero sí números.

Piense que los elementos de un array tridimensional C están definidos por C (número de la página, número de línea y número de columna).

Por ejemplo, para definir un array bidimensional B con dimensiones 3 y 6, Ud. usa la sentencia DIM.

DIM B (3,6)
Esto proporciona 3 x 6 = 18 variables con subíndice
B(1.1), B(1.2), . . ., B (1.6)
B(2.1), B(2.2), . . ., B(2.6)
B(3.1), B(3.2), . . ., B(3.6)

El mismo principio funciona para cualquier número de dimensiones.

Aunque Ud. puede tener un número y un array con el mismo nombre, no puede tener dos arrays con el mismo nombre, aunque ellos tuvieran números diferentes de dimensiones.

Hay también arrays tipo string. Las strings en un array difieren de los strings simples, por ser de una longitud fija y la atribución a ellas siempre es PROCRUSTEANA. Otra manera de considerarlas, es como arrays (con una dimensión extra) de un caracter único. El nombre de un array string, es sólo una letra seguida de $, y una array string y una string simple no pueden tener el mismo nombre (al contrario del caso de los números).

Suponga, que Ud. quiere un array A$ de cinco strings. Ud. debe decidir cual es la longitud que esas strings deben tener — supongamos que 10 caracteres para cada una sea suficiente.

Entonces

DIM A$ (5,10) (digite eso)

Eso define un array 5 x 10 caracteres, mas puede considerar también cada fila como unastring.

A$ (1) = A$ (1, 1) A$ (1, 2) . . . A$ (1, 10)
A$ (2) = A$ (2,1) A$ (2,2) . . . A$ (2,10)
A$ (5) = A$ (5,1) A$ (5,2) . . . A$ (5,10)

Si Ud. proporciona el mismo número de subíndices (en este caso dos) que fueron dimensionados en la sentencia DIM obtiene sólo un caracter, pero si Ud. omite el último, obtiene la string de longitud fija. Así, por ejemplo, A$ (2.7) es el séptimo caracter en la string A$ (2)

Usando una noción de slicing, podríamos también escribir eso como A$(2) (7).

Ahora digite

```
DIM A$ (5,10)
LET A$ (2) = "1234567890"
```

y

```
PRINT A$ (2), A$ (2,7)
```

Ud. obtiene

```
1234567890          7
```

Para el último subíndice (el que puede ser omitido), Ud. también puede tener una operación de slicing, de manera que, por ejemplo

```
A$ (2,4 TO 8) = A$ (2) (4 TO 8) = "45678"
```

Recuerde

En una string array, todas las strings tienen lamisma longitud fija.

La sentencia DIM, tiene un número extra (el último) para especificar su largo.

Cuando Ud. escribe un variable con subíndice para una array, puede aumentar un número, o una operación slicing, para corresponder al número extra en la sentencia DIM.

RESUMEN

Arrays
Sentencia: DIM

EJERCICIOS

1. Defina un array M$ de doce strings, en la cual M$ es el nombre del enésimo mes.

(SUGERENCIA: la sentencia DIM será DIM M$ (12,9). Pruebe imprimiendo todos los M$(N) (Use un LOOP). Digite:

```
PRINT "este es el mes de"; M$ (5);
```

"El mes de María".

¿ Qué puede ser hecho sobre esos espacios?

2. Ud. puede tener un array string sin dimensión. Digite

```
DIM A$ (10)
```

y Ud. verá que A$ se comporta como una variable string, excepto que por tener siempre longitud 10, la atribución será PROCRUSTEANA.

capítulo

23

Cuando el computador está lleno

El TK 85 tiene un almacenamiento integral limitado, lo que no es difícil de notar. La mejor indicación que el área de un almacenamiento está llena, es normalmente una indicación de error 4. Sin embargo, otras cosas pueden acontecer y algunas de ellas son muy extrañas. Si Ud. tiene una expansión de memoria (RAM) desconéctela por un momento (habiendo desconectado primero el computador).

El *archivo de imagen,* o sea, el área donde el computador guarda las imágenes de la televisión es ingeniosamente dibujada, de tal manera, que sólo ocupa espacio con lo que fué impreso hasta entonces. Una línea de imagen está formada por 32 caracteres y un caracter *newline.* Esto significa que Ud. puede quedar sin espacio imprimiendo algo, y el momento más obvio, es cuando hace una lista. Digite:

```
POKE 16389,72
NEW
DIM A (355)
10 FOR I = 1 TO 15
20 PRINT I
```

Aquí aparece la primera sorpresa: la línea 10 desaparece de la lista. La lista debería incluir la línea actual, 20, y no hay espacio en la memoria para las dos líneas. Ahora digite:

```
30 NEXT I
```

Normalmente, sólo hay espacio para la línea 30 en la lista.
Ahora digite:

```
40 REM X                    (sin NEWLINE)
```

Ud. verá desaparecer la línea 30 y la línea 40 saltar para la parte superior de la pantalla. Ud. aún tiene el cursor **L** y puede moverlo.

Todo lo que Ud. vió es un mecanismo oscuro, que da a la mitad inferior de la pantalla 24 líneas, para darle prioridad sobre la mitad superior. Ahora digite:

```
XXXXXX (todavía sin NEWLINE)
```

y el cursor desaparecerá — no hay espacio para imprimirlo.

Digite otra X, sin NEWLINE. Todo desaparecerá, pero el programa aún se encuentra en el computador, como Ud. podrá comprobar, dejando sin efecto la línea 10, usando ⇩ y ⇧.

Ahora digite:

```
10 FOR I = 1 TO 15
```

Nuevamente esta línea se va a mover para la parte superior de la pantalla, como hizo la línea 40. Pero cuando Ud. presione NEWLINE, ella no entrará, a pesar de no tener mensaje de error, ni la **S** indicadora de error de sintáxis, para indicar algún error más. Esto es el resultado por no haber espacio en la memoria, para verificar la sintáxis de líneas que contienen números. (otro además del número de la línea en el inicio).

La solución es hacer espacio, de alguna manera, pero primero, deje sin efecto la línea 10 y presione EDIT. La pantalla quedará blanca, porque no hay espacio para traer el programa para la pantalla.

Presione NEWLINE y obtendrá parte de la lista.

Ahora deje sin efecto la línea 40 (la cual Ud. no quería, de ninguna manera) digitando:

40 (y NEWLINE)

Ahora, digitando la línea 10 nuevamente, ella aún no será aceptada. Nuevamente déjela sin efecto; Ud. de alguna manera tiene que conseguir espacio.

Tenga "in mente", que la razón por la cual la línea 10 fué rechazada, fué probablemente porque no había espacio para verificar la sintáxis de los números, 1 y 15. Entonces deje sin efecto la línea 20 del programa; Ud. debe tener espacio para entrar con la línea 10 y con la línea 20 (que no contiene números)

Pruebe esto; digite:

```
20
10 FOR I = 1 TO 20
20 PRINT I
```

y el programa está digitando correctamente. Digite:

```
GOTO 10
```

nuevamente Ud. verá que esta línea no es aceptada, pués su sintáxis no puede ser verificada. Sin embargo, si Ud, deja sin efecto y digita:

```
RUN
```

funcionará (RUN limpia el *array,* abriendo bastante espacio).

Ahora digite lo mismo que antes, desde el NEW hasta la línea 30, y entonces:

```
40 REM xxxxxxxxxxx
```

(11 veces x) lo cual acabará pareciendo a 40 REM.

Cuando Ud. presione NEWLINE, la lista será formada sólo por la línea 30 y en realidad, la línea 40 habrá sido totalmente perdida. Simplemente aconteció eso, porque ella era muy larga para caber en el programa; Ud. perderá ambas: la línea antigua del programa y la nueva, que debía reemplazarla.

Ahora, imprimir y hacer listas no hará que su computador quede sin memoria y Ud. no verá esas listas simplificadas y saltos de líneas, mas verá líneas perdiéndose y nuevamente, la solución será pedir espacio.

Para resumir,

1. Si el listado comienza a quedar resumido o las cosas comienzan a saltar por la pantalla, Ud. está quedando sin espacio.
2. Si NEWLINE parece no tener efecto en el fín de una línea probablemente no hay espacio para tratar con un número. Deje sin efecto la línea, usando EDIT — NEWLINE o RUBOUT.
3. NEWLINE puede hechar a perder toda una línea.

Para todas esas cosas, la solución es la misma: no se asuste y busque espacio libre.

La primera cosa que hay que hacer es considerar el CLEAR; Si Ud. tiene variables y no le importa perderlos, lo debe hacer.

Si falla eso, busque sentencias innecesarias en el programa, tales como REM y déjelas sin efecto.

RESUMEN

Cuando la memoria se llena, pueden acontecer cosas terribles; pero normalmente ellas no son fatales.

capítulo

24

Contando

A pesar que los ingenieros usan un sistema binario cuando están construyendo computadores (vea la columna de la derecha en la tabla a seguir), otro sistema numérico, el hexadecimal, que tiene como base 16, es útil porque es más fácil para leer, y puede ser convertido fácilmente en binario. Este comienza así:

HEXADECIMAL	CASTELLANO
0	CERO
1	UNO
2	DOS
.	.
.	.
.	.
9	NUEVE
A	DIEZ
B	ONCE
C	DOCE
D	TRECE
E	CATORCE
F	QUINCE
10	DIECISEIS
11	DIECISIETE
.	.
.	.
.	.
19	VEINTECINCO
1A	VIENTESEIS
1B	VIENTISIETE
.	.
.	.
.	.
1F	TREINTA Y UNO
20	TREINTA Y DOS
21	TREINTA Y TRES
.	.
.	.
.	.
9E	CIENTO CINCUENTA Y OCHO
9F	CIENTO CINCUENTA Y NUEVE
40	CIENTO SETENTA
41	CIENTO SETENTA Y UNO
.	.
.	.
.	.
FE	DOCIENTOS CINCUENTA Y CUATRO
FF	DOCIENTOS CINCUENTA Y CINCO
100	DOCIENTOS CINCUENTA Y SEIS

Si Ud. estuviera usando notación HEXADECIMAL y de hecho quiere dejarlo bien en claro, escriba "h" al final de los números. Por ejemplo: ciento cincuenta y ocho, escriba "9Eh" y lea "nueve E hexa".

Los diferentes sistemas numéricos comienzan así:

CASTELLANO	DECIMAL	HEXADECIMAL	BINARIO
CERO	0	0	0 ó 0000
UNO	1	1	1 o 0001
DOS	2	2	10 ó 0010
TRES	3	3	11 ó 0011
CUATRO	4	4	100 ó 0100
CINCO	5	5	101 ó 0101
SEIS	6	6	110 ó 0110
SIETE	7	7	111 ó 0111
OCHO	8	8	1000
NUEVE	9	9	1001
DIEZ	10	A	1010
ONCE	11	B	1011
DOCE	12	C	1100
TRECE	13	D	1101
CATORCE	14	E	1110
QUINCE	15	F	1111
DIECISEIS	16	10	10000

El punto importante es que dieciseis es igual a 2 elevado a la potencia de 4, lo que hace muy fácil la conversión entre HEXEDECIMALES y BINARIOS.

Para convertir de HEXADECIMAL para BINARIO, cambie cada dígito HEXADECIMAL por cuatro BITS, usando la tabla anterior. Los dígitos binarios **0** y 1 son considerados como BITS.

Para convertir BINARIOS en HEXADECIMALES, divida el número binario en grupos de cuatro, iniciando por la derecha, y luego cambie cada grupo por un correspondiente hexadecimal.

Dentro del computador, en su gran mayoría, los BITS están agrupados en conjuntos de ocho, o *bytes.* Un byte único, puede representar a qualquier número de cero hasta 255 (11111111 ó FF HEXA); o, cualquier caracter del conjunto de caracteres del TK 85. Su valor puede ser escrito con dos dígitos hexadecimales.

Dos bytes pueden ser agrupados para formar lo que técnicamente es llamado de *palabra.* Una palabra puede ser escrita usando 16 bits o cuatro dígitos hexadecimales y representa un número desde **0** hasta (en decimal) $2^{16} - 1 = 65535$.

Un byte tiene siempre 8 bits, pero las palabras varían de un computador para otro.

RESUMEN

Sistemas DECIMAL, HEXADECIMAL y BINARIO, BITS y BITES (no los confunda) y palabras.

EJERCICIOS

1. ¿ Cómo haría Ud. una converción entre DECIMAL Y HEXADECIMAL? Escriba un programa en el TK 85 para convertir valores numéricos en strings, dando su representación en HEXADECIMAL y vice-versa (esto es lo que STR$ y VAL hacen de la representación decimal).

capítulo

25

Como funciona el computador

No es el objetivo de este manual describir en detalles la parte electrónica del TK 85 y su operación, pero podemos dar una idea de la función de sus componentes.

El circuito integrado más importante del TK 85 es la CPU (Unidad Central de Procesamiento), que es un microprocesador Z80A. El procesador hace la aritmética y electrónicamente controla el resto del computador de acuerdo con el programa del sistema operacional.

El sistema operacional será contenido en una memoria ROM, esto es, en un dispositivo electrónico de almacenamiento, que tiene un programa permanente para hacer funcionar la CPU.

El programa en forma simbólica, es una gran secuencia de BYTES (BITE es un número entre 0 y 255). Cada byte tiene una DIRECCIÓN mostrando su posición en la EPROM. El primero tiene dirección 0, el segundo tiene dirección 1, y así en adelante, hasta 10239, porque el TK 85 tiene un BASIC de 10K.

Ud. puede ver que BYTE está en una determinada dirección, usando la función PEEK. Por ejemplo este programa imprime los primeros 21 BYTES de la EPROM (y sus direcciones).

```
10 PRINT "DIRECCIÓN"; TAB 8; "BYTE"
20 FOR A = 0 TO 20
30 PRINT A; TAB 8; PEEK A
40 NEXT A
```

La memoria RAM es un "BLOQUE DE PROYECTO" electrónico, que está ligado a la CPU. El programa BASIC que Ud. digita es almacenado electrónicamente ahí, como son: las variables del programa, la imagen de la televisión y las variables del sistema.

Al igual que en la ROM, en la RAM también el almacenamiento es hecho en BYTES, cada uno con una dirección, estos varían de 16384 hasta 32768 o 65280 si el TK 85 tuviera 16 ó 48K. Como en la ROM, Ud. puede encontrar los valores de esos BYTES usando PEEK. La diferencia es que Ud. puede cambiarlos.

Digite:

```
POKE 20000,57
```

Esto da al BYTE en la dirección 20000 el valor 57. Si Ud. ahora digita

```
PRINT PEEK 20000
```

Obtendrá nuevamente el número 57 (Pruebe hacer POKE de otros valores). Fíjese que la dirección tiene que estar entre 0 y 65535. El valor debe estar entre 0 y + 255.

La posición de POKE le da un poder inmenso al tratar con el computador, si Ud. supiera como usarlo; pero, el conocimiento necesario es mucho mayor del que puede haber en un manual de introducción como es éste.

Fuera de los componentes ya descriptos, existen otros circuitos integrados de menor complejidad que desempeñan varias funciones lógicas. Estos circuitos integrados son esenciales para el funcionamiento del computador.

El modulador, convierte la salida del computador para la televisión, en una forma adecuada y el regulador, convierte los 10 volts contínuos en 5 volts regulados.

RESUMEN

CIRCUITO
Sentencias: POKE
Funciones: PEEK

capítulo

26

Empleando lenguaje de máquina

Este capítulo es escrito para aquellos que entienden el lenguaje de máquina del Z80, el conjunto de sentencias que el Z80 usa.

Si Ud. desea aprenderlo, el mejor medio es consultar el Z80 ASSEMBLY LENGUAJE PROGRAMMING MANUAL, conjuntamente con el Z80-CPU, Z80 A-CPU TECHNICAL MANUAL, publicado por la ZILOG; aunque no son recomendables para principiantes.

Ud. puede también buscar libros sobre el Z80. Existen en Inglés y otros idiomas. Hay poco sobre esa materia en castellano.

En lenguaje de máquina, las rutinas pueden ser ejecutadas en un programa BASIC usando la función USR. El argumento de USR es la dirección de inicio de la sub-rutina y su resultado es un entero sin signo, de Bytes, el contenido del par de registros bc en retorno. La dirección de retorno para EL BASIC es mantenida en forma normal; así el retorno es hecho a través de una sentencia RET del TK 85.

Hay algunas restricciones en rutinas USR:

(I) En retorno, los registradores iy e i deben tener valor 4000 h y 1E h.

(II) La rutina de display usa los registros a', f', x, iy; una rutina USR no los debe usar si estuviera operando en la modalidad slow (ver apéndice C) (tampoco es seguro leer el par af').

Todas las líneas del procesador están expuestas en la parte posterior del TK 85. Así, en principio, Ud. puede hacer todo lo que se hace con el Z-80 en el TK 85, aquí tenemos un diagrama de las partes expuestas:

Una parte del código de máquina en el medio de la memoria corre el riesgo de ser cubierta por el sistema BASIC.

Algunos puntos más seguros son:

(I) En una sentencia REM: Digite una sentencia REM con suficientes caracteres para contener los códigos de máquina, en el cual Ud. hará un POKE. Evite sentencias de HALT, ya que esto será reconocido como fín de la sentencia REM.

(II) En una string: Defina una string larga y luego atribuya un código de máquina a cada caracter.

Em ambos casos el código está resguardado, pero sujeto a cambios, especialmente en el caso con string. En el apéndice A, en el conjunto de caracteres, Ud. encontrará los caracteres y las instucciones del Z-80, escrito uno al lado del otro. Ud. los encontrará útiles cuando tenga que entrar con los códigos.

(III) En el tope de la memoria: el TK 85 es conectado, efectúa una prueba para ver cuanta memoria hay y coloca la pila de la máquina exactamente en el tope, de manera que no queda espacio para rutinas. USR — Ahí guarda la dirección del primer Byte no existente, en una variable del sistema, conocida como RTP, en los dos Bytes con direcciones 16388 y 16389. Por otro lado NEW, no hace una prueba de memoria llena, sólo verifica hasta antes de la dirección en RTP. Así, si Ud. hiciera un POKE en la dirección de un byte existente para RTP, NEW para toda la memoria de ese byte en adelante, está fuera del sistema BASIC y es dejada a un lado. Por ejemplo, suponga que Ud. tiene 16K de memoria y recién conectó el computador.

```
PRINT PEEK 16388 + 256*PEEK 16389
```

proporciona la dirección del primer BYTE inexistente.

Ahora supongamos que Ud. quiere cambiar RTP para 18412, por ejemplo, observando que 18412 = 236 + 256*71.

Entonces, para efectuar el cambio debemos digitar los siguientes comandos:

```
POKE 16388,236
POKE 16389,71
```

y a continuación NEW. Verifique ahora cual es el tope de la memoria. Si digita NEW nuevamente, no alterará lo que fué depositado en una dirección mayor que la del indicador por RTP.

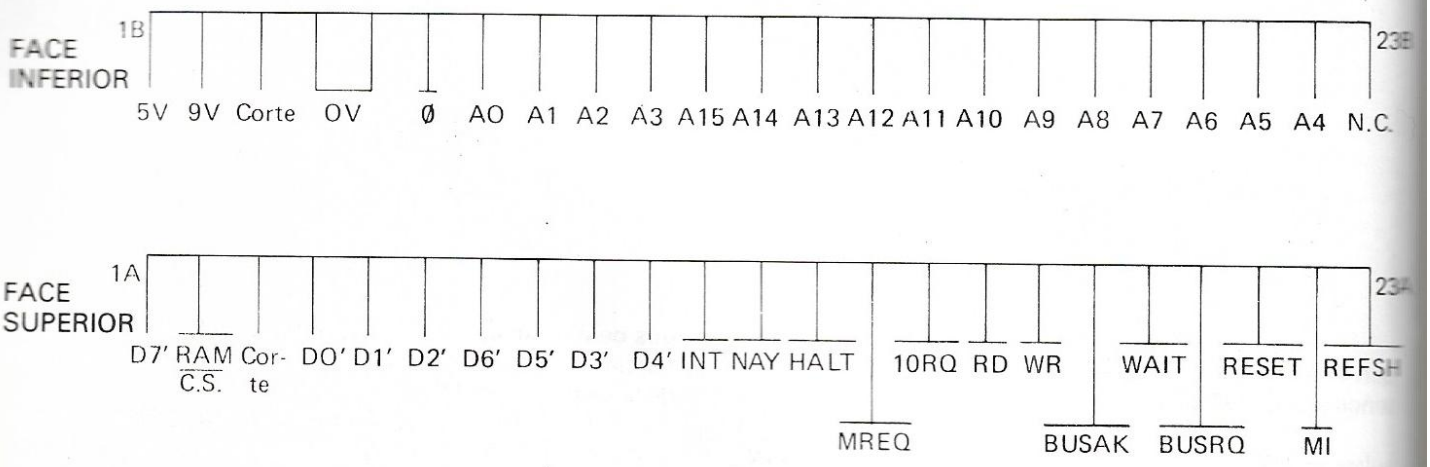
El tope de la memoria es un buen lugar para rutinas USR; seguro (inclusive del NEW) y móvil. La principal desventaja es que no es conservada por el SAVE.

RESUMEN.

Funciones: USR
Sentencias: NEW

EJERCICIOS

1. Haga RTP igual a 16700 y luego ejecute NEW. Ud. tendrá una idea de lo que sucede cuando la memoria se llena.



BIBLIOGRAFIA:

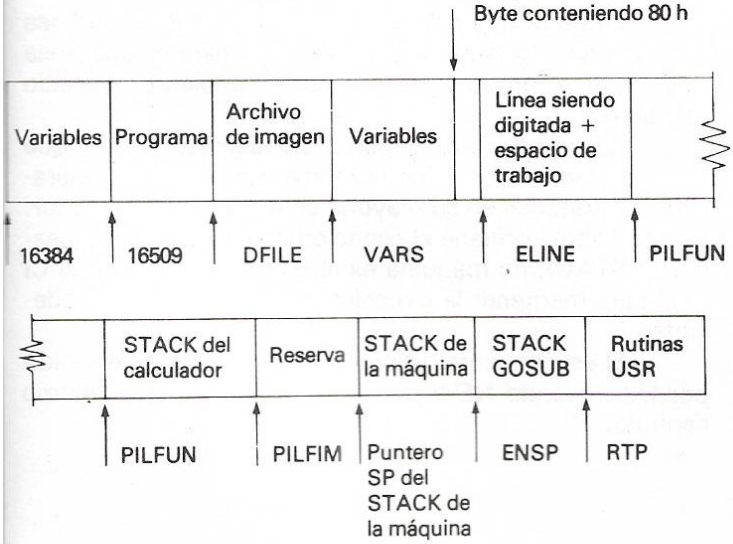
(Z80)
— Z80 CPU TECHNICAL MANUAL de ZILOG o MOSTEK
— PROGRAMMING THE Z80 — RODNEY ZAKS
— Z80 PROGRAMMING FOR LOGIC DESIGN — OSBORNE
— Z80 ASSEMBLY LANGUAGE PROGRAMMING — OSBORNE

capítulo

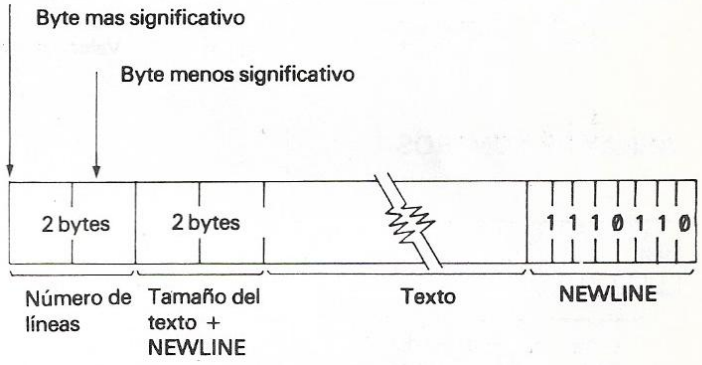
27

Organización de almacenamiento

La memoria es dividida en diferentes áreas, para almacenar diferentes tipos de informaciones. Las áreas son de tamaño suficiente para contener sólo las informaciones que ellas realmente contienen; y si Ud. inserta más, en un determinado lugar (por ejemplo, agregando una línea de programa o variable), el espacio es creado desplazando hacia arriba todo lo que hay antes de ese punto. De la misma forma, si Ud. deja sin efecto una información, todo lo que está arriba de la información dejada sin efecto, es desplazado para abajo.



Las variables del sistema contienen muchas informaciones, que dicen al computador el estado en el cual se encuentra. Ellas están todas en una lista en el próximo capítulo, pero por ahora, fíjese que hay algunas (llamadas DFILE, VARS, ELINE, por ejemplo) que contienen las direcciones de límite entre las diferentes áreas de la memoria. Esas no son variables y sus nombres no serán reconocidos por el computador. En el programa, cada línea es almacenada como:



Observe que, en oposición a todos los otros casos de números de dos bytes, en el Z80, el número de la línea aquí (es también una variable de control de LOOP FOR-NEXT) es almacenada con su primer byte, más significativo, esto es, en el orden que Ud. los escribiría.

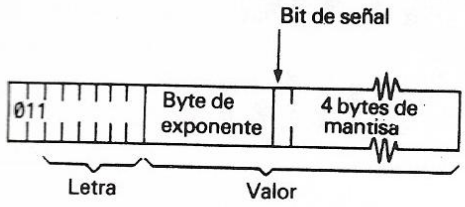
Una constante numérica en el programa es seguida por su forma binaria, usando el caracter CHR$ 126, seguido de 5 BYTES para el número.

El archivo de imagen es la copia, en la memoria, de la imagen de la televisión. Comienza con un caracter NEWLINE, seguido de 24 líneas de texto, cada una terminando con NEWLINE. El sistema es dibujado de forma, que la línea de texto no necesite siempre de 32 caracteres. Los espacios finales pueden ser omitidos. Eso es hecho para economizar espacio cuando la memoria es pequeña.

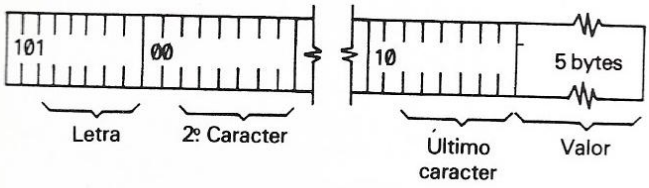
Cuando el total de memoria (de acuerdo con la variable de sistema RTP) es menor que 3 1/4 K, una limpieza de pantalla — como es hecho en el inicio o por un CLS — consiste de sólo 25 NEWLINES. Cuando la memoria es más grande, es hecha una limpieza de pantalla en 24*32 espacios; el SCROLL, mientras tanto y ciertas condiciones donde la parte inferior de la pantalla se expande para más de dos líneas, pueden perturbar esto, produciendo líneas cortas en la parte inferior de la pantalla.

Las variables tienen formatos diferentes, dependiendo de su naturaleza.

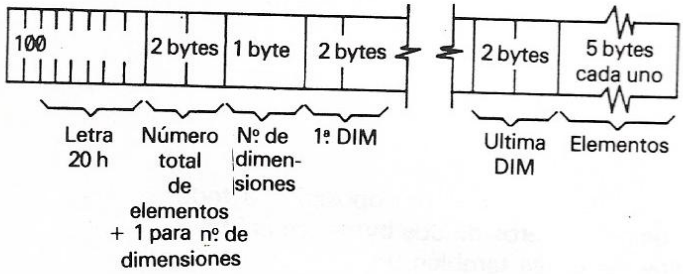
El nombre de un número, si fuera sólo una letra, será



Número cuyo nombre es mas largo que una letra:



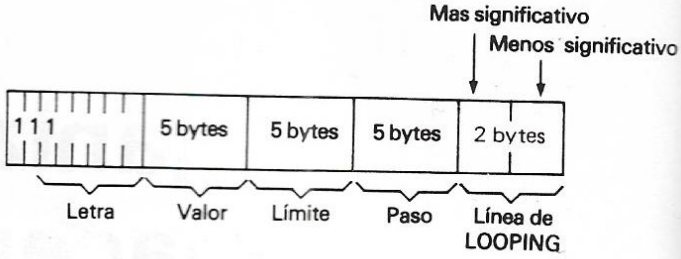
ARRAY DE NÚMEROS



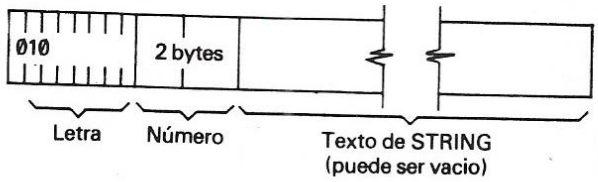
El orden de los elementos es:
Primero, el nombre del elemento para el cual el primer subíndice es 1.
En seguida, el elemento para el cual el primer subíndice es 2.
Luego, los elementos para los cuales el primer subíndice es 3.
Así en adelante para todos los valores posibles del primer subíndice.
Los elementos con el primer subíndice dado, están ordenados de la misma manera, usando el segundo subíndice y así en adelante, hasta llegar al último.

Como ejemplo, el elemento de array B 3x6, en el capítulo 22, están almacenados en el órden B(1,1), B(1.2), B(1.3), B(1.4), B(1.5), B(1.6), B(2.1), B(2.2), . . ., B(2.6), B(3.1), B(3.2), . . ., B(3.6).

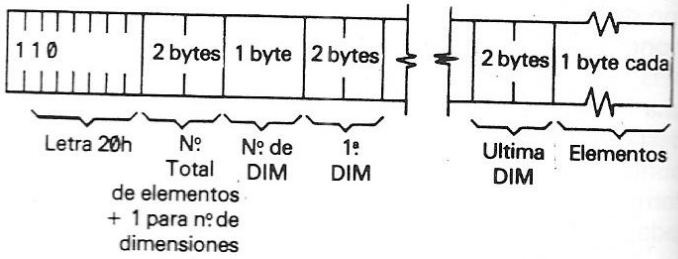
Variable de control de un LOOP FOR-NEXT;



CADENA (STRING)



ARRAY DE CARACTER (STRING)



La parte que empieza en ELINE, contiene la línea que está siendo digitada (como un comando, una línea de programa, un dato de entrada) y también un espacio de trabajo.

La calculadora es parte del sistema BASIC que trata con aritmética y los números que están en operación son tratados en su mayoría en el STACK calculador. La parte libre contiene el espacio hasta entonces no usado. El STACK de máquina es el STACK usado por el CI Z-80 para mantener la dirección de retorno y así en adelante.

El espacio para rutina USR tiene que ser separado por Ud., usando NEW como fue descripto en el último capítulo.

capítulo

28

Variables del sistema

Los Bytes en la memoria, de 16384 hasta 16508, están reservados para usos específicos del sistema. Ud. puede hacer un PEEK con ellos, para descubrir varias cosas sobre el sistema, y algunas de ellas pueden ser útiles. Aquí daremos un listado de ellas, con sus respectivos usos.

Estas son las llamadas variables del sistema, no las confunda con las variables usadas por el BASIC. Ud. no puede usar los mismos nombres en un programa BASIC; son sólo mnemónicos, que son usados para hacer más fácil la referencia de las variables.

Las abreviaciones en la columna 1 tienen los siguientes significados:

X no puede hacer un POKE, porque será destruido por el sistema.

N hace un poke, la variable no tendrá efecto.

S las variables son preservadas por el SAVE.

El número de la columna 1 es el número de Bytes de la variable.

Para 2 Bytes, el primero es el menos significativo-al contrario de lo que Ud. esperaría.

Para hacer un POKE de un valor N para una variable en la dirección N, use:

```
POKE n,v – 256*INT (v/256)
POKE n + 1, INT (v/256)
```

y para hacer un PEEK de su valor, use la expresión:

```
PEEK n + 256*PEEK (n + 1)
```

NOTAS	DIRECCIÓN	NOMBRE	CONTENIDO
1	16384	CODR	1 menos que el código registrado. Se inicia en 255 (por –1), así, PEEK 16384, si funciona, dará 255. POKE 16384, n puede ser usado para forzar un error HALT: $0 \leqslant n \leqslant 14$ da una de las indicaciones, $15 \leqslant N \leqslant 34$ ó $99 \leqslant n \leqslant 127$ da una indicación no standard y $35 \leqslant n \leqslant 98$ probablemente desordenará el archivo de imagen.
X1	16385	BAND	Varias Banderas para controlar el sistema BASIC.
X2	16386	ENSP	Dirección del primer Ítem en el Stack de máquina (después de GOSUB retorno).

2	16388	RTP	Dirección del primer Byte arriba del área de sistema BASIC. Ud. puede hacer un POKE, con el fín de que NEW reserve espacio arriba de esa área (vea capítulo 26) o para engañar a la CLS, para reservar un archivo de imagen más pequeño (capítulo 27).
N1	16390	MODO	Especifica cursor K, L, F ó G. Línea de la sentencia que está siendo ejecutada. Un POKE no tiene consecuencia, excepto en la última línea del programa.
N2	16391	CPB	Número de la línea que está siendo ejecutada.
S1	16393	VERSN	Ø identifica ROM de 8K en programas resguardados.
S2	16394	LPC	Número de línea actual.
SX2	16396	DFILE	Vea capítulo 27.
S2	16398	POSPR	Dirección de la posición del PRINT en el archivo de imagen. Puede ser hecho en POKE, de tal manera que una salida PRINT sea mandada para otro lugar.
SX2	16400	VARS	Vea capítulo 27.
SN2	16402	DEST	Dirección de la variable en atribución.
SX2	16404	ELINE	Vea capítulo 27.
SX2	16406	ENCAR	Dirección del próximo caracter a ser interpretado; el caracter después del argumento del PEEK, o el NEWLINE en el final de una sentencia POKE.
S2	16408	ENSX	Dirección del caracter precediendo el indicador, **S** .
SX2	16410	PILFUN	Vea capítulo 27.
SX2	16412	PILFIM	Vea capítulo 27.
SN1	16414	CALREG	Registrador del calculador.
SN2	16415	MEM	Dirección usada para cálculos en la memoria.
S1	16417	NO USADO	
SX1	16418	DFSZ	Número de líneas (incluyendo una línea en blanco) en la parte inferior de la pantalla.
S2	16419	LTOP	El número de la línea superior del programa en lista automática.
SN2	16421	ULTK	Muestra la última tecla presionada.
SN1	16423	—	Estado de *debounce* del teclado.
SN1	16424	HARG	Número de líneas en blanco, arriba o abajo de la imagen — 31.
SX2	16425	PXLN	Dirección de la próxima línea de programa a ser ejecutada.

S2	16427	VCPB	La línea para la cual salta CONT.
SN1	16429	BANDX	Varias banderas.
SN2	16430	LENCA	Longitual de la designación del tipo de *string* en atribución.
SN2	16432	SXEN	Dirección del próximo Ítem en la tabla de sintaxis.
S2	16434	SEMT	Da origen al RND. Esta es la variable actualizada por el RAND, la "semilla".
S2	16436	QUAD	Cuenta los cuadros presentados en la televisión. BIT 15 es 1. BITS del 0 al 14 van disminuyendo por cada cuatro enviados a la televisión. Eso puede ser usado como temporización, pero PAUSE también la usa — PAUSE reajusta el bit 15 para 0 e introduce en los bits de 0 a 14 la duración de la pausa. De esa forma, cuando hayan llegado a cero, la pausa termina. Si la pausa (PAUSE) se detiene, por haber sido presionada una tecla, el bit 15 es ajustado para 1 nuevamente.
S1	16438	CORDX	Coordenada X del último punto marcado.
S1	16439	CORDY	Coordenada Y del último punto marcado.
S1	16440	PR-CC	Es el BYTE de dirección menos significativo de la próxima posición para LPRINT imprimir (PRBUFF).
SX1	16441	COLPR	Número de la columna para la posición PRINT.
SX1	16442	LINPR	Número de la línea para la posición PRINT.
S1	16443	BANCO	BANDERAS variadas. El bit 7 está activo (1) durante la modalidad SLOW.
S33	16444	PRBUFF	BUFFER de la impresora (33º caracter es NEWLINE).
SN30	16477	MEMBO	Área de la memoria para cálculo, usada para almacenar los números que no pueden ser colocados convenientemente en el STACK del calculador.
S2	16507	NO USADO	

EJERCICIOS.

1. Plantee éste programa:

```
10 FOR N = 0 TO 21
20 PRINT PEEK (PEEK 16400 + 256*PEEK 16401
+ N)
30 NEXT N
```

Esto muestra los primeros 22 bytes del área de variables.

Trate de combinar las variables de control N con la discusión en el capítulo 27.

2. En el programa de arriba, cambie la línea 20 para

```
20 PRINT PEEK (16509 + N)
```

Esto muestra los 22 primeros bytes del área de programa.

Combine eso con el propio programa.

capítulo

29

Funciones especiales de almacenamiento

Una de las funciones elementales de los computadores, es el almacenamiento y la recuperación de informaciones en medios "permanentes". El sistema de grabador y cassette común, es sin duda, el medio "permanente" más compatible con un computador personal, por su personal, por su facilidad de transporte, costo, confiabilidad y popularidad de uso.

Debido a las propias características de este sistema, cuya finalidad original es la reproducción en la área de audio, la velocidad de la transferencia para y desde, se encuentra limitada.

En el TK 85 se puede optar por la transferencia en 2 velocidades: 300 (Formato I) y 4200 (Formato II) bauds (bits por segundo), siendo que cada una de ellas, tiene su propia importancia: Formato II reduce el tiempo de transferencia considerablemente en relación al Formato I. Por ejemplo: 16K demora 30 segundos, por lo tanto, 48K demora 90 segundos aproximadamente. El Formato I posee características de frecuencia tales, que lo hacen compatible con el uso de cintas de las más variadas calidades. Sin embargo, en el Formato II, es recomendable la utilización de cintas de bajo ruido (LOW NOICE), como también, alta relación de señal/ruido en la cabeza lectora del grabador.

Una explicación del funcionamiento en el Formato I, es dada en el capítulo 16.

Además de la operación especial en HIGH-SPEED, el TK 85 incluye funciones de I/Ø (Input/Output). Este grupo de funciones, permite almacenar y recuperar informaciones específicas, independientes de programas en cinta cassette.

Estas informaciones de I/Ø permiten el tratamiento de un conjunto de datos por diversos programas, el tratamiento de un gran archivo por un programa único, u otras aplicaciones que requieran operar con datos, independientes de su origen.

El ROM del TK 85 contiene el Tape Operating System T.O.S. que incluye las funciones especiales de I/Ø y HIGH-SPEED, a las cuales ellas tienen acceso, vía función USR del BASIC. Estas funciones pueden ser usadas en líneas del programa o como comandos directos.

USR es una función del BASIC, que le permite a Ud. ejecutar una sub-rutina en lenguaje de máquina. En verdad, USR trabaja como una función, lo que significa que ella sigue las reglas de la sintáxis de funciones y forma parte de una expresión aritmética. Todos los USR necesitan tener un parámetro único (el cual es realmente la dirección inicial de ejecución de la sub-rutina en languaje de máquina), y todos ellos resultan con un valor único al volver para el interpretador BASIC. ellos pueden ser usados en una sentencia como:

```
100 LET STATUS = USR 8192
```

Donde 8192 es la dirección inicial de la ejecución del USR y STATUS es la variable numérica que permite al BASIC "hacer algo" con el valor de vuelta del USR (puede ser usado cualquier nombre válido). En el ejemplo de arriba, el valor es otorgado a la variable STATUS, la cual el usuario puede aprovechar como estime conveniente.

El concepto de retornar un valor, consiste en la convencional idea de una función como SQR (raíz cuadrada), donde Ud. da un número y recibe otro valor, en este caso, la raíz cuadrada.

Si bien las funciones USR$ no devuelven necesariamente un valor útil a la finalidad exigida, en el caso de los comandos DSAVE, DVERIFY y DLOAD, puede ser aprovechado el valor devuelto, para indicar un código de reportaje especial.

Este código indica si el resultado de la operación referida fué satisfactorio o no. El uso de esos códigos será discutido posteriormente.

HIGH-SPEED

El formato II, HIGH-SPEED, ofrece tres comandos: HISAVE, HIVERIFY y HILOAD, cumpliendo las fun-

ciones de almacenar, verificar y leer las informaciones para y desde la cinta cassete.

Para garantizar un correcto desempeño de HIGH-SPEED se recomienda el uso de cintas de bajo ruido, así como grabadores cuya cabeza reproductora posea una buena relación señal/ruido.

a) Para almacenar un programa en HIGH-SPEED, se usa el comando HISAVE obtenido a través de la sentencia:

RAND USR 8405

pero, antes de presionar NEWLINE, espere algunos segundos, después de haber conectado el grabador, para asegurarse que la parte no magnetizada de la cinta, no se encuentra sobre la cabeza de la grabación.

Después de presionar NEWLINE, asegúrese que el puntero del grabador, que indica el nivel de la grabación no ultrapase la zona (normalmente roja) indicada por el fabricante.

En el caso que ultrapase, Ud. deberá disminuir el nivel de grabación, manipuleando el control correspondiente en su grabador, en caso contrario, la señal será grabada distorcionada y por lo tanto sin posibilidad de ser recuperada.

HISAVE almacena tanto el programa, como las variables que se encuentram definidas en la memoria del computador al ejecutar el comando.

b) En caso que Ud. tenga dudas con respecto a la calidad de la grabación realizada en HIGH-SPEED, Ud. podrá utilizar el comando HIVERIFY, obtenido a través de la sentencia:

RAND USR 8539

Rebobine la cinta cassette hasta antes del programa, ajuste el volúmen inicialmente en 30% de la zona, con el control de tono en el agudo máximo.

Conecte el grabador en reproducción y digite el comando HIVERIFY y NEWLINE. Después de algunos segundos aparecerá una figura característica en su video, correspondiendo al "eco" de la lectura del computador: deberá aparecer la indicación "OK" si la grabación estuviera correcta. En caso que aparezca la indicación "ERROR", significa que la grabación fué incorrecta o que Ud. no ajustó adecuadamente el volúmen de la reproducción. En este caso, pruebe con otro volúmen.

Si en su video no aparece ninguna indicación y la imagen permanece oscura, probablemente el nivel de reproducción se encuentra bajo, ajústelo e intente nuevamente.

c) Cuando Ud. quiera "cargar" un programa almacenado en cinta con HISAVE, deberá usar el comando HILOAD obtenido a través de la sentencia:

RAND USR 8630

en el cual Ud. deberá actuar en forma idéntica al HIVERIFY, diferenciándose en el resultado, ya que HILOAD "carga" el programa y las variables asociadas de la cinta, indicando esto con código de reportaje 0/0 en la parte inferior izquierda del video o a través de la ejecución directa del programa en cuestión.

Si desea detener la ejecución de cualquiera de los comandos HIGH-SPEED, Ud. deberá apretar la tecla BREAK.

FUNCIONES DE I/0.

ALMACENAMIENTO Y RECUPERACIÓN DE INFORMACIÓN.

Las funciones de I/0 (INPUT/OUTPUT) permiten cargar y recuperar exactamente como se hace al usar los comandos SAVE y LOAD. Lo único nuevo que Ud. tiene que hacer, es controlar el grabador adecuadamente con una única unidad de salida para SAVE o una unidad de entrada para LOAD. Si Ud. se olvida de conectar el grabador en la hora cierta, el comando SAVE dará salida para todas las informaciones, pero entonces, no irá para ningún lugar. Por otro lado, LOAD, quedará inmóvil esperando alguna entrada.

En ambos casos, la tecla BREAK liberará el TK 85 y Ud. podrá intentarlo nuevamente.

ALMACENAMIENTO DE ARCHIVOS.

Vamos a considerar algunos principios básicos de almacenamiento de datos. Considerando que todos los datos usados por el programa pueden estar en el RAM, a través del código BASIC, no hay necesidad de ninguna memoria auxiliar, pero como en varias aplicaciones del computador, el objetivo es hacer operaciones con extensas listas de datos, las cuales no tendrán cabida en la memoria RAM; Por ejemplo, si Ud. quiere procesar una lista de direcciones y usar 100 bytes para cada usuario, con el fín de almacenar nombre, domicilio, etc, 16 Kbytes soportarán solamente 160 usuarios, sin considerar el espacio usado para el área de trabajo y su programa.

En el inicio de la era de la computación, cuando un Kbyte costaba carísimo, ese problema era cada vez más crítico, pero aunque ahora la memoria sea barata, el tamaño es todavía limitado por el procesador. En el caso del Z80 el límite de 64K, de los cuales 10 Kbytes de la ROM son usados por el sistema operacional.

Un modo de escapar de esa limitación, es almacenar datos en medios secundarios, como cinta cassette. La ventaja es almacenar los datos como una serie de bloques en la cinta, los cuales podrán ser leídos en su programa de a uno. Aunque los datos totales en la cinta sean más que su RAM, Ud. podrá trabajar por partes. En el caso de la lista de domicilios, Ud. necesitará un bloque en la cinta, para cada usuario.

Luego, si desea imprimir una serie de domicilios para todos los usuarios, su programa tendrá simplesmente que leer un bloque, imprimir el domicilio del usuario, leer el próximo bloque, imprimir el domicilio, y así en adelante.

Su TK 85 puede usar cinta para resguardar programas (con SAVE) y cargar (con LOAD) programas y datos en conjunto.

Cuando un programa es cargado éste cargará todo lo que lee en el RAM, por lo tanto, LOAD no puede ser usado por un programa para leer datos de un archivo de cintas, ya que el acto de cargar, borra el propio programa. Lo que es necesario, es un mecanismo donde los datos puedan ser leídos de la cinta en una área, sin afectar el programa en otra información.

Una área como esa es generalmente llamada de "buffer".

BUFFERS.

Las funciones de I/Ø del TK 85 usan variables, tipo string como entrada y salida de los buffers. Para salida, primero es colocada la información en el buffer (String) deseado, y luego llama al DSAVE con la función USR.

Tanto para leer como para escribir en el buffer, el TK 85 necesita saber cual string (array) Ud. tiene "in mente". Normalmente, Ud. necesita usar más de un buffer en sus programas.

Un caso típico es, cuando se está escribiendo un informe y se desea guardar la página intacta en un buffer, mientras usa otro, para construir las líneas de detalles para cada página. Ud. deberá recordar que el nombre de una variable string, consiste de una única letra seguida del signo peso ($), como ejemplo A$. Para permitir total flexibilidad al respecto de cuantos buffer Ud. va a usar, el TK 85 no necesita que se use nombres específicos para sus buffers, sin embargo, la variable string Z$ es usada como apuntador del buffer.

También otra variable, esta vez, una variable numérica Z, es usada específicamente por las funciones de 1/Ø. Las variables que son usadas para propósitos especiales como éstas, son siempre referidas como variables reservadas, pero Ud. notará que hay libertad para usar estas variables, de cualquier modo, cuando no están siendo utilizadas las funciones especiales con el comando USR. En otras palabras, programas existentes que usen estas variables, no serán afectados, ya que el TK 85 no "amarra" los nombres de las variables. Eso sólo acontecerá cuando Ud. ejecute el USR, que examina las variables del programa para hallar la que necesita.

Si Ud. se hubiera olvidado de definirlas en su programa, entonces le será indicado esto a través de un reportaje especial.

La comunicación entre su programa y las funciones especiales, es hecha a través de variables con nombres particulares y la comunicación hecha entre estas funciones y Ud. es a través de códigos de reportaje.

Supongamos que Ud. quiere C$ como buffer y pretende hacerlo tener 250 bytes, Ud. lo dimensiona de la siguiente forma:

```
100 DIM C$ (250)
```

(Los parámetros de dimensión deben aparecer en el comienzo, o bien cerca de él en su programa, pués ellos necesitan ser definidos, para ser localizado el espacio necesario en la memoria RAM).

Como el punto más apartado en su programa, Ud. tendrá los datos que quiere escribir en el C$, y entonces podrá accionar el TK 85 para ejecutar la operación de salida.

Antes de hacer esto, Ud. necesitará indicar de cual string serán recogidos los datos, esto es obtenido cargando la letra C, como primer caracter de la variable string Z$. Entonces la secuencia será parecida a esto:

```
100 DIM C$ (250)    Definición del buffer
    . . .           (se ejecuta solamente
    . . .           una vez)
    . . .
500 LET Z$ = "C"    Acertando la dirección
510 LET STATUS
= USR 8288          Pasando para cinta la
                    string C$
```

Para ver la flexibilidad de acceso, vamos a suponer que tenemos un programa que lee de la cinta e imprime en la impresora.

El programa aparecerá más o menos así:

```
100 DIM T$ (100)    Buffer de entrada
    . . .
    . . .
    . . .
500 REM
LEER BLOQUE DE LA CINTA
510 LET Z$ = "T"    T$ es seleccionado
    . . .           como buffer de entrada
    . . .
    . . .
    . . .
520 LET STATUS =    La información es leída en T$
USR (8305)
    . . .
    . . .
    . . .
700 REM — IMPRIMIR
    . . .
    . . .
    . . .
710 LPRINT T$       Imprime T$
    . . .
    . . .
    . . .
990 GOTO 500        Va a buscar el próximo
                    bloque.
```

Como Ud. puede notar, el parámetro de dimensión debe ser definido solamente una vez, utilizando Z$ para indicar cual buffer deberá ser usado por el TK 85.

Acuérdese que el USR es, de hecho, una sub-rutina assembler, que es ejecutada por el procesador central del TK 85. Si fuera ejecutado un comando USR para leer o grabar, la primera cosa que él haría, sería buscar entre las variables existentes, hasta hallar Z$. Habiendo encontrado Z$, él extraerá el primer caracter y descubrirá

la identidad de la string (array) deseada para la utilización en su buffer.

Después, nuevamente busca entre sus variables para encontrar el buffer. Si Ud. se hubiera olvidado de designar Z$ o dimensionar el buffer, o haber definido los dos incorrectamente, toda la acción será abandonada y un código de reportaje será devuelto, para informar su error. Siempre su programa deverá verificar el código de reportaje después de leer o grabar con USR y tomar las acciones necesarias, si el código no indica lo que se espera.

El ejemplo anterior, puede ser mejorado así:

```
510 LET Z$ = "T"
520 LET STATUS =
USR (8305)              Leer la cinta
530 IF STATUS =
0 THEN GOTO 600         Condición normal
540 IF STATUS =
1 THEN GOTO 570         Fín de los datos
550 PRINT "ERRORES"
STATUS                  Otras condiciones
560 STOP
570 PRINT "FÍN DE
LA CINTA"
580 STOP
600 . . .
```

Si su programa tuviera un error, el imprimir el código de reportaje, lo ayudará a entender lo que salió errado. Ud. podrá inpeccionar STATUS por el comando PRINT, dado directamente desde el teclado, después que el programa se haya detenido. Talvez eso sea adecuado, mientras Ud. aún esté armando un programa. La solución más adecuada sería examinar el error detectado en una sub-rutina. Si Ud. tuviera muchos comandos para leer y escribir en su programa, eso le ayudará a guardar repetitivos e inútiles códigos.

Volviendo al uso del Z$, si Ud. tuviera un programa que use solamente un buffer, no es errado poner Z$ en el comienzo del programa, así, será ejecutado sólo una vez.

```
100 DIM A$ (500)
110 LET Z$ = "A"
    . . .
500 . . .
520 LET STATUS = USR 8305
    . . .
900 GOTO 500
```

Es importante percibir, que es necesario tener varios buffers de entrada y de salida, al mismo tiempo que se puede aprovechar un buffer único para entrada, así como para salida.

Aunque la extensión de un buffer esté fijada, es ventajoso tener condición de variar la cantidad de los datos a ser leídos.

El tamaño del bloque leído de la cinta, también puede ser diferente del tamaño del buffer, en el cual Ud. esté intentando colocar la información.

Por eso, necesitamos tener algunos medios para indicar el número de caracteres, que serán realmente transferidos.

ESPECIFICANDO LA EXTENSIÓN DEL BLOQUE DE DATOS.

Cuando un buffer es dimensionado, por ejemplo:

```
100 DIM A$ (500)
```

El tamaño referido representa el número máximo de bytes de datos, que deberán ser transferidos para adentro o para afuera del buffer. Para especificar el número real de bytes que van a ser transferidos en una operación, usamos otras variables reservadas, en esta ocasión, sólo una variable numérica, Z. Cuando se desee grabar un bloque en la cinta, Ud. necesitará primero atribuir a Z, el número de bytes, que desea transmitir, comenzando siempre con el primer byte del buffer.

Ejemplo:

```
100 DIM P$ (133)
    . . .
    . . .
650 LET P$ =
"ESTO ES UN EJEMPLO"
600 LET Z = 17          (largo incluyendo
                        espacios).
710 LET Z$ = "p"
720 LET STATUS =
USR DSAVE               Graba en la cinta.
```

En el ejemplo anterior, aunque el tamaño del buffer es 133 bytes, solamente 17 caracteres habrán salido para el grabador. En realidad esto es sólo un ejemplo, ya que la cantidad mínima de bytes enviados del buffer es 40. Si trata con un número más pequeño, indicará sutuación irregular.

Al leer de la cinta, el TK 85 coloca automáticamente el tamaño del bloque que fué leído en Z. Como la USR no puede crear variables, Ud. necesitará incluir algunas referencias para Z, antes de efectuar una lectura, con el fín de garantizar que ella esté disponible para que el TK 85 la escriba. Esto puede ser hecho con el parámetro LET, atribuyéndole cualquier valor a Z antes de su primer comando de lectura.

```
100 DIM TZ (500)
110 LET Z = 0           Creando Z en la lista de
                        variables
    . . .
510 LET Z$ = "T"
520 LET STATUS =
USR 8305                Leer la cinta
530 IF STATUS =
0 THEN STOP             Si diera error
540 REM — Z AHORA CONTIENE EL LARGO
DEL BLOQUE.
```

Si el tamaño del bloque leído fuera más grande que el buffer que Ud. dimensionó, será atribuido a Z el mismo tamaño del buffer y un código de reportaje será devuelto, indicando que hay más datos que deberían ir en el buffer. Dado que Z es usada en toda transacción de entrada y salida, Ud. podrá copiar su contenido para otra variable, si necesitara preservar el bloque para uso posterior. Acuérdese que la próxima operación de entrada y salida, irá a escribir el contenido de Z.

BACK-UP.

Las cintas se desgastan después de un largo uso, o pueden ser damnificadas por accidente. Aunque su cinta esté buena, su programa puede no estarlo, o Ud. puede mandar ejecutar un trabajo, para darse cuenta después que usó datos errados, eso puede sonar excesivamente pesimista, pero en estos casos, ni siquiera el computador puede corregir errores humanos, así, Ud. necesitará asegurarse.

Los archivos de datos, deben ser alterados cada cierto tiempo. Posiblemente Ud. querrá agregar nuevos nombres y domicilios a su archivo y eliminar fichas de personas que no deben ser incluidas. Con el propósito de facilitar la búsqueda de fichas, Ud. probablemente las archiva por orden alfabético, de manera que nuevos nombres puedan ser "encajados" en sus posiciones correctas en el archivo, en lugar de simplemente colocarlas al final del mismo. Aunque fuese posible rebobina la cinta, Ud. comprenderá que es imposible abrir espacios para que puedan ser aceptadas nuevas entradas.

Entonces, el camino correcto para actualizar un archivo sería, no el de escribir sobre la cinta anterior, sino crear una nueva cinta, copiar el material que no será alterado, e incorporar los cambios requeridos a medida que la cinta avanza.

Resumiendo, Ud. tiene una cinta que contiene la copia "master" de su archivo. Para efectuar correcciones, Ud. deberá escribir un programa que permita la actualización. Éste deberá tener condiciones de leer y escribir fichas, pero también deberá permitir otras adiciones e dejar sin efecto fichas a través del teclado, y avisar a través de mensajes, la medida a tomar con el grabador (conectar o desconectar). Observe que Ud. podrá tener un grabador, para permitir la lectura de la cinta anterior, al mismo tiempo que puede ser conectado otro grabador, donde está siendo construido el nuevo archivo.

Su programa de actualización, debería determinar la "Administración" de estos grabadores.

Él podría ver la posición de una ficha por vez, o posiblemente leer el archivo automaticamente, hasta encontrar la posición apropriada, para la próxima alteración que Ud. quiera hacer. Eso sería hecho por ejemplo, comparando el nombre de su nueva ficha con el nombre en cada ficha, a medida que éste las lee.

Un método que podrá ser usado, es tener un índice general en el comienzo de la cinta. Para que el computador pueda determinar entre cuales de esas fichas deberá entrar la nueva. Otros métodos son posibles.

Después de una sesión de actualización Ud. tendrá dos (2) cintas, la que se torna su nueva cinta master y la original, referida como la "vieja master".

No descarte todavía la "vieja master". Ud. no tiene certeza que su nueva cinta esté buena. Puebe haber acontecido cualquier problema con la grabación o con la cinta. Es recomendable muchas veces, operar con tres generaciones de archivos masters, conocidos como "abuelo", "padre" e "hijo" por su aparición. Si Ud. considera que la última generación del archivo está mala, vuelva a la generación anterior y adiciona las últimas alteraciones.

GRABANDO EN LA CINTA (DSAVE).

El comando para grabar en la cinta DSAVE es obtenido con una sentencia del tipo:

```
LET CCODE = USR (8288)
```

Ya hemos visto en el capítulo anterior, las diferentes medidas que necesitan ser tomadas antes de poder ser dado el comando.
La secuencia completa del evento es:

1. Definir el buffer de salida

```
100 DIM W$ (300)
```

2. Cargar los datos que serán escritos en el buffer.

3. Definir en Z el número de caracteres que Ud. quiere grabar,

```
280 LET Z = 240
```

4. Apuntar el buffer vía Z$

```
300 LET Z$ = "W"
```

5. Conectar el grabador.

6. Dar el comando DSAVE.

```
320 LET CC = USR 8288
```

7. Cambiar el TK 85 para el modo SLOW

```
330 SLOW
```

8. Analizar el código de reportaje y tomar las medidas apropiadas:

```
340 IF CC < > 0 THEN STOP
```

Observe que si Ud. quiere que su programa vuelva al estado normal, deberá dar el comando SLOW, o sino, que su programa corra en FAST, no haga nada.

LEER LA CINTA (DLOAD).

El comando para leer la cinta, DLOAD, es obtenido con una sentencia del tipo:

```
LET CODE = USR 8305
```

La secuencia completa de la operación es:

1. Definir el buffer de entrada

```
110 DIM R$ (400)
```

2. Crear Z como variable

```
120 LET Z = 0
```

3. Apuntar el buffer, vía Z$

```
400 LET Z$ = "R"
```

4. Conectar el grabador.

5. Dar comando DLOAD

```
420 LET CC = USR 8305
```

6. Volver a SLOW

```
430 SLOW
```

7. Análisis de los códigos de respuesta

```
440 IF CC = 1 THEN
GOTO 1000          Prueba para fín de
                   archivo.
450 IF CC <> 0 THEN STOP ERROR
```

8. Preservar el recuento de los datos de entrada, si Z fuera usado antes de haber terminado el recuento.

```
460 LET INLEN = Z
```

9. Finalmente haga lo que Ud. desee con los datos que acaban de ser leídos.

```
1000 . . . PRINT INLEN
```

VERIFICACIÓN DEL BLOQUE (DVERIFY).

Esta sentencia es muy útil, especialmente cuando es digitada directamente del teclado. Es similar al comando HI-VERIFY.

En realidad, ella actúa exactamente como si estuviese leyendo la cinta, pero no almacena los datos en RAM. Verifica los datos y devuelve un código de reportaje, que indicará si ocurrió algún error. Ella también puede ser usada para verificar la validez de la cinta de datos, sin destruir el contenido del RAM, y por lo tanto, beneficia el desarrollo del programa, ya que le permite a Ud. determinar si la información fué almacenada correctamente, sin destruir el original, que le dió tanto trabajo para construir.

Dado que DVERIFY no almacena informaciones en la RAM, él no utiliza buffer y por lo tanto, es más simple de usar que DSAVE o DLOAD.

La secuencia es simple:

1. Conecte el grabador

2. Mande ejecutar el comando DVERIFY, con el siguiente tipo de sentencia.

```
LET CC = USR 9816
```

AGRUPANDO FICHAS EN BLOQUES.

Cada vez que se graba una información en cinta, antes es insertado un espacio de 5 segundos, siendo por lo tanto un desperdicio escribir pequeños bloques, ya que Ud. podrá eliminar más espacios que datos. Una opción es agrupar varias fichas y grabarlas como un bloque único. Cuando esto fuera hecho, las pequeñas fichas son llamadas de "fichas lógicas", mientras que el bloque que las agrupa es llamado de "ficha-física".

Agregar fichas, implica que su programa sea más complejo. Una forma de trabajar de este modo, es crear diferentes buffers, donde cada uno de los cuales puede representar una "ficha lógica".

Este esfuerzo tiene validez, si Ud. desea almacenar el máximo de datos y procesarlos a la velocidad máxima.

CÓDIGO DE REPORTAJE DE LAS FUNCIONES DE I/0.

Los códigos de reportaje (C.R.) le indican con respecto a fallas, por ejemplo, en el proceso de lectura. El último CR deberá ser 1, indicando que no fué posible encontrar datos en 15 segundos, lo que es interpretado como fín del bloque de datos. Si todos los C.R. precedentes son cero, entonces no hay ningún problema, ya que todos los datos fueron leídos correctamente.

Se hace notar que los códigos descriptos a continuación, son exclusivos de las funciones I/0.

Los siguientes C.R. son reproducidos directamente en la parte inferior izquierda del video.

C.R.	Indicación.
D	Presiona la tecla BREAK durante algunas de las funciones de 1/0.
G	Espacio insuficiente de memoria en el área de trabajo temporal.
H	Especificación del largo en la variable Z indefinido o incorrectamente definido. Vea más detalles en los C.R. numéricos.

I El indicador de buffer Z$, es indefinido o incorrectamente definido, o el buffer indicado es indefinido e incorrectamente definido. Vea más detalles en los C.R. numéricos.

Ø Sin errores. Ejecución completada.

1 Fué hecha la lectura por más de 15 segundos, pero no fué detectada ninguna información. Esta es la condición normal de fín-de-datos, pero también podría haber sido causada, porque el grabador fué desconectado, o el grabador no tenía cinta, el PLAY del grabador na estaba conectado, o el cable entre el TK 85 (EAR) y el grabador estaba desconectado, etc.

2 Fué detectada información incoherente en el área de variables.

3 Z$, una variable reservada, está indefinida. Ella debería ser una STRING simple, en la cual el primer caracter es la letra del nombre del buffer indicado.

4 Z$ esta definido como una matriz (array) STRING, y en este caso, él debería ser definido como un simple STRING.

5 Z$ está definida, pero tiene longitud cero.

6 El primer caracter de Z$ no es una letra entre A-Y.

7 No fué encontrado Buffer con el nombre definido en Z$.

8 La STRING identificado como buffer (vía Z$) es una STRING simple y deberá ser definida como una matriz (array) STRING. Esto quiere decir, que deberá ser definida por un comando DIM.

9 La STRING identificada como buffer (vía Z$) es una matriz multidimensional y debe ser una matriz del tipo DIM A$ (500).

10 La variable reservada Z, está indefinida. Ella es usada para indicar el largo de datos que serán grabados, y definida por un comando LET.

11 Z está definida, pero su valor está fuera de la zona y no es un número entero, debe ser positivo entre Ø y 65535.

12 El valor de Z, es más grande que el largo del buffer de entrada o salida (identificación vía Z$).
Si el valor de Z parece correcto, verifique que Z$ esté apuntando para el buffer correcto.

C.R. Indicación

13 Ud. trató de transferir un bloque menor de 40 bytes, que es el límite inferior permitido.

14 El bloque leído en la cinta fué más grande que el buffer de entrada. Los datos fueron cargados en el buffer hasta que él se completa y el resto del bloque fué ignorado.

15 Una operación fué interrumpida por el uso de la tecla BREAK.

16-22 Estos códigos indican errores detectados al leer la cinta. Indican diferentes combinaciones de tipos de errores.

	TIPO A	TIPO B	TIPO C
16	V	—	—
17	—	V	—
18	V	V	—
19	—	—	V

	TIPO A	TIPO B	TIPO C
20	V	—	V
21	—	V	V
22	V	V	V

Tipo "A" — Posiblemente nivel bajo del volúmen del grabador.

Tipo "B" — Fluctuación en el nivel de lectura.

Tipo "C" — Posiblemente nivel muy alto del volúmen del grabador.

Además de los seis comandos ya descriptos en este capítulo, fueron incluidos dos más: DHSAVE y DHLOAD, cuyas características son semejantes a los dos mencionados anteriormente.

DSAVE y DLOAD almacenan los datos en el formato I, los nuevos comandos DHSAVE y DHLOAD, efectúan la misma tarea, pero en formato II (HIGH--SPEED).

Los procedimientos de operación son idénticos, y sólo se diferencian en la tabla de códigos de reportajes especiales, donde no será reportado del código 17 en adelante. El código 16 indicará que la lectura fué hecha con error en la información.

Para verificación en su grabación, defina un "buffer" específico para esa función.

DHSAVE — PRINT USR 9008
DHLOAD — PRINT USR 9189

APÉNDICES

a·b·c

APÉNDICE A
El conjunto de caracteres

Este es el conjunto completo de caracteres del TK 85, con códigos en decimal y hexadecimal. Imaginando los códigos en forma de lenguaje de máquina del Z80, vamos a encontrar los mnemónicos correspondientes en la columna de la derecha. Como Ud. debe saber, algunas instrucciones del Z80 son compuestas, comenzando con CBh e EDh, como se puede verificar en las dos columnas de la derecha.

Código	*Caracter*	*Hexa-decimal*	*Z80 assembler*	*DESPUÉS CB*	*Después de ED*
0	espacio (space)	00	nop	ric b	
1	▘	01	ld bc,NN	ric c	
2	▝	02	ld (bc),a	ric d	
3	▀	03	inc bc	ric e	
4	▖	04	inc b	ric h	
5	▌	05	dec b	ric l	
6	▞	06	ld b,N	ric (hl)	
7	▛	07	rlca	ric a	
8	▗	08	ex af,af'	rrc b	
9	▚	09	add hl,bc	rrc c	
10	▐	0A	ld a,(bc)	rrc d	
11	"	0B	dec bc	rrc e	
12	£	0C	inc c	rrc h	
13	$	0D	dec c	rrc l	
14	:	0E	ld,c,N	rrc (hl)	
15	?	0F	rrca	rrc a	
16	(	10	djnz DIS	rl b	
17	)	11	ld de,NN	rl c	
18	>	12	ld (de),a	rl d	
19	<	13	inc de	rl e	
20	=	14	inc d	rl h	
21	+	15	dec d	rl l	
22	–	16	ld d,N	rl (hl)	
23	*	17	rla	rl a	
24	/	18	jr DIS	rr b	
25	;	19	add hl,de	rr c	
26	,	1A	ld a,(de)	rr d	
27	.	1B	dec de	rr e	
28	0	1C	inc e	rr h	

Código	Caracter	Hexa-decimal	Z80 assembler	Después CB	Después ED
29	1	1D	dec e	rr l	
30	2	1E	ld e,N	rr (hl)	
31	3	1F	rra	rr a	
32	4	20	jr nz,DIS	sla b	
33	5	21	ld hl,NN	sla c	
34	6	22	ld (NN),hl	sla d	
35	7	23	inc hl	sla e	
36	8	24	inc h	sla h	
37	9	25	dec h	sla l	
38	A	26	ld,h,N	sla (hl)	
39	B	27	daa	sla a	
40	C	28	jr z,DIS	sra b	
41	D	29	add hl,hl	sra c	
42	E	2A	ld hl,(NN)	sra d	
43	F	2B	dec hl	sra e	
44	G	2C	Inc l	sra h	
45	H	2D	dec l	sra l	
46	I	2E	ld,l,N	sra (hl)	
47	J	2F	cpl	sra a	
48	K	30	jr nc,DIS		
49	L	31	ld sp,NN		
50	M	32	ld (NN),a		
51	N	33	inc sp		
52	O	34	inc (hl)		
53	P	35	dec (hl)		
54	Q	36	ld (hl),N		
55	R	37	scf		
56	S	38	jr c,DIS	srl b	
57	T	39	add hl,sp	srl c	
58	U	3A	ld a,(NN)	srl d	
59	V	3B	dec sp	srl e	
60	W	3C	inc a	srl h	
61	X	3D	dec a	srl l	
62	Y	3E	ld, a,N	srl (hl)	
63	Z	3F	ccf	srl a	
64	RND	40	ld b,b	bit 0,b	in b,(c)
65	INKEY$	41	ld b,c	bit 0,c	out (c),b
66	PI	42	ld b,d	bit 0,d	sbc hl,bc
67		43	ld b,e	bit 0,e	ld (NN),bc
68		44	ld b,h	bit 0,h	neg
69		45	ld b,l	bit 0,l	retn
70		46	ld b,(hl)	bit 0,(hl)	im 0
71		47	ld b,a	bit 0,a	ld i,a
72		48	ld c,b	bit 1,b	inc c,(c)
73		49	ld c,c	bit 1,c	out (c),c
74		4A	ld c,d	bit 1,d	adc hl,bc
75		4B	ld c,e	bit 1,e	ld bc,(NN)
76	no usado	4C	ld c,h	bit 1,h	
77		4D	ld c,l	bit, 1,l	reti
78		4E	ld c,(hl)	bit 1,(hl)	
79		4F	ld c,a	bit 1,a	ld r,a
80		50	ld d,b	bit 2,b	in d,(c)
81		51	ld d,c	bit 2,c	out (c),d
82		52	ld d,d	bit 2,d	sbc hl,de
83		53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	

Código	*Caracter*	*Hexa-decimal*	*Z80 assembler*	*Después CB*	*Después ED*
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	ld a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl, de
91		5B	ld e,e	bit 3,e	ld de,(NN)
92		5C	ld e,h	bit 3 h	
93		5D	ld e,l	bit 3,l	
94		5E	ld e,(hl)	bit 3,(hl)	im 2
95		5F	ld e,a	bit 3,a	ld a,r
96		60	ld h,b	bit 4,b	in h,(c)
97		61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99	no usado	63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	cursor p/ arriba	70	ld (hl),b	bit 6,b	
113	cursor p/ abajo	71	ld (hl),c	bit 6,c	
114	cursor p/ izquierda	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor p/ derecha	73	ld (hl),e	bit 6,e	ld (NN),sp
116	GRAPHICS	74	ld (hl),h	bit 6,h	
117	EDIT	75	ld (hl),l	bit 6,l	
118	NEWLINE	76	halt	bit 6,(hl)	
119	RUBOUT	77	ld (hl),a	bit 6,a	
120	/ modo	78	ld a,b	bit 7,b	in a,(c)
121	FUNCTION	79	ld a,c	bit 7,c	out (c),a
122	no usado	7A	ld a,d	bit 7,d	adc hl,sp
123	no usado	7B	ld a,e	bit 7,e	ld sp,(NN)
124	no usado	7C	ld a,h	bit 7,h	
125	no usado	7D	ld a,l	bit 7,l	
126	número	7E	id a, (hl)	bit 7,(hi)	
127	cursor	7F	ld a,a	bit 7,a	
128	■	80	add a,b	res 0,b	
129	▟	81	add a,c	res 0,c	
130	▙	82	add a,d	res 0,d	
131	▄	83	add a,e	res 0,e	
132	▛	84	add a,h	res 0,h	
133	▌	85	add a,l	res 0,l	
134	▚	86	add a,(hl)	res 0,(hl)	
135	▜	87	add a,a	res 0,a	
136	▒	88	adc a,b	res 1,b	
137	▒	89	adc a,c	res 1,c	
138	▒	8A	adc a,d	res 1,d	
139	inverso "	8B	adc a,e	res 1,e	
140	inverso £	8C	adc a,h	res 1,h	
141	inverso $	8D	adc a,l	res 1,l	

Código	*Caracter*	*Hexa-decimal*	*Z80 assembler*	*Después CB*	*Después ED*
142	inverso :	8E	adc a,(hl)	res 1,(hl)	
143	inverso ?	8F	adc a,a	res 1,a	
144	inverso (	90	sub b	res 2,b	
145	inverso)	91	sub c	res 2,c	
146	inverso >	92	sub d	res 2,d	
147	inverso <	93	sub e	res 2,e	
148	inverso =	94	sub h	res 2,h	
149	inverso +	95	sub l	res 2,l	
150	inverso —	96	sub (hl)	res 2,(hl)	
151	inverso *	97	sub a	res 2,a	
152	inverso /	98	sbc a,b	res 3,b	
153	inverso ;	99	sbc a,c	res 3,c	
154	inverso ,	9A	sbc a,d	res 3,d	
155	inverso .	9B	sbc a,e	res 3,e	
156	inverso 0	9C	sbc a,h	res 3,h	
157	inverso 1	9D	sbc a,l	res 3,l	
158	inverso 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverso 3	9F	sbc a,a	res 3,a	
160	inverso 4	A0	and b	res 4,b	ldi
161	inverso 5	A1	and c	res 4,c	cpi
162	inverso 6	A2	and d	res 4,d	ini
163	inverso 7	A3	and e	res 4,e	outi
164	inverso 8	A4	and h	res 4,h	
165	inverso 9	A5	and l	res 4,l	
166	inverso A	A6	and (hl)	res 4,(hl)	
167	inverso B	A7	and a	res 4,a	
168	inverso C	A8	xor b	res 5,b	ldd
169	inverso D	A9	xor c	res 5,c	cpd
170	inverso E	AA	xor d	res 5,d	ind
171	inverso F	AB	xor e	res 5,e	outd
172	inverso G	AC	xor h	res 5,h	
173	inverso H	AD	xor l	res 5,l	
174	inverso I	AE	xor (hl)	res 5,(hl)	
175	inverso J	AF	xor a	res 5,a	
176	inverso K	B0	or b	res 6,b	ldir
177	inverso L	B1	or c	res 6,c	cpir
178	inverso M	B2	or d	res 6,d	imir
179	inverso N	B3	or 3	res 6,e	otir
181	inverso O	B4	or h	res 6,h	
181	inverso P	B5	or l	res 6,l	
182	inverso Q	B6	or (hl)	res 6,(hl)	
183	inverso R	B7	or a	res 6,a	
184	inverso S	B8	cp b	res 7,b	lddr
185	inverso T	B9	cp c	res 7,c	cpdr
186	inverso U	BA	cp d	res 7,d	indr
187	inverso V	BB	cp e	res 7,e	otdr
188	inverso W	BC	cp h	res 7,h	
189	inverso X	BD	cp l	res 7,l	
190	inverso Y	BE	cp (hl)	res 7,(hl)	
191	inverso Z	BF	cp a	res 7,a	
192	" "	C0	ret nz	set 0,b	
193	AT	C1	pop bc	set 0,c	
194	TAB	C2	jp nz,NN	set 0,d	
195	no usado	C3	jp NN	set 0,e	
196	CODE	C4	call nz,NN	set 0,h	

Código	Caracter	Hexa-decimal	Z80 assembler	Después CB	Después ED
197	VAL	C5	push bc	set 0,l	
198	LEN	C6	add a,N	set 0,(hl)	
199	SIN	C7	rst 0	set 0,a	
200	COS	C8	ret z	set 1,b	
201	TAN	C9	ret	set 1,c	
202	ASN	CA	jp z,NN	set 1,d	
203	ACS	CB		set 1,e	
204	ATN	CC	call z,NN	set 1,h	
205	LN	CD	call NN	set 1,l	
206	EXP	CE	adc a,N	set 1(hl)	
207	INT	CF	rst 8	set 1,a	
208	SQR	D0	ret nc	set 2,b	
209	SGN	D1	pop de	set 2,c	
210	ABS	D2	jp nc,NN	set 2,d	
211	PEEK	D3	out N,a	set 2,e	
212	USR	D4	call nc,NN	set 2,h	
213	STR $	D5	push de	set 2,l	
214	CHR $	D6	sub N	set 2(hl)	
215	NOT	D7	rst 16	set 2,a	
216	**	D8	ret c	set 3,b	
217	OR	D9	exx	set 3,c	
218	AND	DA	jp c,NN	set 3,d	
219	<=	DB	in a,N	set 3,e	
220	>=	DC	call c,NN	set 3,h	
221	<>	DD	prefijo de instrucciones usando ix	set 3,l	
222	THEN	DE	sbc a,N	set 3,(hl)	
223	TO	DF	rst 24	set 3,a	
224	STEP	E0	ret po	set 4,b	
225	LPRINT	E1	pop hl	set 4,c	
226	LLIST	E2	jp po,NN	set 4,d	
227	STOP	E3	ex (sp),hl	set 4,e	
228	SLOW	E4	call po,NN	set 4,h	
229	FAST	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4,(hl)	
231	SCROLL	E7	rst 32	set 4,a	
232	CONT	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GOTO	EC	call pe,NN	set 5,h	
237	GOSUB	ED		set 5,l	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RAND	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	

Código	*Caracter*	*Hexa-decimal*	*Z80 assembler*	*Después CB*	*Después ED*
251	CLS	FB	ei	set 7,e	
252	UNPLOT	FC	call m,NN	set 7,h	
253	CLEAR	FD	prefijo de instrucciones usando iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	

APÉNDICE B
Códigos indicadores

Esta tabla proporciona cada uno de los códigos indicadores del computador, con una descripción general y una lista de sentencias y funciones en las cuales estos pueden ocurrir. En el Apéndice C, bajo cada sentencia o función, Ud. podrá encontrar una descripción más detallada de lo que significan los mensajes de error.

Código	Significado	Situaciones
Ø	Ejecución con éxito o salto para una línea de mayor número que cualquiera otra existente. Una indicación Ø no altera el número de línea usado por CONT.	Cualquiera
1	La variable de control no existe, (no fue establecida por una sentencia FOR) pero existe otra variable con el mismo nombre.	NEXT
2	Fué utilizado una variable indefinida. En el caso de una variable simple, esto ocurrirá si esta fuera usada antes de haber sido asignada por una sentencia LET. Para variables con subíndice, esto acontecerá si éstas fueran usadas antes de ser dimensionadas por la sentencia DIM. Y para una variable de control, esto ocurrirá en e caso que ella sea usada antes de haber sido definida como variable de control por una sentencia FOR y si no hubiera ninguna variable simple con el mismo nombre.	Cualquiera
3	Subíndice fuera de la zona (o sea, negativo o arriba de 65535) en ese caso ocurrirá error B.	Variables con subíndice.
4	Espacio insuficiente en la memoria. Observe que el número de línea en la indidicación (después de /) podrá estar incompleto en la pantalla, por falta de memoria; así por ejemplo, 4/2Ø podrá aparecer como 4/2. Vea el capítulo 23.	LET, INPUT, DIM, PRINT, LIST, PLOT, UMPLOT, FOR, GOSUB. (A veces, durante el cálculo de una función).
5	No hay más espacio en la pantalla. CONT creará espacio, borrando la pantalla.	PRINT, LIST.
6	Sobrecarga aritmética: los cálculos producirán un número superior a 10^{38}.	Cualquier cálculo aritmético.
7	RETURN sin un GOSUB correspondiente.	RETURN.
8	Ud. intentó un comando INPUT no permitido.	INPUT.
9	Sentencia STOP ejecutada. CONT no tratará de volver a ejecutar la sentencia STOP.	STOP.
A	Argumento inválido para ciertas funciones.	SQR, LN, ASN, ACS.
B	Número entero fuera de zona. Cuando un número	RUN, RAND, POKE, DIM,

	entero es requerido, el argumento de punto fluctuante es redondeado para el número entero más próximo. En caso que esté fuera de la zona adecuada, se produce error B. Para acceso array, vea también indicador 3.	GOTO, GOSUB, LIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR, LLIST. acesso a array
C	El texto del argumento (de string) de VAL no forma una expresión numérica válida.	VAL.
D	1. Programa interrumpido por BREAK.	al fínal de cualquier sentencia o en LOAD, SAVE, LPRINT, LLIST o COPY.
	2. La línea INPUT comienza con STOP.	INPUT.
E	No utilizado.	
F	El nombre proporcionado por el programa es una string vacía.	SAVE.

APÉNDICE C
El TK85 para los que ya entienden Basic

INTRODUCCIÓN.

Si Ud. ya conoce el lenguaje BASIC, no debería encontrar problemas al utilizar el TK 85. Sin embargo, el exhibe algunas peculiaridades:

1. Las palabras no son deletreadas — sino que poseen sus propias teclas, como se puede verificar consultando los capítulos 2 y 5. A lo largo del texto de este Manual, tales palabras aparecen subrayadas.
2. El TK 85 no posee READ, DATA y RESTORE en su BASIC, ni funciones definidas por el usuario (FN y DEF, a pesar de poderse utilizar VAL, algunas veces), ni líneas de sentencias múltiples.

3. Los recursos para tratar con strings son fáciles de comprender, pero no son padrón — ver los capítulos 21 y 22.
4. El conjunto de caracteres del TK 85 es exclusivo.

5. La pantalla de la televisión, no está planificada en la memoria.

6. Si Ud. está acostumbrado a utilizar PEEK y POKE en otros computadores, acuérdese que todas las direcciones serán diferentes para el TK 85.

VELOCIDAD.

El TK 85 trabaja en una velocidad llamada SLOW.

En la velocidad llamada slow (lenta), la imagen de la TV es generada continuamente, mientras la computación es ejecutada durante los períodos de borrado, en la parte superior e inferior de la imagen.

En la modalidad fast (rápida), la imagen de la TV es desconectada durante el procesamiento y presentada sólo al fín de un programa, mientras está esperando por datos de entrada o durante una pausa (vea PAUSE).

La modalidad fast, es aproximadamente 4 veces más rápida que slow y es de gran utilidad, especialmente en programas que exigen una gran cantidad de procesamiento o extremadamente largos.

TECLADO.

Los caracteres del TK 85 engloban no sólo los símbolos simples (letras, dígitos, etc) sino tambien las señales compuestas (palabras-llave, nombre de funciones, etc, siempre subrayadas, a lo largo del texto) y todos son introducidos a través del teclado. Para hacer posible esto, se llegó a atribuir hasta 6 significados distintos a algunas teclas, diferenciados en parte por el uso de la tecla SHIFT y en parte por la utilización de la máquina en diferentes modalidades.

La modalidad es indicada por el cursor, una letra en video invertido que indica donde será inscripto el caracter siguiente, insertado por el teclado.

La modalidad **K** (para palabras-llaves) aparece automáticamente cuando la máquina está a la espera de un comando o línea de programa y a través de su posición en la línea, sabe que debe esperar un número de línea o una palabra-llave. Esto en el inicio de la línea, o algunos dígitos después del inicio, o aún, después del THEN. Si no estuviera combinada con la tecla SHIFT, la tecla siguiente será interpretada como una palabra-llave (en esto caso, escritas encima de las teclas) o como un dígito.

La modalidad **L** (para letras) se produce normalmente a toda hora. Cuando no está acompañada por la tecla SHIFT, la tecla siguiente será interpretada como el símbolo principal contenido en ella misma.

En las modalidades **K** y **L**, una tecla combinada con SHIFT será interpretada como el caracter señalado en rojo, de la esquina superior derecha de la misma.

La modalidad **F** (para funciones) aparece siempre después que se pulsa la tecla FUNCTION (SHIFT + NEWLINE) y permanece solamente mientras se pulsa una tecla. Esta tecla será interpretada como un nombre de la función, que aparece escrita debajo de cada tecla.

La modalidad **G** (para gráficos) se produce siempre después de pulsar la tecla GRAPHICS (SHIFT + tecla 9) y perdura hasta que es presionada nuevamente. Una tecla no acompañada de SHIFT, proporcionará video invertido en su interpretación de la modalidad **L**.

Una tecla combinada con SHIFT hará lo mismo, siempre que sea un símbolo; pero, si esa tecla combinada proporciona normalmente una señal compuesta, en la modalidad gráfica proporcionará el símbolo para gráficos, en la esquina inferior derecha de la tecla.

LA PANTALLA.

Posee 24 líneas, cada una con 32 caracteres y es dividida en 2 partes. La porción superior llega como máximo a 22 líneas y exhibe tanto los listados como la salida de los programas; la porción inferior, que ocupa sólo 2 líneas, es usada para introducir comandos, líneas de programas y datos, además de presentar a los indicadores.

Entrada vía teclado: aparece en la mitad inferior de la pantalla, a medida que cada caracter es digitado (símbolos simples o compuestos) siempre un poco a la izquierda del cursor. Este puede ser movido para la izquierda a través de ⇦ (SHIFT + 5) y para la derecha por ⇨ (SHIFT + 8); y el caracter localizado después del cursor puede ser removido a través de RUBOUT (SHIFT

Por supuesto que la línea entera puede ser eliminada al asociar EDIT (SHIFT + 1), seguida de NEWLINE.

Siempre que NEWLINE es presionada, la línea es ejecutada, introducida en el programa o empleada como dato de entrada, a menos que contenga un error de sintáxis, en ese caso aparece el símbolo S antes.

A medida que son introducidas las líneas de programación, va siendo impresa una lista en la parte superior de la pantalla. La manera por la cual ésta lista es introducida, está explicado en detalle en el ejercicio 6 del capítulo 9.

La última línea a ser introducida es denominada *línea corriente,* siendo indicada por el símbolo > ; sin embargo, eso puede ser cambiado utilizándose las teclas ⇩ (SHIFT + 6) y ⇧ (SHIFT + 7). Si fuera accionada EDIT, la línea actual puede ser transferida para la parte inferior de la pantalla y ahí se edita.

Cuando un comando es ejecutado o un programa es rodado, la salida de los datos aparece en la parte superior de la pantalla y ahí permanece hasta que sea digitada una línea de programa, o hasta que sea accionado NEWLINE con una línea vacía, o hasta que las teclas ⇩ o ⇧ sean presionadas. En la parte inferior aparece una indicación bajo la forma m/n, donde m es un código que muestra lo que ocurrió (vea apéndice B), mientras que n representa el número de la última línea que fué ejecutada — o es Ø para el caso de comandos. La indicación permanece hasta que una nueva tecla sea accionada (y aparezca la modalidad K).

En determinadas circunstancias, la tecla SPACE actúa como BREAK, parando el computador con indicación D.

Esto es reconocido:

1. Al final de una sentencia, mientras el programa está rodando;
2. Cuando el computador está buscando un programa en la cinta;
3. Cuando el computador está utilizando la impresora (o, accidentalmente, tratará de usarla cuando no está conectada).

EL BASIC.

Los números son almacenados con una precisión de 9 ó 10 dígitos. El número mayor que se puede obtener en el TK 85 es aproximadamente 10^{38}, y el menor de ellos (positivo) es de $4*10^{-39}$.

En el TK 85, los números pueden ser almacenados en punto fluctuante, con un byte de exponente (y, donde $1 \leqslant e \leqslant 255$) y cuatro bytes de mantisa (m, donde $1/2 \leqslant m \leqslant 1$), lo que representa el número $m*2^{e-128}$.

Ya que $1/2 \leqslant m \leqslant 1$, el bit más significativo de la mantisa es siempre 1. Siendo así, podemos cambiarlo por un bit para representar la señal — Ø para números positivos, 1 para negativos.

El Ø ganó una representación especial, en la cual todos, los 5 bits son Ø. Las variantes numéricas reciben nombres de tamaños arbitrarios, comenzando con una letra y continuando con letras y dígitos. Los espacios son ignorados.

Las variables de control de loops FOR-NEXT tienen nombres de sólo una letra. Los arrays numéricos también exigen nombres de una sola letra, que pueden ser los mismos de una variable simple. Pueden exhibir arbitrariamente, varias dimensiones, de tamaño arbitrario. Los subíndices comienzan en 1. Las strings son completamente flexibles en tamaño. El nombre de una string, consiste en una letra seguida por $.

Los arrays tipo string, también pueden presentar arbitrariamente las más variadas dimensiones, de tamaño arbitrario. Su nombre es compuesto por una letra única seguida de $ y puede no ser el mismo nombre de una string simple. Todas las *string* de una *array* poseen el mismo tamaño fijo, especificado como una dimensión final de la sentencia DIM. Las subíndices comienzan en 1.

Slicing: Las sub-strings de strings pueden ser especificadas por el uso de operación de slicing. Tales operaciones pueden ser:

1. Vacías
2. expresiones numéricas
3. expresión numérica opcional, siendo usada para expresar una sub-string por:

   a. expresión de string (operación de slicing) o

   b. variable de array string (subíndice, . . ., subíndice, formato) que significa lo mismo que variable array string (subíndice, . . ., subíndice) (formato).

En (a), suponga que la expresión de string tenga el valor s$. Si el slicer estuviera vacío, el resultado s$ será considerado una sub-string de si mismo. Si el slicer fuera una expresión numérica con valor m, el resultado será el emésimo caracter de s$ (una sub-string de longitud 1).

Si el slicer posee la forma (iii), suponga que la primera expresión numérica tenga el valor m y la segunda, n. Si $1 \leqslant m \leqslant n \leqslant$ Ø el tamaño de s$, el resultado es una sub-string de s$, comenzando el nº caracter y terminando en el nº.

Si $Ø \leqslant n \leqslant m$, el resultado es una string vacía. En caso contrario, aparece el indicador de error 3.

La operación de slicing es ejecutada antes que las funciones o que las operaciones sean calculadas, a no ser que los paréntesis determinem lo contrario. Las sub-strings pueden ser asignadas (vea LET).

El argumento de una función no exige paréntesis si es una constante o una variable (posiblemente con subíndices o pasada por una operación de slicing).

Función	Tipo de Operando (X)	Resultado
ABS	Número	Magnitud absoluta
ACS	Número	arco-coseno en radianes, error A si X no estuviera entre -1 y 1.
AND	Operación binaria, operando de la derecha siempre un número.	
	Operando izquierdo numérico:	A AND B = { A si B = Ø ; Ø si B = Ø
	Operando izquierdo string:	AS AND B = { A$ si B = Ø ; A$ si B = Ø
ASN	Número	Arco-seno em radianes error A si X no estuviera entre -1 y 1.
ATN	Número	Arco-tangente en radianes.
CHR$	Número	El caracter cuyo código es X, redondeado para el número entero más próximo. Error B si X no estuviera en la zona Ø a 255.
CODE	String	El código del 1º caracter en X (o Ø, si X fuera la string vacía).
COS	Números en radianes.	Coseno.
EXP	Número	e^x
INKEYS	Sin operando	Lee el teclado. El resultado es el cáracter que representa (en modo L) la tecla presionada, si esto ocurriera habría string vacía.
INT	Número	Parte entera (siempre redondeada para menos).
LEN	String	Tamaño (longitud).
LN	Número	Logaritmo natural (en la base e). Error A si X Ø.
NOT	Número	Ø si X = Ø, 1 si X = Ø NOT tiene prioridad 4.
OR	Operación binaria Ambos operandos números	A OR B = { 1 si B = Ø ; A si B = Ø Or tiene prioridad 2.
PEEK	Número	El valor del byte de la memoria cuya dirección es X (redondeado para el entero más próximo). Error B si X no estuviera en la zona de Ø a 65535.
PI	Sin operando	π (3,14159265 . . .)
RND	Sin operando	Genera una secuencia pseudo-aleatória de números entre Ø y 1. Ø ≤ RND < 1.
SGN	Número	La señal (Ø,1 ó -1) de X
SIN	Número (en radianes)	Seno
SQR	Número	Raíz cuadrada; error B si X < Ø.
STR$	Número	La string de caracteres que sería mostrada si X fuese impreso.
TAN	Número en radianes	Tangente.
USR	Número	Llama de la sub-rutina en lenguaje de máquina cuya primera dirección es X. Al volver, el resultado es el contenido del par de registradores bc.
VAL	String	Calcula X como si fuese una expresión numérica. Error C si X contiene un error de sintáxis, o proporcionara un valor de string. También son posibles otros errores, dependiendo de la expresión.
—	Número	Negación.

Los siguientes símbolos son operaciones binarias (incluyendo dos números):

+	Adición (en números), o concatenación (en string)
–	Substracción (resta).
*	Multiplicación.
/	División.
**	Elevación a una potencia. Error B si el operando izquierdo fuera negativo.
>, <, ⩽, ⩾, <>, =	Mayor que, Menor que, Menor o igual a, Mayor o igual a, Diferente de, Igual a. Ambos operandos deben ser del mismo tipo. El resultado es un número: 1 si la comparación fuera correcta y Ø si no fuera incorrecta.

Las funciones y operaciones tienen las siguientes prioridades:

Operaciones	Prioridad
Poner subíndices y *slicing*	12
(trocear) todas las funciones excepto NOT y menos unario	11
**	10
Menos unario	9
*, /	8
+, – (– BINÁRIO)	6
=, >, <, <=, <>, >=	5
NOT	4
AND	3
OR	2

SENTENCIAS.

En esta tabla,

∝	representa una sola letra.
V	representa una palabra.
x, y, z	representan expresiones numéricas.
m, n	representan expresiones numéricas que son redondeadas para el entero más próximo.
e	representa una expresión.
f	representa una expresión con valor de *string*
s	representa una sentencia.

Nótese que expresiones arbitrarias son permitidas en cualquier lugar (excepto para el número de línea en el inicio de la sentencia).

Todas las sentencias excepto INPUT, pueden ser usadas tanto como comando como en el programa.

CLEAR Deja sin afecto todas las variables, dejando libre el espacio ocupado por ellas.

CLS Limpia el archivo de la pantalla. (Vea capítulo 27)

CONT Suponiendo que a/b fué la última indicación con no-cero, entonces CONT tiene el efecto
GOTO b si a = 9
GOTO b + 1 si a = 9 (sentencia STOP)

COPY Envía una copia de la pantalla para la impresora, si estuviera conectada; en caso contrario, no hace nada. Indicación D, si BREAK fuera presionado.

DIM ∝ (n_1, . . ., n_k) Deja sin efecto cualquier *array* con el nombre ∝ y define una array de números con k dimensiones n_1, . . ., n_k. Será todos los elementos.

**DIM∝$** ($n_1$, . . ., n_k) Deja sin efecto cualquier *array* de caracteres con k dimensiones (n_1, . . ., n_k) "Blanquea" todos los elementos. Puede ser considerada una *array* de *string* tamaño fijo n_k con k – 1, dimensiones n_1, . . ., n_{k-1}.

Ocurre error 4 si no hay espacio en la memoria para el *array*. Un array es indefinido hasta que sea dimensionado por una sentencia DIM.

FAST Comienza modo FAST, en el cual el display sólo es mostrado al final de la ejecución.

FOR ∝ = x TO Y FOR ∝ = x TO Y STEP 1

FOR ∝ = x TO y STEP z Deja sin efecto cualquier variable ∝ y define una variable de control com valor x, límite e incremento z, y la dirección de LOOP 1 más el número de la línea de sentencia FOR (– 1 si fuera un comando). Verifica si el valor inicial es mayor (si el incremento ⩾ Ø) o menor (si el incremento < Ø) que el límite, y en caso afirmativo, salta a la sentencia NEXT ∝, en el inicio de la línea. Ver NEXT ∝.
Puede aparecer error 4 si no hay espacio en la memoria para la variable de control.

GOSUB n Registra el número de la línea de la sentencia GOSUB en stack. A continuación, actúa como GOTOn. Puede ocurrir error 4 si no hay RETURN's suficientes.

GOTO n Salta para la línea n (o, si no existe, a la primera después de ésta).

IF x THEN's Si x fuera verdadero (≠ cero), entonces es ejecutado S. El formato "IF x THEN número de línea" no está permitida.

INPUT V Se detiene y espera que sea hecha una entrada de datos. El archivo de imagen es impreso en el modo FAST. INPUT no puede ser usado como comando, pués se produciría error 8. Si el primer caracter en una línea de INPUT es STOP, el programa se detiene con indicación D.

LET v = e Atribuye el valor e a la variable v. LET no puede ser omitido. Una variable simple es indefinida, hasta figurar en una sentencia LET o INPUT. Si v es una variable string con subíndice, entonces la atribución es procrusteana: el valor de la *string* es dejada trunca o completada con ceros a la derecha, para tener el mismo tamaño que la variable v.

LIST LIST Ø.

LIST n Hace el listado del programa en la televisión, iniciándose en la línea n, y hace de n la línea corriente. Se producirá error 4 ó 5, si el listado fuera muy largo para la pantalla. CONT hará exactamente lo mismo nuevamente.

LLIST LLIST Ø.

LLIST n Como LIST, pero usando la impresora y no la

TV. Si no hubiera impresora, no hace nada; se detiene con indicación D, si fuera presionada BREAK.

LOAD f Busca el programa llamando f en la cinta y lo carga con sus variables. Si f = " ", entonces carga el primer programa encontrado.
Si fuera presionado BREAK, entonces:
(I) si ningún programa fuera leído en la cinta, se detiene con indicación D y el programa anterior;
(II) si fué leída una parte del programa, entonces ejecuta NEW.

LPRINT ... Como PRINT, pero usando la impresora. Una línea del texto es enviada a la impresora:
(I) Cuando la impresión salta de una línea para la próxima;
(II) Después de un LPRINT que no termina en una coma o punto y coma.
(III) Cuando una coma, o ítem TAB requiere una nueva línea.
(IV) En el final del programa, si hay algo que no fué impreso. En un ítem AT, sólo el número de la columna tiene efecto; el número de la línea es ignorado. Un ítem AT nunca manda una línea de texto para la impresora.
Si la impresora estuviera ausente, no hay ningún efecto.
Se detiene con indicación D, si fuera presionado BREAK.

NEW Inicia el sistema BASIC, dejando sin efecto el programa y las variables y usando la memoria hasta (pero, sin incluir) el byte cuya dirección está en la variable RTP (bytes 16388 y 16389).

NEXT $\propto$ (I) Encuentra la variable de control $\propto$.
(II) Adiciona el incremento a su valor.
(III) Si el incremento ≥ 0 y el valor $>$ límite; o si el incremento < 0 y el valor $<$ límite, entonces salta a la línea de loop.
Error 1 si no hay variable de control $\propto$.

PAUSE n Detiene el computo y exhibe el archivo de imagen de n cuadros (a 60 cuadros por segundo) o hasta que se pulse una tecla.
$0 \leq n \leq 65535$, o entonces, error B.
Si $n \geq 32767$, entonces la pausa no está temporizada y continúa hasta que se presiona otra tecla.

PLOT m, n Hace que el elemento (|m| , |n|) quede negro; mueve la posición de impresión para después del elemento $0 \leq |m| < 63, 0 \leq |n| \leq 43$, o error B.

POKE m, n Escribe el valor n del byte de dirección m.

PRINT El ". . ." es una secuencia de ítem PRINT, separados por coma o punto y coma. Ellos están escritos en el archivo de imagen en la televisión.
La posición (línea y columna) donde debe ser impreso el próximo caracter es llamada de posición de impresión.
Un ítem PRINT puede ser:
(I) Vacío, esto significa, nada;
(II) Una expresión numérica.
Primero es impreso un signo menos, si el valor es negativo. En seguida, se atribuye a X el módulo del valor.
Si X es $\leq 10^{-5}$ ó $X \geq 10^{13}$, entonces la impresión usa notación científica. La mantisa tiene hasta 8 dígitos y el punto decimal (ausente en caso de un dígito) está después del primero. El exponente es E, seguido de + ó −, seguido de más de 2 dígitos. En caso contrario, X es impresa en notación decimal ordinaria, de hasta 8 dígitos significativos y sin ceros después del punto decimal. Un punto decimal en el inicio, es siempre seguido de un cero.
0 es impreso con un único dígito 0.
(III) Una impresión *string*.
Las marcas en las *string* son expandidas, probablemente con un espacio antes o después. El caracter de comillas es impreso como ". ¿ Caracteres no usados y caracteres de control son impresos como?.
(IV) AT m, n
La posición de impresión es cambiada para la línea m (contando desde el principio), columna n (contando desde la izquierda), $0 \leq |m| 21, 0 \leq |n| \leq 31$, o entonces, error B.
Si m = 22 ó 23, error 5.
(V) TAB n
n es reducido para módulo 32. Entonces, la posición de la impresión es movida para la columna n, permaneciendo en la misma línea; en este caso, se mueve para la próxima línea. $0 \leq |n| \leq 255$, o entonces, error B.
Un punto y coma entre dos ítem inmobiliza la posición de impresión, de forma que el 2º ítem siga a continuación del primero.
Por otro lado, una coma mueve la posición de impresión como mínimo un lugar; y, después de eso, tantos como fueren necesarios para dejarla en la columna 0 ó 16, introduciendo una nueva línea, si fuere necesario. Al final de una sentencia PRINT, si no termina en coma o punto y coma, será colocada una nueva línea.
Error 4 (fuera de la memoria) se puede producir con 3k o menos memoria.
Error 5 significa que la pantalla está repleta. En ambos casos, la solución es CONT, que limpiará la pantalla y continuará.

RAND RAND 0

RAND n Atribuye valor a la variable del sistema (llamado SEED), usada para generar el próximo valor de RND. Si $n \neq 0$, el SEED recibe el valor n; si n es = 0 le es dado un valor de otra variable del sistema (llamada FRAMES) que cuenta los cuadros impresos hasta ese momento en la televisión, lo que debe ser bastante randómico.

Si n no estuviera en la zona de Ø a 65535, aparecerá error B.

REM . . . Sin efecto, ". . ." puede ser cualquier secuencia de caracteres, excepto NEWLINE.

RETURN Retira el número de la línea del stack de GOSUB y salta para la línea siguiente. Cuando no hay número de línea en el stack, se produce error 7.
Hay errores en el programa; GOSUB no está particularmente balanceada porlos RETURN's.

RUN RUN Ø

RUN n CLEAR, y luego GOTOn

SAVE f Graba el programa y variables en cinta y los llama de f.
Save no debe ser usado en una rutina GOSUB.
Si f fuera una *string* vacía, se produciría error F, lo que no es permitido.

SCROLL Gira el archivo de imagen una línea hacia arriba, perdiendo la línea superior y creando otra abajo.

SLOW El display es mostrado contínuamente de este modo.

STOP Finaliza el programa con indicación 9. CONT reiniciará en la línea próxima.

UNPLOT m,n Como PLOT, pero "blanquea" el elemento de impresión.

SOFTWARE

PROGRAMAS PARA EL COMPUTADOR PERSONAL

TK85

JUEGOS INTELIGENTES

LABERINTO TRIDIMENSIONAL - 16K
Juego en tres dimensiones. El jugador podrá elegir la dificultad del laberinto. Es presentado el mapa del Laberinto durante algunos segundos, y enseguida el jugador deberá moverse en dirección a la salida. El programa presenta la posición del jugador en perspectiva. En cualquier momento es posible pedir auxilio al computador.

TKADREZ I - 16K
Este juego presenta el tablero y las piezas en el video. Permite escoger hasta 6 niveles de dificultad, y el montaje de las piezas en el tablero en la forma deseada, para analizar situaciones y posiciones específicas.

TKADREZ II - 16K
Este juego presenta el tablero y las piezas en el video. Permite escoger hasta 7 niveles de dificultad. El programa proporciona en cualquier momento el listado de las jugadas efectuadas, y almacena en cinta la posición de las piezas. El podrá sugerir jugadas.

JUEGO DE BACK-GAMMON - 16K
Juego clásico de habilidad y suerte, transformado en un excitante juego de video. Este programa presenta el tablero en el video y utiliza el eficiente código de máquina, permitiendo 4 niveles de dificultad de juego. Es jugado con el auxilio de 2 dados.

JUEGO DE DAMAS - 16K
Este programa presenta el tablero y las piezas en el video. Es un juego de fácil operación y al mismo tiempo altamente educacional y competitivo.

CUBO MÁGICO - 16K
Consiste en resolver científicamente uno de los mas populares juegos de la década del 80. El programa permite alinear y mover el cubo en cualquier configuración, posibilitando la visualización en el video de cada movimiento según sea en 1,2 ó 3 dimensiones. Se puede solucionar el cubo con ayuda del computador.

JUEGO DEL TA-TE-TI TRIDIMENSIONAL - 16K
Testee su capacidad. El juego es efectuado en 3 dimensiones o en 3 niveles de dificultad. Altamente competitivo y educacional.

TORRE DE HANOI - 2K
El jugador deberá transferir las cinco argollas de la torre 1 para la torre 3. En ningún momento, podrá colocar una argolla grande sobre una mas chica. El display presenta la situación y el número de movimientos jugados. El juego termina cuando estén en la torre 3 todas las argollas de la torre 1. Se juega transfiriendo una argolla por vez.

JUEGO DE ADIVINAR EL NUMERO - 2K
El computador escoge un número de 4 dígitos en forma casual. El jugador puede realizar 10 jugadas para acertar el número. El computador indica en cada jugada cuales son los dígitos acertados y su respectiva posición.

JUEGO DE LOS PALITOS - 2K
Juego inteligente. Se juega contra el computador la partida de palitos. Se tienen 3 líneas de palitos escogidas en forma casual. El jugador y el computador retiran palitos alternadamente. Pierde el juego el que retira el último palito.

MONOPOLIO (Banco Inmobiliario) - 16K
El MONOPOLIO es uno de los juegos más desafiantes y divertidos en la Administración de recursos. Permite jugar hasta 6 personas.

RALLY - 16K
Emocionante corrida de rally en un laberinto, donde podrán ser testados su habilidad y sus reflejos. Para conseguir su finalidad, Ud. deberá evitar autos-ataque y obstáculos en su trayecto.

TK-MAN - 16K
Juego animado donde deberán ser borrados todos los puntos esparcidos en un laberinto (el programa tiene 15 tipos de laberintos). Ud. será impedido de lograrlo a cualquier costo, por 4 extraterrestres, guardianes del laberinto, que podrán ser combatidos con cargas de rayo-laser.

PARQUE DE LAS PESADILLAS - 16K
Juego de aventuras. Realice su paseo por el parque de la muerte. El guardia del parque lo desafía a competir contra desconocidas fuerzas maléficas. Veamos como es su habilidad, hasta donde podrá llegar.

ESTRATEGIA - 16K
Juego inteligente constituído por una compleja simulación de 4 países compitiendo militar, industrial y económicamente para su sobrevivencia. Pueden jugar hasta 4 personas.

TUTOR DE MATEMATICA - 2K
Programa educacional. Optimo para niños. El computador genera operaciones matemáticas (con los operandos: suma, resta, multiplicación y división). El usuario deberá digitar el resultado correcto de la operación y el computador indicará si la respuesta es correcta o no. Se pueden seleccionar hasta 5 niveles de dificultad.

CALENDARIO - 2K
Dado el mes y el año, el computador mostrará el calendario del mes, con los respectivos días de la semana.

JUEGOS ANIMADOS

DEMOLIDOR - 2K
Juego animado tipo fliperama. El jugador deberá demolir una pared con una bola que se encuentra siempre en movimiento. Existen 9 bolas disponibles.

MARCIANO - 2K
Juego de laser. Un marciano está escondido atrás de un árbol en un bosque. El jugador deberá adivinar donde se encuentra el marciano. Ud. será auxiliado por informaciones del computador.

INVASORES DEL ESPACIO - 16K
Consiste en una flota de naves invasoras extraterrestres, descendiendo en el planeta Tierra. Moviéndose hacia derecha e izquierda, ellas detectan la base de rayos laser terrestre, tratando de extinguirla. Su misión es destruir las naves invasoras disponiendo del arma de rayos laser.

TIBURÓN - 16K
En el medio de las aguas del océano, es visto un tiburón. Disponiendo de una cantidad limitada de tiros, Ud. deberá estinguirlo. Es impresionante el efecto de la captura.

OGRO MORTAL - 16K
Ud. deberá salir rápidamente de la oscuridad, antes que el ogro lo encuentre. Son necesarios reflejos precisos y mucha astucia.

MISILES - 16K
Ud. tiene que destruir un Reactor Atómico, evitando los misiles enemigos. Es especial para óptimos pilotos.

MICROSOFT®

PROGRAMAS PARA EL COMPUTADOR PERSONAL

TK85

JUEGOS ANIMADOS

FANTASMAS - 16K
Invasores fantasmas aterrorizan al público. Su misión es destruirlos.

TERRITORIO - 16K
Ud. y su computador representan superpotencias prontas a expandir sus actuales territorios. Para vencer, Ud. deberá limitar la extensión del territorio de su enemigo.

MONSTRUO DE LAS TINIEBLAS - TRIDIMENSIONAL - 16K
Impresionante juego donde Ud. deberá evitar al monstruo de las tinieblas. Todo en 3 dimensión.

OVNIS TRIDIMENSIONAL - 16K
Piloteando una nave espacial, en medio de planetas y meteoritos, Ud. es atacado por varios OVNIS. Su misión es evitar que ellos lo alcancen, lo que requiere mucha habilidad. Para ello Ud. dispone de un radar y un arma de rayos fotónicos.

GUERRA EN LAS ESTRELLAS - 16K
Tenemos en el video una nave espacial ENTERPRISE moviéndose en el espacio sideral. Ud. y el comandante de la nave, deben destruir la galaxia de los Klingons. Además tendrá que enfrentar muchos problemas en su nave, tales como: saboteadores, reparaciones en vuelo, etc.

EL GATO Y EL RATON - 16K
Fascinante juego donde Ud. es el ratón. Su objetivo es entrar en su cueva sin que el terrible gato lo atrape. Además de eso, Ud. tendrá que evitar las trampas que están en su camino. Tiene 50 niveles de dificultades.

DELPHOS - 16K
Delphos es un desafio a su pericia y reflejos. Ud. debe maniobrar su nave espacial dentro de una nube oscura, evitando chocar con los obstáculos que irán a destruirla. Tiene 5 niveles de dificultades.

UTILITARIOS

ASSEMBLER Z80 - 16K
Forme su programa en lenguage de máquina, directamente en mnemónico Z80, que el programa va a convertirlo en código binario. Permite el uso de etiquetas, y así efectua automaticamente el cálculo de direcciones relativas. Herramienta imprescindible para el trabajo en lenguage de máquina, junto con el programa MONITOR.

MONITOR Y DISASSEMBLER Z80 - 16K
Este es un programa para que el usuario desarrolle su propio programa en lenguage de máquina. Permite tabular bloques de memoria en hexa o caracteres, o convertir un bloque en mnemónico de Z80. Ud. podrá consultar y alterar posiciones de la memoria, y también verificar los registros. Permite puntos de parada (breakpoints), para facilitar el bugging de los programas, además de muchos otros recursos. Optima utilización junto con el ASSEMBLER.

SISTEMAS COMERCIALES

SICOM - 16K
Sistema integrado de Aplicaciones Comerciales. Puede ser utilizado para: cadastro de clientes; Funcionarios; Abastecedores; Productos; Duplicados y otros. Cuentas a pagar y a recibir. Control de stock. Agenda personal y telefónica. Bibliografía, etc. El programa permite ademas insertar, cancelar, alterar datos, y efectuar una rápida busqueda.

T-KALC - 16K/64K
Programa realizado para cálculos numéricos en planilla. El usuario define las columnas, las líneas y las fórmulas aplicadas. Es similar al famoso Visicalc. De gran versatilidad, este programa permite la formulación de cálculos científicos y comerciales, análisis de tablas numéricas y otras aplicaciones.

CONTROL DE STOCK - 16K/64K
Controle su stock a través de la relación de material, abastecedores, niveles mínimos y por clasificación. Cada ficha está compuesta por: descripción de material, código del abastecedor, tipo de clasificación, stock mínimo y actual, valor unitario y valor del stock. El programa clasifica y numera las fichas en orden alfabético. Las fichas pueden ser alteradas y canceladas, y serán listadas tablas de abastecedores, materiales con stock abajo del mínimo, o un listado completo. Fácil de operar, permite el cadastro de más de 1000 items en 64K.

PROFESIONALES

ESTADISTICA I - 16K
Se compone de varios programas, entre ellos, cálculo de Desvío Standart, Media, Regresión Lineal con desvio, Histogramas, Variaciones, quicuadrados y gráficas.

MATEMATICA I - 16K/64K
Analiza gráfica de funciones matemáticas, resolución de sistemas de ecuaciones lineales (16K-51 ecuaciones/64K-95 ecuaciones), y Cálculo de integrales definidas.

ANALISIS DE CAMINO CRITICO (PERT) - 16K
El computador permite analizar e identificar el Camino Crítico. Este programa es de gran auxilio en la aplicación de proyectos, agilizando la organización de actividades. Puede ser utilizado en Marketing, Ingeniería y proyectos en general.

VIGA CONTINUA - 16K
Destinado específicamente al área de Ingeniería Civil. Acepta las mas variadas condiciones en el cálculo de viga. Proporciona una tabla de reacciones de apoyo y momentos cortantes, diagramas globales y locales de los esfuerzos en la viga.

INTERES GENERAL

CONTROL BANCARIO
Este eficiente programa controla su Cuenta Bancaria, permitiendo insertar, corregir o cancelar cualquier operación. Indica automáticamente los pagos mensuales a efectuar. Busca cualquier item por el número de cheque, por su descripción o por su valor.

MICROSOFT®

MICRODIGITAL