Texas Instruments
Computadora TI-99/4A

TI BASIC Extendido

PARA TI-99/4A

Un poderoso lenguaje de programación de alto nivel, que amplía la capacidad de la computadora TI-99/4. Incluye las siguientes características:

- *Más de 40 comandos, instrucciones, funciones y subprogramas, nuevos o ampliados.*
- *Líneas con múltiples instrucciones para una mayor velocidad y eficiencia.*
- *Capacidad de Sprites (gráficos en movimiento).*
- *Capacidad que permite almacenar en un diskette los subprogramas más frecuentemente usados para llamarlos cuando se necesitan.*
- *Habilidad para cargar y ejecutar un programa desde otro.*
- *Amplio control sobre errores, advertencias y "break-points" en programación.*
- *Control directo de la entrada y la salida por pantalla.*
- *Posibilidad de cargar y ejecutar programas en lenguaje Assembler TMS9900 si tiene conectada la Expansión de Memoria opcional (vendida por separado).*

TI BASIC Extendido

PARA TI-99/4A

Un poderoso lenguaje de programación de alto nivel, que amplía la capacidad de la computadora TI-99/4. Incluye las siguientes características:

- Más de 40 comandos, instrucciones, funciones y subprogramas, nuevos o ampliados.
- Líneas con múltiples instrucciones para una mayor velocidad y eficiencia.
- Capacidad de Sprites (gráficos en movimiento).
- Capacidad que permite almacenar en un diskette los subprogramas más frecuentemente usados para llamarlos cuando se necesitan.
- Habilidad para cargar y ejecutar un programa desde otro.
- Amplio control sobre errores, advertencias y "breakpoints" en programación.
- Control directo de la entrada y la salida por pantalla.
- Posibilidad de cargar y ejecutar programas en lenguaje Assembler TMS9900 si tiene conectada la Expansión de Memoria opcional (vendida por separado).

TEXAS INSTRUMENTS
ARGENTINA S.A.I.C.F.

Este libro fue desarrollado y escrito por:

Robert E. Whitsitt, II
y un conjunto de miembros del Texas Instruments Learning Center y el Texas Instruments Personal Computer Division.

Con la contribución de:

Tom M. Ferrio
Stanley R. Hume
Jacquelyn F. Quiram

Trabajo de arte y planificación fueron coordinados y ejecutados por:
Schenck Design Associates, Inc.

ISBN # 0-89 512-04 5-3
Catálogo Biblioteca del Congreso N° 80-54899

Traducido e Impreso en Argentina.

Indice

INDICE

APENDICES

CAPITULO

1

Introducción

CARACTERISTICAS

El BASIC Extendido de Texas Instruments es un poderoso lenguaje de programación para usar con la Computadora TI-99/4. Posee características de un lenguaje de alto nivel además de otras adicionales no disponibles en otros lenguajes, incluso diseñadas para use de grandes computadoras.
El TI BASIC Extendido va más allá del BASIC de Texas Instruments para ampliar la capacidad y flexibilidad de su computadoras, agregando las siguientes características:

- *Entrada y Salida* - La instrucción ACCEPT permite la entrada de datos desde cualquier lugar de la pantalla. Se puede limpiar la pantalla, aceptar sólo ciertos caracteres y limitar el mínimo de caracteres ingresados usando esta instrucción. La instrucción DISPLAY ha sido mejorada para que permita colocar caracteres en cualquier lugar de la pantalla, y se agregaron las instrucciones DISPLAY ... USING, PRINT ... USING e IMAGE para formatear datos en el "display" de pantalla y en los dispositivos periféricos.
- *Subprogramas* - TI BASIC Extendido permite la escritura de subprogramas que usan variables locales (afectando sólo los valores del subprograma). En general los subprogramas usados pueden almacenarse en un diskette y agregarse a los programas a medida que se los necesita. Las instrucciones incluidas son SUB, SUBEND y SUBEXIT. Se ha agregado el comando MERGE y se modificó el comando SAVE para permitir la mezcla de programas desde los diskettes.
- *Sprites* - Los sprites son gráficos definidos especialmente, que tienen la propiedad de moverse suavemente por la pantalla. Para implementar la posibilidad de sprites se han incluido en el TI BASIC Extendido los siguientes subprogramas: COINC, DELSPRITE, DISTANCE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, y se rediseñaron SPRITE, COLOR y CHAR para que puedan afectar también a los sprites.
- *Funciones* - Se han incluido en TI BASIC Extendido las funciones: MAX que devuelve el mayor de dos números, MIN que devuelve el más pequeño de dos números y PI que da el valor de ¶.
- *Arreglos* - Los arreglos pueden tener hasta 7 dimensiones en lugar de 3.
- *Manejo de Cadena de Caracteres* — La función RPT$ permite la repetición de una cadena de caracteres.
- *Manejo de Errores* - Con TI BASIC Extendido se puede elegir qué acción tomar si ocurre un error menor (que en TI BASIC provoca un mensaje de advertencia), un error mayor (que en TI BASIC provoca un mensaje de error y detiene la ejecución del programa), o un "breakpoint" (que en TI BASIC provoca una parada del programa). Las nuevas instrucciones que permiten este control son: ON WARNING, ON ERROR, ON BREAK. La instrucción RETURN ha sido modificada para el mensaje de errores. Se puede usar la instrucción CALL ERR para determinar la naturaleza de un error que ha ocurrido en un programa.

- *RUN como Instrucción* - RUN puede ser usado como comando y como instrucción. También puede modificarse para que permita especificar qué programa se desea ejecutar. Como resultado de esto un programa puede cargar y ejecutar otro programa desde un diskette. Se pueden además escribir programas casi sin límite de longitud fraccionándolos en trozos y haciendo que cada segmento ejecute al próximo.
- *Potencia en la Ejecución de Programas* - Cuando se elige TI BASIC Extendido la primera vez, éste busca un programa llamado LOAD en el diskette que está drive 1. Si el programa existe, lo pone en memoria y lo ejecuta.
- *Líneas con Múltiples Instrucciones* - TI BASIC Extendido permite que haya más de una instrucción por línea. Esta característica hace que el programa se ejecute a mayor velocidad, ahorra memoria, y permite que las unidades lógicas (por ejemplo los ciclos FOR-NEXT ocupen una única línea.
- *Protección del SAVE y LIST* - Se pueden proteger los programas para que no sean almacenados o listados, evitando copias no autorizadas o cambios en esos programas. Esto, junto con la característica de protección de copia del Módulo Administrador de Discos asegura completamente los programas escritos en TI BASIC Extendido.
- *IF-THEN-ELSE* - La instrucción IF-THEN-ELSE permite ahora colocar instrucciones como consecuencia de la comparación. Esta expansión permite instrucciones de la forma "IF X <4 THEN GOSUB 240 ELSE X = X + 1".
- *Asignaciones Múltiples* - TI BASIC Extendido permite asignar un valor a más de una variable en una instrucción LET, ahorrando instrucciones y haciendo más eficiente la programación.
- *Comentarios* - Además de la instrucción REM se pueden agregar comentarios al final de las líneas, permitiendo una documentación interna detallada de los programas.
- *Soporte de Lenguaje Assembler* - Con la unidad opcional de Expansión de Memoria (disponible por separado) se pueden cargar y ejecutar subprogramas en lenguaje Assembler TMS9900. Los subprogramas INIT, LOAD, LINK y PEEK se usan para tener acceso a subprogramas en lenguaje assembler. No es posible escribir programas en lenguaje assembler con la Computadora TI-99/4.
- *Información* - Se ha agregado el comando SIZE que informa qué cantidad de memoria libre queda en la computadora. El subprograma VERSION devuelve un valor que indica la versión de BASIC que se está usando. El subprograma CHARPAT devuelve una cadena de caracteres que describe el modelo que define el carácter.
- *Expansión de Memoria* - El TI BASIC extendido acepta el use de un periférico opcional de Expansión de Memoria que permite la escritura de programas mucho más grandes.

CAMBIOS DEL TI BASIC

Las mejoras descriptas anteriormente provocan algunos leves cambios en otras áreas del TI BASIC. Por esta causa, algunos programas escritos en TI-99/4 BASIC no pueden ejecutarse con el TI BASIC Extendido.

- Ahora la longitud máxima de programa es 864 bytes más pequeña que con TI BASIC. Si posee la Expansión de Memoria podrá escribir programas mucho más grandes.
- Los caracteres de los conjuntos 15 y 16 ya no están disponibles. Esa área de memoria la usa TI BASIC Extendido para el control de los sprites.
- La mayoría de los programas en TI BASIC se podrán ejecutar con el TI BASIC Extendido sin dificultad. Sin embargo, en algunos casos, los programas TI BASIC que usen palabras reservadas del TI BASIC Extendido no podrán ser ejecutadas con éste último. Asimismo siempre se podrán cargar programas escritos en TI BASIC con el TI BASIC Extendido. Los programas que usen las ampliaciones del TI BASIC Extendido no ejecutarán correctamente con TI BASIC.

COMO USAR ESTE MANUAL

Este manual supone que el lector ya tiene alguna experiencia de programación con TI BASIC. Las instrucciones, comandos y funciones que son las mismas que en TI BASIC solo se discutirán brevemente. Para una discusión completa vea la Guía de Referencia del Usuario que vino con su Computadora TI-99/4.

Las características adicionales del TI BASIC Extendido se explican en detalle y están ilustrados con ejemplos y programas. Para obtener la máxima utilidad el TI BASIC Extendido, lea este manual cuidadosamente, ingresando y ejecutando los programas de ejemplo para ver cómo trabajan. Aún las características que no han cambiado deben ser repasadas. Puede encontrar que hay instrucciones que no usaba o descubrir una nueva manera de usar instrucciones en distintas combinaciones.

El resto de este capítulo repasa las bases para operar con el TI BASIC Extendido. El Segundo capítulo discute sus características e incluye un ejemplo detallado del ingreso de un programa. El tercer capítulo es una sección de referencia que discute, en orden alfabético, todos los comandos, instrucciones y funciones del TI BASIC Extendido.

Los 14 apéndices contienen mucha información útil, incluyendo los códigos de caracteres ASCII, códigos de error, códigos de color, códigos de teclado, e instrucciones sobre cómo agregar sufijos a las palabras habladas.

COMO USAR LA COMPUTADORA

Antes de usar la computadora con el TI BASIC Extendido deberá insertar el Módulo de Comando en Estado Sólido en la computadora. Si la computadora está apagada, deslice suavemente el módulo en la abertura de la consola.

Luego encienda la computadora. (Si tiene periféricos, enciéndalos antes de encender la computadora). Aparecerá la pantalla con el título principal. Si la computadora ya estaba encendida, vuelva a la pantalla con el título principal. Luego deslice el módulo en la abertura de la consola.

Presione cualquier tecla para que aparezca la lista de selección principal. El título del modulo es TI EXTENDED BASIC, es el segundo de la lista. Mecanografíe 2 para seleccionar el TI BASIC Extendido.

OPERANDO CON TI BASIC EXTENDIDO

Hay tres modos de operación en TI BASIC: Modo Comando, Modo Edición y Modo Ejecución.

Cuando se selecciona TI BASIC Extendido de la lista de selección principal, la computadora entra en Modo Comando. En el Modo Comando se pueden ingresar comandos, instrucciones que pueden ser usadas como comandos y líneas de programa.

El Modo de Edición se usa para editar líneas de programa ya existentes. Para entrar en Modo Edición, mecanografíe el número de línea y luego presione FCTN E (↑) o FCTN X (↓). (TI BASIC también permite el comando EDIT seguido de un número de línea, TI BASIC Extendido no lo permite). La línea especificada se despliega en la pantalla. Puede cambiarla mecanografiando una nueva línea, corrigiendo una parte de la vieja línea, o usando las teclas de edición que se explican más adelante. También está el Modo Edit cuando se presiona FCTN 8 (REDO) para repetir una línea de programa o un comando.

En Modo Ejecución, se ejecuta un programa en TI BASIC Extendido. Puede detener la ejecución de un programa presionando FCTN 4 (CLEAR), lo que provocará un "breakpoint" o con FCTN 4 (QUIT). Nota: FCTN + (QUIT) también borra de la memoria el programa completo, le devuelve la pantalla principal y puede borrar información de alguno de sus archivos. Se recomienda el use de BYE en lugar de FCTN + (QUIT) para abandonar el TI BASIC Extendido.

TECLAS CON FUNCIONES ESPECIALES

Las siguientes son teclas que tienen una función especial cuando se presionan junto con la tecla **FCTN: E, D, S, X, 1, 2, 3, 4, 8,** +. A continuación se discute cada una de estas teclas.

FCTN E (↑) se usa en Modo Edición. Si no está en Modo Edición, se puede entrar en él mecanografiando un número de línea y luego **FCTN E (↑)**. La línea especificada se despliega en la pantalla y puede ser editada. Si ya está en Modo Edición presionando **FCTN E (↑)** ingresa la línea presente tal como ha sido cambiada, y despliega la siguiente línea de programa con número menor. Presionando **FCNT E (↑)** cuando se está en el número más bajo del programa se regresa al Modo Comando. Si está ingresando una línea en Modo Comando, **FCTN E (↑)** tiene el mismo efecto que **ENTER.**

FCTN D (→) mueve el cursor un espacio a la derecha. El cursor no borra o cambia los caracteres, sólo pasa sobre ellos. Al llegar al final de una línea en la pantalla el cursor salta a la primera posición de la línea siguiente. Cuando el cursor está al final de una línea de entrada, no se mueve.

FCTN S (←) mueve el cursor un espacio hacia la izquierda. El cursor no borra ni cambia caracteres a medida que pasa sobre ellos. Si el cursor está al comienzo de una línea, no se mueve. Si está ubicado en el margen izquierdo pero no al principio de una línea de entrada, el cursor pasa al margen derecho de la línea inmediatamente superior.

FCTN X (↓) Se usa en Modo Edición. Si no está en Modo Edición, se puede entrar en él mecanografiando un número de línea y luego presionando **FCTN X (↓)**. La línea especificada por el número de línea se despliega en la pantalla y puede ser editada. Si ya está en Modo Edición, al presionar **FCTN X (↓)** ingresa la línea presente y despliega la próxima línea de programa con número de secuencia más alto. Presionando **FCTN X (↓)** cuando se está en la línea del programa con el número más alto se vuelve al Modo Comando. Si se está ingresando una Línea en Modo Comando, **FCTN X (↓)** tiene en mismo efecto que el ENTER.

FCTN 8 (REDO) hace que los caracteres de la línea ingresada previamente reaparezcan en la pantalla. Así se desea entrar una línea similar a la última ingresada, presione **FCTN 8 (REDO)** y corríjala usando las características del Modo Edición. Esta tecla le evita tener que mecanografiar una línea muy larga.

FCTN 3 (ERASE) bona los caracteres de la línea actual, pero deja el cursor en esa línea. Si está en Modo Comando el cursor regresa al margen izquierdo de la pantalla y se puede ingresar una nueva línea , incluyendo el número de línea. De otro modo, si está editando una línea o si la computadora está entregando números de línea (a través del use de NUM), el número de línea no se borrará.

FCTN 2 (INSERT) Hace que la computadora acepte la inserción de caracteres. Cada tecla subsiguiente que se mecanografía, se inserta en la posición del cursor y el carácter que ocupa la posición del cursor y todos los caracteres a la derecha del mismo se corren una posición hacia la derecha. La inserción continúa con cada carácter mecanografiado hasta que se presiona **ENTER** o cualquiera de las otras teclas de función especiales. Los caracteres al final de la línea de entrada pueden perderse.

FCTN 1 (DELETE) borra el carácter que está sobre el cursor y corre todos los caracteres que están a la derecha del cursor, una posición hacia la izquierda.

FCTN 4 (CLEAR) cumple distintas funciones dependiendo del modo en que se encuentra. Si está en Modo Edit, se ignoran los cambios realizados en la línea, incluso el **FCTN 3 (ERASE)**, y la computadora vuelve a Modo Comando. Si está en Modo Ejecución el programa se detiene en un "breakpoint".
Si está en Modo Comando, se borran los caracteres mecanografiándolos en la línea corriente.
Cuando use **FCTN 4 (CLEAR)** para detener un programa, sostenga las teclas hasta que el TI BASIC Extendido reconozca el "breakpoint".

FCTN + (QUIT) devuelve a la computadora la pantalla principal. Cuando presiona **FCTN + (QUIT)**, se borran todos los datos y los programas de la memoria. Si está usando un sistema de discos, pueden perderse algunos de sus archivos de datos. Abandone el TI BASIC Extendido con BYE en lugar de usar **FCTN + (QUIT)**.

ENTER indica que se ha terminado de mecanografiar la información de la línea corriente y ésta queda lista para que la computadora la procese.

CAPITULO

2

Generalidades del TI BASIC Extendido

GENERALIDADES DEL TI BASIC Extendido

Este capítulo describe brevemente los comandos, instrucciones y funciones del TI BASIC Extendido, y sugiere algunas formas de usarlos.

Las primeras ocho secciones son Comandos; Asignaciones y Entrada; Salida; Funciones; Subrutinas, y Subprogramas: Sound, Speech y Color; Sprites; Debugging y Manejo de errores.

La sección final es un programa ejemplo, que muestra el proceso de entrada y el use de algunos de los elementos del TI BASIC Extendido.

COMANDOS

Los comandos le dicen a la computadora que ejecute una tarea inmediatamente (es decir tan pronto como se presiona **ENTER**. Y las instrucciones se ejecutan cuando se ejecuta el programa. En TI BASIC Extendido muchos comandos se usan como instrucciones y la mayoría de las instrucciones pueden ser usadas como comandos. En el *Apéndice B* se da una lista de todas las instrucciones, comandos y funciones, indicando qué comandos pueden usarse como instrucciones y qué instrucciones pueden ser usadas como comandos.

NEW

Este comando se usa para remover un programa de TI BASIC Extendido y prepara la computadora para aceptar un nuevo programa. Los programas también pueden removerse de la memoria mediante el comando OLD y el comando RUN usados con un nombre de archivo.

NUMBER Y RESEQUENCE

Cuando se ingresa un programa, la computadora asigna automáticamente los números de línea si se ha usado el comando NUMBER. Si se desea resecuenciar los números de línea de un programa que ya está escrito, hay que usar RESEQUENCE.

LIST

Este comando se usa para ver un programa que ya ha sido ingresado. El programa podrá listarse en la pantalla o en un dispositivo periférico.

RUN

El comando RUN ordena a la computadora que ejecute un programa. El comando RUN puede ir seguido por un número de línea para que comience la ejecución a partir de ese número de línea determinado. 0 bien puede ir seguido por un nombre de archivo para cargar y ejecutar un programa desde diskette.

TRACE, UNTRACE, BREAK, UNBREAK y CONTINUE

Todos estos programas están relacionados con el "seguimiento" de un programa para encontrar la causa de una condición de error o un resultado incorrecto. Estos comandos se discuten en la sección "Seguimiento y Manejo de Errores".

SAVE, OLD, MERGE y DELETE

Cuando se ha terminado de trabajar con un programa puede desearse almacenarlo en un cassette o diskette para su use posterior. El comando SAVE seguido del nombre del dispositivo y el nombre del programa, realiza esa tarea. Luego cuando se requiere volver a usarlo, listarlo, editarlo o cambiar el programa, se lo puede cargar en la memoria con el comando OLD.
Si un programa ha sido almacenado usando la opción merge, podrá combinarlo con un programa que ya esté en memoria usando el comando MERGE. Cuando ya no se necesite mas usar un programa que ha sido almacenado en un diskette, se lo podrá remover con el comando DELETE.

SIZE

Este comando le permite determinar cuanta memoria queda libre. Así podrá decidir si sigue agregando líneas al programa o lo termina y crea otro programa que se ejecutará desde el primero usando RUN como instrucción.

BYE

Cuando ha finalizado de usar el TI BASIC Extendido, use el comando BYE para volver a la pantalla principal.
Muchos de estos comandos (RUN, BREAK, UNBREAK, TRACE, UNTRACE, y DELETE) pueden también ser usados como instrucciones en programas.

ASIGNACIONES Y INPUT (ENTRADA)

Esta sección discute las instrucciones de TI BASIC Extendido que asignan valores a variables e ingresan datos en los programas.

LET Y READ

Si se desconocen los valores a asignar a ciertas variables se puede usar los comandos LET o READ.
LET se usa cuando hay que asignar una pequeña cantidad de valores o estos deben ser calculados.
READ se usa junto con DATA y RESTORE cuando se debe asignar una gran cantidad de valores.

INPUT y LINPUT

Cuando se desea que el usuario de un programa asigne valores, es costumbre hacer una pregunta cuya respuesta dé la información necesaria. INPUT le permite hacer la pregunta y aceptar la respuesta.
INPUT permite solamente la entrada de valores desde la parte inferior de la pantalla y no puede verificar si los datos ingresados son el tipo de información que espera el programa. La limitación final de INPUT es que las comas y comillas afectan lo que se ingresa. Con LINPUT no se edita la entrada, de esta manera se pueden ingresar comas y comillas. Tanto INPUT como LINPUT pueden usarse para traer datos de archivos que están en cassettes o diskettes.

ACCEPT

Esta instrucción permite la entrada de datos desde casi todas las posiciones de la pantalla. El use de ACCEPT elimina la necesidad de entrar los datos desde la parte inferior de la pantalla y elimina el movimiento que provoca la instrucción. INPUT. Sin embargo ACCEPT no permite una pregunta del tipo INPUT. Además deberá incluirse en el programa una instrucción PRINT o DISPLAY que diga al usuario qué tipo de entrada se requiere.
ACCEPT puede verificar la entrada para ver si es numérica, alfabética o caracter especial. ACCEPT se usa solamente con pantalla y teclado.

CALL KEY Y CAL JOYST

Si lo único que se necesita del usuario es que presione una tecla, entonces se puede usar CALL KEY. Por ejemplo, si la respuesta requerida es "S" para "Si" y "N" para "No", se puede usar CALL KEY para requerir la entrada. El subprograma busca en el teclado o en una porción de el para ver si se ha presionado una tecla determinada. La mayor limitación del CALL KEY es que sólo se puede presionar una tecla por vez. El dato no se graba como un caracter sino como su equivalente ASCII o algún otro código (Véase los apéndices C y J para la lista de los códigos usados). Si desea mostrar cual fue la tecla que presionó deberá usar DISPLAY, PRINT, CALL VCHAR, o CALL HCHAR.
La entrada desde un Controlador Remoto puede hacerse con CALL JOYST. Tal como en el caso de CALL KEY, los datos no se muestran en la pantalla y no hay movimiento de pantalla.

CALL CHARPAT, CALL COINC, CALL DISTANCE, CALL ERR, FOR-TO-STEP, CALL GCHAR, CALL POSITTION, NEXT, CALL SPGET y CALL VERSION.

Cada una de estas instrucciones asigna uno o más valores a una variable. CALL CHARPAT asigna un valor que especifica el modo de un caracter. CALL COINC asigna un valor para determinar si varios sprites o si un sprite y un punto de la pantalla están cerca o en la misma posición de la pantalla. CALL DISTANCE indica la distancia entre dos sprites o entre un sprite y un punto de la pantalla. CALL ERR especifica qué error ocurrió y cuando ocurrió . CALL GCHAR lee un caracter ubicado en una posición de memoria dada. CALL POSITTION lee la posición de un sprite en la pantalla. CALL SPGET asigna el valor codificado de una frase a una variable que será usada con CALL SAY. CALL VERSION indica la versión de BASIC que se está usando.

FOR-TO-STEP y NEXT merecen un comentario especial. La instrucción FOR-TOSTEP define el valor de una variable que será usada como control del número de veces que se ejecuta el ciclo. Cada vez que se encuentra un NEXT, cambia el valor de la variable. Luego que se ha completado el ciclo, la variable tiene un valor que es el 1er. valor fuera del rango especificado en la instrucción FOR-TO-STEP.

OUTPUT (salida)

Esta sección discute las instrucciones del TI BASIC Extendido que se usan para salidas de datos en la ejecución de programas. Generalmente, la salida consiste en el despliegue de la información en la pantalla, impresión de datos en la impresora, o el almacenamiento de datos en un dispositivo externo. Sin embargo, la salida también puede involucrar un cambio de color de la pantalla, cambio de color de los caracteres, ruidos, palabras habladas o el envío de datos a dispositivos periféricos.

PRINT, DISPLAY, PRINT... USING, DISPLAY... USING e IMAGE.

Las instrucciones más usadas para salida son PRINT y DISPLAY. Los separadores de impresión (coma, punto y coma y dos puntos) y la función TAB se usan para controlar la ubicación de la información.
PRINT despliega los datos en la parte inferior de la pantalla y los desplaza hacia arriba de a una línea por vez.
Con DISPLAY se pueden mostrar datos prácticamente en cualquier lugar de la pantalla sin movimiento. DISPLAY puede también limpiar la pantalla, borrar caracteres en una línea y también provocar un "beep".

PRINT... USING y DISPLAY... USING.

Son los mismos que el PRINT y el DISPLAY, salvo que los formatos impresos o desplegados están determinados por la cláusula USING generalmente en conjunción con una instrucción IMAGE. La cláusula USING permite el control exacto del formato. PRINT y PRINT... USING son posiblemente en conjunción con IMAGE, las únicas instrucciones de salida que pueden usarse para enviar datos a un dispositivo externo.

CALL HCHAR, CALL VCHAR y CALL SPRITE.

CALL HCHAR y CALL VCHAR ubican un caracter en cualquier posición de la pantalla y opcionalmente lo repiten horizontal o verticalmente. CALL SPRITE "muestra" los "sprites" en la pantalla. Los "sprites" son gráficos que se mueven suavemente en la pantalla en cualquier dirección, y a los que se les puede cambiar el modelo, la forma y el color. CALL SPRITE y las otras instrucciones relativas a los sprites se discuten más adelante en este capítulo.

CALL SCREEN y CALL COLOR

Además de mostrar caracteres y datos en la pantalla, se puede cambiar el color de la pantalla y los colores de los caracteres.
CALL SCREEN define el color de la pantalla. CALL COLOR especifica los colores de frente y de fondo de los caracteres o el color de los "sprites".

CALL SOUND Y CALL SAY.

CALL SOUND produce sonidos. Hay disponible un amplio rango de sonidos.
CALL SAY (posiblemente usado con CALL SPGET) hace hablar a la computadora si ésta tiene conectado el "Sintetizador de Palabras en Estado Sólido".

FUNCIONES, SUBRUTINAS y SUBPROGRAMAS

TI BASIC Extendido provee una gran cantidad de funciones y subprogramas para manejar números y caracteres. Además el usuario puede construir sus propias funciones y escribir sus propios subprogramas y subrutinas.
Las funciones son elementos del lenguaje TI BASIC Extendido que devuelven un valor, generalmente basado en parámetros dados a la función. Muchas funciones son de naturaleza

matemática; otras controlan o afectan el resultado o la salida producida por las instrucciones que las contienen. Las funciones de TI BASIC Extendido son: ABS, ASC, ATN, CHR$, COS, EOF, EXP, INT, LEN, LOG, MAX, MIN, PI, POS, REC, RND, RPT$, SEG$, SGN, SIN, SQR, STR$, TAB, TAN y VAL.

El usuario también puede definir sus propias funciones usando DEF. Dichas funciones se usarán junto con las instrucciones del TI BASIC Extendido.

Funciones Propias de TI BASIC Extendido

Se discute brevemente a continuación cada una de las funciones propias del TI BASIC Extendido:

Función	Valor Devuelto y Comentarios
ABS	Valor absoluto de una expresión numérica.
ASC	El código ASCII numérico del primer caracter de una expresión alfa-numérica.
ATN	Arcotangente trigonométrico de una expresión numérica dada en radianes.
CHR$	Caracter que corresponde a un código ASCII.
COS	Coseno trigonométrico de una expresión numérica dada en radianes.
EOF	Condición de fin de archivo para un archivo.
EXP	Valor exponencial de una expresión numérica (ex)
INT	Valor entero de una expresión numérica.
LEN	Cantidad de caracteres en una expresión alfanumérica.
LOG	Logaritmo natural de una expresión numérica.
MAX	El mayor de los valores de dos expresiones numéricas.
MIN	El menor de los valores de dos expresiones numéricas.
PI	r con valor 3.141592654.
POS	Posición de la primera ocurrencia de una expresión alfanumérica de otra.
REC	Posición del registro actual en un archivo.
RND	Número al azar entre 0 y 1.
RPT$	Expresión alfanumérica igual a un número de copias de otra expresión alfanumérica concatenadas juntas.
SEG$	Porción de una expresión alfanumérica, que empieza en un punto determinado y termina luego de una cierta cantidad de caracteres.
SGN	Signo de una expresión numérica.
SIN	Seno trigonométrico de una expresión numérica dada en radianes.
SQR	Raíz cuadrada de una expresión numérica.
STR$	Cadena alfanumérica equivalente a una expresión numérica.
TAB	Posición del próximo ítem en la lista de impresión para PRINT, PRINT...USING, DISPLAY o DISPLAY...USING.
TAN	Tangente trigonométrica de una expresión numérica dada en radianes.
VAL	Valor numérico de una expresión alfanumérica que representa un número.

Función Definidas por el Usuario

También puede usar DEF para definir sus propias funciones.
Se pueden definir funciones de hasta una línea de longitud con un argumento. Si se necesitan funciones de mayor longitud, se las puede construir a partir de otras funciones definidas previamente.
Sin embargo, para funciones muy largas puede ser mas eficiente el use de subrutinas o subprogramas.

Subrutinas

Para llamar subrutinas se usa GOSUB u ON ... GOSUB.
Una subrutina es un conjunto de instrucciones diseñado para realizar una tarea que se repite varias veces dentro de un mismo programa. Usando GOSUB u ON .. GOSUB no es necesario mecanografiar las mismas líneas varias veces. La subrutina puede usar los valores de cualquier variable del programa e incluso cambiar dichos valores.

Subprogramas Provistos por TI BASIC Extendido

Estos subprogramas son elementos del TI BASIC Extendido que realizan funciones especiales. Siempre se puede acceder a ellos mediante la instrucción CALL. Los subprogramas son: CHAR, CHARPAT, CHARSET, CLEAR, COINC, COLOR, DELSPRITE, DISTANCE, ERR, GCHAR, HCHAR, INIT, JOYST, KEY, LINK, LOAD, LOCATE, MAGNIFY, MOTION, PATTERN, PEEK, POSITION, SAY, SCREEN, SOUND, SPGET, SPRITE, VCHAR, Y VERSION.
Los subprogramas provistos por TI BASIC Extendido realizan muchas tarea diferentes. Algunos afectan el "display" y determinan qué tecla se ha presionado en el teclado.

Subprograma	*Acción y Comentarios*
CLEAR	Limpia la pantalla.
COLOR	Especifica los colores de los caracteres con los conjuntos de caracteres o el color de los sprites.
GCHAR	Devuelve el código ASCII de un carácter en una posición determinada de la pantalla.
HCHAR	Muestra un caracter en la pantalla y, opcionalmente lo repite horizontalmente.
JOYST	Devuelve valores que indican la posición de los Controladores Remotos (Opcional)
KEY	Especifica el color de la pantalla.
VCHAR	Muestra un caracter en la pantalla y opcionalmente lo repite verticalmente.

Los subprogramas provistos también definen y controlan los sprites.

Subprogramas	*Acción y Comentarios*
CHAR	Especifica el modelo de un caracter, que se usará como sprite o como gráfico.
COINC	Determina si dos sprites o un sprite y un punto de la pantalla están cerca o en la misma posición de la pantalla.
COLOR	Especifica el color de un sprite o un conjunto de caracteres.
DELSPRITE	Borra sprites.
DISTANCE	Determina la distancia entre dos sprites o entre un sprite y una posición.
LOCATE	Especifica la posición de un sprite.
MAGNIFY	Cambia el tamaño de los sprites.
MOTION	Define el movimiento de un sprite.
PATTERN	Especifica el caracter que define un sprite.
POSITION	Determina la posición de un sprite.
SPRITE	Define los sprites, especificando el caracter que los define, su color, su posición, y su movimiento.

Una tercera categoría de subprogramas provistos por TI BASIC Extendido incluye sonidos y palabras.

Subprograma	*Acción y Comentarios*
SAY	Hace que la computadora emita palabras cuando se lo usa junto con el Sintetizador de Palabras en Estado Sólido.
SOUND	Genera sonidos.
SPGET	Devuelve los códigos de las palabras.

Hay cuatro subprogramas que se usan solamente con subprogramas escritos en lenguaje de máquina, provistos por Texas Instruments o escritos en TMS9900 para otra computadora. No es posible escribir programas en lenguaje de máquina con la Computadora TI-99/4.
Junto con los subprogramas en lenguaje de máquina se da una explicación detallada del use de INIT, LINK, LOAD y PEEK.

Finalmente hay algunos subprogramas varios, provistos por TI BASIC Extendido.

Subprograma	*Acción y Comentarios*
CHARPAT	Devuelve el valor que identifica el modelo de un caracter.
CHARSET	Devuelve a los caracteres 32 a 95 sus modelos originales predefinidos y sus colores.
ERR	Devuelve valores que dan información sobre un error que ha ocurrido.
VERSION	Especifica la versión de BASIC que se está usando.

Subprogramas Escritos por el Usuario

El usuario puede escribir sus propios programas. Estos consisten en conjunto de instrucciones destinadas para realizar una tarea.
Pueden ser usados en un programa cuando se necesita realizar una tarea varias veces o realizar la misma tarea en diferentes programas. Usando la opción MERGE cuando se almacena un subprograma le permitirá luego incluirlo en otros programas.

Cuando un subprograma está dentro de un programa, debe seguir al programa principal.
La estructura de un programa debe ser la siguiente:

Comienzo del programa principal	
.	
.	
.	
Llamados al subprograma	
.	
.	
.	
Fin del programa principal	El programa se detendrá aquí sin un STOP ni un END.
Comienzo del 1er. subprograma	Los subprogramas son opcionales.
.	
.	
.	
Fin del 1er. subprograma	No puede aparecer nada entre los subprogramas, excepto comentarios y la instrucción END.
Comienzo del 2do. Subprograma	
Fin del 2do. Subprograma	Sólo comentarios y END pueden aparecer después de los subprogramas.
Fin del Programa	

Para llamar a los subprogramas se usa el CALL seguido del nombre del subprograma y una lista opcional de parámetros y valores. La primera línea de un subprograma es SUB, seguido por el nombre del subprograma y opcionalmente una lista de parámetros.
Los subprogramas no son parte del programa principal. No pueden usar los valores de las variables del programa principal, de manera que cualquier valor que necesiten deben obtenerlos a través de la lista de parámetros de la instrucción CALL. Los nombres de variables pueden ser duplicados de los que figuran en el programa principal o en otros subprogramas sin que se afecten los valores de dichas variables en el programa principal o subprogramas.
Los subprogramas pueden llamar a otros subprogramas, pero no pueden llamarse a sí mismos directa ni indirectamente.
SUBEND debe ser la última instrucción en un subprograma. Cuando se ejecuta esta instrucción, el control vuelve a la instrucción siguiente a la llamada al Subprograma.
También se puede devolver el control mediante la instrucción SUBEXIT.

SOUND, SPEECH Y COLOR

Se pueden resaltar partes importantes de la salida de un programa mediante el uso de sonidos, palabras y colores. Esta "ingeniería humana" hace que el uso del programa sea más fácil y más interesante.

CALL SOUND

SOUND produce sonidos. Se pueden producir todos desde .001 a 4.25 segundos y volúmenes desde 0 (el más alto) a 30 (el más suave). El rango de frecuencia es desde 110 (A bajo, bajo C) a 44.733 (más allá del rango del oído humano). Además se dispone de 8 tipos de ruidos.
Se pueden producir hasta tres tonos y un ruido al mismo tiempo.
El Apéndice D da una lista de las frecuencias que se usan para producir notas musicales.

CALL SAY Y CALL SPGET

SAY produce palabras cuando la computadora tiene conectado el Sintetizador de Palabras en Estado Sólido (vendido por separado). Se puede elegir entre 373 letras, números, palabras y frases (descriptas en el apéndice L). Además se pueden construir nuevas palabras a partir de las viejas, combinándolas. Por ejemplo SOME + THING produce "something" y THERE + FOUR produce "Therefore".

SPGET se usa para recuperar los códigos de las palabras. Estos modelos pueden usarse para producir un habla más natural y para cambiar las palabras. Dado que la construcción de nuevas palabras es un proceso complejo, no se discute en este manual. Sin embargo se pueden agregar sufijos con relativa simplicidad.
El Apéndice M indica cómo agregar los sufijos ING, S y ED a cualquier palabra de manera que se incluyen palabras como: ANSWERING, ANSWERS, ANSWERED, INSTRUCTING, INSTRUCTS, INSTRUCTED en el vocabulario de la computadora.

CALL COLOR Y CALL SCREEN

COLOR cambia los colores de los conjuntos de caracteres y determina los colores de los sprites.
SCREEN define el color de la pantalla como uno de los dieciséis colores disponibles en la computadora TI99/4A

SPRITES

Los sprites son gráficos que pueden mostrarse y moverse por la pantalla. Una de las ventajas de los sprites sobre los otros caracteres es que pueden ocupar cualquiera de las 49152 posiciones definidas por las 192 filas y las 256 columnas en lugar de las 768 posiciones definidas por las 24 filas y 32 columnas usadas por instrucciones tales como CALL VCHAR y CALL HCHAR. Gracias a su mayor resolución, los sprites se pueden mover más suavemente que los caracteres. También, una vez que han sido puestos en movimiento, los sprites pueden continuar moviéndose más allá del control del programa.

CALL SPRITE

CALL SPRITE define los sprites. Este subprograma especifica el modelo de caracter que usará el sprite, su color, su posición, y opcionalmente su movimiento.

CALL CHAR Y CALL MAGNIFY

Además de poder usar como sprite cualquiera de los caracteres predefinidos con códigos 32 a 95, se puede definir un nuevo modelo de sprite con CALL CHAR. Se pueden usar hasta cuatro caracteres de 8 x 8 puntos para formar un sprite. El subprogramas MAGNIFY controla la resolución y el tamaño de los sprites.

CALL COLOR, CALL LOCATE, CALL PAITERN Y CALL MOTION

Una vez definido un sprite, se lo puede alterar mediante distintos subprogramas. COLOR cambia el color del sprite. LOCATE mueve el sprite a una nueva posición. PATTERN cambia el caracter que define un sprite. MOTION altera el movimiento del sprite.

CALL COINC, CALL DISTANCE y CALL POSITION

Estos tres subprogramas proveen información acerca de los sprites mientras se está ejecutando un programa. COINC devuelve un valor que indica si varios sprites o un sprite y un punto en la pantalla están cerca o en el mismo lugar de la pantalla. DISTANCE devuelve en valor que especifica la distancia entre dos sprites o entre un sprite y un punto de la pantalla. POSITION devuelve valores que indican la posición de un sprite.

CALL DELSPRITE

CALL DELSPRITE permite borrar sprites. Si se prefiere, se puede "esconder" sprites ubicándolos fuera del fondo de la pantalla.

SEGUIMIENTO (DEBUGGING)

El seguimiento de un programa consiste en encontrar errores lógicos o de sintaxis. Los comandos e instrucciones BREAK, CONTINUE, TRACE, UNBREAK, UN-TRACE y FCTN 4 (CLEAR) son los más usados para hacer el seguimiento de un programa.

BREAK, CONTINUE y UNBREAK

BREAK hace que la computadora detenga la ejecución del programa para poder imprimir los valores de las variables, o cambiarlos. BREAK también devuelve los colores standard a los caracteres (negro sobre transparente) devuelve el color standard a la pantalla (azulado), devuelve a los caracteres standard 32 a 95 sus representaciones standard y borra los sprites.

ON BREAK indica a la computadora qué hacer si ocurre una interrupción. Se puede usar esta instrucción para hacer que la computadora ignore los "breakpoint" del programa. CONTINUE hace que la computadora continúe con la ejecución del programa luego de un "breakpoint" UNBREAK cancela todos los "breakpoints" definidos con BREAK.
Nota: Si se ha indicado ON BREAK CONTINUE, la computadora no se detendrá cuando presione FCTN 4 (CLEAR)

TRACE y UNTRACE

TRACE hace que la computadora muestra cada número de línea antes de que la/las instrucciones de esa línea se ejecuten. El use de esta instrucción permite seguir la secuencia de operación de un programa. UNTRACE cancela la operación de TRACE.

MANEJO DE ERRORES

Es posible incluir instrucciones dentro de un programa, que manejen errores que pueden ocurrir mientras se está ejecutando dicho programa.

CALL ERR, ON ERROR, ON WARNING, y RETURN

CALL ERR, devuelve información que indica dónde ocurrió un error y de qué
error se trata. El Apéndice N tiene los códigos de error posibles. ON ERROR define que es lo que tiene que hacer la computadora si ocurre un error.
ON WARNING define que hará la computadora si se llega a una condición que causará un mensaje de advertencia. RETURN se usa con ON ERROR además de usarse con GOSUB. Repite la ejecución de la instrucción que causó el error, vuelve a la instrucción siguiente a la que provocó el error, o transfiere el control a alguna otra parte del programa que no tenga en cuenta el error que ha ocurrido.

EJEMPLO DE UN PROGRAMA CON ENTRADAS

Ahora que ha visto las características generales del TI BASIC Extendido, puede disfrutar revisando e incluso ingresando y experimentando con un programa de demostración. Esta sección muestra una gran variedad de características útiles del TI BASIC Extendido. Siguiendo las sugestiones de esta Sección, podrá aprender algunos métodos breves y útiles en el proceso de entrada.

Este programa le permite jugar un juego llamado "Número Secreto". Al jugarlo usted determina la longitud de un código (1 a 8 dígitos). Luego decide el rango de dígitos que pueden incluirse en el código (hasta 10). La computadora selecciona los dígitos del código sin repetir ningún dígito. Usted debe adivinar cuales son los dígitos y cual es su secuencia. Luego de cada intento la computadora le dice cuántos dígitos adivinó y cuántos están en el lugar correcto (si repite un dígito en su intento, se In cuenta como correcto cada vez que aparece). Usando esta información usted vuelve a adivinar. Gana cuando logró adivinar todos los dígitos correctamente, y ubicarlos en la secuencia apropiada.

Por ejemplo suponga que ha elegido jugar el juego usando cuatro dígitos donde cada dígito puede ser uno de (0, 1, 2, 3, 4, 5, 6, 7 u 8). El código que elige la computadora podría ser 0743.
He aquí una posible secuencia de adivinanzas:

INTENTO	*CORRECTO*	*LUGAR*	*EXPLICACION DE LA RESPUESTA DE LA COMPUTADORA*
0000	4	1	0 es correcto 4 veces, y está 1 vez en el lugar apropiado.
1234	2	0	3 y 4 están bien, pero no están bien ubicados.
5768	1	0	7 está bien pero no en el lugar correcto.
2348	2	1	3 y 4 están bien y 4 está en el lugar correcto.
0347	4	2	Todos bien, 0 y 4 en el lugar correcto.
3047	4	1	Todos bien, 4 en el lugar correcto.
0734	4	2	Todos bien, 0 y 7 en el lugar correcto.
0743	4	4	Todos bien, en el lugar correcto usted gana.

Para comenzar a ingresar el ejemplo, encienda los dispositivos periféricos que tiene conectados a la computadora.
Inserte el Módulo de Comando del TI BASIC Extendido y encienda la computadora. Presione cualquier tecla para obtener la lista principal de selección. Presione 2 para seleccionar TI BASIC Extendido.

En adelante, los caracteres que mecanografía y las teclas que presione están SUBRAYADAS.

PROGRAMA DEL "NUMERO SECRETO"

COMENTARIOS

Numera automáticamente las líneas del programa.

Reserva lugar para los códigos y los intentos, genera códigos al azar.

Limpia la pantalla, emite un "beep" y pone el título "Número secreto" en la fila 11 columna 9.

REDO repite lo que se hizo antes de dar **ENTER**. Usando las teclas de edición **FCTN 2** (INSERT), **FCTN 1** (DELETE), y las flechas cambie la línea 130 a 140 DISPLAY AT (19,1) BEEP: "CANTIDAD DE CODIGOS (1-8)".

Suena un "beep" y coloca "Número de códigos? (1-8) en la fila 19 columna 1.

Presione **FCTN 8** (REDO) nuevamente. Ahora cambie la línea 140 a: 150 DISPLAY AT (21,6) Beep: "Dígitos de cada código?"

Suena un "beep" y coloca "Dígitos de Cada Código?" en la fila 21 columna 6.

Acepta en CODIGOS una entrada que sea solamente de dígitos, en la fila 19 columna 24.

Cambie la línea 160 a: 170

Accept at (21,24)

Validate (digit): Dígitos.

Acepta en DIGITOS una entrada en la fila 21 columna, 24 permitiendo que se ingresen sólo dígitos.

DISPLAY

```
*READY*
> NUM.                                  ENTER
> 100 REM NUMERO SECRETO                ENTER
> 110 DIM CODIGO$(8), INTEN
  TO $(8)                               ENTER
> 120 RANDOMIZE
> 130 DISPLAY AT (11,9) BEEP
  ERASE ALL; "NUMERO SEC
  RETO"                                 ENTER
> 140
                                        FCTN 8

140 DISPLAY AT (19, 1) BEEP:
"CANTIDAD DE CODIGOS?(1-8)"             ENTER

>                                       FCTN 8

150 DISPLAY AT (21,6) BEEP:
"DIGITOS DE CADA CODIGO?"               ENTER

160 ACCEPT AT (19,24)
VALIDATE (DIGIT): CODIGOS               ENTER

>                                       FCTN 8

170 ACCEPT AT (21,24) VALI-
DATE (DIGIT): DIGITOS                   ENTER
```

Muestra el programa tal como se lo ha ingresado.

```
> LIST
100 REM NUMERO SECRETO
110 DIM CODIGO$(8),INTENTO$(8)
120 RANDOMIZE
130 DISPLAY AT (11,9) BEEP ERASE ALL:
"NUMERO SECRETO"
140 DISPLAY AT (19,1) BEEP:
"CANTIDAD DE CODIGOS? (1-8)"
150 DISPLAY AT (21,6) BEEP: "DIGITOS
DE CADA CODIGO?"
160 ACCEPT AT (19,24) VALIDATE
(DIGIT): CODIGOS
170 ACCEPT AT (21,24) VALIDATE
(DIGIT): DIGITOS
```

Ejecuta el programa
La pantalla se limpia y aparece:

```
> RUN

NUMERO SECRETO
CANTIDAD DE CODIGOS? (1-8) ■
DIGITOS DE CADA CODIGO?
```

Ingrese cualquier cosa que no sea un dígito. Suena un "beep" y la computadora lo rechazará.

Ingrese 4. El cursor mueve hacia abajo, a la segunda pregunta.
Ingrese 10. El programa finaliza y se puede seguir ingresando.

Los números de línea comienzan con 180.

Verifica para ver si hay suficientes dígitos para la cantidad de códigos, Si CODIGOS es menor o igual que dígitos, el control pasa

a la próxima línea.
Si CODIGOS es mayor que DIGITOS se muestra el mensaje "HAY MAS CODIGOS QUE DIGITOS en la última línea de la pantalla y el control vuelve a la línea 160 nuevamente.

```
* READY *
> NUM 180
> 180 IF CODIGO > DIGITOS
THEN DISPLAY AT (24,2) BEEP:
"HAY MAS CODIGOS QUE
DIGITOS" : : GOTO 160          ENTER
```

Comienza el ciclo para elegir los códigos. Las palabras luego Elige códigos al azar	> 190 FOR A = 1 TO CODIGOS !ELECCION DE CODIGOS **ENTER** > 200 CODIGO$ (A) =5 STR$ (INT (RND *DIGITOS) **ENTER**
Comienza el ciclo para evitar código duplicados	> 210 FOR B= 0 TO A – 1 !VERIFICACION DE DUPLI-CADOS **ENTER**
Verifica duplicados y elige un nuevo código si los hay	> 220 IF CODIGO$ (A) = CODIGO$ (B) THEN 200 **ENTER**
Termina el ciclo de verificación de duplicados.	> 230 NEXT B **ENTER**
Termina el ciclo de elección de códigos.	> 240 NEXT A **ENTER**
Define una variable para tener control del lugar de la pantalla donde se va mostrando la información.	> 250 FILA = 2 **ENTER**
Limpia la pantalla y muestra el encabezamiento	> 260 DISPLAY AT (1,1) ERASE ALL: "INTENTO CORRECTO Y LUGAR" **ENTER**
REDO línea 260 se cambia a: 270 DISPLAY AT (24,3): "INGRESE N COMO SOLUCION".	> 270 **FCTN 8**
Muestra una instrucción en la parte inferior de la pantalla. Los números de línea comienzan en 280	> 270 DISPLAY AT (24,3): "INGRESE X COMO SOLUCION" **ENTER** NUM 280 **ENTER**
Acepta el intento en la fila apropiada.	> 280 ACCEPT AT (FILA,1): C$ **ENTER**
Verifica si abandona el juego o continúa.	> 290 IF C$ = "X" THEN 470 **ENTER**
Comienza el ciclo que fracciona el intento para acercarse a la respuesta.	> 300 FOR D = 1 TO CODIGO! FRACCIONA EL INTENTO **ENTER**
Separa el intento en dígitos individuales.	> 310 INTENTO (D) = SEG$ (C$, D, 1) **ENTER**
Completa el ciclo para separar el intento.	> 320 NEXT D **ENTER**
Pone cero en CORRECTO Y LUGAR	> 330 CORRECTO, LUGAR 0 **ENTER**
Comienza un ciclo externo para verificar el intento contra el código.	> 340 FOR E = 1 TO CODIGOS! VERIFICA LA CORRECCION DEL INTENTO **ENTER**
Comienza un ciclo interno para verificar el intento.	> 350 FOR F = 1 TO CODIGOS **ENTER**

Si el intento no coincide con el código, va a la línea siguiente.	> 360 IF CODIGO$ (E) = INTENTO$ (F) THEN CORRECTO = CORRECTO + 1 : :	
Si coincide la suma 1 al número correcto. Luego si el intento está en el lugar correcto, suma 1 al lugar.	IF E = F THEN LUGAR = LUGAR + 1	**ENTER**
Completa el ciclo interno.	> 370 NEXT F	**ENTER**
Completa el ciclo externo.	> 380 NEXT E	**ENTER**
Muestra la cantidad de dígitos acertados.	> 390 DISPLAY AT (FILA,14): CORRECTO	**ENTER**
REDO línea 390 se transforma en 400 DISPLAY AT (FILA, 22): LUGAR Muestra qué cantidad de dígitos están en una posición acertada.	> 400 400 DISPLAY AT (FILA, 22): LUGAR	**FCTN 8**
Los números de línea comienzan en la 410	> NUM 410	**ENTER**
Verifica si se ha resuelto el código. Si es así va a la línea siguiente. Si no suma 1 a la fila. Luego si la fila es mayor que 22, va a la línea 470 y da la solución. De otro modo va a al línea 280 y acepta otro código.	> 410 IF LUGAR < > CODIGOS THEN FILA = FILA + 1 : : IF FILA > 22 THEN 470 ELSE 280	**ENTER**
Muestra el mensaje al ganador indicando el número de intentos en la fila 23 columna 1.	> 420 DISPLAY AT (23,1) BEEP: "GANO CON"; FILA - 1; "INTENTOS"	**ENTER**
REDO línea 420 se transforma en: 430 DISPLAY AT (24,1) BEEP: "JUEGA DE NUEVO: (S/N) S".	> 430	**FCTN 8**
Muestra el mensaje en la fila 24 columna 1	> 430 DISPLAY AT (24,1) > BEEP: "JUEGA DE NUEVO? (S/N) S"	**ENTER**
Los números de línea comienzan en 440.	> NUM 440	**ENTER**
Acepta la entrada en X$ en la fila 24 columna 19. No remueve ningún caracter que ya está allí; acepta solo un caracter, emite un "beep" y acepta sólo S ó N.	> 440 ACCEPT AT (24,19) SIZE (-1) BEEP VALIDATE ("SN") : X $	**ENTER**

Al presionar ENTER en este punto, cuando el programa se está ejecutando, confirma la S que se mostró en la línea 430.		
Si se eligió S, vuelve a la línea 190 y elige un nuevo código para otro juego.	> 450 IF X $ = "S" THEN 190	**ENTER**
Detiene el programa.	> 460 STOP	**ENTER**
Coloca el mensaje EL CODIGO ES	> 470 DISPLAY AT (23,1) BEEP: "EL CODIGO ES"	**ENTER**
Comienza el ciclo para mostrar los dígitos.	> 480 FOR G = 1 TO CODIGOS	**ENTER**
Muestra los dígitos	> 490 DISPLAY AT (23,12 ≤ G) CODIGO$ (G)	**ENTER**
Termina el ciclo.	> 500 NEXT G	**ENTER**
Deja el modo number	> 510	**ENTER**
Presione para obtener la línea 430 y poder usar **FCTN 8 (REDO)**	> 430 430 DISPLAY AT (24,1) BEEP: "JUEGA DE NUEVO? (S/N) S"	**FCTN X** **ENTER**
Presione REDO. La línea 510 es duplicada de la 430	>	**FCTN 8**
Propone un nuevo juego en la fila 24 columna 1	510 DISPLAY AT (24,1) BEEP: "JUEGA DE NUEVO? (S/N) S"	**ENTER**
Para poder usar la línea 440	> 440 440 ACCEPT AT (24,19) SIZE (-1) BEEP VALIDATE ("SN"): X$	**FCTN X** **ENTER**
Presione REDO porque la línea 520 es duplicada de la 440	>	**FCTN 8**
Actúa como la línea 440	520 ACCEPT AT (24,19) SIZE (-1) BEEP VALIDATE ("SN"):	**ENTER**
Si se ingresó S, vuelve a la línea 130, permite cambiar la número de dígitos en un código y el número de dígitos aceptable, y comienza un nuevo juego.	> 530 IF X $ = "S" THEN 130	**ENTER**

Antes de ejecutar un programa habrá que revisarlo. Aquí hay un listado completo para verificar contra su programa.

```
100 REM NUMERO SECRETO
110 DIM CODIGO$(8), INTENTO$(8)
120 RANDOMIZE
130 DISPLAY AT (11,9) BEEP ERASE ALL: "NUMERO
SECRETO"
140 DISPLAY AT (19,1) BEEP: "CANTIDAD DE
CODIGOS? (1 —8)"
150 DISPLAY AT (21,6) BEEP: "DIGITOS DE CADA
CODIGO?"
160 ACCEPT AT (19,24) VALIDATE (DIGIT): CODIGOS
170 ACCEPT AT (21,24) VALIDATE (DIGIT): DIGITOS
180 IF CODIGOS > DIGITOS THEN DISPLAY AT (24,2)
BEEP: "HAY MAS CODIGOS QUE DIGITOS" : : GOTO
160
190 FOR A = 1 TO CODIGOS !ELECCION DE CODIGOS
200 CODIGO$ (A) = STR$ (INT (RND * DIGITOS) )
210 FOR B = 0 TO A-1 !VERIFICACION DE
DUPLICADOS
220 IF CODIGO$ (A) = CODIGO$ (B) THEN 200
230 NEXT B
240 NEXT A
250 FILA = 2
260 DISPLAY AT (1,1) ERASE ALL: INTENTO
CORRECTO LUGAR"
270 DISPLAY AT (24,3): "INGRESE X COMO
SOLUCION"
280 ACCEPT AT (FILA,1): C$ 290 IF C$ = "X" THEN 470
300 FOR D = 1 TO CODIGOS !FRACCIONA EL INTENTO
```

```
310 INTENTO$(D) = SEG$ (C$, D, 1)
320 NEXT D
330 CORRECTO, LUGAR = 0
340 FOR E = 1 TO CODIGOS !VERIFICA LA
CORRECCION DEL INTENTO
350 FOR F = 1 TO CODIGOS
360 IF CODIGO$(E) = INTENTO$(F) THEN CORRECTO =
CORRECTO + 1 : : IF E = F THEN LUGAR = LUGAR + 1
370 NEXT F
380 NEXT E
390 DISPLAY AT (FILA, 14): CORRECTO
400 DISPLAY AT (FILA, 22): LUGAR
410 IF LUGAR <> CODIGOS THEN FILA = FILA ≤ 1 : : IF
FILA > 22 THEN 470 ELSE 280
420 DISPLAY AT (23,1) BEEP: "GANO CON";FILA - 1;
"INTENTOS"
430 DISPLAY AT (24,1) BEEP: "JUEGA DE NUEVO?
(S/N) S"
440 ACCEPT AT (24,19) SIZE (-1) BEEP VALIDATE
("SN") : X$
450 IF X $ = "S" THEN 190
460 STOP
470 DISPLAY AT (23,1) BEEP: "EL CODIGO ES"
480 FOR G = 1 TO CODIGOS
490 DISPLAY AT (23,12 ≤ G): CODIGO$ (G)
500 NEXT G
510 DISPLAY AT (24,1) BEEP: "JUEGA DE NUEVO?
(S/N) S"
520 ACCEPT AT (24,19) SIZE (-1) BEEP VALIDATE
("SN"): X
530 IF X$ = "S" THEN 130
```

Ahora ejecute el programa mecanografiando RUN y presionando ENTER. Elija 4 códigos con 10 dígitos (0,1,2,3,4,5,6,7,8,9) posibles cada uno. Si adivina el código en 6 intentos es excelente. En ocho es muy bueno.

Si desea usar el programa nuevamente almacénelo en cassette o en diskette. Para almacenarlo en cassette asegúrese que el grabador está enchufado. Luego ingrese SAVE CS1 y siga las instrucciones de la pantalla.

Para almacenarlo en diskette ingrese SAVE DSK1. "nombre-archivo" con el nombre que eligió para su programa.

Luego de almacenar el programa, si ya no desea usarlo ingrese NEW. Se remueve el programa y puede ingresar otro.

Si almacenó el programa podrá volverlo a cargar en la computadora para volverlo usar. Desde cassette se usa OLD CS1 y desde diskette OLD DSK1. "nombre-archivo" con el nombre de archivo con que lo almacenó.

Cuando terminó de usar TI BASIC Extendido ingrese BYE para volver a la pantalla principal.

CAPITULO

3

Convenciones del TI BASIC Extendido

Este capítulo discute el formato que deben tener los programas escritos con TI BASIC Extendido y la forma en que funciona este lenguaje.

EJECUCION DE UN PROGRAMA DE ENCENDIDO

Si cuando se elige TI BASIC Extendido, el programa llamado LOAD se halla en el drive 1, este programa se carga y se ejecuta. El efecto es el mismo que si se hubiera mecanografiado RUN "DSK1.LOAD". Si el programa no existe habrá una pequeña demora mientras TI BASIC Extendido lo busca.

ARCHIVOS

Los archivos son grupos de datos colocados en dispositivos externos. Los archivos más comunes están en cassettes o diskettes, pero también se consideran archivos a los datos que se envían a través de dispositivos externos tales como la Interfase RS232 y la impresora térmica opcional.

NUMEROS DE LINEA

Los programas TI BASIC Extendido requieren números de línea.
Los números de línea especifican el orden en que se ejecutarán las líneas y se usan para identificar qué línea se ejecutará, con las instrucciones IF-THEN-ELSE, GOTO GOSUB, ON ERROR, ON ... GOTO y ON...GOSUB. Los números de línea también pueden usarse con BREAK, LIST, NUM, RESTORE, RETURN y RUN. Los números de línea pueden ser cualquier entero desde 1 hasta 32767.

La computadora genera automáticamente números de línea si se utiliza el comando NUM. Cuando no se ha indicado número de línea, el comando genera números a - partir de 100 con incrementos de 10 en 10. Se pueden resecuenciar los números de línea con el comando RES.

LINEAS

Las líneas pueden tener hasta 140 caracteres de longitud incluyendo el número de línea y los espacios. Si ha llegado al final de la línea, los caracteres adicionales que ingrese reemplazarán al caracter 140° . Es posible construir una línea de más de 140 caracteres en el Modo Edit. usando FCTN 2 (INSERT).

SIMBOLOS ESPECIALES

Cos símbolos especiales separan instrucciones y comentarios en una misma línea. Una línea de TI BASIC Extendido está formada por: un número de línea, una o más instrucciones y un comentario opcional.
Por ejemplo:
100 FOR A = 1 TO 100 :: PRINT A; SQR (A) :: NEXT A! IMPRESION DE RAICES CUADRADAS.

El símbolo separador de instrucciones :: se usa para separar instrucciones dentro de una misma línea. El signo de admiración! se usa para separar un comentario explicatorio del resto de la línea. Los comentarios no se ejecutan cuando se ejecuta el programa.

ESPACIOS

TI BASIC Extendido requiere espacios entre los elementos que conforman las instrucciones para poder distinguir los nombres de variables de los elementos del lenguaje propiamente dicho. Sin embargo no se requieren espacios antes o después de símbolos relacionados, o antes o después del símbolo !, o de un símbolo separador de instrucciones. Se pueden insertar espacios extra cuando se ingresan comandos e instrucciones, pero TI BASIC Extendido los eliminará. Cuando se listen programas, TI BASIC Extendido puede agregar espacios alrededor del símbolo ! o de un símbolo separador de instrucciones.

CONSTANTES NUMERICAS

Se pueden ingresar constantes numéricas con cualquier cantidad de dígitos. Sin embargo la computadora las redondeará a 13 ó 14 dígitos debido al método interno de almacenamiento que usa. Y generalmente los mostrará con un máximo de 10 dígitos. Para números extremadamente grandes o pequeños, se aconseja usar notación científica para su ingreso. La computadora, normalmente usa este tipo de notación para imprimir números muy grandes o muy pequeños.

En notación científica, un amen) se da como una mantisa (un número con un lugar a la izquierda del punto decimal) multiplicada por 10 elevado a una potencia entera. En notación científica 15 se expresa como 1.5×10^1 . 150 se expresa como 1.5×10^2; - 1500 se expresa como - 1.5×10^3; 156789000000000 se expresa como 1.56789×10^{14} y 0.156789 se expresa como 1.56789×10^{-1} . El TI BASIC EXTENDIDO el " X 10" se representa por "E". Así 1.5×10^3 se transforma en 1.5E3.

Las constantes numéricas están definidas en el rango - 9.9999999999999E127 a - 1E – 128.0, y 1E - 128 a 9.9999999999999E127. Si el exponente de un número calculado es mayor que 99, se imprimirá ** en lugar del exponente. El exponente completo se guarda internamente y puede mostrarse con USING en una instrucción PRINT o DISPLAY.

CONSTANTES ALFANUMERICAS

Las constantes alfanuméricas pueden tener hasta una longitud igual al largo de una línea de entrada. Si la cadena está encerrada entre comillas, las comillas que están dentro de la cadena, se representarán mediante doble comillas.

VARIABLES

Los nombres de variable en TI BASIC Extendido pueden tener de 1 a 15 caracteres. El primer caracter de una variable debe ser una letra del alfabeto, el símbolo @ o una línea (-). Los caracteres subsiguientes pueden ser cualquiera de esos símbolos más los dígitos. El último caracter de una variable alfanumérica debe ser siempre un signo $.
Las variables pueden ser escalares o arreglos de hasta siete dimensiones.

CONVENCIONES DEL TI BASIC EXTENDIDO

Ciertas palabras están reservadas para use del TI BASIC Extendido. Son comandos, instrucciones, funciones y operadores que conforman el lenguaje.
Estas palabras no pueden usarse como nombres de variable pero sí pueden formar parte de un nombre de variable. A continuación se da una lista completa de las palabras reservadas para TI BASIC Extendido.

ABS	EDF	NUMBER	SEQUENTIAL
ACCEPT	ERASE	NUMERIC	SGN
ALL	ERROR	OLD	SIN
AND	EXP	ON	SIZE
APPEND	FIXED	OPEN	SQR
ASC	FOR	OPTION	STEP
AT	GO	OR	STOP
ATN	GOSUB	OUTPUT	STR$
BASE	GOTO	PERMANENT	SUB
BEEP	IF	PI	SUBEND
BREAK	IMAGE	POS	SUBEXIT
BYE	INPUT	PRINT	TAB
CALL	INT	RANDOMIZE	TAN
CHR$	INTERNAL	READ	THEN
CLOSE	LEN	REC	TO
CON	LET	RELATIVE	TRACE
CONTINUE	LINPUT	REM	UALPHA
COS	LIST	RES	UNBREAK
DATA	LOG	RESEQUENCE	UNTRACE
DEF	MAX	RESTORE	UPDATE
DELETE	MERGE	RETURN	USING
DIGIT	MIN	RND	VAL
DIM	NEW	RPT$	VALIDATE
DISPLAY	NEXT	RUN	VARIABLE
ELSE	NOT	SAVE	WARNING
END	NUM	SEG$	XOR

Los siguientes son ejemplos de nombres de variables válidas:
NUMERICAS: X, A9, ALPHA, BASE-PAY, V(3), T(X, Y, Z, Q, A, R, P6)
TABLE (Q37, M/4)
ALFANUMERICAS: S$, YZ2$, NAME$, Q5$ (X, 7, L/Z), ADRESS$(4)

EXPRESIONES NUMERICAS

Las expresiones numéricas se construyen con constantes numéricas, variables numéricas y funciones. usando los operadores aritméticos para suma (+), resta (-), multiplicación (*), división (/) y exponenciación (^).
El signo menos (-) puede usarse ya sea como signo de sustracción o como un menos unario. De esta forma, invierte el signo del número que lo sigue. Por ejemplo, - 3 ^ 2 es igual a - 9 dado que se considera - (3 ^ 2).
La jerarquía normal para evaluar una expresión numérica es: exponenciación, seguido de multiplicación y división, y luego por suma y resta. Sin embargo cualquier parte de una expresión numérica que esté encerrada entre paréntesis, se evalúa primero. Las siguientes expresiones muestran el efecto de los paréntesis en la evaluación de la expresión:

Expresión	*Resultados Intermedios*		*Valor Final*
4 + 2 ^ 2 / 2 - 6	4 + 4 / 2 - 6	4 + 2 - 6	0
(4 + 2) ^ 2 / 2 - 6	6 ^ 2 / 2 - 6	36 / 2 - 6	12
4 + 2 ^ 2 / (2 - 6)	4 + 4 / (- 4)	4 - 1	3

EXPRESIONES ALFANUMERICAS

Las expresiones alfanuméricas se construyen a partir de constantes y variables alfanuméricas y referencias a funciones, usando la operación de concatenación (&) para combinar las cadenas. Si se excede la longitud de 255 caracteres, se truncan los caracteres de la derecha y se produce un mensaje de error. El siguiente es un ejemplo de concatenación:

```
100 A$ = "HI" & "THERE! "
```

A$ = "HI" & " THERE!" define A$ igual a "HI THERE! "

EXPRESIONES RELACIONALES

Las expresiones relacionales se usan frecuentemente en la instrucción IF-THEN-ELSE, pero también pueden aparecer en cualquier lugar donde estén permitidas las expresiones numéricas. Una expresión relacional tiene el valor —1 si es verdadera y 0 si es falsa.
Los operadores relacionales se procesan de izquierda a derecha, luego que se han completado las operaciones aritméticas y antes de que se realice ninguna concatenación (operador &).
Las expresiones relacionales son:

Igual a (=)
Menor que (<)
Mayor que (>)
No igual a (< >)
Menor o igual que (< =)
Mayor o igual que (> =)

Los siguientes ejemplos ilustran el use de las expresiones relacionales.

IF X < Y THEN 200 ELSE GOSUB 420. Si X es menor que Y ejecuta la instrucción 200. Si X es mayor o igual que Y se ejecuta GOSUB 420.	> 100 IF X < Y THEN 200 ELSE GOSUB 420
IF L(C) = 12 THEN C = S + 1 ELSE COUNT = COUNT + 1 :: GOTO 140 Si L(C) es igual a 12 hace C = S + 1. Si L(C) no es igual a 12 suma 1 a COUNT y lo almacena en COUNT y luego ejecuta la instrucción 140.	> 100 IF L (C) = 12 THEN C = S + 1 ELSE COUNT = COUNT + 1 ::GOTO 140
A = 2 < 5 hace A = -1 por ser verdadero que 2 es menor que 5	> 100 A = 2 < 5
PRINT “THIS” = “THAT” imprime 0 si no es verdad que “THIS” es igual a “THAT”	> 100 PRINT "THIS" = "THAT"
A = B = 7 hace A = -1 si B es igual a 7, y 0 si B no es igual a 7. No tiene efecto sobre B. Note que esta expresión no tiene el significado aritmético usual de A = B = 7.	> 100 A = B = 7

EXPRESIONES LOGICAS

Las expresiones lógicas se usan junto con las expresiones relacionales. Los operadores lógicos son AND, OR, NOT, y XOR. Si son verdaderas, las expresiones lógicas toman un valor —1. Si son falsas toman un valor 0. El orden de precedencia para las expresiones lógicas de mayor a menor es NOT, XOR, AND y OR.

Una expresión lógica que usa AND es verdadera si sus cláusulas derecha e izquierda son ambas verdaderas.

Una expresión lógica que usa OR es verdadera, si su cláusula izquierda es verdadera, o si su cláusula derecha es verdadera, o si ambas lo son.

Una expresión lógica que usa NOT es verdadera si la cláusula que la sigue no lo es.

Una expresión lógica que usa XOR (or excluyente) es verdadera. Si su cláusula derecha es verdadera o si su cláusula izquierda lo es, pero no si ambas son verdaderas.

Los siguientes ejemplos ilustran el uso de las expresiones lógicas:

IF 3 < 4 AND 5 > 6 THEN L = 7 hace L = 7 dado que 3 es menor que 4 y 5 es menor que 6	100 IF 3 < 4 AND 5 < 6 THEN L = 7
IF 3 < 4 AND 5 > 6 THEN L = 7 no define L = 7 dado que 3 es menor que 4 pero 5 no es mayor que 6	100 IF 3 < 4 AND 5 > 6 THEN L = 7
IF 3 < 4 OR 5 > 6 THEN L = 7 define L= 7 porque 3 es menor que 4	100 IF 3 < 4 OR 5 > 6 THEN L= 7
IF 3 < 4 XOR 5 > 6 THEN L= 7 hace L= 7 porque 3 es menor que 4 pero 5 no es mayor que 6	100 IF 3 < 4 XOR 5 < 6 THEN L = 7
IF 3 < 4 XOR 5 < 6 THEN L = 7 no hace L= 7 porque 3 es menor que 4 y 5 es menor que 6	100 IF 3 < 4 XOR 5 < 6 THEN L = 7
IF NOT 3 = 4 THEN L = 7 define L= 7 porque 3 no es igual a 4	100 IF NOT 3 = 4 THEN L = 7
IF NOT 3 = 4 AND (NOT 6 = 5 XOR 2 = 2) THEN 200 no pasa el control a la línea 200 porque mientras que es verdad que es verdad que 3 no es igual a 4, es verdad también que 6 no es igual a 5 y que 2 es igual a 2, por lo tanto la cláusula entre paréntesis no es verdadera.	100 IF NOT 3 = 4 AND (NOT 6 = 5 XOR 2 = 2) THEN 200
IF (A OR B) AND (C XOR D) THEN 200 pasa el control a la línea 200 si A o B, o ambas A y B son verdaderas (igual a-1), y C o D, pero no C y D son verdaderas (igual a -1)	100 IF (A OR B) AND (C XOR D) THEN 200

Los operadores lógicos pueden usarse directamente sobre números. Convierten los números a notación binaria, realizan la operación requerida a nivel de bit, y luego convierten el resultado nuevamente a la representación decimal. Se podrá encontrar una discusión más detallada sobre el uso de los operadores lógicos con números en un texto de matemáticas o ingeniería que trata de lógica.
Los números deben estar entre —32768 y 32767. Representados en notación binaria entre 1000000000000000 y 0111111111111111.
Los números negativos se indican como complemento de 2 con un 1 en el bit más significativo. En notación binaria, cada lugar corresponde a una potencia de 2 en lugar de una potencia de 10 como en la notación decimal.
El cuadro siguiente muestra algunos números escritos en ambas notaciones.

	LUGAR DECIMAL		
-	100	10	1
	0	0	1
	0	0	6
	0	2	5
-	0	1	3

	LUGAR BINARIO														
-	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1

Lo anterior es equivalente a:

$1_{10} = 0000000000000001_2 = 1_2$ $\quad 25_{10} = 0000000000011001_2 = 11001_2$

$6_{10} = 0000000000000110_2 = 110_2$ $\quad -13_{10} = 1111111111110011_2$

AND pone un 1 en la posición binaria correspondiente si hay un 1 en ambos números, el que lo precede y el que lo sigue. De otro modo pone cero.
OR pone un 1 en la posición binaria correspondiente si hay un 1 en cualquiera de los dos números, en el que lo precede, en el que lo sigue, o en ambos. De otro modo pone cero.
XOR pone un I en la posición binaria correspondiente si hay un 1 en alguno de los números, el que lo precede o el que lo sigue, pero no en ambos. De otro modo pone cero.
NOT pone un 1 en la posición binaria correspondiente si hay un cero en el número que lo sigue. De otro modo pone cero.
Los ejemplos siguientes ilustran el resultado de los operadores lógicos usados con números:

	DECIMAL	BINARIO		DECIMAL	BINARIO
A:	1	0000000000000001	A:	1	0000000000000001
B:	2	0000000000000010	B:	3	0000000000000011
A AND B:	0	0000000000000000	A AND B:	1	0000000000000001
	6	0000000000000110	A:	47	0000000000101111
B:	5	0000000000000101	B:	62	0000000000111110
A AND B:	4	0000000000000100	A AND B:	46	0000000000101110

	DECIMAL	BINARIO		DECIMAL	BINARIO
A:	1	0000000000000001	A:	1	0000000000000001
B:	2	0000000000000010	B:	3	0000000000000011
A OR B:	3	0000000000000011	A OR B:	3	0000000000000011
A:	6	0000000000000110	A:	47	0000000000101111
B:	5	0000000000000101	B:	62	0000000000111110
A OR B:	7	0000000000000111	A OR B:	63	0000000000111111

	DECIMAL	BINARIO		DECIMAL	BINARIO
A:	1	0000000000000001	A:	1	0000000000000001
B:	2	0000000000000010	B:	3	0000000000000011
A XOR B:	3	0000000000000011	A XOR B:	2	0000000000000010
A:	6	0000000000000110	A:	47	0000000000101111
B:	5	0000000000000101	B:	62	0000000000111110
A XOR B:	3	0000000000000011	A XOR B:	17	0000000000010001

	DECIMAL	BINARIO		DECIMAL	BINARIO
A:	1	0000000000000001	A:	2	0000000000000010
NOT A:	-2	1111111111111110	NOT A:	-3	1111111111111101
A:	6	0000000000000110	A:	47	0000000000101111
NOT A:	-7	1111111111111001	NOT A:	-48	1111111111010000

CAPITULO

4

Sección de Referencia

Este capítulo es una lista alfabética de todos los comandos, instrucciones y funciones del TI BASIC Extendido, con una explicación detallada de cómo trabaja cada uno. Se incluyen ejemplos y programas ejemplo para mayor claridad.

En el formato de los elementos, las palabras clave aparecen en MAYUSCULAS. Las variables están en itálicas. Las porciones opcionales están entre [corchetes] . Los ítems que se pueden repetir se indican mediante puntos suspensivos (. . .). Las formas alternativas se presentan una sobre otra.

El Apéndice A contiene una lista de los programas ilustrativos. El índice da las páginas en las que se usa cada elemento del TI BASIC Extendido en un programa ilustrativo.

ABS

Formato: ABS (expresión-numérica)

Descripción:

La función ABS da el valor absoluto de una expresión-numérica.
Si la expresión-numérica es positiva, ABS da el valor de la expresión. Si la expresión-numérica es negativa, ABS da su negativo (un valor positivo). Si la expresión-numérica es cero, ABS devuelve un cero. El resultado de ABS es siempre un número no negativo.

Ejemplos:

PRINT ABS (42.3) imprime 42.3 > 100 PRINT ABS (42.3)
VV = ABS (-6.124) hace VV = 6.124 > 100 VV = ABS(-6.124)

ACCEPT

CAPITULO

4

Formato:

ACCEPT [[AT (fila, columna)][VALIDATE(tipo-de-dato,…)][BEEP]
[ERASE ALL][SIZE(expresión-numérica)]:] variable

Descripción:

La instrucción ACCEPT detiene la ejecución del programa hasta que se ingresan datos desde el teclado. Hay muchas opciones disponibles para el ACCEPT, que lo hacen más versátil que el INPUT. Puede aceptar datos desde cualquier lugar de la pantalla, emitir un tono audible (beep) cuando está lista para aceptar los datos borrar todos los caracteres de la pantalla antes de aceptar los datos, limitar los datos aceptados a un cierto número de caracteres, y limitar el tipo de los caracteres aceptados.

Opciones:

Las siguientes opciones pueden aparecer en cualquier orden luego del ACCEPT.

AT(fila, columna): ubica el comienzo del campo especificado, en una fila y columna determinada. Las filas se numeran de 1 a 24 y las columnar de 1 a 28: donde la columna 1 se corresponde con la columna 3 de los subprogramas VCHAR, HCHAR y GCHAR.

VALIDATE (tipo-de-dato…) permite solo el ingreso de ciertos caracteres. El tipo-de-dato especifica que caracteres son aceptables. Si se especifican más de un tipo-de-datos, se aceptará cualquier caracter perteneciente a esos tipos. Los siguientes son tipos de datos:

UALPHA permite todos los caracteres alfabéticos, en mayúsculas.
DIGIT permite de 0 a 9.
NUMERIC permite de 0 a 9, ".","+", "-", y "E".
Expresión-alfanumérica permite todos los caracteres contenidos en dicha expresión alfanumérica.

BEEP emite un tono breve que indica que la computadora está lista para aceptar la entrada.

ERASE ALL llena la pantalla completa con caracteres en blanco antes de aceptar la entrada.

SIZE (expresión-numérica) limita la cantidad de caracteres aceptados, al valor absoluto de la expresión-numérica. Si la expresión numérica es positiva, el campo en que se ingresarán los datos, se blanquearán antes de aceptar la entrada. Si la expresión-numérica es negativa el campo no se blanquea. Esto permite colocar un valor standard en el campo e ingresarlo sólo presionando ENTER. Si hay cláusula SIZE, la línea se blanquea desde la posición de comienzo, hasta el final de la línea.

Si se usa la instrucción ACCEPT sin la cláusula AT, los dos últimos caracteres de la pantalla (esquina inferior derecha) se cambian a "Caracteres de borde" (código ASCII 31).

Ejemplos:

ACCEPT AT (5,7): Y acepta datos en la quinta fila, séptima columna de la pantalla, en la variable Y

```
> 100 ACCEPT AT (5,7): Y
```

ACCEPT VALIDATE ("YN"): R$ acepta Y ó N en la variable R$

```
> 100 ACCEPT VALIDATE ("YN"): R$
```

ACCEPT ERASE ALL: B acepta datos en la variable B luego de poner blanco en todas las posiciones de la pantalla.

```
> 100 ACCEPT ERASE ALL: B
```

ACCEPT AT (R, C) SIZE (FIELDLEN) BEEP VALIDATE (DIGIT, "AYN"): X$ acepta un dígito o las letras A, Y ó N en la variable X.$. La longitud del campo puede ser de hasta FIELDLEN caracteres. El dato se acepta en la fila R, columna C y se produce un "beep" antes de aceptar los datos.

```
> 100 ACCEPT AT (R, C) SIZE (FIELD LEN) BEEP VALIDATE (DIGIT, "AYN"): X $
```

Programa:

El programa de la derecha muestra un uso típico del ACCEPT.
Permite la entrada de hasta 20 nombres y direcciones, y luego los muestra juntos en la pantalla.

```
> 100 DIM NOM$ (20), DIREC$(20)
> 110 DISPLAY AT (5,1) ERASE ALL : "NOMBRE:"
> 120 DISPLAY AT (7,1): "DIREC-CION:"
> 130 DISPLAY AT (23,1): "MECA-NOGRAFIE UN ? PARA FINALIZAR LA ENTRADA".
> 140 FOR S = 1 TO 20
> 150 ACCEPT AT (5,7) VALIDATE (UALPHA "?") BEEP SIZE (13): NOM$ (5)
> 160 IF NOM$ (S) = `?" THEN 200
> 170 ACCEPT AT (7,10) SIZE (12): DIREC$ (S)
> 180 DISPLAY AT (7,10): "
```

```
> 190 NEXT S
> 200 CALL CLEAR
> 210 DISPLAY AT (1,1): "NOMBRE",
  "DIRECCION"
> 220 FOR T = 1 TO S-1
> 230 DISPLAY AT(T+2.1): NOM$ (T),
  DIREC$ (T)
> 240 NEXT T
> 250 GOTO 250
```

(Presione **FCTN 4** para detener el programa).

ASC

Formato:

ASC (expresión-alfanumérica)

Descripción:

La función ASC devuelve el código del carácter ASCII que corresponde al primer caracter de la expresión. En el Apéndice C se da una lista de los códigos ASCII. La función ASC es la inversa de la función CHR$.

Ejemplos:

PRINT ASC ("A") imprime 65	100 PRINT ASC ("A")
B = ASC ("1") da a B el valor 49	100 B = ASC ("1")
DISPLAY ASC ("HOLA") muestra el valor 72.	100 DISPLAY ASC ("HOLA")

ATN

Formato:

ATN (expresión-numérica)

Descripción:

La función ATN devuelve la medida del ángulo (en radianes) cuya tangente es el valor de la expresión-numérica. Si desea el equivalente del ángulo en grados, multiplíquelo por 180/PI. El valor dado por la función ATN está siempre en el rango - PI/2 < ATN (X) < PI/2.

Ejemplos:

PRINT ATN(0) imprime 0

```
> 100 PRINT ATN (0)
```

Q = ATN(.44) asigna a Q es valor .4145068746

```
> 100 Q = ATN (.44)
```

BREAK

Formato:

BREAK Lista-de-número-de-línea

Descripción:

El comando BREAK requiere una lista-de-número-de-línea. Hace que el programa se detenga inmediatamente antes que se ejecuten las líneas que figuran en la lista.
Luego que ocurre un "breakpoint" (interrupción) a raíz de que ese número de línea forma parte de la lista-de-números-de-línea, este "breakpoint" se remueve y no volverá a tener efecto sobre esa línea a menos que se dé un nuevo comando BREAK o una nueva instrucción BREAK.

La instrucción BREAK sin número de línea hace que el programa se detenga cuando encuentra dicha instrucción. La línea en que se detiene el programa es llamada un 'breakpoint" (interrupción).
Cada vez que encuentra una instrucción BREAK sin número de línea, el programa se detiene aún si se ha ejecutado una instrucción ON BREAK NEXT.

También se puede provocar una interrupción en un programa presionando la tecla **FCTN 4** (CLEAR) mientras el programa se está ejecutando, a menos que las interrupciones se estén manejando de otra manera mediante la acción ON BREAK.
BREAK es útil para encontrar porque un programa no se está ejecutando como uno espera. Cuando el programa se ha detenido, se pueden imprimir los valores de las variables para ver que está sucediendo.
Se puede ingresar cualquier comando o cualquier instrucción usada como comando. Si en cambio se edita el programa, no se puede retomar su ejecución con un CONTINUE

Una manera de remover los "breakpoint" establecidos con BREAK seguido de números de línea, es mediante el comando UNBREAK También si se ha definido un "breakpoint" en una línea de programa y se borra dicha línea, se remueve el "breakpoint". También se remueven los breakpoints cuando un programa se almacena con el comando SAVE.
Véase ON BREAK para ver las maneras de manejar los "breakpoints".
Cada vez que ocurre un "breakpoint", el conjunto de caracteres standard se recupera. Así los caracteres standard que se hayan redefinido con CALL CHAR, vuelven a los caracteres normales. Un "breakpoint" también recupera los colores standard, borra los sprites y define la magnificación de sprites con el valor standard de 1.

Opciones:

La lista-de-números-de-línea es opcional cuando se usa BREAK como instrucción, pero es obligatoria cuando se lo usa como comando. Cuando está presente, hace que el programa se detenga inmediatamente antes que se ejecuten las líneas que figuran en la lista. Luego que ocurre un "breakpoint" (interrupción) a causa de una línea que figura en la lista-de-números-de-línea, el "breakpoint" se remueve y no volverá a tener efecto sobre esa línea a menos que se dé un nuevo comando BREAK o una nueva instrucción BREAK.

Ejemplos:

BREAK como instrucción hace que se produzca una interrupción cuando se ejecuta la instrucción.	> 150 BREAK
BREAK 120, 130 como instrucción provoca interrupciones antes de la ejecución de los números de líneas listados.	> 110 BREAK 120, 130
BREAK 200, 300, 1105 como comando provoca interrupciones antes de la ejecución de los números de línea listados.	> BREAK 200, 300, 1105

BYE

Formato:

BYE

Descripción:

El comando BYE termina el TI BASIC Extendido y vuelve a la pantalla principal. Se cierran todos los archivos que estaban abiertos, se borran todas las líneas de programa y la computadora queda desactivada. Use siempre el comando BYE en lugar de FCTN + (QUIT) para dejar el TI BASIC Extendido. FCTN + (QUIT) no cierra los archivos, de lo que puede resultar que se pierdan datos de los dispositivos periféricos.

CALL

CAPITULO
4

Formato:

CALL nombre-de-subprograma [(lista-de-parámetros)
Descripción:
La instrucción CALL transfiere el control a nombre-de-subprograma. El subprograma puede ser uno de los provistos por TI BASIC Extendido, o bien un subprograma escrito por el usuario.
Luego que ha sido ejecutado el subprograma, se ejecuta la siguiente instrucción después del CALL. CALL puede ser tanto una instrucción como un comando cuando se trata de subprogramas provistos por TI BASIC Extendido, pero debe ser sólo una instrucción cuando se trata de subprogramas escritos por el usuario.

Opciones:

La lista-de-parámetros se define de acuerdo al subprograma que se está llamando. Algunos no requieren parámetros, otros sí los requieren y otros tienen parámetros opcionales. Cada uno de los subprogramas provistos por TI BASIC Extendido se discute en su propio capítulo en este manual. Los subprogramas que puede escribir el usuario se discuten en el Capítulo II bajo el título de subprogramas y SUB.
Los siguientes son nombres de subprogramas provistos por TI BASIC Extendido.

CHAR	HCHAR	PATTERN
CHARPAT	INIT	PEEK
CHARSET	JOYST	POSITION
CLEAR	KEY	SAY
COINC	LINK	SCREEN
COLOR	LOAD	SOUND
DELSPRITE	LOCATE	SPGET
DISTANCE	MAGNIFY	SPRITE
ERR	MOTION	VCHAR
GCHAR		VERSION

Programa:

Este programa ilustra el uso de CALL con un subprograma provisto (CLEAR) y el uso de un subprograma escrito por el usuario TIEMPO en la línea 120

```
100 CALL CLEAR
110 X = 4
120 CALL TIEMPO (X)
130 PRINT X
140 STOP
200 SUB TIEMPO(Z)
210 Z = Z* PI
220 SUBEND
RUN
- - - la pantalla se limpia
    12.56637061
```

CHAR Subprograma

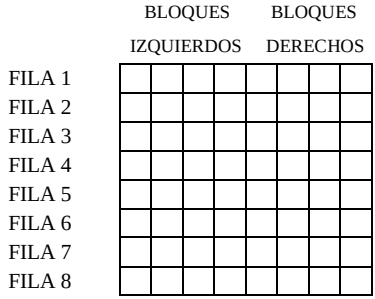
Formato:

CALL CHAR (código-caracter, identificador-de-modelo [, . . .])

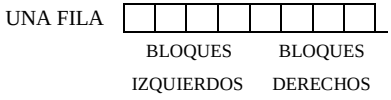
Descripción:

El subprograma CHAR le permite definir caracteres gráficos especiales. Se puede redefinir el conjunto standard de caracteres (códigos ASCII 32-95) y los caracteres no definidos, con códigos ASCII 96-143. Note que en TI BASIC Extendido hay nuevos caracteres definibles por programa que el TI BASIC donde se permitirán códigos ASCII de 96-156. El subprograma CHAR es el inverso del subprograma CHARPAT.
El código-de-caracter especifica el caracter que se desea definir y debe ser una expresión numérica con un valor entre 32 y 143.
El modelo-de-caracter es una cadena de 0 a 64 caracteres que especifica el modelo del / de los caracteres que se definen. Esta cadena es una representación codificada de los puntos que forman un caracter en la pantalla.

Cada caracter se compone de 64 puntos distribuidos en una grilla de 8 x 8 como se muestra abajo.



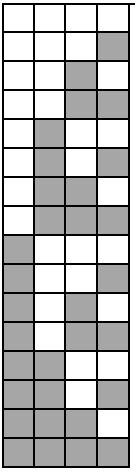
Cada fila está dividida en dos bloques de 4 puntos cada uno:



Cada caracter en el identificador-de-modelo, describe el modelo de un bloque de una fila. Las filas se describen de izquierda a derecha y de arriba hacia abajo. Así, los primeros dos caracteres del identificador de modelo describen el modelo para la fila 1 de la grilla, los dos siguientes, la segunda fila, y así sucesivamente.

SUBPROGRAMA CHAR

Los caracteres se crean "encendiendo" algunos puntos y "apagando" otros. El caracter "espacio" (ASCII 32) es el caracter donde todos los puntos están "apagados". El encendido de todos los puntos produce un bloque sólido. El color de los puntos "encendido" es el color de frente. El color en los puntos "apagados" es el color de fondo.
Todos los caracteres standard están definidos con los puntos apropiados "encendidos". Para crear un nuevo caracter hay que especificar que puntos quedan "encendidos" y cuales "apagados". En el código binario de la computadora se usa un número por cada 64 puntos para especificar que puntos están "encendidos" o "apagados" para un bloque especial. Una forma de lectura más accesible que la binaria es la hexadecimal. La tabla siguiente muestra todas las condiciones de "encendido"/"apagado" para los 4 puntos de un bloque dado y los códigos binario y hexadecimal para cada condición.

BLOQUES	Código Binario (0: apagado)(1: encendido)	Código Hexadecimal
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

Si el identificador-de-modelo es menor de 16 caracteres, la computadora asuma que los caracteres restantes son cero.
Si el identificador-de-modelo tiene de 17-32 caracteres se definen dos códigos de caracter, el primero con los primeros 16 caracteres y el segundo con los restantes, con adición de ceros si es necesario.
Si el identificador-de-modelo tiene de 33-48 caracteres, se definen 3 códigos de caracteres, el primero toma los primeros 16 caracteres, el segundo desde el 170 al 32^0 caracter y el tercero con los caracteres restantes con adición de ceros si es necesario.
Si el identificador-de-modelo tiene de 49-64 caracteres, se definen cuatro códigos de caracter; el primero toma los primeros 16 caracteres, el segundo desde el 170 al 32^0 caracter, el tercero desde el 33^0 al 48^0 caracter y el cuarto con los caracteres restantes y ceros adicionales si fuese necesario.
Si el identificador-de-modelo es más largo que 64 caracteres o es lo suficientemente largo como para definir caracteres con código mayor que 143, el exceso se ignora.

Programas:

Para describir el modelo de puntos dibujados a continuación se codificará una cadena de caracteres para el CALL CHAR:

"1898FF3D3C3CE404"

	BLOQUES IZQUIERDOS				BLOQUES DERECHOS			
FILA 1				■	■			18
FILA 2	■			■	■			98
FILA 3	■	■	■	■	■	■	■	FF
FILA 4			■	■	■	■		3D
FILA 5			■	■	■	■		3C
FILA 6			■	■	■	■		3C
FILA 7	■	■	■			■		E4
FILA 8						■		04

El programa de la derecha usa ésta y otra cadena de caracteres para hacer que la figura "baile".

```
>100 CALL CLEAR
>110 A$ = "1898FF3D3C3CE404'
>120 B$ = "1819FFBC3C3C2720"
>130 CALL COLOR (9, 7, 12)
>140 CALL VCHAR (12, 16, 96)
>150 CALL CHAR (96, A$)
>160 GOSUB 200
>170 CALL CHAR (96,B$)
>180 GOSUB 200
>190 GO TO 150
>200 FOR DELAY = 1 TO 50
>210 NEXT DELAY
>220 RETURN
>RUN
- - la pantalla se borra
- - el caracter se mueve (Presione
FCTN 4 para detener el programa)
```

Si un programa es detenido por un breakpoint. los caracteres ASCII predefinidos (códigos 32 a 95) vuelven a su modelo standard. Los que tienen códigos entre 96 y 143 conservan el modelo definido por el programa. Cuando el programa termina normalmente o a causa de un error, todos los caracteres predefinidos retoman su modelo.

```
>100 CALL CLEAR
>110 CALL CHAR (96,"FFFFFFFFF
  FFFFFFF")
>120 CALL CHAR (42, "OFOFOFOFO
  FOFOFOF")
>130 CALL HCHAR (12, 17, 42)
>140 CALL VCHAR (14, 17, 96)
>150 FOR DELAY = 1 TO 500
>160 NEXT DELAY
>RUN
```

CHARPAT Subprograma

CAPITULO **4**

Formato:

CALL CHARPAT (código-de-caracter, variable-alfanumérica [,…])

Descripción:

El subprograma CHARPAT devuelve en la variable alfanumérica el identificador de modelo de 16 caracteres que define el modelo de ese código-de-caracter. El subprograma CHARPAT es el inverso del Subprograma CHAR.
Véase el subprograma CHAR para una explicación del valor devuelto en la variable alfanumérica.

Ejemplo:

CALL CHARPAT (33.C$) hace C$ igual a "0010101010001000", el identificador de modelo para el caracter 33, el signo !

```
> 100 CALL CHARPAT (33,C$)
```

CHARSET Subprograma

Formato:

CALL CHARSET

Descripción:

El subprograma CHARSET recupera los modelos standard y los colores para los caracteres de 32 a 95. Generalmente cuando un programa se ejecuta desde otro programa usando RUN como instrucción, los caracteres 32-95 no vuelven a sus modelos y colores standard. CHARSET es útil cuando no se desea que pasa eso.

Ejemplo:

Recupera los caracteres standard y sus colores.

```
> 100 CALL CHARSET
```

CHR$

Formato:

CHR$ (expresión-numérica)

Descripción:

La función CHR$ devuelve el carácter correspondiente al código de carácter ASCII especificado por la expresión-numérica. La función CHR$ es la inversa de la función ASC. En el Apéndice C se da una lista de los códigos de caracter ASCII para cada caracter.

Ejemplos:

PRINT CHR$(72) imprime H.

```
> 100 PRINT CHR$ (72)
```

X$ = CHR$ (33) hace X$ igual a !

```
> 100 X$ = CHR$ (33)
```

Programa:

Para una lista completa de todos los caracteres ASCII y sus correspondientes valores ejecute el programa de la derecha.

```
> 100 CALL CLEAR
> 110 FOR A = 32 TO 95
> 120 PRINT A; " " " ";CHR$ (A);
    " " " ";
> 130 NEXT A
```

CLEAR Subprograma

CAPITULO **4**

Formato:

CALL CLEAR

Descripción:

El subprograma CLEAR se usa para limpiar (borrar) la pantalla completa. Cuando se llama a este programa, se coloca el caracter "espacios" (código ASCII 32) en todas las posiciones de la pantalla.

Programas:

Cuando se ejecuta el programa de la derecha la pantalla se limpia antes de ejecutarse los PRINT

```
> 100 CALL CLEAR
> 110 PRINT "HOLA"
> 120 PRINT "COMO ESTA?"
> RUN
  - - - la pantalla se borra
  HOLA
  COMO ESTA?
```

Si el caracter espacio (código ASCII 32) ha sido redefinido mediante el subprograma CALL CHAR la pantalla se llena con el nuevo caracter cuando se ejecuta CALL CLEAR

```
> 100 CALL CHAR (32,"0103070F1F
  3F7FFF")
> 110 CALL CLEAR
> 120 GO TO 120
> RUN
- - - la pantalla se llena con
 (presione FCTN 4 para
detener el programa).
```

CLOSE

Formato:

CLOSE # N^{o} -de-archivo [:DELETE]

Descripción:

La instrucción CLOSE hace que el programa deje de usar el archivo referenciado en # N^{o} -de-archivo. Después que se ejecute la instrucción CLOSE, el archivo no puede ser usado por el programa a menos que se lo abra (OPEN) nuevamente. La computadora ya no asocia el N^{o}-de-archivo con el archivo cerrado, de modo que se puede asignar ese número a otro archivo.
Cuando no se está ejecutando ningún programa, las siguientes acciones cierran todos los archivos abiertos:

Editar el programa
Ingresar el comando BYE
Ingresar el comando RUN
Ingresar el comando NEW
Ingresar el comando OLD
Ingresar el comando SAVE
Ingresar el comando LIST para un dispositivo.

Se utiliza **FCTN +** (QUIT) para dejar el TI BASIC Extendido, la computadora no cierra ningún archivo abierto y se pueden perder datos de esos archivos. Para evitar esta posibilidad, debe. abandonar TI BASIC Extendido con una instrucción BYE en lugar de **FCTN +** (QUIT).

Opciones:

Se puede borrar un archivo de un diskette al mismo tiempo que se lo cierra agregando ":DELETE" a la instrucción.
Otros dispositivos tales como grabadoras de cassette no admiten DELETE. El manual de cada dispositivo discute el use de DELETE.

Ejemplos:

Cuando la computadora realiza un close para un cassette, se reciben instrucciones sobre como operar el grabador

```
> 100 OPEN# 24: "CS1", INTERNAL, INPUT,
FIXED
●
●
●
- - - líneas de programa
●
●
●
> 200 CLOSE # 24
> RUN
- - - instrucciones de apertura
●
●
●
- - el programa se ejecuta
●
●
●
* PRESS CASSETTE STOP
THEN PRESS ENTER          CS1
```

La instrucción CLOSE para diskettes no requiere ninguna acción de su parte

```
> 100 OPEN# 24: "DSK1. MIDATO",
  INTERNAL, INPUT, FIXED
●
●
●
- - -líneas de programa
●
●
●
> 200 CLOSE # 24
> RUN
- - el programa se ejecuta.
```

COINC Subprograma

Formato:

CALL COINC (# Nº sprite,# Nº sprite, tolerancia, variable-numérica)
CALL COINC (# Nº sprite, punto-fila, punto-columna, tolerancia, variable-numérica)
CALL COINC (ALL, variable-numérica)

Descripción:

El subprograma COINC detecta una coincidencia entre un sprite y otro sprite o una posición de la pantalla. El valor devuelto en la variable numérica es -1 si hay coincidencia y 0 si no la hay.

Si se da la palabra clave ALL, se informa de la coincidencia de dos sprites cualesquiera. Si se identifica # No de sprite y posición en la pantalla, también se informa su coincidencia.

Si se da la palabra clave ALL se consideran sprites coincidentes sólo si uno o más puntos que los componen, ocupan la misma posición en la pantalla. Si se dan dos sprites o un sprite y una posición, debe especificarse también una tolerancia, y dos sprites se considerarán coincidentes si sus esquinas izquierdas superiores están dentro del valor especificado por la tolerancia. Un sprite y una posición son coincidentes si la esquina superior izquierda del sprite y la posición especificada por punto-fila y punto columna están dentro del valor especificado en tolerancia. Estas coincidencias se informan y no hay superposición aparente de los sprites o del sprite y la posición.

El punto-de-fila y punto-de-columna se renumeran consecutivamente comenzando con 1 en la esquina superior izquierda de la pantalla.

El punto de fila puede ir de 1 a 192 y el de columna va de 1 a 256. En realidad el punto de fila puede llegar a 256 pero las posiciones 193 a 256 caen fuera de la pantalla. Si cualquier parte del sprite ocupa una posición dada, entonces hay coincidencia.

La detección o no de la coincidencia depende de varias variables. Si los sprites se mueven muy rápidamente, COINC puede no llegar a detectar la coincidencia. También, COINC verifica la coincidencia sólo cuando se lo llama de modo que puede pasar que la coincidencia se dé cuando se está ejecutando alguna otra instrucción del programa.

SUBPROGRAMA COINC

Programa:

Este programa define dos sprites que consisten de un triángulo

```
> 100 CALL CLEAR
> 110 S$ = "0103070F1F3F7FFF"
> 120 CALL CHAR (96,S$)
> 130 CALL CHAR (100,S$)
> 140 CALL SPRITE (# 1,96, 7, 8, 8)
> 150 CALL SPRITE (# 2,100, 5, 1, 1)
```

La línea 160 muestra una coincidencia porque los sprites están a 10 puntos uno de otro.

```
> 160 CALL COINC (# 1,#2, 10,C)
> 170 PRINT C
```

La línea 180 muestra que no hay coincidencia porque las áreas sombreadas de los sprites no son coincidentes

```
> 180 CALL COINC (ALL, C)
> 190 PRINT C
> RUN
  -1
   0
```

COLOR Subprograma

Formato:

CALL COLOR (# N°-de-sprite, color-fondo [,…])
CALL COLOR (conjunto-de-caract, color-de-fondo, color-de-frente [,…])

Descripción:

El subprograma COLOR permite especificar ya sea un color de frente para un sprite o bien color de frente y color de fondo para un conjunto de caracteres. En un CALL COLOR dado se puede especificar color para un sprite o para un con-junto de caracteres pero no para ambos.

Cada caracter tiene dos colores. El color de los puntos que forman el caracter es llamado el color de frente. El color que ocupa el resto de la posición del caracter en la pantalla es el color de fondo. En los sprites, el color de fondo es siempre 1, transparente, que permite ver los caracteres a través del color de la pantalla. Para cambiar el color de la pantalla vea el subprograma SCREEN.
Tanto el color de frente como el color de fondo deben tener valores entre 1 y 16. Los códigos de color son:

CODIGO DECOLOR	COLOR
1	Transparente
2	Negro
3	Verde Mediano
4	Verde Claro
5	Azul Oscuro
6	Azul Claro
7	Rojo Oscuro
8	Ciánico (Azul)
9	Rojo Mediano
10	Rojo Claro
11	Amarillo Oscuro
12	Amarillo Claro
13	Verde Oscuro
14	Magenta
15	Gris
16	Blanco

Hasta que se ejecuta un CALL COLOR, el color de frente standard es negro (código 2) y el color de fondo standard es transparente (código 1) para todos los caracteres. Los sprites tienen el color que se les asignó cuando fueron creados..Cuando ocurre un "breakpoint" todos los caracteres vuelven a sus colores standard.

SUBPROGRAMA COLOR

Para usar el subprograma CALL COLOR se debe especificar también a cual de los 15 conjuntos de caracteres pertenece el caracter. (Note que en TI BASIC hay 16 conjuntos de caracteres mientras que TI BASIC Extendido tiene 15).
La lista de los códigos ASCII para caracteres standard se da en el Apéndice C. Los números de los conjuntos de caracteres son los siguientes:

NUMERO DEL CONJUNTO	CODIGOS DE CARACTER
0	30-31
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143

Ejemplos:

CALL COLOR (3, 5, 8) define el color de frente de los caracteres 48 a 55 en 5 (azul oscuro) y el color de fondo en 8 (ciánico)

> 100 CALL COLOR (3, 5, 8)

CALL COLOR (#5, 16) hace que el sprite número 5 tenga color de frente 16 (blanco). El color de fondo es siempre 1 (transparente)

> 100 CALL COLOR (#5, 16)

CALL COLOR (#7,INT(RND x 16 + 1)) define al sprite número 7 con color de frente elegido al azar de los 16 colores disponibles.
El color de fondo es 1 (transparente)

> 100 CALL COLOR (# 7, INT (RND x 16 + 1))

CONTINUE

Formato:

CONTINUE
CON

Descripción:

El comando CONTINUE recomienza un programa que fue detenido por un "breakpoint". Puede ingresarse siempre que el programa haya parado su ejecución a causa de un comando BREAK, o una instrucción, o por FCTN 4 (CLEAR). Sin embargo no se puede usar CONTINUE si se ha editado el programa. CONTINUE puede abreviarse como CON.

Cuando ocurre un "breakpoint", el conjunto de caracteres standard y los colores standard se recuperan. Los sprites dejan de existir CONTINUE no devuelve los caracteres ni colores standard que habrán sido cambiados. De otro modo, el programa continúa como si no hubiera ocurrido el "breakpoint"!

COS

Formato:

COS (expresión-en-radianes)

Descripción:

La función coseno da el coseno trigonométrico de la expresión-en-radianes. Si el ángulo está en grados, multiplique el número por P1/180 para conseguir el ángulo equivalente en radianes.

Programa:

El programa de la derecha da el coseno de varios ángulos.

```
> 100 A = 1.047197551196
> 110 B = 60
> 120 C = 45* PI/180
> 130 PRINT COS(A); COS(B)
> 140 PRINT COS(B* PI/180)
> 150 PRINT COS(C)
> RUN
  .5 –.9524129804
  .5
  .7071067812
```

DATA

Formato:

DATA Lista-de-datos

Descripción:

La instrucción DATA permite almacenar datos dentro del programa. Los datos, que pueden ser constantes numéricos o alfanuméricos se describen en la lista-de-datos separados por comas. Durante la ejecución del programa, la instrucción READ asigna los valores de la lista-de-datos a las variables especificadas en la lista de variables de la instrucción READ.

Las instrucciones DATA pueden incluirse en cualquier punto del programa. Sin embargo, el orden en que aparecen es importante. Los datos de varias instrucciones DATA se lean secuencialmente comenzando con el primer dato de la primera instrucción DATA. Si un programa tiene más de una instrucción DATA, las instrucciones DATA se leen en el orden en que aparecen en el programa a menos que se especifique otra cosa en una instrucción RESTORE. Así el orden en que aparecen los datos en el programa determina el orden en que serán leídos. Las instrucciones DATA no pueden formar parte de instrucciones múltiples.

Los datos en la lista-de-datos deben corresponderse con el tipo de variable que se le asigna en la instrucción READ. Así, si se especifica una variable numérica en la instrucción READ, deberá haber una constante numérica en la posición correspondiente de la instrucción DATA. De la misma forma, si se especifica una variable alfanumérica deberá indicarse la constante alfanumérica correspondiente. Un número es una variable alfanumérica válida, de manera que se podrá tener una constante numérica en la instrucción DATA, que se corresponda con una variable alfanumérica en la instrucción READ. Si la instrucción DATA contiene comas adyacentes, la computadora asume que se intenta ingresar una variable nula (una cadena de caracteres).

Cuando se usa una cadena alfanumérica en una instrucción DATA, se la puede encerrar entre comillas. Sin embargo si la cadena contiene una coma, o espacios al principio o al final, **deberá** encerrársela entre comillas. Si la cadena está encerrada entre comillas, las comillas que están dentro de la cadena se representarán mediante doble comillas.

Programa:

El programa de la derecha lee e imprime varias constantes numéricas y alfanuméricas. La línea 100 a 130 leen cinco conjuntos de datos e imprimen sus valores de a dos por línea.

```
> 100 FOR A = 1 TO 5
> 110 READ B, C
> 120 PRINT B;C
> 130 NEXT A
> 140 DATA 2, 4, 6, 7, 8
> 150 DATA 1, 2, 3, 4, 5
> 160 DATA """"AQUI HAY
  COMILLAS""""
> 170 DATA "AQUI, NO HAY COMI
  LLAS"
> 180 DATA AQUI TAMPOCO HAY
  COMILLAS
```

Las líneas 190 a 220 leen siete datos y los imprimen cada uno en una línea.

```
> 190 FOR A = 1 TO 7
> 200 READ B$
> 210 PRINT B$
> 220 NEXT A
> 230 DATA 1, NUMERO,,TI
> RUN
```

Los primeros dos elementos de la línea 140

```
2      4
```

Los segundos dos elementos de la línea 140

```
6      7
```

El último elemento de la línea 140 y el primero de la línea 150

```
8      1
```

El segundo y tercer elementos de la línea 150

```
2      3
```

El cuarto y quinto elementos de la línea 150

```
4      5
```

La línea 160

```
"AQUI HAY COMILLAS"
```

La línea 170

```
AQUI, NO HAY COMILLAS
```

La línea 180

```
AQUI TAMPOCO HAY COMILLAS
```

El primer elemento de la línea 230

```
1
```

El segundo elemento de la línea 230

```
NUMERO
```

Cadena nula para las dos comas de la línea 230

Ultimo elemento de la línea 230

```
TI
```

DEF

Formato:

DEF nombre-de-función [(parámetro) = expresión

Descripción:

La instrucción DEF le permite definir sus propias funciones. El nombre-de-función puede ser cualquier nombre de variable. Si especifica un parámetro, éste debe estar encerrado entre paréntesis y puede ser cualquier nombre de variable escalar. Si la expresión es una cadena alfanumérica, el nombre de función debe ser el nombre de una variable alfanumérica (el último caracter debe ser un signo $).

La instrucción DEF debe aparecer siempre en una línea con número de secuencia mas bajo que cualquier referencia a la función que define. Asimismo, una instrucción DEF no puede aparecer en una instrucción IF-THEN-ELSE. Cuando la computadora encuentra una instrucción DEF durante la ejecución de un programa, prosigue con la siguiente instrucción si realiza ninguna acción.
Una función puede usarse ya sea para una expresión numérica o alfanumérica empleando el nombre-de-función seguido de una expresión encerrada entre paréntesis si es que se especificó un parámetro para la instrucción DEF.

Cuando se encuentra una referencia a la función en una expresión (mediante el use del nombre-de-función), la función se evalúa usando los valores actuales de las variables especificados en la instrucción DEF y el valor del parámetro si lo hay. Una instrucción DEF puede referirse a otra función definida.
Sin embargo no podrá referirse a si misma directamente (por ej.: DEF B = Bx2) ni indirectamente (Por ej.: DEF F = G : : DEF G = F).

Si la Expansión de Memoria está conectada a la computadora no podrá imprimirse el valor de una función con PRINT usado como comando.

Opciones:

Si se especifica un parámetro para una función, cuando se encuentra una referencia a ella en una expresión se asigna el valor al parámetro. Luego, el valor de la función queda determinado usando el valor del parámetro y los valores de las otras variables usadas en la instrucción DEF. Si se definió un parámetro para una instrucción DEF habrá que dar un argumento cada vez que se haga referencia a la función.

El nombre del parámetro usado en una instrucción DEF sólo afecta a la instrucción que lo usa. Esto significa que es distinto de cualquier otra variable que aparezca con el mismo nombre en cualquier lugar del programa.

Un parámetro no puede ser usado como un arreglo. Se puede emplear un elemento de un arreglo en una función siempre que no tenga el mismo nombre que el parámetro. Por ejemplo se puede hacer DEF F(A) = B(Z) pero no DEF F(A) = A(Z).

Ejemplos:

DEF PAGO (HE) = 40* SALARIO + 1.5 * SALARIO * HE definir PAGO de manera que cada vez que aparece en el programa el pago se calcula multiplicando por 40 el SALARIO más 1.5 del SALARIO por las horas extra.

```
> 100 DEF PAGO(HE) = 40 * SALARIO +
  1.5 * SALARIO * HE
```

DEF RND20 = INT (RND * 20 + 1) define RND20 de modo que cada vez que se encuentra en el programa la función produce un entero de 1 a 20.

```
> 100 DEF RND20 = INT (RND * 20 + 1)
```

DEF PRIMPAL$(NOMBRE$) = SEG$(NOMBRE$,1,POS(NOMBRE$," ",1) - 1) define a PRIMPAL$ como la porción de NOMBRE$ producida por un espacio.

```
> 100 DEF PRIMPAL$(NOMBRE$) =
  SEG$(NOMBRE$,1,POS(NOMBRE$,"
  ",1)  - 1)
```

DELETE

Formato:

DELETE dispositivo-nombre-de-archivo

Descripción:

El comando DELETE le permite remover un programa o un archivo de datos del sistema de archivos de la computadora. Dispositivo-nombre-de-archivo es una expresión alfanumérica. Si se usa una constante alfanumérica, debe estar numerada entre comillas. También se pueden borrar archivos de datos usando la palabra clave DELETE en la instrucción CLOSE.

Algunos dispositivos (tales como diskettes) permiten el borrado de archivos; otros (tales como cassettes) no lo permiten. Vea el manual del dispositivo específico para mayor información.

Ejemplo:

DELETE "DSK1.MYFILE" borra el archivo MYFILE en el drive 1

```
> DELETE "DSK1.MYFILE"
```

Programa:

El programa de la derecha ilustra el use de DELETE

```
> 100 INPUT "NOM-ARCH: ":X$
> 110 DELETE X$
```

DELSPRITE Subprograma

CAPITULO 4

Formato:

CALL DELSPRITE (# N°-de-sprite [,…])
CALL DELSPRITE (ALL)

Descripción:

El subprograma DELSPRITE remueve sprites. Se pueden' borrar uno o más sprites especificando sus números precedidos por el signo (#) y separados por comas o se pueden borrar todos los sprites especificando ALL. Luego que ha sido borrado con DELSPRITE un sprite puede volver a crearse con el subprograma SPRITE.

Ejemplos:

CALL DELSPRITE (#3) borra el sprite número 3.	> 100 CALL DELSPRITE (#3)
CALL DELSPRITE (#4,#3*C) borra el sprite 4 y el sprite cuyo número se halla multiplicando 3*C	> 100 CALL DELSPRITE (#4,#3*C)
CALL DELSPRITE (ALL) borra todos los sprites.	> 100 CALL DELSPRITE (ALL)

DIM

Formato:

DIM nombre-arreglo (entero 1 [,entero 2]…[,entero 7][,...])

Descripción:

La instrucción DIM reserva espacio en la memoria de la computadora para arreglos numéricos y alfanuméricos. Un arreglo sólo puede dimensionarse una vez en un programa. Si se dimensiona un arreglo, la instrucción DIM debe aparecer en una línea con número de secuencia inferior a cualquier referencia a dicho arreglo. Si se dimensiona más de un arreglo en una única instrucción DIM, los nombres-de-arreglo deben estar separados por comas. Nombre-arreglo puede ser cualquier nombre de variable Una instrucción DIM no puede aparecer en una instrucción IF-THEN-ELSE.

En TI BASIC Extendido se pueden tener arreglos de hasta 7 dimensiones. La cantidad de "enteros" separados por comas que siguen al nombre del arreglo, determina la dimensión de dicho arreglo. Los valores de los "enteros" determinan la cantidad de elementos en cada dimensión.

El espacio que ocupará el arreglo se reserva luego de haber, ingresado el comando RUN, pero antes de que se ejecute la primera instrucción. Cada elemento de un arreglo alfanumérico será una cadena nula, y cada elemento de un arreglo numérico será cero hasta tanto se reemplace por otro valor.

Los valores de los "enteros" determinan el valor máximo de cada subíndice del arreglo. Si se utiliza un arreglo no definido en una instrucción DIM, el máximo valor para cada subíndice es 10. El subíndice del primer elemento es cero a menos que una instrucción OPTION BASE defina el subíndice mínimo en 1. De esta forma, un arreglo definido como A (6) es un arreglo unidimensional de 7 elementos, a menos que el subíndice 0 se haya eliminado con una instrucción OPTION BASE.

Ejemplos:

DIM X$ (30) reserva espacio en la memoria de la computadora para los 31 miembros del arreglo llamado X$

```
> 100 DIM X$ (30)
```

DIM D(100), B(10,9) reserva espacio en la memoria de la computadora para los 101 miembros del arreglo D y 110 (11 veces 10) miembros del arreglo llamado B.

```
> 100 DIM D(100) ,B(10,9)
```

DISPLAY

CAPITULO

4

Formato:

DISPLAY [[AT(fila, columna)][BEEP][ERASE ALL]
[SIZE (expresión-numérica)]: lista-de-variables

Descripción:

La instrucción DISPLAY despliega información en la pantalla. Hay muchas opciones disponibles para DISPLAY, lo que la hace mucho más versátil que el PRINT. Se pueden desplegar datos en cualquier lugar de la pantalla, producir un tono audible (beep) cuando se despliegan los datos, blanquear posiciones de la pantalla y borrar todos los caracteres antes de desplegar los datos.

Opciones:

AT (fila, columna) define el comienzo del campo a desplegar en una fila y columna determinada. Las filas se numeran de 1 a 24. Las columnas se numeran de 1 a 28, donde la columna 1 corresponde a la columna 3 que utilizan los subprogramas VCHAR, GCHAR y HCHAR. Si no aparece la opción AT, los datos se despliegan a partir de la fila 24, columna 1 como para la instrucción PRINT.

BEEP produce un tono corto cuando se despliegan los datos.

ERASE ALL llena la pantalla completa con caracteres blancos antes de desplegar los datos.

SIZE (expresión-numérica) coloca "expresión-numérica" cantidad de blancos en 1 pantalla comenzando en fila y columna. Si no se usa esta opción se blanquea el resto de la fila en la que se despliega el dato. Si el valor de expresión-numérica es más grande que las posiciones que quedan en la fila, sólo se blanquea el resto de la fila.

Ejemplos:

DISPLAY AT (5,7):Y muestra el valor de Y en la quinta fila, séptima columna de la pantalla

```
> 100 DISPLAY AT (5,7):Y
```

DISPLAY ERASE ALL: B coloca el caracter blanco en todas las posiciones de la pantalla antes de mostrar el valor de B.

```
> 100 DISPLAY ERASE ALL: B
```

DISPLAY AT (R,C) SIZE (LONG) BEEP:X$ muestra el valor de X$ en la fila R columna C. Primero emite un "beep" y se blanquean LONG cantidad de caracteres.

```
> 100 DISPLAY AT (R,C) SIZE
(LONG) BEEP:X$
```

Programa:

El programa de la derecha ilustra el uso de DISPLAY
Permite posicionar bloques en cualquier lugar de la pantalla para dibujar una figura.

```
> 100 CALL CLEAR
> 110 CALL COLOR (9,5,5)
> 120 DISPLAY AT (23,1): "INGRESE FILA Y
  COLUMNA"
> 130 DISPLAY AT (24,1): "FILA: COLUMNA:"
> 140 FOR CONTAR = 1 TO 2
> 150 CALL KEY (0,FILA(CONTAR),S)
> 160 IF S <= 0 THEN 150
> 170 DISPLAY AT (24,5 + CONTAR) SIZE (1):
  STR$(FILA(CONTAR) - 48)
> 180 NEXT CONTAR
> 190 FOR CONTAR = 1 TO 2
> 200 CALL KEY (0,COLUMNA(CONTAR), S)
> 210 IF S <= 0 THEN 200
> 220 DISPLAY AT (24,16 + CONTAR) SIZE(1):
  STR$ (COLUMNA(CONTAR) - 48)
> 230 NEXT CONTAR
> 240 FILA 1= 1 * (FILA(1) - 48) + FILA(2) - 48
> 250 COLUMNA(1) = 10 * (COLUM NA(1) -
  48) + COLUMNA(2) - 48
> 260 DISPLAY AT (FILA(1),COLUMNA(1))
  SIZE(1):CHR$ (96)
> 270 GOTO 130
```

Presione **FCTN 4** para detener el programa

DISPLAY USING

CAPITULO

4

Formato:

DISPLAY [lista-de-opción:] USING expresión-alfanumérica [:lista-de-variables]
DISPLAY [lista-de-opción:] USING N°-de-línea [:lista-de-variables]

Descripción:

La instrucción DISPLAY...USING es la misma que el DISPLAY pero con el agregado de la cláusula USING que especifica el formato de los datos de la lista-de-variables. Si la expresión-alfanumérica está presente, entonces define el formato. Si se indica N°-de-línea se refiere al N°-de-línea de una instrucción IMAGE. Ver IMAGE para una explicación de como se define el formato.

Ejemplos:

DISPLAY AT(10,4):USING "##.##":N despliega el valor de N en la décima fila y cuarta columna con el formato "##.##".

```
> 100 DISPLAY AT(10,4):USING
"##.##":N
```

DISPLAY USING "##.##":N despliega el valor de N con el formato "##.##" , en la fila 24, columna 1.

```
> 100 DISPLAY USING "##.##":N
```

DISTANCE Subprograma

Formato:

CALL DISTANCE (N^{o}-sprite, N^{o}-sprite, variable-numérica)
CALL DISTANCE (N^{o}-sprite, punto-fila, punto-columna, variable-numérica)

Descripción:

El subprograma DISTANCE devuelve el cuadrado del valor de la distancia entre dos sprites o entre un sprite y una posición.

La posición de cada sprite se define teniendo en cuenta su esquina superior izquierda. Punto-fila y punto-columna van de 1 a 256.

En la variable-numérica se devuelve la distancia al cuadrado.

El número devuelto se calcula: hallando la diferencia entre los puntos-fila de los sprites y elevándola al cuadrado. Luego se halla la diferencia entre los puntos columna de los sprites y se eleva al cuadrado. Luego se suman los dos cuadrados. Si la suma es mayor que 32767, el número devuelto es 32767.

La distancia entre los sprites (o el sprite y la posición) es la raíz cuadrada del valor devuelto.

Ejemplos:

CALL DISTANCE (#3, #4, DIST) define DIST como el cuadrado de la distancia entre las esquinas superiores izquierdas de los sprites #3 y #4

```
> 100 CALL DISTANCE(#3, #4, DIST)
```

CALL DISTANCE (#4, 18, 89, D) define D igual al cuadrado de la distancia entre la esquina superior izquierda del sprite #4 y la posición 18, 89

```
> CALL DISTANCE (#4, 18, 89, D)
```

END

Formato:

END

Descripción:

La instrucción END finaliza un programa y se puede usar en lugar del STOP. Si bien la instrucción END puede aparecer en cualquier lugar, normalmente se lo ubica en la última línea del programa y así se indica su finalización tanto física como lógica. La instrucción STOP se usa generalmente en los lugares en donde se desea que el programa se detenga. No es necesario usar la instrucción END en TI BASIC Extendido. El programa se detiene automáticamente luego de ejecutar la línea con el mayor número de secuencia.

EOF

Formato:

EOF(número de archivo)

Descripción:

La función EOF usa para verificar si queda en el archivo algún registro que no fue leído. El valor de N°-de-archivo indica el archivo a verificar, y debe corresponderse con el número de un archivo abierto. La función EOF no puede usarse con cassettes.

La función EOF supone siempre que el próximo registro será leído secuencialmente, aún si se trata de un archivo RELATIVE.

El valor de la función EOF depende del lugar en que está posicionado el archivo. Si no se encuentra en el último registro del archivo, la función devuelve un valor 0. Si está en el último registro del archivo, devuelve un valor 1.

Si el diskette u otro medio de almacenamiento está lleno, está al final del archivo, y no hay mas lugar para ningún dato, la función devuelve en valor - 1.

Para más información vea el manual "Sistema de Discos"

Ejemplos:

PRINT EOF(3) imprime un valor que indica si ha llegado o no al final del archivo abierto como # 3.	> 100 PRINT EOF (3)
IF EOF(27) <> 0 THEN 1150 transfiere el control a la línea 1150 si llegó al final del archivo abierto como #27.	> 100 IF EOF(27) <> 0 THEN 1150
IF EOF(27) THEN 1150 transfiere al control a la línea 1150 si se llegó al final del archivo abierto como # 27	> 100 IF EOF(27) THEN 1150

ERR Subprograma

CAPITULO

4

Formato:

CALL ERR (código-de-error, tipo-de-error [,severidad-del-error, número-de-línea])

Descripción:

El subprograma ERR devuelve el código-de-error y el tipo-de-error del error más reciente no aclarado. Un error queda aclarado cuando ha sido accedido por el subprograma ERR, cuando ha ocurrido otro error, o el programa ha finalizado.

El código-de-error está formado por un número de dos o tres dígitos. El significado de cada uno de los códigos aparece en el Apéndice N.

Si el tipo-de-error es un numero negativo, el error ocurrió en la ejecución del programa. Si el código de error es 130 (Error de E/S) el tipo-de-error es un número positivo y es el 'número del archivo que provocó el error.

Si no ocurrió ningún error, CALL ERR devuelve ceros en todas las variables. CALL ERR se usa junto con ON ERROR.

Opciones:

Opcionalmente se puede obtener la severidad-del-error y el número-de-línea donde ocurrió. La severidad-del-error es siempre 9. El número-de-línea es el número de la línea que se estaba ejecutando cuando ocurrió el error. No siempre es la línea la fuente del problema dado que un error puede ocurrir por causa de valores generados o acciones que se han tomado en cualquier lugar del programa.

Ejemplos:

CALL ERR (A, B) pone en A. el código-de-error y en B el tipo-de-error correspondiente al error más reciente.

```
> 100 CALL ERR (A, B)
```

CALL ERR (W, X, Y, Z) pone en W el código-de-error, en X el tipo-de-error, en Y la severidad-de-error y en Z el número-de-línea del error más reciente.

```
> 100 CALL ERR (W, X, Y, Z)
```

SUBPROGRAMA ERR

Programa:

El programa de la derecha ilustra el use de CALL ERR. Se provoca un error en la línea 110 pidiendo un color de pantalla ilegal. A través de la línea 100 el control de transfiere a la línea 130.
La línea 140 imprime los valores obtenidos. El 79 indica que se dio un valor incorrecto. El —1 indica que el error fue en una instrucción. El 9 es la severidad-del-error. El 110 indica que el error ocurrió en la línea 110.

```
> 100 ON ERROR 130
> 110 CALL SCREEN (18)
> 120 STOP
> 130 CALL ERR (W. X, Y, Z)
> 140 PRINT W;X;Y;Z
> RUN
79  - 1 9 110
```

EXP

Formato:

EXP (expresión-numérica)

Descripción:

La función EXP devuelve el valor exponencial (e^x) de la expresión-numérica. El valor de e es: 2.718281828459.

Ejemplos:

Y = EXP(7) pone en Y el valor de e elevado a la séptima potencia, que es 1096.633158429

```
> 100 Y = EXP(7)
```

L = EXP (4.394960467) asigna a L el valor de e elevado a la 4.394960467 potencia que es 81.04142688868

```
> 100 L = EXP(4.394960467)
```

FOR TO [STEP]

Formato:

FOR variable-de-control = valor-inicial TO límite STEP incremento

Descripción:

La instrucción FOR-TO-STEP repite la ejecución de las instrucciones que están entre FOR-TO-STEP y NEXT hasta que la variable-de-control caiga fuera del rango formado por el valor-inicial y el límite. La instrucción FOR-TO-STEP es útil cuando se necesitan repetir los mismos pasos en un ciclo. La instrucción FOR-TO-STEP no puede usarse en una instrucción IF-THEN-ELSE.

La variable-de-control debe ser una variable numérica sin subíndice. Y actúa como contador para el ciclo. El valor-inicial y el límite son expresiones numéricas. La segunda vez que se ejecuta el ciclo, el valor de la variable-de-control se incrementa en 1 o en el incremento indicado. Este incremento puede ser un número positivo o negativo. Así continúa hasta que el valor de la variable-de-control cae fuera del rango definido por el valor-inicial y el límite. Luego se ejecuta la instrucción que sigue al NEXT.

El valor de la variable-de-control no cambia cuando la computadora abandona el ciclo.

El valor de la variable-de-control puede cambiarse dentro del ciclo, pero se lo debe hacer con cuidado para evitar resultados inesperados. Los ciclos pueden estar "anidados", es decir que un ciclo puede estar contenido completamente dentro de otro. Se puede abandonar un ciclo usando GOTO, GOSUB, IF-THEN-ELSE, y luego volver. Sin embargo no se puede entrar a un ciclo FOR-NEXT en ningún punto excepto en el comienzo.

Si el valor inicial excede el límite al comienzo del ciclo FOR-NEXT, no se ejecutará ninguna de las instrucciones del ciclo. En cambio la ejecución continuará con la primera instrucción después de la instrucción NEXT.

Ejemplos:

FOR A = 1 TO 5 STEP 2 ejecuta las instrucciones que están entre FOR y NEXT A. tres veces con A tomando los valores 1, 3 y 5. Cuando finaliza el ciclo, A vale 7.

```
> 100 FOR A = 1 TO 5 STEP 2
```

FOR J = 7 TO -5 STEP -.5 ejecuta las instrucciones entre FOR y NEXT J 25 veces, con J cuando los valores 7, 6. 5, 6,… -4, -4.5 y -5. Cuando finalizó el ciclo, J vale -5.5

```
> 100 FOR J = 7 TO -5 STEP -.5
```

Programa:

El programa de la derecha ilustra el uso de la instrucción FOR-TO-STEP. Hay tres ciclos FOR-NEXT con variables de control CAR, FILA y COLU.

```
>100 CALL CLEAR
>110 D = 0
>120 FOR CAR = 33 TO 63 STEP 30
>130 FOR FILA = 1 + D TO 21 + D
  STEP 4
>140 FOR COLU = I + D TO 29 + D
  STEP 4
>150 CALL VCHAR (FILA, COLU,
  CAR)
>160 NEXT COLU
>170 NEXT FILA
>180 D = 2
>190 NEXT CAR
>200 GOTO 200
```

(Presione **FCTN 4** para detener el programa)

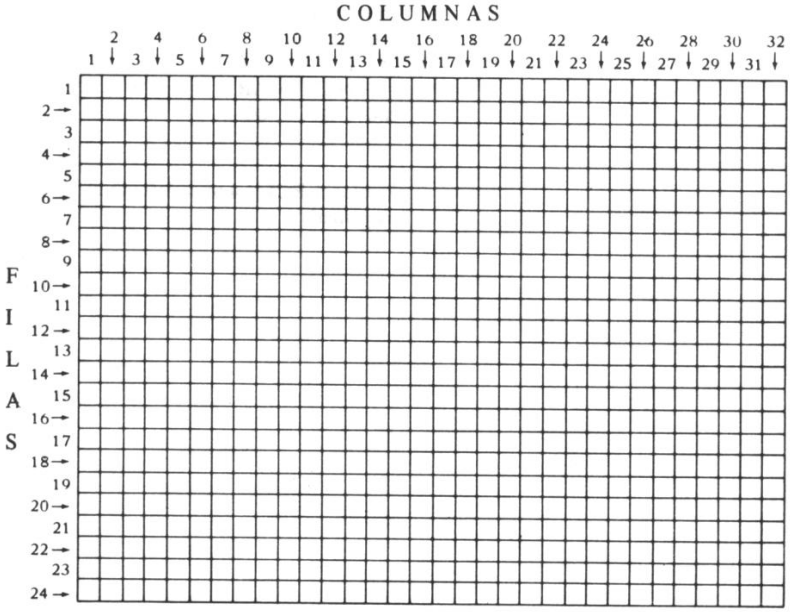
GCHAR Subprograma

Formato:

CALL GCHAR (fila, columna, variable-numérica)

Descripción:

El subprograma GCHAR lee un caracter ubicado en cualquier lugar de la pantalla. La computadora devuelve en la variable-numérica el código ASCII para el caracter que está en la posición indicada por fila y columna.
Fila y columna son expresiones numéricas. Un valor 1 para fila indica la parte superior de la pantalla. Un valor 1 para columna indica el costado izquierdo de la pantalla. La pantalla puede imaginarse como una grilla tal como se muestra a continuación



Ejemplos:

CALL GCHAR (12, 16,X) asigna a X el código ASCII del caracter que está en la fila 12, columna 16.

```
> 100 CALL GCHAR (12, 16, X)
```

CALL GCHAR (R, C, K) pone en K el código ASCII del caracter que está en la fila R, columna C.

```
> 100 CALL GCHAR (R, C, K)
```

GOSUB

Formato:

GOSUB número-de-línea
GO SUB número-de-línea

Descripción:

La instrucción GOSUB permite la transferencia a una subrutina. Cuando se ejecuta, el control se transfiere al número-de-línea y se ejecutan esa instrucción y las siguientes (las cuales pueden incluir cualquier instrucción, incluso GOTO y GOSUB).

Cuando encuentra una instrucción RETURN, el control vuelve a la instrucción que sigue al GOSUB. Las subrutinas son útiles cuando se debe realizar una misma acción en distintas partes de un programa. Vea también ON . . . GOSUB. El TI BASIC Extendido las subrutinas pueden llamarse a sí mismas.

Ejemplos:

GOSUB 200 transfiere el control a la instrucción 200. Se ejecuta esa instrucción y todas las que la siguen hasta el RETURN. Luego el control vuelve a la instrucción siguiente a la de llamada.

```
> 100 GOSUB 200
```

Programa:

El programa de la derecha ilustra el use de GOSUB. La subrutina de la línea 260 calcula el factorial para la variable NUMB. El programa completo calcula la solución de la ecuación

$$NUMB = \frac{X!}{Y!*(X-Y)!}$$

donde el símbolo ! significa factorial. Esta fórmula se usa en el cálculo de probabilidades. Por ejemplo:
Si se da a X el valor 52 y a Y el valor 5 se encontrarán el número de posibles combinaciones de cinco cartas de poker

```
> 100 CALL CLEAR
> 110 INPUT "INGRESE X E Y:": X, Y
> 120 IF X < Y THEN 110
> 130 IF X > 69 OR Y > 69 THEN 110
> 140 NUM = X
> 150 GOSUB 260
> 160 NUMERADOR = NUMB
> 170 NUMB = Y
> 180 GOSUB 270
> 190 DENOMINADOR = NUMB
> 200 NUMB = X - Y
> 210 GOSUB 260
> 220 DENOMINADOR =
  DENOMINADOR * NUMB
> 230 NUMB = NUMERADOR /
  DENOMINADOR
> 240 PRINT "EL NUMERO ES",
  NUMB
> 250 STOP
> 260 REM CALCULO DEL
  FACTORIAL
> 270 IF NUMB < 0 THEN PRINT
  "NEGATIVO" : : GOTO 110
> 280 IF NUMB  < 2 THEN NUMB =  I
  : : GOTO 330
> 290 MULT = NUMB - 1
> 300 NUMB = NUMB * MULT
> 310 MULT = MULT - 1
> 320 IF MULT > 1 THEN 300
> 330 RETURN
```

GOTO

Formato:

GOTO número-de-línea
GO TO número-de-línea

Descripción:

La instrucción GOTO permite la transferencia incondicional del control a otra línea dentro del programa. Cuando se ejecuta la instrucción GOTO, el control se transfiere a la primera instrucción de la línea especificada por número-de-línea.
La instrucción GOTO no debe usarse para transferir el control entre subprogramas.

Programa:

El programa de la derecha muestra el use del GOTO en la línea 160. Cada vez que el programa llegue a esa línea, se ejecutará la línea 130 y las siguientes a partir de este nuevo punto.

```
> 100 REM SUMA DE 1 A 100
> 110 RESPUESTA = 0
> 120 NUMB = 1
> 130 RESPUESTA = RESPUESTA +
  NUMB
> 140 NUMB = NUMB + 1
> 150 IF NUMB > 100 THEN 170
> 160 GOTO 130
> 170 PRINT "LA RESPUESTA ES" ;
  RESPUESTA
> RUN
LA RESPUESTA ES 5050
```

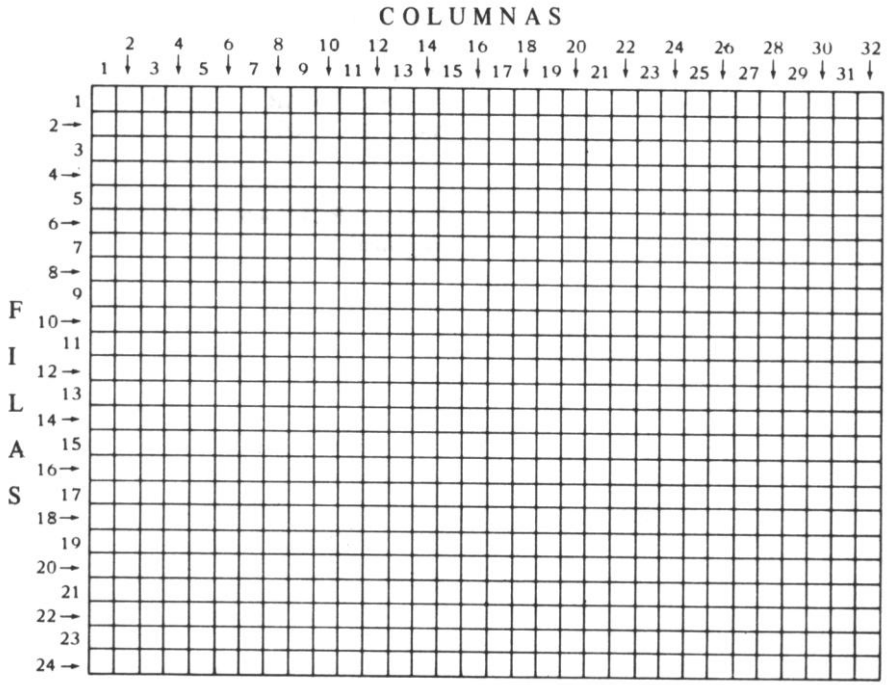
HCHAR Subprograma

Formato:

CALL HCHAR (fila, columna, código-de-caracter [, repeticiones])

Descripción:

El subprograma HCHAR muestra un caracter en cualquier posición de la pantalla y opcionalmente lo repite horizontalmente. El caracter, que tiene el valor ASCII indicado por código-de-caracter, se ubica en la posición descripta por fila y columna y se repite horizontalmente tantas veces como indique repeticiones.
Un valor 1 para fila indica la parte superior de la pantalla. El valor 24 es la última. fila El valor 1 para columna indica el costado izquierdo de la pantalla; y un valor 32 el costado derecho.
La pantalla puede imaginarse como una grilla, tal como se muestra a continuación.



Ejemplos:

CALL HCHAR (12,16, 33) coloca el caracter 33 (un signo de admiración) en la fila 12, columna 16.

```
> 100 CALL HCHAR(12, 16, 33)
```

CALL HCHAR (1, 1, ASC ("!"), 768) pone un signo de admiración en la fila 1, columna 1, y lo repite 768 veces, completando la pantalla.

```
> 100 CALL HCHAR(1, 1, ASC ("!"), 768)
```

CALL HCHAR (R, C, K, T) coloca el caracter con el código ASCII especificado por el valor de K, en la fila R, columna C y lo repite T veces.

```
> 100 CALL HCHAR(R, C, K, T)
```

IF THEN [ELSE]

Formato:

IF expresión-relacional THEN número-de-línea-1 [ELSE número-de-línea-2]
IF expresión-relacional THEN instrucción-1 [ELSE instrucción-2]
IF expresión-numérica THEN número-de-línea-1 [ELSE número-de-línea-2]
IF expresión-numérica THEN instrucción-1 [ELSE instrucción-2]

Descripción:

La instrucción IF-THEN-ELSE permite transferir el control a número-de-línea-1 o ejecutar la instrucción-1 si la expresión-relacional es verdadera o si la expresión-numérica es distinta de cero.

Si no es así, el control pasa a la instrucción siguiente u opcionalmente al número-de-línea-2 o instrucción-2.

Instrucción-1 e instrucción-2 pueden a su vez estar formadas cada una por varias instrucciones separadas por el símbolo separador de instrucciones. Estas instrucciones se ejecutan solamente si se ejecuta la cláusula inmediatamente anterior a ellas.

La instrucción IF-THEN-ELSE no puede contener DATA, DEF, DIM, FOR, NEXT, OPTION BASE, SUB o SUBEND.

Ejemplos:

IF X > 5 THEN GOSUB 300 ELSE X = X + 5 opera de la siguiente manera: Si X es mayor que 5 entonces se ejecuta GOSUB 300. Cuando termina la subrutina el control vuelve a la línea que sigue a esta línea. Si es menor o igual a 5, X toma el valor X + 5 y el control pasa a la próxima línea.

```
> 100 IF X > 5 THEN 300 GOSUB
ELSE X = X + 5
```

IF Q THEN C = C + 1 :: GOTO 500 :: ELSE L = L/C :: GOTO 300 opera de la siguiente manera: Si Q no es cero C toma el valor C + 1 y se transfiere el control a la línea 500. Si Q es cero, L toma el valor L/C y el control se transfiere a la línea 300

```
> 100 IF Q THEN C = C + 1 :: GOTO
500 :: ELSE L = L/C :: GOTO 300
```

IF A > 3 THEN 300 ELSE A = 0 :: GOTO 10 opera de la siguiente manera: Si A es mayor que 3, el control se transfiere a la línea 300. Sino A toma el valor 0 y se transfiere el control a la línea 10.

```
> 100 IF A > 3 THEN 300 ELSE A = 0 ::
GOTO 10
```

IF A$ = "Y" THEN CONT = CONT + 1 :: DISPLAY AT (24, 1): "OTRA VEZ!" :: GOTO 300 opera de la siguiente manera: Si A$ no es igual a Y el control pasa a la línea siguiente. Si A$ es igual a Y, CONT se incrementa en 1, se muestra un mensaje y se transfiere el control a la línea 300.

```
> 100 IF A$ = "Y" THEN CONT = CONT
+ 1 :: DISPLAY AT (24,1): "OTRA VEZ!"
:: GOTO 300
```

IF HORAS <= 40 THEN PAGO = HORAS * SUELDO ELSE PAGO = HORAS * SUELDO + .5 * SUELDO * (HORAS - 40) :: OT = 1 opera de la siguiente manera: Si HORAS es menor o igual que 40, PAGO se calcula como HORAS * SUELDO y el control pasa a la siguiente línea. Si HORAS es mayor que 40, PAGO se calcula como HORAS * SUELDO + .5 * SUELDO (HORAS - 40), OT toma el valor 1 y el control pasa a la línea siguiente.

```
> 100 IF HORAS <= 40 THEN PAGO =
HORAS * SUELDO ELSE PAGO =
HORAS * SUELDO + .5 * SUELDO *
(HORAS - 40) :: OT = 1
```

IF A = 1 THEN IF B = 2 THEN C = 3 ELSE D = 4 ELSE E = 5 opera de la siguiente manera: Si A no es igual a 1, E toma el valor 5 y el control pasa a la línea siguiente. Si A es igual a 1 y B no es igual a 2, D toma el valor 4 y el control pasa a la siguiente línea. Si A es igual a 1 y B es igual a 2, C toma el valor 3 y el control pasa a la línea siguiente.

```
> 100 IF A = 1 THEN IF B = 2 THEN C =
3 ELSE D = 4 ELSE E = 5
```

IF THEN [ELSE]

Programa :

El programa de la derecha ilustra el use de IF THEN ELSE. Acepta hasta 1000 números y luego los imprime ordenados de menor a mayor.

```
> 100 CALL CLEAR
> 110 DIM VALOR (1000)
> 120 PRINT "INGRESE LOS VALORES A
  ORDENAR.":"INGRESE '9999' PARA
  TERMINAR LA ENTRADA DE DATOS"
> 130 FOR CONT = I TO 1000
> 140 INPUT VALOR (CONT)
> 150 IF VALOR (CONT) = 9999 THEN 170
> 160 NEXT CONT
> 170 CONT = CONT —1
> 180 PRINT "ORDENANDO"
> 190 FOR SORTI = 1 TO CONT-1
> 200 FOR SORT2 = SORT1 + 1 TO CONT
> 210 IF VALOR (SORT1) > VALOR (SORT2)
  THEN TEMP = VALOR (SORT1) :: VALOR
  (SORT1)  =  VALOR (SORT2) ::
  VALOR(SORT2) = TEMP
> 220 NEXT SORT2
> 230 NEXT SORT1
> 240 FOR ORDEN = 1 TO CONT
> 250 PRINT VALOR (ORDEN)
> 260 NEXT ORDEN
```

IMAGE

CAPITULO

4

Formato:

IMAGE Formato

Descripción:

La instrucción IMAGE especifica el formato en que se imprimirán o mostrarán los números cuando se ha usado la cláusula USING con PRINT o DISPLAY. Cuando se encuentra una cláusula IMAGE durante la ejecución del programa, no se la tiene en cuenta. La instrucción IMAGE debe ser la única de la línea. La siguiente descripción de formato se aplica también al use de una imagen explícita luego de la cláusula USING en un PRINT.. USING o DISPLAY... USING.

El formato puede tener hasta 254 caracteres e incluye cualquier caracter. Este formato se trata de la siguiente manera:

Los signos (#) se reemplazan por los valores de la lista-de-impresión dada en PRINT… USING o DISPLAY... USING. Se deberá indicar un signo # para cada dígito del valor y uno para el signo negativo, si lo tiene, o bien uno para cada caracter que se desea imprimir. Si no hubiera lugar suficiente para imprimir el número o los caracteres en el espacio permitido, cada uno de estos signos será reemplazado por un asterisco (*). Si hay más números luego del punto decimal que los permitidos por los signos (L) en la instrucción IMAGE, estos se redondean hasta que coincidan con los lugares decimales. Si hay menos caracteres no numéricos que los permitidos para una cadena de impresión, los lugares restantes se completan con blancos.

Para indicar que un número está dado en notación científica es necesario el signo (A) para la E de las potencias. Cuando se use el formato E deberá haber cuatro o cinco signos (A) y no más de 10 caracteres (signo (-), signo (#) y punto decimal).

El punto decimal separa las partes entera y decimal de los números y se imprime en la posición que aparece en la instrucción IMAGE.

Todas las otras letras, números y caracteres se imprimen exactamente como aparecen en la instrucción IMAGE.

El formato debe estar encerrado entre comillas. Si no es así, todos los espacios anteriores y posteriores serán ignorados. Sin embargo cuando se usa con PRINT... USING o DISPLAY… USING, debe estar encerrado entre comillas.

Cada instrucción IMAGE puede estar formada por varios formatos separados por cualquier caracter que no será el punto decimal.

Si en un PRINT... USING o DISPLAY... USING se dan más valores que formatos, estos se vuelven a usar para los valores restantes, recomenzando desde el primer formato.

Si se desea, se puede colocar un formato directamente en la instrucción PRINT... USING o DISPLAY... USING inmediatamente después del USING.

Sin embargo si un formato se usa con frecuencia, es más efectivo referirse a él a través de una instrucción IMAGE.

Ejemplos:

IMAGE $####.### permite imprimir cualquier número desde -999.999 a 9999.999. El siguiente ejemplo muestra como se imprimían ciertos valores:

```
> 100 IMAGE $####.###
> 110 PRINT USING 100:A
```

VALORES	ASPECTO
-999.999	$-999.999
-34.5	$ -34.500
0	$ 0.000
12.4565	$ 12.457
6312.9991	$6,312.999
99999999	$********

IMAGE LAS RESPUESTAS SON ### y ##.## permite la impresión de dos números. El primero va desde -99 a 999 y el segundo puede estar entre - 9.99 y 99.99. El siguiente ejemplo muestra como se imprimirán ciertos valores.

```
> 200 IMAGE LAS RESPUESTAS SON ### y ##.##
> 210 PRINT USING 200: A, B
```

VALORES		ASPECTO
-99	-9.99	LAS RESPUESTAS SON -99 y -9.99
7	-3.459	LAS RESPUESTAS SON -7 y -3.46
0	0	LAS RESPUESTAS SON 0 y 0.00
14.8	12.75	LAS RESPUESTAS SON 15 y 12.75
795	852	LAS RESPUESTAS SON 795 y *****
-984	64.7	LAS RESPUESTAS SON *** y 64.70

IMAGE

CAPITULO

4

IMAGE QUERIDO permite la impresión de una cadena de cuatro caracteres. He aquí algunos ejemplos del uso:

```
> 300 IMAGE QUERIDO ####,
> 310 PRINT USING 300: X$
```

VALORES	ASPECTO
JUAN	QUERIDO JUAN
LUIS	QUERIDO LUIS
CARLOS	QUERIDO****,

Programa:

El programa de la derecha ilustra el uso de IMAGE. Lee e imprime siete números y su total. Las líneas 110 a 120 definen los formatos. Son iguales excepto por el signo $ en la línea 110. Para conservar el espacio en blanco en el lugar del signo $, el formato de la línea 120 fue encerrado entre comillas.

La línea 180 imprime los valores usando las instrucciones IMAGE

La línea 210 muestra cómo el formato puede incluirse directamente en la instrucción PRINT... USING. Las cantidades se imprimen con los puntos decimales alineados

```
> 100 CALL CLEAR
> 110 IMAGE $####.##
> 120 IMAGE "####.##"
> 130 DATA 233.45, -147.95, 8.4,
  37.263, -51.299, 85.2, 464
> 140 TOTAL = 0
> 150 FOR A = 1 TO 7
> 160 READ CANTIDAD
> 170 TOTAL = TOTAL + CANTIDAD
> 180 IF A = 1 THEN PRINT USING
  110: CANTIDAD ELSE PRINT
  USING 120: CANTIDAD
> 190 NEXT A
> 200 PRINT "------“
> 210 PRINT USING "$####.##":
  TOTAL
> RUN
 $      233.45
       -147.95
          8.40
         37.26
        -51.30
         85.20
        464.00
        ------
 $      629.06
```

El programa de la derecha muestra el efecto de usar más valores en la instrucción PRINT… USING que los formatos que hay en la instrucción IMAGE

```
> 100 IMAGE ###.## ###.##
> 110 PRINT USING 100:50.34,50.34,
  37.26, 37.26
> RUN
  50.34, 50.3
  37.26, 37.3
```

INIT Subprograma

CAPITULO 4

Formato:

CALL INIT

Descripción:

El subprograma INIT se usa junto con LINK, LOAD y PEEK para acceder a subprogramas en lenguaje assembler. El subprograma INIT verifica que la Expansión de Memoria esté conectada, prepara la computadora para ejecutar programas en lenguaje assembler, y carga un conjunto de subrutinas de soporte en la Expansión de Memoria.

El subprograma INIT debe ser llamado antes que los subprogramas LOAD y LINK. INIT remueve de la Expansión de Memoria cualquier subprograma cargado previamente. El efecto de INIT dura hasta que se apaga la Expansión de Memoria y no necesita ser llamado desde cada programa que lo use.

Si la Expansión de Memoria no está conectada se recibe un mensaje de error.

INPUT

Formato:

INPUT [señal-de-input:] lista-de-variables.
Para información sobre el uso de INPUT con archivos, vea INPUT (con archivos).

Descripción:

Esta forma de instrucción INPUT se utiliza para ingresar datos desde el teclado. La instrucción INPUT suspende la ejecución del programa hasta que reciben los datos requeridos. La señal-de-input opcional puede indicar en la pantalla cuales son los datos que se esperan.

La lista-de-variables contiene las variables (escalares o elementos de arreglos, numéricos o alfanuméricos) a los que se asignarán valores cuando se ejecute la instrucción INPUT. Las variables están separadas por comas. Si se ha ingresado un valor de la lista-de-variables, este mismo valor puede ser usado como subíndice dentro de la misma instrucción INPUT.

Cuando se ingresan valores alfanuméricos, pueden estar opcionalmente encerrados entre comillas. Pero si contienen blancos al principio o al final es obligatoria la colocación de las comillas. Si se ingresa más de un valor, deberá separárselos con comas.

Opciones:

La señal-de-input opcional es una expresión alfanumérica. Debe estar seguida por dos puntos (:) y se muestra en la pantalla cuando se ejecuta la instrucción INPUT. Si no se incluye la señal-de-input, aparecerá en la pantalla un signo de interrogación y un espacio para indicar que se está aguardando una entrada. Si existe una señal-de-input el texto de ella ocupará el lugar del signo de interrogación y el espacio.

Ejemplos:

INPUT X permite la entrada de un número	> 100 INPUT X
INPUT X$, 4 permite la entrada de una cadena alfanumérica y un número	> 100 INPUT X$, Y
INPUT "INGRESE DOS NUMEROS:": A, B imprime el mensaje INGRESE DOS NUMEROS y luego permite la entrada de los dos números.	> 100 INPUT "INGRESE DOS NUMEROS: ":A, B
INPUT A(J), J primero evalúa el subíndice de A y luego acepta el dato en A. Luego se acepta el valor de J.	> 100 INPUT A(J), J

INPUT J, A(J) primero acepta el dato en J y luego acepta el dato en el j-ésimo elemento del arreglo A.

```
100 INPUT J, A(7)
```

Programa:

El programa de la derecha ilustra el use de INPUT desde el teclado. Las líneas 110 a 140 permiten a la persona que usa el programa ingresar datos a medida que lo solicitan las señales de input (entrada).

Las líneas 170 a 250 constituyen una carta basada en la entrada

```
> 100 CALL CLEAR
> 110 INPUT "INGRESE SU PRIMER
  NOMBRE: ":FNOMBRE$
> 120 INPUT "INGRESE SU ULTIMO
  NOMBRE: ":LNOMBRE$
> 130 INPUT "INGRESE UN NUMERO DE
  3 DIGITOS: ":DOLARES
> 140 INPUT "INGRESE UN NUMERO DE
  2 DIGITOS: ":CENTAVOS
> 150 IMAGE DE $###.## Y QUE SI TU
> 160 CALL CLEAR
> 170 PRINT "QUERIDO"; FNOMBRE$ ; ",
  ": :
> 180 PRINT "ESTO ES PARA
  RECORDARTE"
> 190 PRINT "QUE NOS DEBES LA
  CANTIDAD"
> 200 PRINT USING 150: DOLARES +
  CENTAVOS / 100
> 210 PRINT "NO NOS PAGAS, PRONTO
  RECIBIRAS"
> 220 PRINT "UNA CARTA DE
  NUESTRO"
> 230 PRINT "ABOGADO, DIRIGIDA A"
> 240 PRINT FNOMBRE$;“ “; LNOMBRE$;
  “ ! ::
> 250 PRINT TAB (15); "SINCERA-
  MENTE,” :::TAB(15);"TE MOLESTA" ::::
> 260 GOTO 260
```

(Presione **FCTN 4** para detener el programa)

INPUT (con archivos)

Formato:

INPUT # N°-de-archivo [, REC N°-de-registro]: lista-de-variables.
(Para información sobre ingreso de datos desde el teclado vea INPUT)

Descripción:

La instrucción INPUT usada con archivos permite leer datos desde esos archivos. La instrucción INPUT sólo puede usarse con archivos abiertos en modo INPUT o UPDATE. Los archivos DISPLAY no pueden tener más de 160 caracteres en cada registro.

N°-de-archivo y lista-de-variables son obligatorios en la instrucción INPUT. N°-de-registro se puede incluir opcionalmente cuando se está trabajando con archivos tipo RELATIVE en diskettes.

Todas las instrucciones que hacen referencia a archivos, necesitan un N°-de-archivo entre 0 y 255. El N°-de-archivo se asigna a un archivo en particular mediante la instrucción OPEN. El archivo número 0 está dedicado a la consola y el teclado de la computadora.

No se puede usar para otros archivos y está siempre abierto. El N°-de-archivo se ingresa con el símbolo # seguido de una expresión numérica, que al ser redondeada da un número entre 0 y 255 y es el número de un archivo abierto.

Lista-de-variables es la lista de las variables en que se ubicarán los datos a medida que sean leídos. Está formada por variables numéricas o alfanuméricas separadas por coma, con una coma opcional al final.

Opciones:

Opcionalmente se puede especificar el número del registro que se desea leer como N°-de-registro. Esto vale solamente para archivos en diskette que han sido abiertos como RELATIVE.
El primer registro de un archivo es el número 0.

Ejemplos:

INPUT # 1:X$ coloca en X$ el próximo valor disponible en el archivo que se abrió como # 1	> 100 INPUT # 1:X$
INPUT # 23:X, A, LL$ coloca en X, A y LL$ los próximos tres valores del archivo que se abrió como # 23.	> 100 INPUT # 23:X, A, LL$
INPUT# 11, REC 44: TAX coloca un TAX el primer valor del registro número 44 del archivo que se abrió como # 11	> 100 INPUT # 11, REC 44: TAX
INPUT # 3: A, B, C, coloca en A, B y C los siguientes tres valores del archivo que se abrió como # 3. La coma después de C crea una condición de entrada pendiente. Cuando se ejecute la próxima instrucción INPUT o LINPUT que use este archivo, ocurrirá una de las siguientes acciones: Si la próxima instrucción INPUT o LINPUT no tiene cláusula REC, la computadora usa los datos que comienzan donde se detuvo el INPUT previo. Si la siguiente instrucción INPUT o LINPUT incluye una cláusula REC, la computadora termina la condición de entrada pendiente y lee el registro especificado.	> 100 INPUT # 3: A, B, C,

INPUT (con archivos)

Programa:

El programa de la derecha ilustra uno de los usos de la instrucción INPUT. Abre un archivo en un grabador de cassettes y graba cinco registros. Luego vuelve hacia atrás lee los registros y los muestra en la pantalla

```
> 100 OPEN # 1: "CS1", SEQUENTIAL,
  INTERNAL, OUTPUT, FIXED 64
> 110 FOR A = 1 TO 5
> 120 PRINT # 1: "ESTE ES EL
  REGISTRO", A
> 130 NEXT A
> 140 CLOSE # 1
> 150 CALL CLEAR
> 160 OPEN # 1: "CS1", SEQUENTIAL,
  INTERNAL, INPUT, FIXED 64
> 170 FOR B 1 TO 5
> 180 INPUT # 1: A$, C
> 190 DISPLAY AT (B,1):A$; C
> 200 NEXT B
> 210 CLOSE # 1
> RUN
  *REWIND CASSETTE TAPE CS1 THEN
  PRESS ENTER
  *PRESS CASSETTE RECORD CS1
  THEN PRESS ENTER *PRESS
  CASSETTE STOP CS1 THEN PRESS
  ENTER
  *REWIND CASSETTE TAPE CS1 THEN
  PRESS ENTER
  *PRESS CASSETTE PLAY CS1 THEN
  PRESS ENTER ESTE ES EL REGISTRO
  1 ESTE ES EL REGISTRO 2 ESTE ES EL
  REGISTRO 3 ESTE ES EL REGISTRO 4
  ESTE ES EL REGISTRO 5 *PRESS
  CASSETTE STOP CS1 THEN PRESS
  ENTER
```

Vea el manual del Sistema de Discos para el uso de diskettes.

INT

CAPITULO

4

Formato:

INT (expresión-numérica)
Descripción:
La función INT devuelve el entero menor que o igual a la expresión numérica.

Ejemplos:

PRINT INT (3,4) imprime 3	> 100 PRINT INT (3,4)
X = INT (3.9) hace por igual a 3	> 100 X = INT (3.9)
P = INT (3.99999999) hace P igual a 3	> 100 P = INT (3.99999999)
DISPLAY AT (3,7) = INT (4.0) coloca 4 en la fila 3 columna 7	> 100 DISPLAY AT (3,7) : INT(4.0)
N = INT (-3.9) hace N igual a – 4	> 100 N = INT (— 3.9)
K = INT (-3.0000001) hace K igual a – 4	> 100 K = INT (-3.0000001)

JOYST Subprograma

Formato:

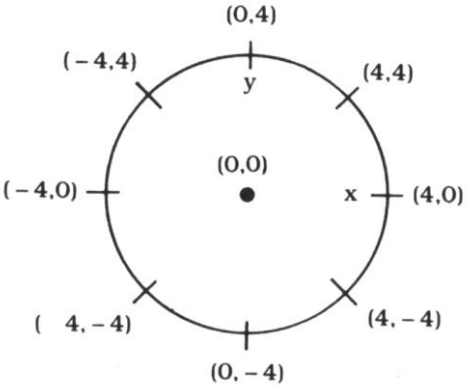
CALL JOYST (unidad-clave, x, y)

Descripción:

El subprograma JOYST entrega datos en x e y de acuerdo con la posición de la palabra de control del Controlador Remoto (disponible por separado) rotulada como unidad-clave.

Unidad-clave es una expresión numérica con un valor entre 1 y 4. Los valores 1 y 2 corresponden a las palancas 1 y 2; los valores 3 y 4 están reservados para un posible uso futuro.

Los valores devueltos en x e y dependen de la posición de la palanca. Esos valores se muestran más abajo. El primer valor entre paréntesis corresponde a la x y el segundo valor corresponde a la y.



Ejemplo:

CALL JOYST (1, X, Y) devuelve los valores de x e y de acuerdo con la posición de la palanca de control número 1.

```
> 100 CALL JOYST (1, X, Y)
```

Programa:

El programa de la derecha ilustra el uso del subprograma JOYST. Crea un sprite y le da movimiento de acuerdo con la entrada desde una palanca de control

```
> 100 CALL CLEAR
> 110 CALL SPRITE (# 1, 33, 5, 96, 1 28)
> 120 CALL JOYST (1, X, Y)
> 130 CALL MOTION (# 1, —Y, X)
> 140 GOTO 120
```

(Presione **FCTN 4** para detener el programa)

KEY Subprograma

CAPITULO 4

Formato:

CALL KEY (unidad-clave, variable-de-salida, variable-de-estado)

Descripción:

El subprograma KEY asigna el código de la tecla que se ha presionado, a la variable-de-salida. El valor asignado depende de la unidad-clave especificada. Si unidad-clave es 0, la entrada se toma del teclado completo y el valor que asume la variable-de-salida es el código ASCII de la tecla que se ha presionado. Si no se ha presionado ninguna tecla la variable-de-salida tendrá un valor -1. Vea en el Apéndice C la lista de códigos ASCII.

Si unidad-clave es 1, la entrada se toma del lado izquierdo del teclado. Si es 2, la entrada se toma del lado derecho. En el Apéndice J se dan los posibles valores que puede tomar la variable-de-salida. Los valores 3, 4 y 5 están reservados para posibles usos futuros.

La variable-de-estado indica si se ha presionado una nueva tecla o no. Un valor 1 significa que se ha presionado una nueva tecla desde que se ejecutó el último CALL KEY. Un valor -1 significa que se ha presionado la misma tecla que en el último CALL KEY. Un valor cero implica que no se ha presionado ninguna tecla.

Ejemplo:

CALL KEY (0, K, S) devuelve en K el código ASCII de cualquiera de las teclas del teclado, y en S devuelve un valor que indica si se ha presionado alguna tecla.

```
> 100 CALL KEY (0,K, S)
```

Programa:

El programa de la derecha ilustra el use del subprograma KEY Crea un sprite y lo mueve de acuerdo a la entrada desde el lado izquierdo del teclado. Note que la línea 130 vuelve a la 120 si no se ha presionado ninguna tecla.

```
> 100 CALL CLEAR
> 110 CALL SPRITE (#1, 33,5,96,1 28)
> 120 CALL KEY (1, K, S)
> 130 IF S = 0 THEN 120
> 140 IF K = 5 THEN Y = -4
> 150 IF K = 0 THEN Y = 4
> 160 IF K = 2 THEN X = —4
> 170 IF K = 3 THEN X = 4
> 180 IF K = 1 THEN X, Y = 0
> 190 IF K > 5 THEN X,Y = 0
> 200 CALL MOTION (#1, Y, X)
> 210 GOTO 120
```

(Presione FCTN 4 para detener el programa)

LEN

Formato:

LEN (expresión-alfanumérica)

Descripción:

La función LEN devuelve la cantidad de caracteres de una expresión-alfanumérica Un espacio se cuenta como un caracter.

Ejemplos:

PRINT LEN ("ABCDE") imprime 5	> 100 PRINT LEN ("ABCDE")
X = LEN ("ESTO ES UNA INSTRUCCION.") pone en X el valor 24	> 100 X = LEN ("ESTO ES UNA INSTRUCCION.")
DISPLAY LEN ("") muestra un 0	> 100 DISPLAY LEN ("")
DISPLAY LEN (" ") muestra un 1	> 100 DISPLAY LEN ("")

LET

CAPITULO

4

Formato:

LET variable-numérica [, variable-numérica,…] = expresión-numérica
LET variable-alfanumérica [, variable-alfanumérica,…] = expresión- alfanumérica

Descripción:

La instrucción LET asigna el valor de una expresión a la/las variable(s) especificada(s). La computadora evalúa la expresión de la derecha y coloca el valor en la(s) variable(s) de la izquierda. Si se indica más de una variable a la izquierda, éstas deben estar separadas por comas. LET es opcional, y se omite en los ejemplos de este manual. Todos los subíndices de las variables de la izquierda se evalúan antes de hacer las asignaciones.

Se pueden usar operadores relacionales y lógicos en la expresión numérica. Si la relación o el valor lógico es verdadero, se asigna un -1 a la variable-numérica. Si es falso se le asigna un valor 0.

Ejemplos:

T = 4 coloca el valor 4 en T	> 100 T = 4
X, Y, Z = 12.4 coloca el valor 12.4 en X, Y y Z.	> 100 X, Y, Z =12.4
A = 3 < 5 coloca -1 en A dado que se verdad que 3 es menor que cinco.	> 100 A = 3 < 5
B = 12 < 7 coloca 0 en B dado que es falso que 12 sea menor que 7	> 100 B = 12 < 7
I, A(I) = 3 coloca 3 en A(I) con el valor que pudiera haber tomado antes I. Luego coloca 3 en 1	> 100 I, A (I) = 3
L$, D$, B$ = "B" coloca "B" en L$ D$ y B$	> 100 L$, D$, B$ = "B"

LINK Subprograma

Formato:

CALL LINK (nombre-subprograma [, lista-de-argumentos])

Descripción:

El subprograma LINK se usa junto con INIT, LOAD y PEEK para acceder a subprogramas en lenguaje assembler. El subprograma LINK pasa el control y opcionalmente, una lista de parámetros desde un programa en TI BASIC Extendido a un subprograma en lenguaje assembler.

El nombre-subprograma es el nombre del subprograma que se está llamando. Previamente se lo debe cargar en la Expansión de Memoria con la instrucción o el comando CALL LOAD.

La lista-de-argumentos es una lista de variables y expresiones que requiere el subprograma assembler al que se está llamando.

LINPUT

CAPITULO

4

Formato:

LINPUT [[#N°-de-archivo][, REC N°-de-registro]:] variable-alfanumérica
LINPUT [señal-de-input:] variable-alfanumérica

Descripción:

La instrucción LINPUT permite la asignación de una línea completa, un registro de un archivo, o (si hay un registro de entrada pendiente) la porción restante de un registro de un archivo. en una variable-alfanumérica. No se realiza ninguna edición de lo que entra, de manera que las comas, blancos iniciales y finales, puntos y comas, dos puntos, y comillas se incluyen en la variable-alfanumérica tal como entran.

Opciones:

Debe especificarse un N°-de-archivo. Si el archivo tiene formato RELATIVE, se puede hacer referencia a un registro específico con REC. El archivo debe ser de tipo DISPLAY. Si no se especifica archivo se puede emitir una señal-de-input antes de aceptar la entrada desde el teclado.

Ejemplos:

LINPUT L$ asigna a L$ cualquier cosa que se mecanografía antes de presionar ENTER

```
> 100 LINPUT L$
```

LINPUT "NOMBRE:":NM$ muestra el mensaje NOMBRE: y asigna a NM$ cualquier cosa que se mecanografía antes de presionar ENTER.

```
> 100 LINPUT "NOMBRE:":NM$
```

LINPUT #1, REC M: L$(M) asigna a L$(M) el valor que estaba en el registro M, del archivo abierto como # 1.

```
> 100 LINPUT # 1, REC M:L$ (M)
```

Programa:

El programa de la derecha ilustra el use de LINPUT Lee un archivo ya existente y muestra sólo las líneas que contienen la palabra "SOL"

```
> 100 OPEN # 1: "DSK1.TEXT1,
  INPUT, FIXED 80, DISPLAY
> 110 IF EOF(1) THEN CLOSE #1 ::
  STOP
> 120 LINPUT # 1:A$
> 130 I = POS(A$, "SOL", 1)
> 140 IF I < > 0 THEN PRINT A$
> 150 GOTO 110
```

LIST

Formato:

LIST ["nombre-dispositivo":][N°-de-línea]
LIST ["nombre-dispositivo":][N°-línea-comienzo]-[N°-línea-final]

Descripción:

El comando LIST permite mostrar en la pantalla las líneas de programa. Si se ingresa LIST sin indicar ningún N° de línea, se lista el programa completo tal como está en la memoria. Si se indica un número de línea, sólo se lista la línea especificada. Si se indica un número de línea seguido de un guión se listan esa línea y las líneas subsiguientes. Si luego de LIST aparece un número de línea precedido por un guión, se listan esa línea y todas las que la preceden. Si se indican dos números separados por un guión, se listan las líneas indicadas y todas las líneas que están entre ellas.

Presionando y sosteniendo cualquier tecla mientras se lista el programa, se puede detener temporalmente el listado para poder ver una porción determinada en la pantalla.
Presionando nuevamente cualquier otra tecla, continúa el listado. Asimismo, presionando **FCTN 4** (CLEAR) se detiene el listado

Opciones:

El listado normalmente se muestra en la pantalla. Si se desea que salga por algún otro dispositivo, tal como la impresora térmica opcional o la interfase RS232 se debe especificar el nombre-dispositivo.

Ejemplos:

LIST lista por la pantalla el programa completo tal como está en la memoria	> LIST
LIST 100 lista la línea 100	> LIST 100
LIST 100 lista la línea 100 y todas las líneas que lo siguen	> LIST 100 –
LIST-200 lista todas las líneas anteriores a la 200 y la 200 inclusive.	> LIST -200
LIST 100-200 lista todas las líneas desde 100 hasta 200	> LIST 100-200
LIST "TP" lista el programa completo en la impresora térmica opcional.	> LIST "TP"
LIST "TP": -200 lista las líneas desde el principio del programa hasta la 200 por la impresora técnica opcional.	> LIST "TP": -200

LOAD Subprograma

CAPITULO

4

Formato:

CALL LOAD ("nombre-acceso" [, dirección, bytes 1][...], campo-archivo,…])

Descripción:

El subprograma LOAD se usa junto con INIT, LINK y PEEK para acceder a subprogramas en lenguaje assembler. El subprograma LOAD carga un archivo objeto en lenguaje assembler o datos directos en la Expansión de Memoria para su posterior ejecución usando la instrucción LINK.

El subprograma LOAD puede especificar uno o más archivos de los cuales se cargarán datos objeto o listas de datos de carga directa que consisten de una dirección seguida de bytes de datos.

La dirección y los bytes de datos están separados por comas.

Los datos de carga directa están separadas por campo-archivo, que es una expresión alfanumérica que especifica un archivo del cual se cargará un código-objeto en lenguaje assembler. Campo-archivo puede ser una cadena nula cuando se usa solo para separar datos de carga directa.

El use del subprograma LOAD con valores incorrectos puede provocar una interrupción en la computadora, que obligue a apagarla y encenderla nuevamente.

Los nombres de subprogramas en lenguaje assembler (ver LINK) se incluyen en el archivo.

LOCATE Subprograma

Formato:

CALL LOCATE (#N°-de-sprite, punto-fila, punto-columna [,...])

Descripción:

El subprograma LOCATE se usa para cambiar la posición de el/los sprite(s) dado(s) al punto-fila, punto-columna indicados. El punto-fila y punto-columna se numeran consecutivamente comenzando con 1 en la esquina superior izquierda de la pantalla. Punto-fila va desde 1 a 192 y punto-columna va desde 1 a 256. (En realidad el punto-fila puede llegar hasta 256 pero las posiciones 193 a 256 caen fuera de la pantalla).

La ubicación del sprite está dada por la posición de la esquina superior izquierda del caracter que lo define.

Programa:

El programa de la derecha ilustra el use del subprograma LOCATE. La línea 110 crea un sprite con la forma de un signo ! rojo que se mueve rápidamente
La línea 140 coloca el sprite en una ubicación elegida al azar en las líneas 120 a 130 La línea 150 repite el proceso

```
> 100 CALL CLEAR
> 110 CALL SPRITE ( # 1, 33, 7, 1, 1, 25, 25)
> 120 YLOC = INT (RND*150 + 1)
> 130 XLOC = INT (RND*200 + 1)
> 140 CALL LOCATE (# 1, YLOC, XLOC)
> 150 GOTO 120
```

(Presione **FCTN 4** para detener el programa)

Vea también el tercer ejemplo del subprograma SPRITE.

LOG

Formato:

LOG (expresión-numérica)

Descripción:

La función LOG devuelve el logaritmo natural de la expresión-numérica donde expresión-numérica es mayor que cero. La función LOG es la inversa de la función EXP.

Ejemplos:

PRINT LOG (3.4) imprime el logaritmo natural de 3.4 que es 1.22377543122

```
> 100 PRINT LOG(3.4)
```

X = LOG(EXP(7.2)) pone en X el logaritmo natural de elevado a la 7.2 que es 7.2

```
> 100 X = LOG(EXP(7.2) )
```

S = LOG(SQR(T)) hace S igual al logaritmo natural de la raíz cuadrada del valor de T.

```
> 100 S = LOG(SQR (T) )
```

Programa:

El programa de la derecha devuelve el logaritmo de cualquier número positivo en cualquier base.

```
> 100 CALL CLEAR
> 110 INPUT "BASE: ": B
> 120 IF B <= 1 THEN 110
> 130 INPUT "NUMERO: ": N
> 140 IF N < = THEN 130
> 150 LG = LOG(N)/LOG(B)
> 160 PRINT "LOG BASE"; B; "DE"; N; "ES";
  LG
> 170 GOTO 110
```

(Presione **FCTN 4** para detener el programa)

MAGNIFY Subprograma

Formato:

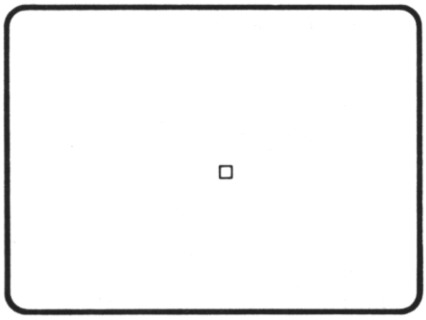
CALL MAGNIFY (factor-de-aumento)

Descripción:

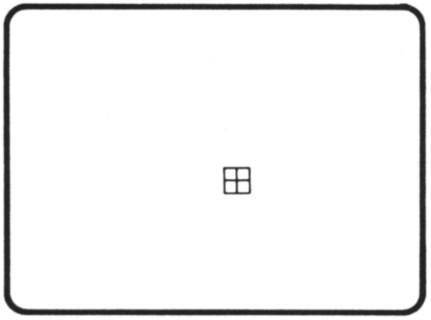
El subprograma MAGNIFY permite determinar el tamaño de los sprites y cuántos caracteres conforman cada sprite. El MAGNIFY afecta a todos los sprites. Los factores-de-aumento pueden ser 1, 2, 3 ó 4.

Si no hay ningún CALL MAGNIFY en el programa, el factor-de-aumento standard es 1.

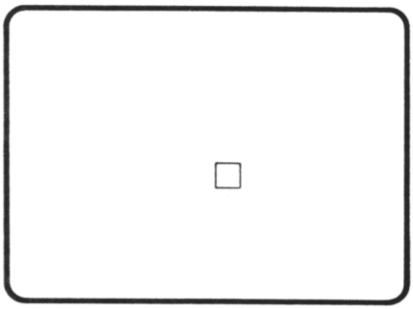
Un factor-de-aumento igual a 1 hace que todos los sprites, sean de tamaño simple no aumento. Esto quiere decir que cada sprite, está definido por el único caracter especificado cuando se creó y ocupa la posición de un solo caracter en la pantalla.



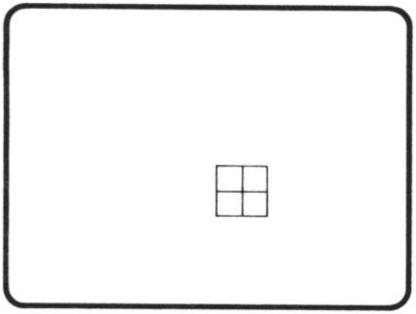
Un factor de aumento igual a 2 hace que todos los sprites sean de tamaño simple aumentado. Es decir que cada sprite está definido por el único caracter especificado cuando se creó, pero ocupa la posición de cuatro caracteres en la pantalla. Cada posición de punto en el caracter especificado se expande para ocupar cuatro posiciones de punto en la pantalla. La expansión a partir de un factor-de-aumento de 1 es hacia abajo y a la derecha.



Un factor-de-aumento igual a 3 hace que todos los sprites tengan tamaño doble no aumentado. Esto significa que cada sprite está definido por cuatro posiciones de caracter que incluyen el caracter especificado. El primer caracter es el que se definió al crear el sprite si su número es divisible exactamente por cuatro, o el número siguiente más pequeño que sea divisible exactamente por cuatro.
Ese caracter ocupa el cuarto superior izquierda del sprite. El siguiente caracter es el cuarto inferior izquierdo del sprite. El siguiente caracter es el cuarto superior derecho del sprite. Y el caracter final es el cuarto inferior derecho del sprite. El caracter definido cuando se creó el sprite es uno de los cuatro caracteres que constituyen el sprite. El sprite ocupa cuatro posiciones de caracter en la pantalla.



Un factor-de-aumento igual a 4 hace que todos los sprites tengan tamaño doble aumentado. Es decir que cada sprite queda definido por cuatro posiciones que incluyen el caracter especificado. El primer caracter es el que se definió al crear el sprite si su número es divisible exactamente por cuatro, o el número siguiente más pequeño que sea divisible exactamente por cuatro. Ese caracter ocupa el cuarto superior izquierdo del sprite. El siguiente caracter ocupa el cuarto inferior izquierdo. El siguiente ocupa el cuarto superior derecho. El caracter final ocupa el cuarto inferior derecho del sprite.
El caracter definido cuando se creó el sprite es uno de los cuatro caracteres que lo constituyen. El sprite ocupa dieciséis posiciones de caracter en la pantalla. La expansión desde un factor-de-aumento igual a 3 es hacia abajo y a la derecha.



SUBPROGRAMA MAGNIFY

Programa:

El siguiente programa ilustra el use del subprograma MAGNIFY. Cuando se ejecuta, aparece una pequeña figura en el centro de la pantalla. En seguida aumenta al doble de su tamaño, ocupando cuatro posiciones de caracter.
El otro momento es reemplazada por la esquina superior izquierda de una figura más grande, ocupando todavía cuatro posiciones. Luego aparece la figura completa, ocupando dieciséis posiciones. Finalmente se reduce en tamaño a cuatro posiciones de caracter.

La línea 110 define el caracter 96.
La línea 120 define un sprite que usa el caracter 96. Por omisión, el factor de aumento es 1. La línea 140 cambia el factor-de-aumento a 2
La línea 160 redefine el caracter 96. Dado que la definición
tiene 64 caracteres de longitud también define a los caracteres 97, 98 y 99. La línea 180 cambia el factor-de-aumento a 4.
La línea 200 cambia el factor-de-aumento a 3.

```
> 100 CALL CLEAR
> 110 CALL CHAR (96,"1898FF3D3C
  3CE404")
> 120 CALL SPRITE (# 1, 96, 5, 92, 1
  24)
> 130 GOSUB 230
> 140 CALL MAGNIFY (2)
> 150 GOSUB 230
> 160 CALL CHAR (96, "0103C3417
  F3F07070707077E7C40000080C0C0
  80FCFEE2E3E0E0E06060606070")
> 170 GOSUB 230
> 180 CALL MAGNIFY (4)
> 190 GOSUB 230
> 200 CALL MAGNIFY (3)
> 210 GOSUB 230
> 220 STOP
> 230 REM DEMORA
> 240 FOR DEMORA = 1 TO 500
> 250 NEXT DEMORA
> 260 RETURN
```

MAX

Formato:

MAX (expresión-numérica-1, expresión-numérica-2)

Descripción:

La función MAX devuelve el mayor valor entre expresión-numérica-1 y expresión-numérica-2. Si son iguales, luego devuelve ese valor.

Ejemplos:

PRINT MAX (3,8) imprime 8	> 100 PRINT MAX (3,8)
F = MAX (3E12, 1800000) hace F igual a 3E12.	> 100 F = MAX (3E12, 1800000)
G = MAX (-12, -4) hace G igual a -4.	> 100 G = MAX (-12, -4)
L = MAX (A, B) hace L igual a 7. Si A es 7 y B es -5	> 100 L = MAX (A, B)

MERGE

Formato:

MERGE ["] dispositivo-nombre-archivo ["]

Descripción:

El comando MERGE mezcla líneas del nombre-archivo desde un dispositivo dado, con las líneas de programa que se hallan en la memoria de la computadora. Si un número de línea del nombre-archivo es duplicado de un mínimo de línea del programa en la memoria, la nueva línea reemplaza a la vieja. De otra manera, las líneas se insertan entre las que están en la memoria. El comando MERGE no borra "breakpoints". También MERGE sólo puede ser usado con diskettes.

NOTA: Los archivos se pueden mezclar en la memoria sólo si han sido almacenado con la opción MERGE. Para más detalles vea el comando SAVE.

Ejemplo:

MERGE DSK 1 SUB mezcla el programa SUB con el programa que está en la memoria.

```
> MERGE DSK1.SUB
```

Programa:

Si el programa de la derecha fue almacenado en el DSK1 como REBOTAR con la opción merge, podrá ser mezclado con programas como el que se muestra en la página siguiente

```
> 100 CALL CLEAR
> 110 RANDOMIZE
> 140 DEF RND50 = INT (RND*50 25)
> 150 GOSUB 10000
> 10000 FOR AA = 1 TO 20
> 10010 QQ = RND50
> 10020 LL = RND50
> 10030 CALL MOTION (# 1, QQ, LL)
> 10040 NEXT AA
> 10050 RETURN
> SAVE "DSK1.REBOTAR", MERGE
```

Ingrese el programa de la derecha en la memoria de la computadora

```
> 120 CALL CHAR
  (96,"18183CFFFF3C1818")
> 130 CALL SPRITE (#1, 96, 7, 92, 1
  28)
> 150 GOSUB 500
> 160 STOP
```

Ahora mezcla REBOTAR con el programa anterior.

```
> MERGE DSK1.REBOTAR
```

A la derecha se muestra el programa resultante de mezclar REBOTAR con el programa anterior

```
> LIST
> 100 CALL CLEAR
> 110 RANDOMIZE
> 120 CALL CHAR (96, "18183CFFFF
  3C1818")
> 130 CALL SPRITE (# 1, 96, 7, 92, 1
  28)
> 140 DEF RND50 = INT (RND * 50 -
  25)
```

Note que la línea 150 pertenece al programa que se mezcló y no al que estaba en la memoria.

```
> 150 GOSUB 10000
> 160 STOP
> 100000 FOR AA = 1 TO 20
> 10010 QQ = RND50
> 10020 LL = RND50
> 10030 CALL MOTION (# 1, QQ, LL)
> 10040 NEXT AA
> 10050 RETURN
```

MIN

Formato:

MIN (expresión-numérica-1, expresión-numérica-2)

Descripción:

La función MIN devuelve el valor más pequeño entre expresión-numérica-1 y expresión-numérica-2. Si son iguales se devuelve su valor.

Ejemplos:

PRINT MIN(3,8) imprime 3	> 100 PRINT MIN (3,8)
F = MIN(3E12,1800000) hace F igual a 1800000	> 100 F = MIN (3E12,1800000)
G = MIN(-12, -4) hace G igual -12	> 100 G = MIN (-12, -4)
L = MIN(A, B) hace L igual a -5 si A es 7 y B es -5	> 100 L = MIN (A, B)

MOTION Subprograma

CAPITULO **4**

Formato:

CALL MOTION (#N°-sprite, velocidad-fila, velocidad-columna [,...])

Descripción:

El subprograma MOTION se usa para especificar la velocidad-fila y velocidad-columna de un sprite. Si ambas velocidades son cero, el sprite está estacionario. Una velocidad-fila positiva mueve el sprite hacia abajo y una negativa lo mueve hacia arriba. Una velocidad-columna positiva mueve el sprite a la derecha, y una negativa lo mueve a la izquierda.
Si ambas velocidades son distintas de cero, el sprite se mueve suavemente en un ángulo determinado por los valores de las dos velocidades.

Las velocidades de fila y columna pueden estar entre —128 y 127. Un valor cercano a 0 es muy lento. Un valor lejos de cero es muy rápido. Cuando un sprite llega al borde de la pantalla desaparece, y reaparece en la posición correspondiente sobre el borde opuesto de la misma.

Programa:

El programa de la derecha ilustra un ejemplo del subprograma Motion.

```
> 100 CALL CLEAR
```

La línea 110 crea un sprite.

```
> 110 CALL SPRITE (# 1, 33, 5, 92, 1, 24)
```

Las líneas 120 y 130 definen los valores para el movimiento del sprite.

```
> 120 FOR XVEL = —16 TO 16 STEP 2
> 130 FOR YVEL = —16 TO 16 STEP 2
```

La línea 140 muestra los valores de ambas velocidades.

```
> 140 DISPLAY AT (12,11): XVEL; YVEL
```

La línea 150 define el movimiento del sprite.

```
> 150 CALL MOTION (# 1, YVEL, XVEL)
```

Las líneas 160 y 170 completan los ciclos que definen el movimiento del sprite.

```
> 160 NEXT YVEL
> 170 NEXT XVEL
```

NEW

Formato:

NEW

Descripción:

El comando NEW limpia la memoria y la pantalla y prepara la computadora para un nuevo programa.
Todos los valores y caracteres definidos se vuelven indefinidos. Cualquier archivo que esté abierto, se cierra. Los caracteres 32 a 95 vuelven a su representación standard. Se cancelan los comandos BREAK Y TRACE.

Debe asegurarse de almacenar el programa en el que ha estado trabajando antes de ingresar NEW dado que una vez dado el comando el programa es irrecuperable.

NEXT

Formato:

NEXT variable-de-control

Vea ON BREAK, ON WARNING Y RETURN (con ON ERROR) para el uso de la cláusula NEXT con eras construcciones.

Descripción:

La instrucción NEXT está siempre asociada con la instrucción FOR TO STEP formando un ciclo. La variable-de-control debe ser la misma variable-de-control de la instrucción FOR-TO-STEP. La instrucción NEXT no puede aparecer en una instrucción IF-THEN-ELSE.

La instrucción NEXT controla la repetición del ciclo. Cada vez que se ejecuta esta instrucción, al valor de la variable-de-control cambia de acuerdo al STEP definido en la instrucción FOR-TO-STEP, o se encuentra en 1 si no hay cláusula STEP. Así, la variable-de-control al finalizar el ciclo contiene siempre el primer valor fuera del rango definido por la instrucción FOR-TO-STEP. Vea FOR-TO-STEP para mayor información.

Programa:

El programa de la derecha ilustra el uso de la instrucción NEXT en las líneas 130 y 140

```
> 100 TOTAL = 0
> 110 FOR CONT = 10 TO 0 STEP -2
> 120 TOTAL = TOTAL + CONT
> 130 NEXT CONT
> 140 FOR DEMORA 1 TO 100 ::
  NEXT DEMORA
> 150 PRINT TOTAL,CONT; DEMORA
> RUN
  30 -2 101
```

NUMBER

Formato:

NUMBER [línea-inicial][, incremento]
NUM [línea-inicial][, incremento]

Descripción:

El comando NUMBER genera números de línea secuenciales, facilitando el ingreso de líneas de programa sin necesidad de mecanografiar los números de línea. Si no se especifica línea-inicial e incremento, los números de línea comienzan con el 100 con incrementos de 10. Si se está en Modo Comando se puede indicar NUMBER en cualquier momento.
Si la línea solicitada ya existe, se muestra en la pantalla y se puede cambiar o corregir con las funciones de edición, o presionar ENTER para confirmarla. Para dejar el modo NUMBER presione ENTER cuando aparece un número de línea sin instrucción o presione FCTN 4 (CLEAR) si se ha desplegado alguna línea. NUMBER se puede abreviar como NUM.

Opciones:

Se puede especificar línea inicial y/o incremento.

Ejemplo:

En el siguiente ejemplo está SUBRAYADO lo que se debe mecanografiar. Presione ENTER luego de cada línea NUM pide que la numeración comience con 100 con incrementos de 10

NUM 110 pide que la numeración comience con 110 con incrementos de 10. Cambie la línea a Z = 11.

NUM 105,5 pide que la numeración comience en la línea 105 con incrementos de 5
La línea 110 ya existía

```
> NUM
> 100 X = 4
> 110 Z = 10
> 120
> 110 Z = 11
> 120 PRINT  (Y + X) / Z
> 130
> NUM 105,5
> 105 Y = 7
> 110 Z = 11
> 115
> LIST
  100 X = 4
  105 Y = 7
  110 Z = 11
  120 PRINT (Y + X) / Z
```

OLD

Formato:

OLD ["] dispositivo-nombre-programa ["]

Descripción:

El comando OLD carga en la memoria el programa nombre-programa desde un dispositivo. El programa tiene que haber sido almacenado en el dispositivo con el comando SAVE. OLD cierra todos los archivos que están abiertos y remueve el programa que está en la memoria antes de cargar el nombre-programa. Para agregar línea de otro programa al que ya está en la memoria, vea el comando MERGE.

Dispositivo puede tener varios significados. Si es CS1 ó CS2 designa a uno de los dos posibles grabados cassette, por lo tanto no es necesario dar nombre-programa. El programa que se carga es el que está en el cassette. En la pantalla se muestran instrucciones para operar el grabador de cassettes.

Vea el manual del Sistema de Discos para usar OLD con diskettes.

Ejemplos:

OLD CS1 carga un programa desde el grabador de cassettes en la memoria de la computadora	> OLD CS1
OLD "DSK1.MYPROG" carga el programa MYPROG en la memoria desde el drive 1.	> OLD "DSK1.MYPROG"
OLD DSK.DISK3.UPDATE80 carga el programa UPDATE80 desde el diskette llamado DISK3 en la memoria de la computadora.	> OLD DSK.DISK3.UPDATE80

ON BREAK

Formato:

ON BREAK STOP
ON BREAK NEXT

Descripción:

La instrucción ON BREAK determina que acción tomar si se encuentra un "breakpoint" durante la ejecución del programa. La acción por omisión es STOP, lo que hace que el programa se detenga y se imprima el mensaje standard de "breakpoint". La alternativa es NEXT, que transfiere el control a la próxima instrucción sin tener en cuenta el "breakpoint".

Se puede usar ON BREAN NEXT para evitar "breakpoints" que se pusieran en un programa, con propósitos de seguimiento. (NOTA: ON BREAK NEXT no tiene ningún efecto sobre una instrucción BREAK que no está seguida por un número de línea. En este caso el "breakpoint" ocurrirá aún cuando se haya ejecutado ON BREAK NEXT). Cuando ON BREAK NEXT está actuando, la interrupción externa **FCTN 4** (CLEAR) no detiene el programa. En ese caso solamente **FCTN +** (QUIT) puede detenerlo. **FCTN +** (QUIT) borra el programa, devuelve la pantalla principal y puede interferir la correcta operación de algunos dispositivos externos, tales como diskettes.

Programa:

El programa de la derecha ilustra el uso de ON BREAK. La línea 110 define un "breakpoint" para la línea 150. La línea 120 hace que el "breakpoint" se ignore y continúa con la próxima línea. A pesar de la línea 120 ocurrirá un "breakpoint" en la línea 130. Ingresa CONTINUE No ocurre ningún "breakpoint" en la línea 150 a causa de la línea 120. **FCTN 4** (CLEAR) no
tiene efecto durante la ejecución de la líneas 140 a 160 a causa de la línea 120. La línea 170 devuelve el uso normal de **FCTN 4** (CLEAR)

```
> 100 CALL CLEAR
> 110 BREAK 150
> 120 ON BREAK NEXT
> 130 BREAK
> 140 FOR A = 1 TO 50
> 150 PRINT FCTN 4 NO TIENE
  EFECTO"
> 160 NEXT A
> 170 ON BREAK STOP
> 180 FOR A = 1 TO 50
> 190 PRINT "AHORA TODO ES
  NORMAL"
> 200 NEXT A
```

ON ERROR

CAPITULO 4

Formato:

ON ERROR STOP
ON ERROR N°-de-línea

Descripción:

La instrucción ON ERROR determina la acción a tomar si ocurre un error durante la ejecución de un programa. La acción por omisión es STOP, que hace que se imprima el mensaje de error standard y se detenga la ejecución del programa. La alternativa es dar un N°-de-línea para transferir el control en caso de error.

Una vez ocurrido el error y transferido el control, el manejo de errores vuelve a su acción normal, STOP. Si se desea que los nuevos errores se manejen diferente, habrá que ejecutar una nueva instrucción ON ERROR.

Si se especifica un N°-de-línea, éste deberá corresponder al comienzo de una subrutina similar a las que se llaman con GOSUB. Esta subrutina deberá terminar con RETURN. Vea RETURN (con ON ERROR) para mayor información.

NOTA:

Una transferencia de control a continuación de una instrucción ON ERROR actúa como la ejecución de una instrucción GOSUB. Al igual que con GOSUB y GOTO, deben evitarse las transferencias entre subprogramas. El resultado más común de una transferencia ilegal, en un subprograma, es un error de sintaxis en una instrucción que parece correcta.

Programa:

El programa de la derecha ilustra el use de ON ERROR. La línea 110 hace que cualquier error que ocurra pase en control a la línea 160. Ocurre un error en la línea 130 y el control pasa a la 160.

La línea 170 hace que el próximo error pase el control a la línea 230.
La línea 180 de información acerca del error ocurrido. La línea 190 transfiere el control a la línea 230 si el error no está en la línea esperada.
La línea 200 transfiere el control a la línea 230 si el error no está en la línea esperada.
La línea 210 combina el valor de X$ a un valor aceptable. La línea 220 devuelve el control a la línea donde ocurrió el error.
La línea 240 informa la naturaleza del error inesperado y el programa se detiene.

```
> 100 CALL CLEAR
> 110 ON ERROR 160
> 130 X = VAL (X$)

> 140 PRINT X; "CUADRADO" X*X
> 150 STOP
> 160 REM SUBRUTINA DE ERROR
> 170 ON ERROR 230
> 180 CALL ERR (COD, TIPO, LINEA)
> 190 IF LINEA <> 130 THEN
  RETURN 230
> 200 IF COD <> 74 THEN RETURN
  230

> 210 X$ = "S"
> 220 RETURN
> 230 REM ERROR DESCONOCIDO
> 240 PRINT "ERROR"; COD, "EN LA
  LINEA "; LINEA
> RUN
  5 CUADRADO ES 25
```

ON GOSUB

CAPITULO

4

Formato:

ON expresión-numérica GOSUB N°-de-línea [,…]
ON expresión-numérica GO SUB N°-de-línea [,…]

Descripción:

La instrucción ON…GOSUB transfiere el control a la subrutina que comienza en N°-de-línea, de acuerdo al valor de la expresión-numérica. Esta instrucción, más que dar una opción actúa como el GOSUB, pero es más eficiente desde el punto de vista que requiere menos líneas de programa que cuando se usa la instrucción IF-THEN-ELSE.

La expresión-numérica debe tomar un valor entre 1 y el No de línea.

Ejemplos:

ON X GOSUB 1000, 2000, 3000
transfiere el control a 1000 si X vale 1, a 2000 si X vale 2 y a 3000 si X vale 3.

```
> 100 ON X GOSUB 1000, 2000, 3000
```

ON P-4 GOSUB 200, 250, 300 800, 170
transfiere el control a 200 si P-4 es 1 (P = 5), a 250 si P-4 es 2, a 300 si P-4 es 3, a 800 si P-4 es 4 y a 170 si P-4 es 5.

```
> 100 ON P4 GOSUB, 200,250, 300 800,
  170
```

Programa:

El programa de la derecha ilustra el use de ON…GOSUB
La línea 220 determina donde ir, de acuerdo con el valor de ELECCION.

```
> 100 CALL CLEAR
> 110 DISPLAY AT (11,1): "ELIJA UNA DE
  LAS SIGUIENTES "OPCIONES"
> 120 DISPLAY AT (13,1): "1. SUMA DE DOS
  NUMEROS".
> 130 DISPLAY AT (14,1): "2.
  MULTIPLICACION DE 2 NUMEROS".
> 140 DISPLAY AT (15,1): "3. RESTA DE
  DOS NUMEROS".
> 150 DISPLAY AT (20,1): "SU ELECCION",
> 160 DISPLAY AT (22,2): "PRIMER
  NUMERO:"
> 170 DISPLAY AT (23,1): "SEGUNDO
  NUMERO:"
> 180 ACCEPT AT (20,14) VALIDATE
  (NUMERIC): ELECCION
> 190 IF ELECCION < 1 OR ELECCION > 3
  THEN 180
> 200 ACCEPT AT (22,16) VALIDATE
  (NUMERIC): PRIMERO
> 210 ACCEPT AT (23,16) VALIDATE
  (NUMERIC): SEGUNDO
> 220 ON ELECCION GOSUB 240, 260, 280
> 230 GO TO 180
> 240 DISPLAY AT (3,1): PRIMERO; "MAS";
  SEGUNDO; "IGUAL"; PRIMERO +
  SEGUNDO
> 250 RETURN
> 260 DISPLAY AT (3,1): PRIMERO "POR" ,
  SEGUNDO; "IGUAL"; PRIMERO
  *SEGUNDO
> 270 RETURN
> 280 DISPLAY AT (3,1): PRIMERO;
  "MENOS"; SEGUNDO; "IGUAL";
  PRIMERO-SEGUNDO
> 290 RETURN
```

(Presione **FCTN 4** para detener el programa)

ON GOTO

CAPITULO

4

Formato:

ON expresión-numérica GOTO N°-de-línea [,…]
ON expresión-numérica GO TO N°-de-línea [,…]

Descripción:

La instrucción ON…GOTO transfiere el control al N°-de-línea en la posición correspondiente al valor de la expresión-numérica. La instrucción, más que dar una opción, actúa de la misma forma que la instrucción GOTO, pero es más eficiente en el sentido que requiere menos líneas de programa que el use de instrucción IFTHEN-ELSE.

Expresión-numérica debe tener un valor entre 1 y la cantidad de N°-de-línea.

Ejemplos:

ON X GOTO 1000,2000, 300 transfiere el control a 1000 si X vale 1, a 2000 si X vale 2 y a 300 si X vale 3.
La instrucción equivalente usando IF-THEN-ELSE es IF X = 1 THEN 1000 ELSE IF X = 2 THEN 2000 ELSE IF X = 3 THEN 300 ELSE PRINT "ERROR!" :: STOP

```
> 100 ON X GOTO 1000, 2000, 300
```

ON P-4 GOTO 200, 250, 300, 800, 170 transfiere el control a 200 si P-4 es 1 (P es 5), 250 si P-4 es 2,300 si P-4 es 3,800 si P-4 es 4, y 170 si P-4 es 5.

```
> 100 ON P-4 GOTO 200, 250 300, 800, 170
```

Programa:

El programa de la derecha ilustra el use de la instrucción ON…GOTO
La línea 220 determina donde ir de acuerdo con el valor de ELECCION.

```
> 100 CALL CLEAR
> 110 DISPLAY AT (11,1): "ELIJA UNA DE
  LAS OPCIONES"
> 120 DISPLAY AT (13,1): "1. SUMA DE
  DOS NUMEROS".
> 130 DISPLAY AT (14,1): "2.
  MULTIPLICACION DE DOS NUMEROS".
> 140 DISPLAY AT (15.1): "3. RESTA DE
  DOS NUMEROS".
> 150 DISPLAY AT (20,1): "SU
  ELECCION:"
> 160 DISPLAY AT (22,2): "PRIMER
  NUMERO:"
> 170 DISPLAY AT (23,1): "SEGUNDO
  NUMERO:"
> 180 ACCEPT AT (20,14) VALIDATE
  (NUMERIC): ELECCION
> 190 INF ELECCION < 1 OR ELECCION >
  3 THEN 180
> 200 ACCEPT AT (22,16) VALIDATE
  (NUMERIC): PRIMERO
> 210 ACCEPT AT (23,16) VALIDATE
  (NUMERIC): SEGUNDO
> 220 ON ELECCION GOTO 230, 250 270
> 230 DISPLAY AT (3,1): PRIMERO;
  "MAS"; SEGUNDO; "IGUAL"; PRIMERO
  SEGUNDO
> 240 GOTO 180
> 250 DISPLAY AT (3,1): PRIMERO; "POR";
  SEGUNDO; "IGUAL": PRIMERO
  SEGUNDO
> 260 GOTO 180
> 270 DISPLAY AT (3,1): PRIMERO
  "MENOS"; SEGUNDO; "IGUAL";
  PRIMERO-SEGUNDO
> 280 GO TO 180
```

(Presione **FCTN 4** para detener el programa)

ON WARNING

CAPITULO **4**

Formato:

ON WARNING PRINT
ON WARNING STOP
ON WARNING NEXT

Descripción:

La instrucción ON WARNING determina que acción tomar si ocurre un "warning" (advertencia) durante la ejecución de un programa. La acción por omisión es PRINT, lo que hace que se imprima un mensaje standard de advertencia y el programa continúa la ejecución. Una alternativa es STOP que hace que el programa se detenga luego de la impresión del mensaje. La otra alternativa es NEXT que hace que el programa continúe la ejecución sin imprimir ningún mensaje.

Programa:

El programa de la derecha ilustra el uso de ON WARNING. La línea 110 hace que en caso de "warning" el programa continúa con la línea siguiente. La línea 120 imprime el resultado sin ningún mensaje.

```
> 100 CALL CLEAR
> 110 ON WARNING NEXT
> 120 PRINT 120, 5/0
```

La línea 130 define el manejo de "warning" en su valor standard imprimiendo el mensaje y continuando con la ejecución. La línea 140 además, imprime 140 luego del "warning" y continúa.

```
> 130 ON WARNING PRINT
> 140 PRINT 140, 5/0
```

La línea 150 hace que en presencia de un "warning" se imprima un mensaje y el proceso se detenga. La línea 160 imprime 160 y luego del mensaje de advertencia, detiene el programa.

```
> 150 ON WARNING STOP
> 160 PRINT 160, 5/0
> 170 PRINT 170
> RUN
  120              9.99999E***
  140
  *WARNING
  NUMERIC OVERFLOW IN 140 160
  * WARNING
  NUMERIC OVERFLOW IN 160
```

OPEN

Formato:

OPEN # N°-archivo: dispositivo-nombre-archivo [, organización-archivo][, tipo-archivo]
[, modo-apertura][, tipo-registro]

Descripción:

La instrucción OPEN prepara un programa BASIC para usar archivos almacenados en diskette o cassette, estableciendo un vínculo entre el N°-archivo y el archivo. Para definir este vínculo, la instrucción OPEN describe las características del archivo.

Si el archivo existe, la descripción dada en el programa debe coincidir con las características reales del archivo. Sin embargo las características de los archivos en cassette no se verifican, por lo que pueden ocurrir errores si estas son incorrectas.

N°-de-archivo debe figurar en la instrucción OPEN. Las instrucciones que se refieren a archivos, lo hacen mencionando un número entre 1 y 255. El archivo número 0 es el teclado y la pantalla de la computadora. No puede usarse para otros archivos y está siempre abierto. Se pueden asignar los otros números como se desee, dando a cada archivo un número diferente. N°-archivo se ingresa con un símbolo (#) seguido por una expresión numérica que, al evaluarse es redondeada al numero entero más próximo, dando un número entre 1 y 255. No deberá ser además el !número de un archivo que ya esté abierto.

También debe incluirse el dispositivo en la instrucción OPEN. Si dispositivo es CSI ó CS2 se refiere a uno de los dos grabadores de cassettes, por lo tanto no es necesario dar nombre-archivo. En la pantalla aparecen las instrucciones para operar el grabador de cassettes.

Si su dispositivo es DSK1, DSK2 o DSK3 se refiere a los drives de discos y nombre-archivo deberá ser el nombre de un archivo que está en el diskette del drive que se indica. Si dispositivo es DSK.nombre-diskette, donde nombre-diskette es el nombre de un diskette en uno de los drives, el nombre-archivo debe ser el nombre de un archivo que esté en nombre-diskette. La computadora busca entre los drives comenzando por DSK1 hasta que encuentra el diskette solicitado. Luego busca el nombre-archivo en el diskette.

La otra información puede aparecer en cualquier orden o bien puede omitirse. Si se omite un ítem la computadora asume ciertos valores standard que se describen más adelante.

Organización-archivo puede ser secuencial o al azar. En un archivo secuencial los registros se leen o graban uno después del otro. En un archivo al azar los registros se pueden leer o grabar en cualquier orden. Los archivos al azar también pueden procesarse secuencialmente. Para indicar la estructura del archivo deberá ingresarse SEQUENTIAL para archivos secuenciales y RELATIVE para archivos al azar. Se puede especificar opcionalmente la cantidad inicial de registros colocando a continuación de la palabra SEQUENTIAL o RELATIVE, una expresión numérica. Si no se especifica organización-archivo, se asume SEQUENTIAL.

Tipo-archivo puede ser DISPLAY o INTERNAL. Los archivos pueden grabarse en una forma de fácil lectura llamada ASCII (DISPLAY), o en una forma que sólo puede ser leída por la máquina, llamada binaria (INTERNAL), Los registros binarios ocupan menos lugar y se procesan más rápidamente. Sin embargo, si la información debe ser impresa o mostrada en la pantalla, la mejor elección es el formato ASCII.

Para especificar que se desea que el archivo esté en código ASCII se debe indicar DISPLAY. Para formato binario corresponde INTERNAL. Si no se especifica tipo-archivo se asume DISPLAY generalmente cuando se usan archivos en cassettes o diskettes. INTERNAL es la menor elección. Para archivos en la impresora térmica o la interfase RS232, la mejor es DISPLAY/.

Modo-apertura puede ser UPDATE, INPUT, OUTPUT o APPEND. Se puede indicar a la computadora que un archivo puede ser leído o grabado, que sólo puede leerse, que sólo puede grabarse o solamente se le pueden agregar registro. Sin embargo, si el archivo está marcado como protegido, no podrá grabarse y sólo se podrá abrir como INPUT.

Para poder leer y grabar un archivo debe especificarse UPDATE.

Para leerlo solamente: INPUT. Sólo para grabado: OUTPUT. Para agregar registros APPEND. El modo APPEND solo puede especificarse para registros de longitud variable. Si no se especifica modo-apertura, se asume UPDATE.

Nótese que si existe un archivo protegido en el diskette, al especificar un modo apertura OUTPUT sobre el mismo archivo, se grabarán los nuevos datos sobre los ya existentes. Se puede evitar esto moviéndose hasta el final del archivo con una instrucción RESTORE con el registro apropiado o bien abriendo el archivo con modo APPEND.

Tipo-registro puede ver VARIABLE o FIXED. Los archivos pueden tener todos los registros de longitud fija o variable. Si todos los registros tienen la misma longitud, aquellos que son más cortos son completados para salvar la diferencia. Los que tienen longitud mayor se truncan a la longitud apropiada. Se pueden especificar registros de longitud variable, ingresando VARIABLE. Y registros de longitud fija, ingresando FIXED.

Si se desea se puede especificar la longitud máxima de registro colocando luego de VARIABLE o FIXED una expresión numérica.

La máxima longitud de registro depende del dispositivo usado. Si no se específica longitud de registro los valores standard son: 80 para diskettes, 64 para cassettes, 80 para la interfase RS232 y 32 para la impresora térmica.

Los archivos RELATIVE deben tener registros de longitud FIXED. Si no se específica tipo-registro para un archivo RELATIVE, se asume FIXED.

Los archivos SEQUENTIAL pueden ser FIXED o VARIABLE. Si no se especifica tipo registro para un archivo SEQUENTIAL, se asume VARIABLE. Un archivo de longitud fija puede ser re-abierto con acceso SEQUENTIAL o VARIABLE independientemente de las asignaciones de organización-archivo.

Ejemplos:

OPEN # 1: "CS1", FIXED, OUTPUT, abre un archivo en el cassette uno. El archivo es SEQUENTIAL, con formato DISPLAY, modo OUTPUT con registros FIXED con longitud máxima de 64 bytes.	> 100 OPEN # 1: "CS1", FIXED, OUTPUT
OPEN # 23: "DSK.MYDISK.X" RELATIVE 100, INTERNAL, UPDATE, FIXED abre un archivo llamado X. El archivo está en el diskette llamado MYDISK que puede estar colocado en cualquier drive. El archivo es RELATIVE, con formato INTERNAL, registros de longitud FIXED con una longitud máxima de 80 bytes. El archivo se abre en modo UPDATE y comienza con 100 registros disponibles.	> 300 OPEN# 23: "DSK.MYDISK.X", RELATIVE 100, INTERNAL, UPDATE, FIXED
OPEN # 243 = A$, INTERNAL, si A$ es igual a "DSK2.ABC", asume un archivo en el diskette DSK2 con el nombre ABC. El archivo es SEQUENTIAL, almacenado en formato INTERNAL, modo UPDATE con registros de longitud máxima de 80 bytes.	> 100 OPEN # 243: A$, INTERNAL
OPEN # 17: "TP", OUTPUT para impresión, la impresora térmica.	> 100 OPEN # 17: "TP", OUTPUT

OPTION BASE

CAPITULO
4

Formato:

OPTION BASE 0
OPTION BASE 1

Descripción:

La instalación OPTION BASE define el subíndice más bajo disponible para arreglos, en cero o uno. El valor standard es cero. Si se usa una instrucción OPTION BASE deberá tener un número de línea más bajo que la instrucción DIM o cualquier otra instrucción que haga referencia al arreglo. Sólo puede haber un OPTION BASE en un programa y se aplica a todos los arreglos.
La instrucción OPTION BASE no puede aparecer una instrucción IF-THEN-ELSE.

Ejemplos:

OPTION BASE 1 define en 1 el valor del subíndice más bajo disponible para todos los arreglos.

```
> 100 OPTION BASE 1
```

PATTERN Subprograma

Formato:

CALL PATTERN (# N°-sprite, valor-caracter [,…])

Descripción:

El subprograma PATTERN permite cambiar el modelo de un sprite sin afectar ninguna de sus características.
N°-sprite específica qué sprite se está usando. Valor-caracter puede ser cualquier entero de 32 a 143. Vea el subprograma CHAR para información acerca de cómo definir un caracter. Vea el subprograma MAGNIFY para mayor información.

Programa:

El programa de la derecha ilustra el uso del subprograma PATTERN. Las líneas 110 a 140 construyen un piso.

```
> 100 CALL CLEAR
> 110 CALL COLOR (12, 16, 16)
> 120 FOR A= 19 TO 24
> 130 CALL HCHAR (A, 1, 120, 32)
> 140 NEXT A
```

De la línea 150 a la 200 se definen los caracteres 96 a 104.

```
> 150 A$ = "01071821214141FFFF41
  4121211907008E9884848282FF
  FF8282848498E000"
> 160 B$ = "01061820305C46818142
  46242C1807008060183424624281
  81623A000$18E000"
> 170 X$ = "0106182C244642818146
  5C3020180700806018040C346281
  814262243418E000"
> 180 CALL CHAR = (96,A$)
> 190 CALL CHAR = (100,B$)
> 200 CALL CHAR = (104,C$)
```

La línea 210 crea un sprite con forma de una rueda que comienza a moverse hacia la derecha. La línea 220 aumenta el sprite al doble de su tamaño.

```
> 210 CALL SPRITE (# 1, 96, 5, 130, 1.0.8)
> 220 CALL MAGNIFY (3)
```

Las líneas 230 a 270 hacen que los rayón de la rueda se muevan a medida que el caracter se muestra y cambia.
Vea también el tercer ejemplo del subprograma SPRITE

```
> 230 FOR A = 96 TO 104 STEP 4
> 240 CALL PATTERN (# 1 A)
> 250 FOR DEMORA = 1 TO 5 : : NEXT
  DEMORA
> 260 NEXT A
> 270 GOTO 230
```

(Presione **FCTN 4** para detener el programa)

PEEK Subprograma

CAPITULO
4

Formato:

CALL PEEK (dirección, lista-variables-numéricas)

Descripción:

El subprograma PEEK se usa junto con INIT, LINK y LOAD para acceder a subprogramas en lenguaje aseembler. El subprograma PEEK devuelve en las variables de la lista-variables-numéricas los valores que se corresponden con los valores del byte especificado por dirección y los bytes subsiguientes. PEEK puede usarse sin subprogramas en lenguaje assembler pero la información obtenida es de poca utilidad.

Con los programas en venta disponibles en cassettes o diskettes, se dan instrucciones detalladas del uso de INIT, LINK, LOAD y PEEK.

El use indiscriminado de PEEK puede hacer que la computadora "se bloque" y deba ser apagada y encendida nuevamente para poder usarla.

Ejemplo:

CALL PEEK (8192, X1, X2, X3, X4) devuelve los valores de las posiciones de memoria 8192, 8193, 8194 y 8195 en X1, X2, X3 y X4 respectivamente.

```
> 100 CALL PEEK (8192, XI, X2, X3, X4)
```

PI

Formato:

PI

Descripción:

La función PI devuelve el valor de ¶ = 3.41459265359

Ejemplo:

VOLUME = 4 / 3 * PI * 6 ^ 3 define VOLUME igual a cuatro tercios de pi multiplicado por seis al cubo. Es decir el volumen de una esfera de radio seis.

```
> 100 VOLUME = 4 / 3 * PI * 6 ^ 3
```

POS

CAPITULO

4

Formato:

POS (cadena-1, cadena-2, expresión-numérica)

Descripción:

La función POS devuelve la posición de la primera ocurrencia de la cadena-2 en la cadena-1. La búsqueda comienza en la posición especificada por expresión-numérica. Si no hay coincidencia la función devuelve el valor cero.

Ejemplos:

X = POS ("PAN", "A", 1) define el valor X igual a 2 porque A es la segunda letra de PAN.

```
> 100 X = POS ("PAN", "A", 1)
```

Y = POS ("APAN", "A", 2) define Y igual a 1 porque A es la primera letra de APAN a partir de la segunda letra

```
> 100 Y = POS ("APAN", "A", 2)
```

Z = POS ("PAN", "A", 3) define Z a cero porque A no está en la porción de PAN donde se busca.

```
> 100 Z = POS ("PAN", "A", 3)
```

R = POS ("PABNAN", "AN", 1) define R igual a 5 porque la primera ocurrencia de AN comienza con la A en la quinta posición de PABNAN

```
> 100 R = POS ("PABNAN", "AN", 1)
```

Programa:

El programa de la derecha ilustra un use de POS. En cada entrada se buscan los espacios y luego se imprime cada palabra en una línea

```
> 100 CALL CLEAR
> 110 PRINT "INGRESE UNA FRASE"
> 120 INPUT X$
> 130 S = POS (X$, " ", 1}
> 140 IF S = 0 THEN PRINT X$: : PRINT : :
  GOTO 110
> 150 Y$ = SEG$ (X$, 1, S) : : PRINT Y$
> 160 X$ = SEG$ (X$,S + 1, LEN (X$) )
> 170 GOTO 130
```

(Presione **FCTN 4** para detener el programa)

POSITION Subprograma

Formato:

CALL POSITION (# N°-sprite, punto-fila, punto-columna [,…])

Descripción:

El subprograma POSITION devuelve la posición del/de los sprite/s especificados, en punto-fila y punto-columna, en la forma de dos números entre 1 y 256. Estos puntos definen la posición de la esquina superior, izquierda del sprite. Si el sprite no está definido, punto-fila y punto-columna toman el valor cero.

El sprite continúa moviéndose luego que se le devuelve su posición. La distancia que se mueve depende de la velocidad del sprite.

Ejemplo:

CALL POSITION (# 1, Y, X) devuelve la posición de la esquina superior izquierda del sprite # 1.

```
> 100 CALL POSITION (# 1, Y, X)
```

Vea también el tercer ejemplo del subprograma SPRITE.

PRINT

CAPITULO

4

Formato:

PRINT [# N°-archivo [, REC N°-registro] :][lista-de-impresión]

Descripción:

La instrucción PRINT permite transferir los valores de los elementos de la lista-de-impresión opcional, a la pantalla o a un archivo externo o dispositivo. La lista-de-impresión está formada por constantes alfanuméricas, constantes numéricas, variables alfanuméricas y/o funciones TAB. Cada elemento de la lista-de-impresión se separa de los otros mediante un punto y coma, una coma o dos puntos.

El punto y coma, la coma y los dos puntos controlan el espaciado en la pantalla o en un archivo abierto con formato DISPLAY. Un punto y coma hace que el próximo elemento se ubique inmediatamente a continuación del elemento previo. Una coma hace que el próximo elemento de la lista-de-impresión se ubique en el siguiente campo de impresión. Cada campo de impresión tiene 14 caracteres de la longitud del registro que se está usando. En la pantalla, los campos de impresión están en las opciones 1 y 15. Si el cursor se encuentra pasando el comienzo del último campo de impresión, el próximo ítem se imprime en la línea siguiente. El símbolo dos puntos (:) hacen que el próximo elemento se ubique en la línea siguiente. Para imprimir varias líneas en blanco, se pueden poner varios símbolos (:) luego de la instrucción PRINT. Sin embargo se deben dejar espacios entre ellos para que no se confundan con el símbolo separador (::)

Se puede colocar un separador a continuación del último elemento de la lista-de-impresión, lo que afectará la ubicación del próximo PRINT, PRINT…USING, DISPLAY (sin AT), o DISPLAY…USING (sin AT) para el mismo dispositivo. Esto hace que la próxima instrucción de salida se considere como una continuación de la anterior a menos que contenga una cláusula REC.
Cuando se imprime una línea en la pantalla, todo lo que hay en ella (menos los sprites) se desplaza una línea hacia arriba (de manera que la primera línea se pierde) y una nueva línea aparece en la parte inferior.

Opciones:

El N°-archivo determina el archivo que se imprimirá. Si se omite o se coloca #0, se utiliza la pantalla.
De otro modo N°-de-archivo debe ser el número de un archivo que esté abierto. Vea OPEN.

Cláusula REC se usa para especificar el registro en el que se desea grabar los elementos de la lista-de-impresión. REC sólo se puede usar con archivos abiertos como RELATIVE. Vea OPEN.

Al grabar archivos INTERNAL, tanto la coma como el punto y coma colocan los elementos de la lista-de-impresión uno a continuación del otro. Para archivos de tipo DISPLAY, la coma y el punto y coma actúan como se indicó anteriormente; el punto y coma coloca un elemento al lado del otro, y la coma coloca el próximo elemento en el siguiente campo de impresión.

Ejemplos:

PRINT hace que aparezca una línea en blanco en la pantalla	> 100 PRINT
PRINT "LA RESPUESTA ES"; RESPUESTA imprime el texto LA RESPUESTA ES seguido inmediatamente del valor de la variable RESPUESTA. Si RESPUESTA es positiva habrá un blanco para el signo positivo después de ES	> 100 PRINT "LA RESPUESTA ES"; RESPUESTA
PRINT X: Y / 2 hace que el valor de X se imprima en una línea y el valor de Y/2 en la línea siguiente	> 100 PRINT X: Y / 2
PRINT# 12, REC 7: A hace que el valor de A se grabe en el 8° registro del archivo que fue abierto como # 12 (El registro 0 es el primer registro).	> 100 PRINT # 12, REC 7: A
PRINT # 32: A, B, C, hace que los valores de A, B y C se graben en el próximo registro del archivo que se abrió con el número 32. La próxima instrucción PRINT dirigida al archivo 32 grabará en el mismo registro que esta instrucción PRINT, a menos que se especifique un número de registro que cancela la condición de salida pendiente.	> 100 PRINT # 32: A, B, C,

PRINT # 1, REC 3: A, B seguido de PRINT # 1: C, D hace que A y B se graben en el registro 3 del archivo que se abrió con el número 1 y C y D en el registro 4 del mismo archivo.

```
> 100 PRINT # 1, REC 3: A, B
> 150 PRINT # 1: C, D
```

Programa:

El programa de la derecha imprime varios valores en diferentes posiciones de la pantalla

```
> 100 CALL CLEAR
> 110 PRINT 1;2;3;4;5;6;7;8;9;
> 120 PRINT 1, 2, 3, 4, 5, 6,
> 130 PRINT 1: 2: 3
> 140 PRINT
> 150 PRINT 1;2;3;
> 160 PRINT 4;5;6/4
> RUN
1 2 3 4 5 6 7 8 9
1          2
3          4
5          6
1
2
3
1 2 3 4 5 1.5
```

PRINT USING

Formato:

PRINT [# N°-archivo [, REC N°-registro] USING expresión-alfanumérica: lista-de-impresión
PRINT [# N° archivo [, REC N°-registro] USING N°-línea: lista-de-impresión.

Descripción:

La instrucción PRINT…USING, que especifica el formato a usar. La expresión-alfanumérica describe el formato tal Como se hace en la instrucción IMAGE. El No línea se refiere al N° de línea de una instrucción IMAGE. Vea la instrucción IMAGE para mayor información acerca del use de la expresión-alfanumérica.

Ejemplos:

PRINT USING "###.##": 32.5 imprime 32.50

```
> 100 PRINT USING "###.##": 32.5
```

PRINT USING "LA RESPUESTA ES ###.#": 123.98 imprime LA RESPUESTA ES 124.0

```
> 100 PRINT USING "LA RESPUESTA
  ES ###.#": 123.98

 LA RESPUESTA ES 124.0
```

PRINT USING 185: 37.4, -86.2 imprime los valores de 37.4 y -86.2 usando la instrucción IMAGE de la línea 185

```
> 100 PRINT USING 185: 37.4, - 86.2
```

RANDOMIZE

Formato:

RANDOMIZE [expresión-numérica]

Descripción:

La instrucción RANDOMIZE redefine una secuencia no predecible para el generador de números al azar. Si RANDOMIZE está seguido de una expresión-numérica cada vez que se ejecuta la instrucción se producirá la misma secuencia de números al azar. Valores diferentes dan secuencias diferentes.

Programa:

El programa de la derecha ilustra el use de la instrucción RANDOMIZE. Acepta un valor para expresión-numérica, e imprime los primeros 10 valores obtenidos usando la función RND

```
> 100 CALL CLEAR
> 110 INPUT "VALOR INICIAL": S
> 120 RANDOMIZE S
> 130 FOR A = 1 TO 10 :: PRINT A;
  RND :: NEXT A :: PRINT
> 140 GOTO 110
```

(Presione **FCTN 4** para detener el programa)

READ

Formato:

READ lista-variables.

Descripción:

La instrucción READ permite asignar a las variables de la lista-variables, constantes numéricas y alfanuméricas de una instrucción DATA. La lista-variables, está formada por variables numéricas y alfanuméricas separadas por comas.

Los datos generalmente se leen comenzando con el primer DATA del programa. Luego de haber leído los datos, la computadora marca el lugar donde se ha detenido, para poder continuar cuando se ejecute la próxima instrucción READ. Se puede cambiar el orden en que se leen los datos usando la instrucción RESTORE.

Vea los ejemplos de la instrucción DATA.

REC

Formato:

REC (N°-archivo)

Descripción:

La función REC devuelve el número del registro en el archivo abierto como N°-archivo, que será el próximo accedido por una instrucción PRINT, INPUT, o LINPUT.
Los registros en un archivo se numeran desde 0, es decir que el registro número 3 es el cuadro del archivo.

Ejemplo:

PRINT REC(4) imprime la posición del registro actual en el archivo que se abrió como número 4.

```
> 100 PRINT REC(4)
```

Programa:

El programa de la derecha ilustra el use de la función REC. La línea 110 abre el archivo.
Las líneas 120 a 140 graban registros en el archivo

La línea 150 posiciona el archivo en el comienzo. Las líneas 160 a 200 imprimen la posición del archivo y leen e imprimen los valores en esa posición.

La línea 210 cierra el archivo

```
> 100 CALL CLEAR
> 110 OPEN # 1: "DSK1.RNDFILE",
  RELATIVE, INTERNAL
> 120 FOR A = 0 TO 3
> 130 PRIN # 1: "ESTE ES EL REGISTRO",
  A
> 140 NEXT A
> 150 RESTORE # 1
> 160 FOR A = 0 TO 3
> 170 PRINT REC(1)
> 180 INPUT # 1  A$, B
> 190 PRINT A$; B
> 200 NEXT A
> 210 CLOSE # 1
  RUN
  0
  ESTE ES EL REGISTRO 0
  1
  ESTE ES EL REGISTRO 1
  2
  ESTE ES EL REGISTRO 2
  3
  ESTE ES EL REGISTRO 3
```

REM

Formato:

REM cadena de caracteres

Descripción:

La instrucción REM permite incluir comentarios en el programa. Los comentarios pueden tener cualquier contenido, pero generalmente se los usa para dividir secciones de programas y explicar en qué consiste la sección siguiente. No importa lo que haya a continuación de REM, incluso el separador de instrucciones (::).
Los comentarios no se ejecutan y no tiene efecto en la ejecución del programa. Sin embargo, ocupan espacio en memoria.

Ejemplos:

REM SUBRUTINA DE COMIENZO identifica una sección de comienzo de una subrutina.

```
> 100 REM SUBRUTINA DE
  COMIENZO
```

RESEQUENCE

CAPITULO
4

Formato:

RESEQUENCE [línea-inicial][, incremento]
RES [línea-inicial][, incremento]

Descripción:

El comando RESEQUENCE cambia los números de línea del programa que está en la memoria. Si no se indica línea-inicial, la renumeración comienza con 100. Si no se indica incremento, se utiliza 10 a tal fin. RESEQUENCE se puede abreviar como RES.

Además de renumerar las líneas, se cambia también cualquier referencia a ellas en instrucciones BREAK, DISPLAY…USING, GOSUB, GOTO, IF-THEN-ELSE, ON ERROR, ON…GOSUB, ON…GOTO, PRINT…USING, RESTORE, RETURN y RUN de manera que conserven la correspondencia que tenían antes del RESEQUENCE.
Si una línea a que se hace referencia en una instrucción, no existe, su número de línea se cambia por 32767.

Si, por causa de la línea-inicial y el incremento elegidos, el programa requiere números de línea mayores que 32767, el proceso de resecuencia se detiene y el programa queda sin cambios.

Ejemplos:

RES resecuencia las líneas del programa que está en la memoria comenzando con 100 con incrementos de 10	> RES
RES 1000 resecuencia las líneas del programa que está en la memoria comenzando con 1000 con incremento de 10	> RES 1000
RES 1000, 15 resecuencia las líneas del programa que está en la memoria comenzando con 1000 con incrementos de 15	> RES 1000, 15
RES, 15 resecuencia las líneas del programa que está en la memoria comenzando con 100 con incrementos de 15	> RES, 15

RESTORE

Formato:

RESTORE [Nº línea]
RESTORE # Nº archivo [, REC Nº-registro]

Descripción:

La instrucción RESTORE puede usarse ya sea con instrucciones DATA o con archivos. RESTORE determina cual será la instrucción DATA que se usará en la próxima instrucción READ. Si no se indican número-línea, la próxima instrucción READ tomará la instrucción DATA que tenga el número-línea más bajo.

Si se indica número-línea se usará la instrucción DATA que tenga ese número-línea (si no corresponde a una instrucción DATA se usa la DATA siguiente al número-línea).

Cuando se usa con archivos, la instrucción RESTORE indica qué registro se usará en la próxima instrucción PRINT, INPUT ó LINPUT refiriéndose a ese número-archivo. Si no se especifica la cláusula REC el próximo registro será el primer registro del archivo, el registro número cero. Si la cláusula REC está presente, número-registro especifica el próximo registro a usar.

Si hay una condición de salida pendiente a causa de un PRINT, DISPLAY, PRINT…USING o DISPLAY…USING, el registro pendiente se graba en el archivo antes de que se ejecute la instrucción RESTORE. La instrucción RESTORE, remueve los datos de entrada pendientes.

Ejemplos:

RESTORE hace que la próxima instrucción DATA a usar sea la primera instrucción DATA del programa	> 100 RESTORE
RESTORE 130 hace que la próxima instrucción DATA a usar sea la instrucción de la línea 130. Si la línea 130 no es una instrucción DATA se tomará la instrucción DATA que sigue a la línea 130.	> 100 RESTORE 130
RESTORE # 1 hace que el registro a usar por la próxima instrucción PRINT, INPUT o LINPUT sea el primer registro del archivo numerado como # 1	> 100 RESTORE # 1
RESTORE # 4, REC H5 hace que el próximo registro que use una instrucción PRINT, INPUT o LINPUT sea el registro H5 del archivo # 4.	> 100 RESTORE # 4, REC H5

RETURN (con GOSUB)

CAPITULO **4**

Formato:

RETURN

Descripción:

Vea también RETURN (con ON ERROR)

RETURN usado con GOSUB transfiere el control de vuelta a la instrucción siguiente al GOSUB u ON…GOSUB que se ejecutó la última vez.

Programa:

El programa de la derecha ilustra el uso de RETURN con GOSUB. El programa calcula el interés de una cantidad de dinero puesta en depósito

```
> 100 CALL CLEAR
> 110 INPUT "CANTIDAD DEPOSITADA: ":
  CANTIDAD
> 120 INPUT "TASA ANUAL DE INTERES:
  ": TASA
> 130 IF TASA < 1 THEN TASA = TASA*100
> 140 PRINT "NUMERO DE TIEMPO
  COMPUESTO"
> 150 INPUT "ANUALMENTE: ": COMP
> 160 INPUT "AÑO DE COMIENZO: ": Y
> 170 INPUT "CANTIDAD DE AROS: ": N
> 180 CALL CLEAR
> 190 FOR A = Y TO Y + N
> 200 GOSUB 240
> 210 PRINT A, INT (CANTIDAD * 100 + .5 /
  100
> 220 NEXT A
> 230 STOP
> 240 FOR B = 1 TO COMP
> 250 CANTIDAD = CANTIDAD +
  CANTIDAD * TASA / COMP * 100)
> 260 NEXT B
> 270 RETURN
```

RETURN (con ON ERROR)

Formato:

RETURN [No-línea]
RETURN NEXT

Descripción:

Vea también RETURN (con GOSUB)
RETURN se usa con ON ERROR. Luego de ejecutarse la instrucción ON ERROR se produce la transferencia de control a la línea especificada por la instrucción ON ERROR. Esa línea o lo que le sigue debería tener una instrucción RETURN. Si se especifica RETURN sin nada que la siga, el control vuelve a la instrucción donde ocurrió el error y el programa la ejecuta nuevamente.
Si se especifica RETURN seguida de un No línea, el control pasa a la línea especificada y la ejecución comienza a partir de esa línea.
Si RETURN va seguido de NEXT, se transfiere el control a la instrucción siguiente a la que causó el error.

Programa:

El programa de la derecha ilustra el uso de RETURN con ON ERROR
La línea 120 hace que si ocurre un error el control se transfiere a la línea 170. La línea 130 provoca un error. La línea 140, la línea siguiente a la que provocó el error, imprime 140. La línea 170 verifica si el error ha ocurrido cuatro veces y transfiere el control a la 220 la línea 180 incrementa el contador de errores en uno. La línea 190 imprime 190. La línea 200 hace que en caso de error el control pase a la línea 170. La línea 210 vuelve a la línea que provocó el error y la ejecuta nuevamente. La línea 220, que se ejecuta sólo si el error ha ocurrido 4 veces, imprime 220 y vuelve a la línea siguiente a la que provocó el error.

Vea también el ejemplo de la instrucción ON ERROR.

```
> 100 CALL CLEAR
> 110 A= 1
> 120 ON ERROR 170
> 130 X = VAL ("D")
> 140 PRINT 140
> 150 STOP
> 160 REM MANEJO DE ERRORES
> 170 IF A > 4 THEN 220
> 180 A = A + 1
> 190 PRINT 190
> 200 ON ERROR 170
> 210 RETURN
> 220 PRINT 220 :: RETURN
  NEXT
  RUN
  190
  190

  190
  220
  140
```

RND

Formato:

RND

Descripción:

La función RND devuelve el próximo número al azar en la secuencia de número al azar. El número devuelto está entre 0 y 1. La secuencia de números al azar devuelta es la misma cada vez que se ejecuta el programa, a menos que aparezca una instrucción RANDOMIZE.

Ejemplos:

COLOR 16 = INT(RND * 16) + 1 define COLOR 16 como un número entre 1 y 16

```
> 100 COLOR 16 = INT(RND * 16) + 1
```

VALOR = INT(RND * 16) + 10 define VALOR como un número entre 10 y 25

```
> 100 VALOR = INT(RND * 16) + 10
```

LL(8) = INT(RND * (B - A + 1)) + A hace LL(8) igual A un número entre A y B

```
> 100 LL(8) = INT(RND * (B - A + 1)) + A
```

RPT$

Formato:

RPT$ (expresión-alfanumérica, expresión-numérica)

Descripción:

La función RPT$ devuelve una cadena de caracteres formada por la expresión-alfanumérica repetida tantas veces como indica expresión-numérica. Si RPT$ produce una cadena de más de 255 caracteres de longitud, el exceso se descarta y se exhibe un mensaje de advertencia.

Ejemplos:

M$ = RPT$("ABCD",4) define M$ como "ABCDABCDABCDABCD"

> 100 M$ = RPT$ ("ABCD", 4)

CALL CHAR (96, RPT$("0000FFFF",8)) define los caracteres 96 a 99 como la cadena "0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF"

> 100 CALL CHAR (96,RPT$ ("0000FFFF", 8))

PRINT USING:RPT$ ("#",40):x$ imprime el valor de x$ usando un formato con 40 signos #

> 100 PRINT USING:RPT$("#",40):x$

RUN

CAPITULO

4

Formato:

RUN ["dispositivo:nombre-programa”]
RUN [N°-línea]

Descripción:

El comando RUN, que también puede usarse como instrucción inicia la ejecución de un programa. El programa a ejecutar primero se carga en la memoria desde el dispositivo:nombre-programa si se ha especificado esa opción. Luego se verifica el programa para ver si está libre de errores tales como ciclos FOR-NEXT a los que le falta el NEXT, y errores de sintaxis en las instrucciones.
Los valores de todas las variables numéricas se vuelven cero y los alfanuméricos se convierten en variables nulas (una cadena sin caracteres). Luego se ejecuta el programa.

Opciones:

Si se especifica dispositivo.nombre-programa, el programa solicitado se carga desde el dispositivo. El programa y los datos que estaban en la memoria se pierden.
Si se especifica N°-línea la ejecución del programa comienza en ese número de línea.

Ejemplos:

RUN hace que la computadora comience la ejecución del programa que está en la memoria.	> RUN
RUN 200 hace que la computadora comience la ejecución del programa que está en la memoria a partir de la línea 200.	> RUN 200 > 100 RUN 200
RUN "DSK1.PRG3" hace que la computadora cargue el programa PRG3 desde el drive 1, y comience su ejecución.	> RUN "DSK1.PRG3" > 320 RUN "DSK1.PRG3"

Programa:

El programa de la derecha ilustra el uso del comando RUN usado como instrucción. Crea un "menú" donde permite a la persona que opera el programa, elegir que programa desea ejecutar.
Los otros programas deben a su vez ejecutar (dar RUN) este programa de manera que una vez cumplida su tarea se vuelve al menú principal

```
> 100 CALL CLEAR
> 110 PRINT "1 PROGRAMA 1."
> 120 PRINT "2 PROGRAMA 1."
> 130 PRINT "3 PROGRAMA 2."
> 140 PRINT "4 FIN."
> 150 PRINT
> 160 INPUT "SU ELECCION:" :C
> 170 IF C = 1 THEN RUN "DSK1. PRG1"
> 180 IF C = 2 THEN RUN "DSK1.PRG 2"
> 190 IF C = 3 THEN RUN "DSK1.P RG3"
> 200 IF C = 4 THEN STOP
> 210 GOTO 100
```

SAVE

CAPITULO

4

Formato:

SAVE dispositivo.nombre-programa [, PROTECTED]
SAVE dispositivo.nombre-programa [, MERGE]

Descripción:

El comando SAVE permite copiar el programa que está en la memoria en un dispositivo externo, bajo el nombre-programa. Más tarde se podrá traer el programa nuevamente a la memoria con el comando OLD. En el Manual de Referencia del Usuario se da el método para almacenarlo en un grabador de cassettes. En el Manual del Sistema de Discos se da el método para almacenamiento en diskette. El comando SAVE borra los "breakpoints" que hay en el programa.

Opciones:

Para grabadores de cassettes sólo está disponible la opción PROTECTED.
Usando la palabra clave PROTECTED se puede especificar opcionalmente que el programa sólo podrá ser ejecutado o traído a la memoria con OLD. Dicho programa no podrá ser listado, modificado o almacenado. Este no es el mismo tipo de protección que provee el Módulo Administrador de discos. NOTA: Asegúrese de guardar una copia no protegida del programa, dado que esta característica no es reversible. Si desea también proteger el programa para que no se lo copie, use la protección que provee el Módulo Administrador de Discos.
Se puede especificar opcionalmente, que el programa estará disponible para su mezclado con otro programa, usando la palabra clave MERGE. Sólo los programas almacenados con la palabra clave MERGE, pueden mezclarse con otros programas.

Ejemplos:

SAVE DSK1.PRG1 almacena el programa que está en la memoria, en el diskette que está en el drive 1, bajo el nombre PRG1.

```
> SAVE DSK1.PRG1
```

SAVE DSK.PRG1, PROTECTED almacena el programa que está en la memoria, en el diskette que está en el drive 1, bajo el nombre PRG1. Este programa podrá cargarse y ejecutarse pero no podrá ser modificado, listado o vuelto a almacenar.

```
> SAVE DSK1.PRG1, PROTECTED
```

SAVE DSK1.PRG1.MERGE almacena el programa que está en la memoria, en el diskette que está en el drive 1, bajo el nombre PRG1. Más tarde este programa podrá mezclarse con un programa almacenado en la memoria usando el comando MERGE.

```
> SAVE DSK1.PRG1, MERGE
```

SAY Subprograma

Formato:

CALL SAY (cadena-palabras [, cadena-directa [,…])

Descripción:

El subprograma SAY hace que la computadora hable cadena-palabras o el valor especificado por cadena-directa, cuando tiene conectado el Sintetizador de Palabras en Estado Sólido (vendido por separado). Para una descripción completa de SAY, vea el Manual que viene en el Módulo Comando Editor de Palabras y Sintetizador de Palabras (ambos vendidos por separado).

El valor de cadena-palabras es uno de los que figuran en el Apéndice L. Si es da como valor literal deberá estar encerrado entre comillas. El valor de cadena-directa es el que devuelve SPGET. El valor de cadena-directa puede alterarse agregando sufijos como se indica en el Apéndice M.

La cadena-palabras y cadena-directa deben alternarse en el subprograma CALL SAY. Si se desea emitir dos cadenas-directa o cadena-palabras consecutivas se puede colocar una coma extra para indicar la posición del ítem omitido.

Ejemplos:

CALL SAY ("HELLO, HOW ARE YOU") hace que la computadora diga "Hello, how are you".

```
> 100 CALL SAY ("HELLO, HOW ARE
  YOU")
```

CALL SAY (,A$,,B$) hace que la computadora diga las palabras indicadas por A$ y B$ que deben ser provistas por SPGET.

```
> CALL SAY (,A$,,B$)
```

Programa:

El programa de la derecha ilustra el use de CALL SAY con una cadena-palabras y tres cadena-directa

```
> 100 CALL SPGET ("HOW", X$)
> 110 CALL SPGET ("ARE", Y$)
> 120 CALL SPGET ("YOU", Z$)
> 130 CALL SAY ("HELLO", X$, Y$, Z$)
```

SCREEN Subprograma

CAPITULO **4**

Formato:

CALL SCREEN (código-color)

Descripción:

El subprograma SCREEN cambia el color de la pantalla al color especificado en código-color. Todas las posiciones en la pantalla sobre la que no hay ningún caracteres o porciones de caracteres que tienen color 1 (transparente), se mostrarán con el color especificado por código-color. El color standard de la pantalla para TI BASIC Extendido es 8, azulado.

Los códigos de color son:

CODIGO	COLOR	CODIGO	COLOR
1	transparente	9	rojo mediano
2	negro	10	rojo cedro
3	verde mediano	11	amarillo oscuro
4	verde claro	12	amarillo claro
5	azul oscuro	13	verde oscuro
6	azul claro	14	fucsia
7	rojo oscuro	15	gris
8	azulado	16	blanco

Ejemplos:

CALL SCREEN(8) cambia al color standard de la pantalla, el azulado.

```
> 100 CALL SCREEN(8)
```

CALL SCREEN(2) cambia el color de la pantalla a negro.

```
> 100 CALL SCREEN(2)
```

SEG$

Formato:

SEG$ (expresión-alfanumérica, posición, longitud)

Descripción:

La función SEG$ devuelve una porción de expresión-alfanumérica. La cadena que devuelve comienza en la posición de la expresión alfanumérica, y tiene tantos caracteres como indica longitud.
Si la posición está más allá del fin de la expresión-alfanumérica, se devuelve la cadena vacía (""). Si la longitud se extiende más allá del fin de la expresión-alfanumérica sólo se devuelven los caracteres del final.

Ejemplos:

X$ = SEG$ ("PRIMERNOMBRE-ULTIMONOMBRE", 1, 12) hace X$ igual a PRIMERNOMBRE.

> 100 X$ = SEG$ (PRIMERNOMBRE-ULTIMONOMBRE, 1, 12)

X$ = SEG$ ("PRIMERNOMBRE-ULTIMONOMBRE, 13, 12) hace Y$ igual a ULTIMONOMBRE

> 100 Y$ = SEG$ ("PRIMERNOMBRE-ULTIMONOMBRE, 13, 12)

Z$ = SEG$ ("PRIMERNOMBRE-ULTIMONOMBRE, 13, 1) hace Z$ igual ""

> 100 Z$ = SEG$ ("PRIMERNOMBRE-ULTIMONOMBRE, 13, 1)

PRINT SEG$(A$, B, C) imprime la porción de A$ que comienza en el caracter B y se extiende por C caracteres.

> PRINT SEG$ (A$, B, C)

SGN

Formato:

SGN (expresión-numérica)

Descripción:

La función SGN devuelve un 1 si expresión-numérica es positiva, 0 si es cero y -1 si es negativa.

Ejemplos:

IF SGN (X2) = 1 THEN 300 ELSE 400 transfiere el control a la línea 300 si X2 es positiva y a la línea 400 si X2 es cero o negativa.

```
> 100 IF SGN (X2) = 1 THEN 300 ELSE 400
```

ON SGN(X) + 2 GOTO 200, 300, 400 transfiere el control a la línea 200 si X es negativa, a la línea 300 si es cero y a la 400 si X es positiva.

```
> 100 ON SGN (X) + 2 GOTO 200, 300, 400
```

SIN

Formato

SIN (expresión - radianes)

Descripción:

La función seno da el seno trigonométrico de expresión - radianes. Si el ángulo está en grados, multiplique el número de grados por PI/180 para obtener el ángulo equivalente en radianes.

Programa:

El programa de la derecha da el seno de varios ángulos

```
> 100 A = .5235987755982
> 100 B = 30
> 120 C = 54 * PI / 180
> 130 PRINT SIN(A); SIN(B)
> 140 PRINT SIN(B * PI / 180)
> 150 PRINT SIN(C)
> RUN
  .5 -.9880316241
  .5
  .7071067812
```

SIZE

CAPITULO

4

Formato:

SIZE

Descripción:

El comando SIZE indica la cantidad de bytes libres que quedan en la memoria de la computadora. Si tiene conectado el periférico Expansión de Memoria, el número de bytes disponibles está dado por la cantidad de espacio libre del periférico mas el espacio libre de programas. Un byte es el espacio de memoria requerido para almacenar un caracter o un dígito a una palabra clave de TI BASIC Extendido.

Si la Expansión de Memoria no está conectada, la cantidad de memoria disponible, es el espacio que queda luego de restar el espacio que ocupan: el programa, la pantalla, las definiciones de modelos de caracter, las tablas de sprites, tablas de colores valores alfanuméricos, etc.

Si la Expansión de Memoria está conectada, la cantidad de memoria disponible, el espacio que queda luego de restar el espacio que ocupan los valores alfanuméricos, la información acerca de las variables, etc. El espacio de programa es el espacio que queda luego de restar el espacio que ocupa el programa y el que ocupan los valores de las variables numéricas.

Ejemplos:

SIZE de la memoria disponible	> SIZE 13928 BYTES FREE
SIZE de la memoria disponible si la Expansión de Memoria está conectada, se dará el espacio libre de la memoria y el espacio de programas	> SIZE 13928 BYTES OF STACK FREE 24511 BYTES OF PROGRAM SPACE FREE

SOUND Subprograma

Formato:

CALL SOUND (duración, frecuencia 1, volumen 1, … frecuencia 4, volumen 4)

Descripción:

El subprograma SOUND hace que la computadora produzca sonidos o ruidos. Los valores dados controlan tres aspectos del sonido: duración; frecuencia y volumen.

VALOR	RANGO	DESCRIPCION
DURACION		La longitud del sonido en diez milésimas de segundo
FRECUENCIA		qué sonido toca
VOLUMEN		Cómo es de alto el sonido.

Duración es desde .001 a 4.250 segundos, aunque puede variar hasta 1/60 de segundo. La computadora continúa ejecutando las instrucciones del programa mientras emite el sonido. Cuando llama al subprograma SOUND, la computadora espera que se complete el sonido previo antes de ejecutar un nuevo CALL SOUND. Sin embargo si se especifica una duración negativa, el sonido previo se detiene y el nuevo es ejecutado inmediatamente.

Frecuencia específica la frecuencia de la nota que se ejecutará, con un valor entre 110 y 44733 (**NOTA**: este rango está más allá de lo que puede percibir el (oído humano. En general la habilidad de la gente para percibir notas varía, pero el valor más alto aproximado es de 10000). La frecuencia real producida por la computadora puede variar hasta un 10%. En el Apéndice D hay una lista de las notas más comunes.

Un valor entre -1 y -8 especifica uno de ocho diferentes tipos de ruidos.

FRECUENCIA	DESCRIPCION
-1	Ruido periódico Tipo 1
-2	Ruido periódico Tipo 2
-3	Ruido periódico Tipo 3
-4	Ruido periódico que varía la frecuencia del tercer tono especificado.
-5	Ruido blanco Tipo 1
-6	Ruido blanco Tipo 2
-7	Ruido blanco Tipo 3
-8	Ruido blanco que varía con la frecuencia del tercer tono especificado.

Se pueden tocar simultáneamente un máximo de 3 tonos y un ruido.

Ejemplos:

CALL SOUND (1000, 110, 0) toca A bajo C, fuertemente durante un segundo.

```
> 100 CALL SOUND (1000, 110, 0)
```

CALL SOUND (500, 110, 0, 131, 0, 196, 3) toca A bajo C bajo y C bajo fuertemente; y G bajo C no tan fuertemente durante medio segundo.

```
> 100 CALL SOUND (500, 110, 0 131,0, 196,
  3)
```

CALL SOUND 4250,-8,0 toca un ruido blanco fuerte durante 4.250 segundos.

```
> 100 CALL SOUND (4250, -8,0)
```

CALL SOUND (DUR, TONO, VOL) toca el tono indicado por TONO con la duración indicada por DUR y el volumen indicado por VOL.

```
> 100 CALL SOUND (DUR, TONO,VOL)
```

Programa:

El programa de la derecha toca las 13 notas de la primera octava que está disponible en la computadora

```
> 100 X = 2 A(1 / 12)
> 110 FOR A = 1 TO 13
> 120 CALL SOUND (100, 110 * X A, 0)
> 130 NEXT A
```

SPGET Subprograma

Formato:

CALL SPGET (cadena-palabra, cadena-salida)

Descripción:

El subprograma SPGET devuelve en cadena-salida el modelo de palabra que corresponde a cadena-palabra. Para un descripción completa del SPGET, vea el Manual que viene con Módulo Comando Editor de Palabras y Sintetizador de palabras en Estado Sólido (ambos vendidos por separado).

El valor de cadena-palabra es cualquiera de los valores que aparece en el Apéndice L. Si se da como un valor literal, debe estar encerrado entre comillas. El valor de cadena-salida se usa con SAY y puede alterarse agregando sufijos como se explica en el Apéndice M.

Programa:

El programa de la derecha ilustra el use de CALL SPGET

```
> 100 CALL SPGET ("HOW", X$)
> 110 CALL SPGET ("ARE", Y$)
> 120 CALL SAY ("HELLO' , X$)
> 130 CALL SAY ("HELLO", X$, Y$, Z$)
```

SPRITE Subprograma

CAPITULO 4

Formato:

CALL SPRITE (# N°-sprite, valor-caracter, color-sprite, punto-fila, punto-columna, velocidad-fila, velocidad-columna [.....]

Descripción:

El subprograma SPRITE crea sprites. Los sprites son gráficos que poseen color y se pueden ubicar en cualquier posición de la pantalla. Se puede definir su movimiento en cualquier dirección y con variedad de velocidades, y continúa ese movimiento hasta que el programa se detenga o lo cambie. Se mueven más suavemente que los caracteres comunes que saltan de una posición de la pantalla a la otra.

N°-sprite es una expresión numérica del 1 a 28. Si ese valor es el de un sprite que ya está definido, se borra el viejo Sprite y es reemplazado por el nuevo. Si el viejo sprite tenía velocidad de fila o columna y no se define una nueva, el nuevo sprite retiene las viejas velocidades.

Los sprites pasan por sobre los caracteres fijos en la pantalla. Cuando dos o más sprites son coincidentes, el sprite que tiene el número más bajo cubre a los otros sprites. Cuando hay cinco o más sprites sobre la misma fila de la pantalla, aquellos que tienen el número más alto desaparecen.

Valor-caracter puede ser cualquier entero entre 32 y 143.

Vea el subprograma CHAR para información sobre como definir caracteres. El valor caracter puede cambiarse con el subprograma PATTERN. El sprite se define como el carácter dado y, en el caso de sprites de tamaño doble, los próximos tres caracteres. Vea el subprograma MAGNIFY para mayor información.

Color-sprite puede ser cualquier expresión numérica de 1 a 16.

Determina el color de frente del sprite. El color de fondo es siempre 1, transparente. Vea los subprogramas COLOR y SCREEN para más información.

Punto-fila y punto-columna se numeran consecutivamente comenzando con 1 en la esquina superior izquierda de la pantalla. Punto-fila puede estar entre 1 y 129 y punto-columna entre 1 y 256. (En realidad punto-fila puede ir hasta 256, pero las posiciones 193 a 256 caen fuera de la pantalla). La posición del sprite está dada por la esquina superior izquierda del caracter que lo define.

Usando los subprogramas POSITION, COINC, y DISTANCE se puede obtener información acerca de la posición del sprite. Esta puede cambiarse usando el subprograma DELSPRITE. Cuando ocurre un "breakpoint" o el programa se detiene, los sprites dejan de existir y no vuelven a aparecer aunque se indique CONTINUE.

Opciones:

Velocidad-fila y velocidad-columna pueden especificarse opcionalmente cuando se crea el Sprite. Si ambas velocidades son cero el sprite está estacionario. Una velocidad-fila positiva mueve el sprite hacia abajo, y una negativa lo mueve hacia arriba. Una velocidad-columna positiva mueve el sprite hacia la derecha y una negativa lo mueve hacia la izquierda. Si ambas velocidades son distintas de cero, el sprite se mueve en ángulo en la dirección determinada por los valores actuales.

Velocidad-fila y velocidad-columna pueden estar entre -128 y 127. Un valor cercano a cero es muy lento. Un valor lejos de cero es muy rápido. Cuando un sprite llega al borde de la pantalla, desaparece, y vuelve a aparecer en la posición correspondiente del borde opuesto. La velocidad del sprite puede cambiarse usando el subprograma MOTION.

Programas:

Los tres programas siguientes muestran algunos usos posibles de los sprites. El tercer programa usa todos los subprogramas relacionados con sprites, excepto COLOR Y DISTANCE.

```
> 100 CALL CLEAR
> 110 CALL CHAR (96, "FFFFFFFFFFFFFF")
> 120 CALL CHAR (98, "183C7EFFF7E3C18")
> 130 CALL CHAR (100,
  "F00FF00FF00FF00F")
> 140 CALL SPRITE (#1,96,5,92,1
  24,#2,100,7,1, 1)
> 150 CALL SPRITE (#28,33,16,12 48,1,1)
> 160 CALL SPRITE (#15,98,14,1, 1,127,-128)
> 170 GO TO 170
```

La línea 140 crea un sprite azul oscuro, en el centro de la pantalla y un rojo oscuro en la esquina superior derecha y comienza a moverlo lentamente un ángulo de 45° hacia abajo y a la derecha. El sprite es un signo de admiración.

La línea 160 crea un sprite en la esquina superior izquierda de la pantalla y comienza a moverse muy rápido un ángulo de 45° hacia arriba y a la derecha

(Presione **FCTN 4** para detener el programa)

SUBPROGRAMA SPRITE

El programa de la derecha hace un uso espectacular de los sprites. La línea 110 define el caracter 96. La línea 150 define los sprites, 28 en total. El N°-sprite es el valor que toma A. El valor-caracter es 96. El color sprite es INT (A / 3) + 3. Los puntos iniciales de fila y columna son 92 y 124 el centro de la pantalla. Las velocidades de fila y columna se eligen al azar usando el valor A* INT(RND + 4.5) 2. 25 + A / 2 + SGN(RND - .5). La línea 170 hace que la secuencia se repita.

```
> 100 CALL CLEAR
> 110 CALL CHAR
  (96,"008081C7FC0808")
> 120 RANDOMIZE
> 130 CALL SCREEN(2)
> 140 FOR A= 1 TO 28
> 150 CALL SPRITE (#A, 96, INT(A/ 3) +
  3, 92, 124, A* INT(RDN * 4.5) - 2.25 + A
  / 2 = SGN(RND - .5), A* INT(RND * 4.5)
  - 2.25 + A / 2 * SGN(RND - .5) )
> 160 NEXT A
> 170 GOTO 140
```

(Presione **FCTN 4** para detener el programa)

El siguiente programa usa todos los subprogramas que se relacionan con sprites, excepto COLOR y DISTANCE. Estos subprogramas son: CHAR, COING, DELSPRITE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION y SPRITE.

El programa crea dos sprites de tamaño doble aumentados, con la forma de una persona caminando por un piso. Hay una barrera. Uno de ellos la salta y el otro pasa por encima de ella. El que la salta va un poquito más rápido cada vez, de manera que eventualmente alcanza al otro. Cuando lo logra, los dos se convierten en sprites de doble tamaño no aumentado y continuarán caminando. Cuando se encuentran la segunda vez aquel que iba más rápido desaparece, y el otro sigue caminando.

Las líneas 110, 120, 140, 150, 250 y 260 definen los sprites

La línea 130 pone en cero el contador de encuentros

Las líneas 170 a 200 construyen el piso

```
> 100 CALL CLEAR
> 110 S1$ =  "010303010303030303030303030
  303030303038000C0C0C0C0C0C0C0C0C
  0C0C0C0E0"
> 120 S2$ = "0103030103070F1B1B0303030
  60C0C0E8000008000E0F0D8CCC0C0C0C
  060303038"
> 130 CONT = 0
> 140 CALL CHAR(96, S1$)
> 150 CALL CHAR(100, S2$)
> 160 CALL SCREEN(14)
> 170 CALL COLOR(14, 13, 13)
> 180 FOR A = 19 TO 24
> 190 CALL HCHAR(A, 1, 136, 32)
> 200 NEXT A
```

Las líneas 210 a 240 constituyen la barrera

```
> 210 CALL COLOR(13, 15, 15)
> 220 CALL VCHAR(14, 22, 128, 6)
> 230 CALL VCHAR(14, 23, 128, 6)
> 240 CALL VCHAR(14, 24, 128, 6)
> 250 CALL SPRITE(#1, 96, 5, 113, 129, #
  2, 96, 7, 113, 9)
> 260 CALL MAGNIFY(4)
```

La línea 270 define la velocidad inicial que se acelerará

```
> 270 YDIR = 4
> 280 PAT = 2
```

La línea 290 pone los sprites en movimiento

```
> 290 CALL MOTION(#1, 0, YDIR, # 2, 0,
  4)
```

La línea 300 crea la ilusión de la caminata

```
> 300 CALL PATTERN(#1, 98 + PAT, # 2,
  98 - PAT)
> 310 PAT = - PAT
```

La línea 320 verifica si todos los sprites se han encontrado.
Si los sprites se encontraron la línea 330 transfiere el control. Las líneas 340 y 350 verifican si el sprite ha encontrado la barrera y transfiera el control si fue así.

```
> 320 CALL COINC(ALL, CO)
> 330 IF CO <> 0 THEN 370
> 340 CALL POSITION(#1, YPOS1 ,
  YPOS1)
> 350 IF YPOS1 > 136 AND YPOS1 < 192
  THEN 470
```

La línea 360 cierra el ciclo para que continúen caminando. Las líneas 370 a 460 manejan los sprites moviéndose entre ellos

```
> 360 GOTO 300
> 370 REM COINCIDENCIA
> 380 CALL MOTION(#1, 0, 0, #2, 0, 0)
> 390 CALL PATTERN(#1, 96, #2, 96)
```

La línea 400 verifica el primer encuentro.

```
> 400 IF CONT > 0 THEN 540
```

La línea 410 incrementa el contador de encuentros. La línea 420 encuentra su posición.

```
> 410 CONT = CONT + 1
> 420 CALL POSITION(#1, YPOS1,
  YPOS1, #2, YPOS2, YPOS2)
```

La línea 430 los empequeñece

```
> 430 CALL MAGNIFY(3)
```

La línea 440 los coloca sobre el piso y mueve el más rápido hacia adelante.

```
> 440 CALL LOCATE(#1, YPOS1 + 16,
  YPOS1 + 8, #2, YPOS2 + 16,YPOS2)
```

La línea 450 comienza a moverlos nuevamente

```
> 450 CALL MOTION(#1, 0, YDIR, #2, 0,
  4)
> 460 GOTO 340
```

Las líneas 470 a 530 hacen que el sprite salte la barrera La línea 480 lo detiene. La línea 490 encuentra su posición La línea 500 lo coloca en una nueva posición más allá de la barrera.
Las líneas 510 y 520 comienzan a moverlo más rápidamente
Las líneas 540 a 640 manejan

La línea 560 comienza a mover el sprite más lento, mientras que la 570 borra el sprite rápido. Las líneas 580 a 630 hacen que sprite más lento de 20 pasos.

```
> 470 REM# 1 ALCANZA LA
> 480 CALL MOTION (# 1, 0, 0)
> 490 CALL POSITION (# 1, YPOS1,
  YPOS1)
> 500 CALL LOCATE (# 1, YPOS 1 , 193)
> 510 YDIR YDIR + 1
> 520 CALL MOTION (# 1, 0, YDIR)
> 530 GOTO 300
> 540 REM SEGUNDA COINCIDENCIA
> 550 FOR DEMORA = 1 TO 500: : NEYT
  DEMORA
> 560 CALL MOTION (# 2, 0, 4)
> 570 DELSPRITE (# 1)
> 580 FOR PASO 1 = 1 TO 20
> 590 CALL PATTERN (# 2, 100)
> 600 FOR DEMORA = 1 TO 20 : : NEYT
  DEMORA
> 610 CALL PATTERN (# 2, 96)
> 620 FOR DEMORA = 1 TO 20 : : NEYT
  DEMORA
> 630 NEYT PASO1
> 640 CALL CLEAR
```

SQR

Formato:

SQR (expresión-numérica)

Descripción:

La función SQR devuelve la raíz cuadrada positiva de expresión numérica. SQR(X) equivale a X ^ (1 / 2). Expresión-numérica no puede ser un número negativo.

Ejemplos:

PRINT SQR(4) imprimir 2
Y = SQR(2.57E5) hace E igual al valor de la raíz cuadrada de 257000, que es 506, 9516742

```
> 100 PRINT SQR(4)
> 100 Y = SQR(2.57E5)
```

STOP

Formato:

STOP

Descripción:

La instrucción STOP detiene la ejecución del programa. Puede colocarse en el lugar del END, salvo al final de los subprogramas.

Programas:

El programa de la derecha ilustra el uso de la instrucción STOP. El programa suma los números de 1 a 100

```
> 100 CALL CLEAR
> 110 TOT = 0
> 120 NUMB = 1
> 130 TOT = TOT + NUMB
> 140 NUMB = NUMB + 1
> 150 IF NUMB  > 100 THEN PRINT
  TOT :: STOP
> 160 GOTO 130
```

STR$

Formato:

STR$ (expresión-numérica)

Descripción:

La función STR$ devuelve la cadena-alfanumérica equivalente a expresión-numérica. Esto permite que las funciones, instrucciones y comandos, que actúan sobre cadenas alfanuméricas, puedan usar la representación en caracteres, de la expresión-numérica.
La función STR$ es la inversa de la función VAL.

Ejemplos:

NUM$ = STR$ (78.6) hace NUM$ igual a "78.6"

```
> 100 NUM$ = STR$ (78.6)
```

LL$ = STR$ (3E15) hace LL$ igual a "3E15".

```
> 100 LL$ = STR (3E15)
```

I$ = STR$ (A * 4) define I$ igual a la cadena alfanumérica que se obtenga de multiplicar A por 4.
Por ejemplo si A es igual a -8, I$ es igual a "-32".

```
> 100 I$ = STR$ (A * 4)
```

SUB

Formato:

SUB nombre-subprograma [(lista-parámetros)]

Descripción:

La instrucción SUB es la primera instrucción de un subprograma. Los subprogramas se usan cuando se desea separar un grupo de instrucciones, del programa principal. Se pueden usar subprograma para realizar la misma operación varias veces dentro de un mismo programa, o en programas diferentes, o para usar variables que son específicas del subprograma. La instrucción SUB no puede aparecer en un instrucción IF-THEN-ELSE.

Los subprogramas se llaman con CALL nombre-subprograma(lista-parámetros). Los subprogramas se terminan con SUBEND y se pueden abandonar con SUBEND o SUBEYIT. En todos los casos el control vuelve a la instrucción siguiente a la que lo llamó. Nunca se debe transferir el control fuera de un subprograma, con una instrucción que no sea SUBEND o SUBEXIT. Esto incluye el paso de control con ON ERROR.

Cuando un subprograma está en un programa, debe seguir al programa principal. La estructura de un programa debe ser la siguiente:

Comienzo del Programa Principal	
:	
Llamados a subprogramas	
:	
Fin del Programa Principal	El programa se detendrá aquí sin necesidad de instrucción STOP o END
Comienzo del Primer Subprograma	Los subprogramas son opcionales
:	
Fin del Primer Subprograma	
	No debe aparecer nada entre sus programas que no sean comentarios o una instrucción END
Comienzo del Segundo Subprograma	
:	
Fin del Segundo Subprograma	
	Solo pueden aparecer END y comentarios luego de los subprogramas.
Fin del Programa.	

Opciones:

Todas las variables usadas en un subprograma, que no figuren en la lista-parámetros, son locales para ese subprograma. De manera que se pueden usar los mismos nombres de variables, que en el programa principal o en otros subprogramas, y alterar sus valores sin que esto tenga efecto sobre las otras variables. De la misma manera, las variables del programa principal o de los otros subprogramas no afectan las variables del subprograma (sin embargo, las instrucciones DATA están disponibles para subprogramas).

La comunicación de valores desde y hacia el programa principal se hace con la lista-parámetros opcional. Los parámetros no necesitan tener el mismo nombre que en la instrucción de llamada, pero deben ser del mismo tipo de datos (numérico o alfanumérico), y deben tener el mismo orden que en la instrucción CALL. Si las variables simples que pasan a los subprogramas, cambian su valor, también cambiarán los valores en el programa principal. También cambiarán su valor en el programa principal, los elementos de arreglos (por ej.: A(1)) que hayan cambiado su valor en el subprograma.

Un valor que está dado en la instrucción de llamada, como una expresión, se pasa solamente como un valor, y cualquier cambio que se haga en el subprograma, no afectará el valor del programa principal. Se pasan arreglos enteros por referencia, de manera que cualquier cambio de los elementos en el subprograma también produce cambio en el programa principal. Los arreglos se indican colocando un paréntesis a continuación del nombre de parámetro. Si el arreglo tiene más de una dimensión, deberá colocarse dentro del paréntesis una coma por cada dimensión adicional.

Si se desea se pueden pasar valores sólo para variables simples, encerrándolos entre paréntesis. Luego, el valor podrá usarse en el subprograma pero no cambiará en la vuelta al programa principal. Por ejemplo, CALL SPRG 1 ((A)) pasa el valor de A al Subprograma que comienza con SUB SPRGI (Y) y permite que ese valor se use en Y, pero no cambia el valor de A en el programa principal aún si cambia el valor de Y en el subprograma.

Si se llama a un subprograma más de una vez, las variables locales usadas en el subprograma, retienen sus valores entre una llamada y la siguiente.

Ejemplos:

SUB MENU marca el comienzo de un subprograma. No hay parámetros de entrada ni de salida.

```
> 100 SUB MENU
```

SUB MENU(CONT, ELECCION) marca el comienzo de un subprograma. Las variables CONT y ELECCION pueden ser usadas y/a sus valores cambiados en el subprograma y serán devueltas a las variables en la misma posición en la instrucción de llamada.

```
> 100 SUB MENU (CONT, ELECCION)
```

SUB PAYCHECK(DATE, A, SSN, PAYRATE, TABLE(,)) marca el comienzo de un subprograma.
Las variables DATE, Q, SSN, PAYRATE y el arreglo TABLE con dos dimensiones, pueden ser usados y cambiados sus valores y luego se devolverán en la misma posición de la instrucción de llamada.

```
> 100 SUB PAYCHECK (DATE, Q,
  SSN, SSN, PAYRATE, TABLE (,) )
```

Programa:

El programa de la derecha ilustra el use de Sub. El subprograma MENU fue almacenado previamente con la opción merge. Imprimir un MENU y pide una elección. El programa principal dice al subprograma cuántas elecciones hay y cuales son. Luego usa la elección hecha en el subprograma para determinar que programa ejecutar.
Comienzo del subprograma MENU

Note que R no es la misma R que se usa en las líneas 100 y 110 del programa principal

```
> 100 CALL MENU (5.R)
> 110 ON R GOTO 120,130,140,150, 16-
> 120 RUN "DSK1.PAGOS"
> 130 RUN "DSK1. ENTRADAS"
> 140 RUN "DSK1. SUELDOS"
> 150 RUN "DSK1.INVENTARIO"
> 160 RUN "DSK1. LIBRO-MAYOR"
> 170 DATA CUENTAS A PAGAR,
  CUENTAS A COBRAR SUELDOS,
  INVENTARIO, LIBRO MAYOR
> 10000 SUB MENU (CONT,
  ELECCION)
> 10010 CALL CLEAR
> 10020 IF CONT > 22 THEN PRINT
  "DEMASIADOS ITEMS":. ELEC-
  CION = 0 :: SUBEXIT
> 10030 RESTORE
> 10040 FOR R = 1 TO CONT
> 10050 READ TEMP$
> 10060 TEMP$ = SEG$ (TEMP$, 1, 25)
> 10070 DISPLAY AT (R,1) : R;TEMP$
> 10080 NEXT R
> 10090 DISPLAY AT (R + 1, 1): "SU
  ELECCION : 1"
> 10100 ACCEPT AT (R + 1,14) BEEP
  VALIDATE (DIGIT) SIZE (-2) :
  ELECCION
> 10110 IF ELECCION < 1 OR
  ELECCION > CONT THEN 10100
> 10120 SUBEND
```

SUBEND

Formato:

SUBEND

Descripción:

La instrucción SUBEND marca el final de un subprograma. Cuando se ejecuta SUB-END, el control pasa a la instrucción siguiente a aquella que llamó al subprograma. La instrucción SUBEND debe ser siempre la última instrucción del subprograma. No puede formar parte de una instrucción IF-THEN-ELSE.
Las únicas instrucciones que pueden seguir inmediatamente a SUBEND son REM, END, o la instrucción SUB para el próximo subprograma.

SUBEXIT

Formato:

SUBEXIT

Descripción:

La instrucción SUBEXIT permite abandonar un subprograma antes de llegar al final indicado por SUBEND. Cuando se ejecuta, el control pasa a la instrucción que sigue a la que llamó al subprograma.
No es necesario que la instrucción SUBEXIT esté en un subprograma.

TAB

CAPITULO

4

Formato:

TAB (expresión-numérica)

Descripción:

La función TAB especifica la posición de comienzo del próximo ítem de impresión en una instrucción PRINT, PRINT ... USING, DISPLAY o DISPLAY ... USING. Si la expresión-numérica es más larga que la longitud del registro a imprimir según el dispositivo, (por ejemplo: 28 para pantalla, 32 para la impresora térmica, el valor especificado para un cassette o diskette), se reducirá repetidamente en cantidades iguales a la longitud de registro hasta que su longitud esté entre 1 y la long. de registro.

Si el número de caracter que ya se han impreso en el registro es menor o igual que la expresión-numérica, el próximo ítem de impresión se imprime comenzando en la posición indicada por expresión-numérica. Si el número de caracteres que ya se han impreso es mayor que la posición indicada por expresión-numérica, el próximo ítem de impresión se imprimirá en el próximo registro comenzando en la posición indicada por expresión-mínima.

La función TAB se trata como un ítem de impresión, de manera que debe tener un separador en impresión (coma, punto y coma, dos puntos) antes y/o después de ella. El separador de impresión anterior a TAB se evalúa antes de la función. Generalmente se usa (;) después de TAB.

Ejemplos:

PRINT TAB(12); 35 imprime el número 35 en la 12° posición

> 100 PRINT TAB (12); 35

PRINT 356; TAB(18); "NOMBRE" imprime 356 al comienzo de la línea y NOMBRE en la 18o posición de la línea.

> 100 PRINT 356 ; TAB(18); "NOMBRE"

PRINT "ABCDEFGHIJKLM";TAB(5); "NOP" imprime ABCDEFGHIJKLM en el comienzo de la línea y NOP en la 5^0 posición de la próxima línea.

> 100 PRINT "ABCDEFGHIJKLM"; TAB(5); "NOP"

DISPLAY AT (12,1): "NOMBRE"; TAB (15): "DIRECCION" muestra en pantalla la palabra NOMBRE al comienzo de la fila 12, columna 1 y DIRECCION en la posición 15o de la 12o línea.

> 100 DISPLAY AT (12,1): "NOMBRE" TAB(15); "DIRECCION"

VAL

Formato:

TAN (expresión-radianes)

Descripción:

La función TAN de la tangente trigonométrica de la expresión radianes. Si el ángulo está en grados, multiplique el número de grados por PI/180 para obtener el ángulo equivalente en radianes.

Programa:

El programa de la derecha de la tangente de varios ángulos.

```
> 100 A = .7853981633973
> 110 B= 26.565051177
> 120 C = 45*PI/180
> 130 PRINT TAN(A); TAN(B)
> 140 PRINT TAN (B*PI/180)
> 150 PRINT TAN(C)
> RUN
  1.  7.17470553
  .5
  1
```

TRACE

Formato:

TRACE

Descripción:

El comando TRACE hace que se muestre en pantalla los números de línea de un programa en ejecución, antes de que se ejecute cada Esto permite seguir el curso del programa con el propósito de encontrar errores. El comando TRACE se puede usar como instrucción. Su efecto se cancela cuando se de NEW o UNTRACE.

Ejemplo:

TRACE hace que la computadora muestre los números de línea del programa, en la pantalla

```
> TRACE
> 100 TRACE
```

UNBREAK

Formato:

UNBREAK (lista-líneas)

Descripción:

El comando UNBREAK remueve todos los "breakpoints". Opcionalmente se puede definir sólo para los de la lista-líneas. UNBREAK puede ser usado como instrucción.

Ejemplos:

UNBREAK remueve todos los "breakpoints"	> UNBREAK > 420 INBREAK
UNBREAK 100,300 remueve los "breakpoints" de las líneas 100 y 130.	> UNBREAK 100,130 > 320 UNBREAK 100,130

UNTRACE

Formato:

UNTRACE

Descripción:

El comando UNTRACE remueve el efecto del comando TRACE. UNTRACE puede ser usado como instrucción.

Ejemplo:

UNTRACE remueve el efecto del TRACE	> UNTRACE > 420 UNTRACE

VAL

Formato:

VAL(expresión-alfanumérica)

Descripción:

La función VAL devuelve el valor equivalente a la expresión-numérica.
Esto permite que las funciones, instrucciones y comandos que actúan sobre números pueda usar la expresión alfanumérica.
La función VAL es la inversa de la función STR$.

Ejemplos:

NUM = VAL ("78.6") hace NUM igual a 78.6 > 100 NUM = VAL("78.8")

LL = VAL ("3E15") hace LL igual a 3E15 > 100 LL = VAL("3E15")

VCHAR

Formato:

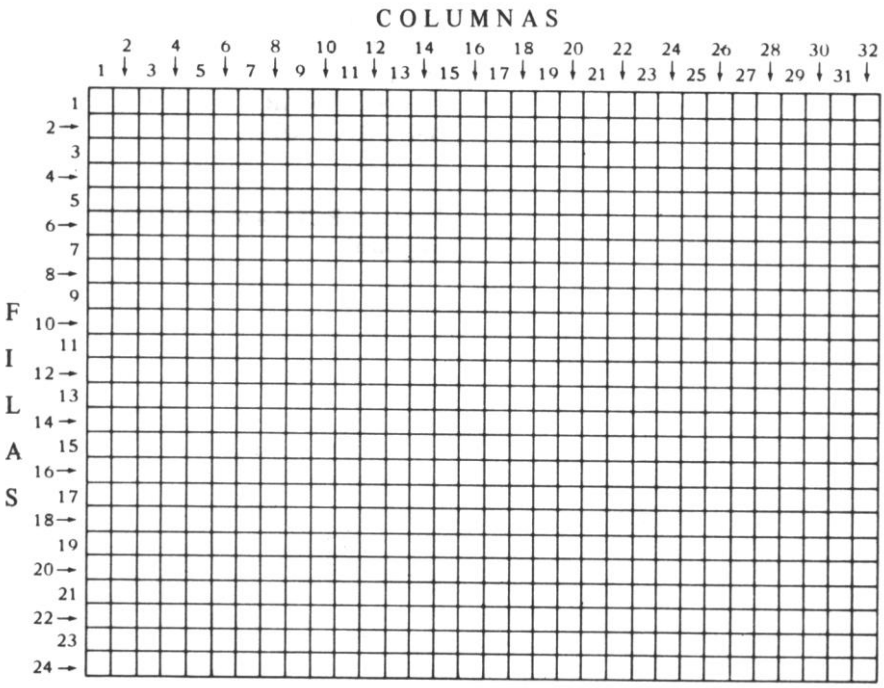
CALL VCHAR(fila, columna, código-caracter [repetir])

Descripción:

El subprograma VCHAR coloca un caracter en cualquier lugar de la pantalla y opcionalmente lo repite verticalmente. El caracter que tiene el código ASCII igual a código-caracter, se ubica en la posición indicada por fila y columna y se repite tantas veces como indique repeticiones.

Un valor 1 para fila indica la parte superior de la pantalla. Un valor 24 es la parte inferior de la pantalla. Un valor 1 para columna indica el lado izquierdo de la pantalla. Un valor 32, el lado derecho.

La pantalla puede pensarse como una grilla tal como se muestra en la página siguiente :



Ejemplos:

CALL VCHAR (12, 16,33) coloca el caracter 33 (un signo de admiración) en la fila 12 columna 16.

> 100 CALL VCHAR (12, 16, 33)

CALL VCHAR(1, 1, ASC("!"), 768) coloca un signo de admiración en la columna 1 fila 1 y lo repite 768 veces, lo que llena totalmente la pantalla.

> 100 CALL VCHAR (1, 1, ASC ("!"), 768)

CALL VCHAR (R, C, K, T) coloca el caracter con el código ASCII K en la fila R columna C y lo repite T veces.

> CALL VCHAR (R, C, K, T)

VERSION Subprograma

Formato:

CALL VERSION(variable-numérica)

Descripción:

El subprograma VERSION devuelve el valor que indica la versión de BASIC que se está usando. TI BASIC Extendido devuelve un valor igual a 100.

Ejemplo:

CALL VERSION (V) define V igual a 100 > 100 CALL VERSION (V)

Apéndices

Los siguientes apéndices contienen información útil acerca del TI BASIC Extendido

APENDICE A: - Lista de programas ilustrativos.
APENDICE B: - Lista de comandos, instrucciones y funciones.
APENDICE C: - Códigos ASCII.
APENDICE D - Frecuencias de las tonalidades musicales.
APENDICE E: - Conjuntos de caracteres.
APENDICE F: - Tabla de conversión para identificadores de modelo.
APENDICE G : - Códigos de color.
APENDICEH: - Combinaciones de colores.
APENDICE I: - División del teclado.
APENDICE J: - Códigos de caracteres para la división del teclado.
APENDICE K: - Funciones matemáticas.
APENDICE L: - Lista de palabras habladas.
APENDICE M: - Agregado de sufijos a las palabras habladas.
APENDICE N: - Mensajes de error.

Listado de Programas Ilustrativos

APENDICE **A**

ELEMENTO ILUSTRADO	LINEAS	DESCRIPCION	PAGINA
	44	Juego Número Secreto	27
ACCEPT	16	Ingreso de 20 nombres	48
CALL	8	Subrutina del usuario y CLEAR	55
CHAR	12	1. Figura en movimiento	58
	7	2. Recuperación de caracteres	58
CHR$	4	Lista de códigos ASCII	60
CLEAR	3	(Ejemplo simple)	61
	3	(Ejemplo simple)	61
COINC	10	(Ejemplo simple)	65
COS	6	(Ejemplo simple)	69
DATA	14	(Ejemplo simple)	71
DELETE	2	(Ejemplo simple)	74
DISPLAY	18	Dibujo en la pantalla	78
ERR	5	(Ejemplo simple)	84
FOR-TO-STEP	11	Dibujo	87
GOSUB	24	Probabilidad	90
GOTO	8	Suma de 1 a 100	91
IF-THEN-ELSE	17	Números de secuencia	96
IMAGE	12	(Ejemplo simple)	99
	2	(Ejemplo simple)	100
INPUT	17	Escribe una carta	103
INPUT (con archivos)	12	(Ejemplo simple)	106
JOYST	5	Mueve un sprite	108
KEY	12	Mueve un sprite	109
LINPUT	6	(Ejemplo simple)	113
LOCATE	6	(Ejemplo simple)	116
LOG	8	Log. de cualquier base	117

LISTADO DE PROGRAMAS ILUSTRATIVOS

ELEMENTO ILUSTRADO	LINEAS	DESCRIPCION	PAGINA
MAGNIFY	17	(Ejemplo simple)	120
MERGE	13	Mueve un sprite	122
MOTION	8	Mueve un sprite	125
NEYT	6	(Ejemplo simple)	127
NUMBER	4	(Ejemplo simple)	128
ON BREAK	11	(Ejemplo simple)	130
ON ERROR	15	(Ejemplo simple)	132
ON ... GOSUB	20	Elección con menú	134
ON ... GOTO	19	Elección con menú	136
ON WARNING	8	(Ejemplo simple)	137
PATTERN	18	Rueda que gira	142
POS	8	Ruptura de una frase	145
PRINT	7	(Ejemplo simple)	149
RANDOMIZE	5	(Ejemplo simple)	151
REC	12	(Ejemplo simple)	153
RETURN (con GOSUB)	18	(Ejemplo simple)	157
RETURN (con On Error)	13	Manejo de error	158
RUN	12	Elección con menú	162
SAY	4	(Ejemplo simple)	164
SIN	6	(Ejemplo simple)	168
SOUND	4	Toca las primeras 13 notas	171
SPGET	4	(Ejemplo simple)	172
SPRITE	8	(Ejemplo simple)	174
	8	(Creación de estrellas)	175
	55	Sprites que caminan	175
STOP	7	Suma de 1 a 100	178
SUB	21	Elección con menú	183
TAN	6	(Ejemplo simple)	186

Comandos, Instrucciones y Funciones

APENDICE **B**

La siguiente es una lista de comandos, instrucciones y funciones de TI BASIC Extendido. Primero se listan los comandos; si un comando también puede ser usado como instrucción, aparecerá la letra S a la derecha del comando. Los comandos que pueden abreviarse tienen subrayadas las abreviaturas aceptadas. La siguiente es una lista de Instrucciones de TI BASIC Extendido; aquéllas que también pueden ser usadas como comando tienen una "C" a la derecha. Finalmente hay una lista de funciones del TI BASIC Extendido.

Comandos del TI BASIC Extendido

BREAKS	MERGE	SAVE
BYE	NUMBER	SIZE
CONTINUE	OLD	TRACE S
DELETE S	RESEQUENCE	UNBREAK S
LIST	RUN S	UNTRACE S

Instrucciones de TI BASIC Extendido

ACCEPT C	CALL HCHAR C	OPTION BASE
CALL	IF THEN ELSE	CALL PATTERN C
CALL CHAR C	IMAGE	CALL PEEK C
CALL CHARPAT C	CALL INIT C	CALL POSITION C
CALL CHARSET C	INPUT	PRINT C
CALL CLEAR C	INPUT REC	PRINT USING C
CLOSE C	CALL JOYST C	RANDOMIZE C
CALL COINC C	CALL KEY C	READ C
CALL COLOR C	[LET] C	REM C
DATA	CALL LINK C	RESTORE C
DEF	LINPUT	RETURN
CALL DELSPRITE C	CALL LOAD C	CALL SAY C
DIM C	CALL LOCATE C	CALL SCREEN C
DISPLAY C	CALL MAGNIFY C	CALL SOUND C
DISPLAY USING C	CALL MOTION C	CALL SPGET C
CALL DISTANCE C	NEXT C	CALL SPRITE C
END	ON BREAK	STOP C
CALL ERR C	ON ERROR	SUB
FOR C	ON GOSUB	SUBEND
CALL GCHAR C	ON GOTO	SUBEXIT
GOSUB	ON WARNING	CALL VCHAR C
GOTO	OPEN C	CALL VERSION C

Funciones del TI BASIC Extendido

ABS	LEN	SEG$
ASC	LOG	SGN
ATN	MAX	SIN
CHR$	MIN	SQR
COS	PI	STR$
EOF	POS	TAB
EXP	REC	TAN
INT	RND	VAL
	RPT$	

Códigos ASCII

APENDICE

C

Los siguientes caracteres predefinidos pueden imprimirse y mostrarse en la pantalla.

CODIGO ASCII	CARÁCTER		CODIGO ASCII	CARACTER	
30		(cursor)	63	?	(signo de interrogación)
31		(caracter de borde)	64	@	
32		espacio	65	A	
33	!	(signo de exlcamación)	66	B	
34	"	(comillas)	67	C	
35	#	(numeral)	68	D	
36	$	(signo dolar)	69	E	
37	%	(por ciento)	70	F	
38	&	(ampersand)	71	G	
39	‘	(apóstrofe)	72	H	
40	(	(paréntesis de apertura)	73	I	
41	)	(paréntesis de cierre)	74	J	
42	*	(asterisco)	75	K	
43	+	(más)	76	L	
44	,	(coma)	77	M	
45	-	(menos)	78	N	
46	.	(punto)	79	O	
47	/	(barra)	80	P	
48	0		81	Q	
49	1		82	R	
50	2		83	S	
51	3		84	T	
52	4		85	U	
53	5		86	V	
54	6		87	W	
55	7		88	Y	
56	8		89	Y	
57	9		90	Z	
58	:	(dos puntos)	91	[	(corchete de apertura)
59	;	(punto y coma)	92	\	(barra invertida)
60	<	(menor que)	93	]	(corchete de cierre)
61	=	(igual)	94	^	(exponenciación)
62	>	(mayor que)	95	_	(subrayado)

La siguientes teclas también pueden ser detectadas por CALL KEY.

1	**FCTN 7** (AID)	2	**FCTN 4** (CLEAR)
3	**FCTN 1** (DELete)	4	**FCTN 2** (INSert)
5	**FCTN =** (QUIT)	6	**FCTN 8** (REDO)
7	**FCTN 3** (ERASE)	8	**FCTN S** (LEFT izquierda)
9	**FCTN D** (RIGHT derecha)	10	**FCTN X** (DOWN abajo)

Tonos y Frecuencias Musicales

APENDICE

D

La siguiente tabla da las frecuencias (redondeadas a enteros) de la escala temperada (medio paso entre notas). Si bien esta tabla no representa la totalidad de sonidos que la computadora puede producir, podrá ser de utilidad para programar música.

FRECUENCIA	NOTA	FRECUENCIA	NOTA
110	A	440	A (por debajo medio C)
117	$A^{\#}$, B^{b}	466	$A^{\#}$, B^{b}
123	B	494	B
131	C (low C)	523	C (alto C)
139	$C^{\#}$, D^{b}	554	$C^{\#}$, D^{b}
147	D	587	D
156	$D^{\#}$, E^{b}	622	$D^{\#}$, E^{b}
165	E	659	E
175	F	698	F
185	$F^{\#}$, G^{b}	740	$F^{\#}$, G^{b}
196	G	784	G
208	$G^{\#}$, A^{b}	831	$G^{\#}$, A^{b}
220	A (por debajo medio C)	880	A (por sobre alto C)
220	A (por sobre medio C)	880	A (por sobre alto C)
233	$A^{\#}$, B^{b}	932	$A^{\#}$, B^{b}
247	B	988	B
262	C (middle C)	1047	C
277	$C^{\#}$, D^{b}	1109	$C^{\#}$, D^{b}
294	D	1175	D
311	$D^{\#}$, E^{b}	1245	$D^{\#}$, E^{b}
330	E	1319	E
349	F	1397	F
370	$F^{\#}$, G^{b}	1480	$F^{\#}$, G^{b}
392	G	1568	G
415	$G^{\#}$, A^{b}	1661	$G^{\#}$, A^{b}
440	A (por sobre medio C)	1760	A

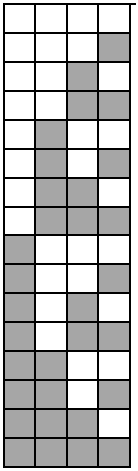
Grupos de Caracteres

APENDICE **E**

SET	ASCII CODES	SET	ASCII CODES
0	30-31		
1	32-39	8	88-95
2	40-47	9	96-103
3	48-55	10	104-111
4	56-63	11	112-119
5	64-71	12	120-127
6	72-79	13	128-135
7	80-87	14	136-143

Tabla de Convención Identificador Secuencial

APENDICE **F**

BLOQUES	CODIGO BINARIO (0: apagado)(1: encendido)	CODIGO HEXADECIMAL
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

Código de Colores

APENDICE

G

COLOR	CODIGO	COLOR	CODIGO
Transparente	1	Rojo mediano	9
Negro	2	Rojo claro	10
Verde mediano	3	Amarillo oscuro	11
Verde claro	4	Amarillo claro	12
Azul oscuro	5	Verde oscuro	13
Azul claro	6	Magenta	14
Rojo oscuro	7	Gris	15
Ciánico	8	Blanco	16

Combinaciones de Color

APENDICE

H

Las siguientes combinaciones en los colores producen una más clara resolución en los caracteres.

LA MEJOR

2, 8	Negro sobre ciánico	2, 13	Negro sobre verde oscuro
2, 7	Negro sobre rojo oscuro	2, 15	Negro sobre gris
2, 6	Negro sobre azul claro	2, 14	Negro sobre magenta
2, 3	Negro sobre verde mediano	2, 9	Negro sobre rojo mediano
5, 8	Azul oscuro sobre ciánico	5,15	Azul oscuro sobre gris
5, 6	Azul oscuro sobre azul claro	5,4	Azul oscuro sobre verde claro
5, 14	Azul oscuro sobre magenta	5, 16	Azul oscuro sobre blanco
13, 8	Verde oscuro sobre ciánico	13, 11	Verde oscuro sobre amarillo oscuro
13,15	Verde oscuro sobre gris	13, 4	Verde oscuro sobre verde claro
13,12	Verde oscuro sobre amarillo claro	13, 3	Verde oscuro sobre verde medio
7, 15	Rojo oscuro sobre gris	7, 10	Rojo oscuro sobre rojo claro
7, 12	Rojo oscuro sobre amarillo claro	14,2	Magenta sobre rojo claro
3, 12	Verde medio sobre amarillo claro	3, 15	Verde mediano sobre blanco

SEGUNDA MEJOR

2, 5	Negro sobre azul oscuro	2, 11	Negro sobre amarillo oscuro
2, 4	Negro sobre verde claro	2, 10	Negro sobre rojo claro
2, 12	Negro sobre amarillo claro	13, 10	Verde oscuro sobre rojo claro
13, 16	Verde oscuro sobre blanco	7, 16	Rojo oscuro sobre blanco
6, 15	Azul claro sobre gris	6, 4	Azul claro sobre verde claro
6, 16	Azul claro sobre blanco	4, 16	Verde claro sobre blanco

TERCERA MEJOR

2, 16	Negro sobre blanco	5, 12	Azul oscuro sobre amarillo claro
7, 9	Rojo oscuro sobre verde medio	4,12	Verde claro sobre amarillo claro
14, 15	Magenta sobre gris	14, 16	Magenta sobre blanco
3, 11	Verde mediano sobre amarillo oscuro	3, 15	Verde mediano sobre gris
9, 15	Rojo mediano sobre gris	9, 10	Rojo mediano sobre rojo claro
9, 12	Rojo mediano sobre amarillo claro	9,16	Rojo mediano sobre blanco
16, 7	Blanco sobre rojo oscuro		

CUARTA MEJOR

8, 2	Ciánico sobre negro	8, 16	Ciánico sobre blanco
7, 2	Rojo oscuro sobre negro	7,4	Rojo oscuro sobre verde claro
15, 16	Gris sobre blanco	5,2	Azul claro sobre negro
4, 2	Verde claro sobre negro	10, 2	Rojo claro sobre negro
10, 16	Rojo claro sobre blanco	14, 12	Magenta sobre amarillo claro
9; 4	Rojo mediano sobre verde claro	16, 6	Blanco sobre azul claro

Descripción del Teclado

APENDICE

I

Figura 1: Teclado dividido
Códigos expuestos = 0-19.

Código de Caracteres Para Teclados Divididos

APENDICE

J

CODIGO	TECLAS*	CODIGO	TECLAS*
0	X, M	10	5, 0
1	A, H	11	T, P
2	S, J	12	F, L
3	D, K	13	V, . (punto)
4	W, U	14	C, , (coma)
5	E, 1	15	Z, N
6	R, 0	16	B, / (diagonal)
7	2, 7	17	G, ;(punto y coma)
8	3, 8	18	Q, Y
9	4, 9	19	1, 6

Note que la primera tecla de la lista está en el lado izquierdo del teclado y la segunda de la lista está en el costado derecho.

Funciones Matemáticas

APENDICE
K

Las siguientes funciones matemáticas pueden definirse con DEF Como se muestra a continuación.

Función	Instrucción en TI BASIC Extendido
Secante	DEF SEC(X)=1/COS(X)
Cosecante	DEF CSC(X)=1/SIN(X)
Cotangente	DEF COT(X)=1/TAN(X)
Inversa del seno	DEF ARCSIN(X)=ATN(X/SQR(1-X*X))
Inversa del coseno	DEF ARCCOS(X)=-ATN(X/SQR(1-X*X))+PI/2
Inversa de la secante	DEF ARCSEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*PI/2
Inversa de la cosecante	DEF ARCCSC(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*PI/2
Inversa de la cotangente	DEF ARCCOT(X)=PI/2-ATN(X) ó =PI/2+ATN(-X)
Seno hiperbólico	DEF SINH(X)=(EXP(X)-EXP(-X))/2
Coseno hiperbólico	DEF COSH(X)=(EXP(X)+EXP(-X))/2
Tangente hiperbólica	DEF TANH(X)=-2*EXP(-X)/(EXP(X)+EXP(-X))+1
Secante hiperbólica	DEF SECH=2/EXP(X)+EXP(-X))
Cosecante hiperbólica	DEF CSCH=2/EXP(X)-EXP(-X))
Cotangente hiperbólica	DEF COTH(X)=2*EXP(-X)/(EXP(X)-EXP(-X))+1
Inversa del seno hiperbólico	DEF ARCSINH(X)=LOG(X+SQR(X*X+1))
Inversa del coseno hiperbólico	DEF ARCCOSH(X)=LOG(X+SQR(X*X-1))
Inversa de la tangente hiperbólica	DEF ARCTANH(X)=LOG((l+X)/(1-X))/2
Inversa de la secante hiperbólica	DEFARCSECH(X)=LOG((l+SQR(1-X*X))/X)
Inversa de la cosecante hiperbólica	DEF ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Inversa de la cotangente hiperbólica	DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2

Lista de Palabras Habladas

APENDICE **L**

La siguiente es una lista de todos los números, letras, palabras y frases a las que se puede acceder con CALL SAY y CALL SPGET. Vea el Apéndice M sobre instrucciones para agregar sufijos a cualquiera de los elementos de esta lista.

- (NEGATIVE)
+ (POSITIVE)
. (POINT)
0
1
2
3
4
5
6
7
8
9
A (a)
A1 (a)
ABOUT
AFTER
AGAIN
ALL
AM
AN
AND
ANSWER
ANY
ARE
AS
ASSUME
AT
B
BACK
BASE
BE
BETWEEN
BLACK
BLUE
BOTH
BOTTOM
BUT
BUY
BY
BYE
C
CAN
CASSETTE
CENTER
CHECK
CHOICE
CLEAR
COLOR
COME
COMES
COMMA
COMMAND
COMPLETE
COMPLETED
COMPUTER
CONNECTED
CONSOLE
CORRECT
COURSE
CYAN
D
DATA
DECIDE
DEVICE
DID
DIFFERENT
DISKETTE
DO
DOES
DOING
DONE
DOUBLE
DOWN
DRAW
DRAWING
E
EACH
EIGHT
EIGHTY
ELEVEN
ELSE
END
ENDS
ENTER
ERROR
EXACTLY
EYE
F
FIFTEEN
FIFTY
FIGURE
FIND
FINE
FINISH
FINISHED
FIRST
FIT
FIVE
FOR
FORTY
FOUR
FOURTEEN
FOURTH
FROM
FRONT
G
GAMES
GET
GETTING
GIVE
GIVES
GO
GOES
GOING
GOOD
GOOD WORK
GOODBYE
GOT
GRAY
GREEN
GUESS
H
HAD
HAND
HANDHELD UNIT
HAS
HAVE
HEAD
HEAR
HELLO
HELP

HERE
HIGHER
HIT
HOME
HOW
HUNDRED
HURRY
I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT
J
JOYSTICK
JUST
K
KEY
KEYBOARD
KNOW
L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER
M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST
N
NAME
NEAR
NEED
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER
O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER
P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
POSITIVE
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING
Q
R
RANDOMLY
READ (read)
READ 1 (red)
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND
S
SAID
SAVE
SAY
SAYS
SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY

LISTA DE PALABRAS HABLADAS

SPACE
SPACES
SPELL
SQUARE
START
STEP
STOP
SUM
SUPPOSED
SUPPOSED TO
SURE
T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS INCORRECT
THAT IS RIGHT
THE (the)
THE1 (tha)
THEIR
THEN
THERE
THESE
THEY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRTY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE
U
UHOH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE
V
VARY
VERY
W
WAIT
WANT
WANTS
WAY
WE
WEIGH
WEIGHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WON
WORD
WORDS
WORK
WORKING
WRITE
X
Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR
Z
ZERO

Agregado de Sufijos a las Palabras Habladas

APENDICE
M

Este Apéndice describe cómo agregar ING, S y ED a cualquier palabra disponible del vocabulario residente del Sintetizador de Palabras en Estado Sólido.

Primero se lee el código de una palabra usando SPGET. Este código está formado por una cantidad de caracteres, uno de los cuales indica la longitud de la palabra. Luego usando los subprogramas que se listan aquí se pueden agregar códigos adicionales que den el sonido de un sufijo.

A menudo las palabras traen datos adicionales que hacen que suenen más naturales, pero impiden el agregado de sufijos. Para poder hacerlo se removerán estos datos adicionales.

El siguiente programa muestra cómo ingresar una palabra y, usando distintos valores de truncamiento, hacer que el sufijo suene como una parte natural de la palabra. Los subprogramas DEF1NG (líneas 1000 a 1130), DEFS1 (líneas 2000 a 2100), DEFS2 (líneas 3000 a 3090), DEFS3 (líneas 4000 a 4120), DEFED1 (líneas 5000 a 5070), DEFED2 (líneas 6000 a 6110), DEFED3 (líneas 7000 a 7130) y MENU (1í- neas 10000 a 10120) deben ingresarse separadamente y almacenarse con la opción merge. (El subprograma MENU, es el mismo del ejemplo usado con SUB). Puede cambiar los números de línea si lo desea. Cada uno de estos subprogramas, excepto MENU, define un sufijo.

DEF1NG define el sonido 1NG.DEFS1 define el sonido S cuando se produce al final (como "cats"). DEFS2 define el sonido S cuando se produce al final (como "cats"). DEFS3 define el sonido S cuando se produce al final (como "wishes"). DEFED1 define el sonido ED cuando se produce al final (como "passed"). DEFED 2 define el sonido ED cuando se produce al final como "caused"). DEFED 3 define el sonido ED cuando se produce al final (como "heated").

Cuando ejecute el programa, ingrese un cero como valor de truncamiento, para dejar la secuencia de truncamiento.

```
100 REM ********************
110 REM REQUIRES MERGE OF:
120 REM MENU (LINES 10000 THROUGH 10120)
130 REM DEFING (LINES 1000 THROUGH 1130)
140 REM DEFS1 (LINES 2000 THROUGH 2100)
150 REM DEFS2 (LINES 3000 THROUGH 3090)
160 REM DEFS3 (LINES 4000 THROUGH 4120)
170 REM DEFED1 (LINES 5000 THROUGH 5070)
180 REM DEFED2 (LINES 6000 THROUGH 6110)
190 REM DEFED3 (LINES 7000 THROUGH 7130)
200 REM ********************
210 CALL CLEAR
220 PRINT "THIS PROGRAM IS USED TO"
```

```
230 PRINT "FIND THE PROPER TRUNCATION"
240 PRINT "VALUE FOR ADDING SUFFIXES"
250 PRINT "TO SPEECH WORDS.": :
260 FOR DELAY=1 TO 300::NEXT DELAY
270 PRINT "CHOOSE WHICH SUFFIX YOU"
280 PRINT "WISH TO ADD.": :
290 FOR DELAY=1 TO 200::NEXT DELAY
300 CALL MENU(8,CHOICE)
310 DATA 'ING','S' AS IN CATS,'S' AS IN CADS,'S' AS IN WISHES,
'ED' AS IN PASSED,'ED' AS IN CAUSED,'ED' AS IN HEATED,END
320 IF CHOICE=O OR CHOICE=8 THEN STOP
330 INPUT "WHAT IS THE WORD? ":WORD$
340 ON CHOICE GOTO 350,370,390,410,430,450,470
350 CALL DEFING(D$)
360 GOTO 480
370 CALL DEFS1(D$)!CATS
380 GOTO 480
390 CALL DEFS2(D$)!CADS
400 GOTO 480
410 CALL DEFS3(D$)!WISHES
420 GOTO 480
430 CALL DEFED1(D$)!PASSED
440 GOTO 480
450 CALL DEFED2(D$)!CAUSED
460 G0TO 480
470 CALL DEFED3(D$)!HEATED
480 REM TRY VALUES
490 CALL CLEAR
500 INPUT "TRUNCATE HOW MANY BYTES? ":L
510 IF L=0 THEN 300
520 CALL SPGET(WORD$,B$)
530 L=LEN(B$)-L-3
540 C$=SEG$(B$,1,2)&CHR$(L)&SEG$(B$,4,L)
550 CALL SAY(,C$&D$)
560 GOT0 500
```

Los datos se han dado en cortas instrucciones DATA para facilitar su entrada. Se pueden agrupar para hacer el programa mas corto.

```
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$= “”
1120 FOR I=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT I
1130 SUBEND

2000 SUB DEFS1(A$)!CATS
2010 DATA 96,0,26
2020 DATA 14,56,130,204,0
2030 DATA 223,177,26,224,103
2040 DATA 85,3,252,106,106
2050 DATA 128,95,44,4,240
2060 DATA 35,11,2,126,16,121
2070 RESTORE 2010
2080 A$=“"
2090 FOR I=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT I
2100 SUBEND

3000 SUB DEFS2(A$)!CADS
3010 DATA 96,0,17
3020 DATA 161,253,158,217
3030 DATA 168,213,198,86,0
3040 DATA 223,153,75,128,0
3050 DATA 95,139,62
3060 RESTORE 3010
3070 A$=""
3080 FOR I=1 TO 20::READ A::A$=A$&CHR$(A)::NEXT I
3090 SUBEND
```

```
4000 SUB DEFS3(A$)!WISHES
4010 DATA 96,0,34
4020 DATA 173,233,33,84,12
4030 DATA 242,205,166,55,173
4040 DATA 93,222,68,197,188
4050 DATA 134,238,123,102
4060 DATA 163,86,27,59,1,124
4070 DATA 103,46,1,2,124,45
4080 DATA 138,129,7
4090 RESTORE 4010
4100 A$=""
4110 FOR I=1 TO 37::READ A::A$=A$&CHR$(A)::NEXT I
4120 SUBEND

5000 SUB DEFED1(A$)!PASSED
5010 DATA 96,0,10
5020 DATA 0,224,128,37
5030 DATA 204,37,240,0,0,0
5040 RESTORE 5010
5050 A$=-"
5060 FOR I=1 TO 13::READ A::A$=A$&CHR$(A)::NEXT I
5070 SUBEND

6000 SUB DEFED2(A$)!CAUSED
6010 DATA 96,0,26
6020 DATA 172,163,214,59,35
6030 DATA 109,170,174,68,21
6040 DATA 22,201,220,250,24
6050 DATA 69,148,162,166,234
6060 DATA 75,84,97,145,204
6070 DATA 15
6080 RESTORE 6010
6090 A$=--
6100 FOR I=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT I
6110 SUBEND
```

```
7000 SUB DEFED3(A$)!HEATED
7010 DATA 96,0,36
7020 DATA 173,233,33,84,12
7030 DATA 242,205,166,183
7040 DATA 172,163,214,59,35
7050 DATA 109,170,174,68,21
7060 DATA 22,201,92,250,24
7070 DATA 69,148,162,38,235
7080 DATA 75,84,97,145,204
7090 DATA 178,127
7100 RESTORE 7010
7110 A$=-'
7120 FOR I=1 TO 39::READ A::A$=A$&CHR$(A)::NEXT I
7130 SUBEND

10000 SUB MENU(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT>22 THEN PRINT "TOO MANY ITEMS" :: CHOICE=0 :: SUBEXIT
10030 RESTORE
10040 FOR I=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEG$(TEMP$,1,25)
10070 DISPLAY AT(I,1):I;TEMP$
10080 NEXT I
10090 DISPLAY AT(I+1,1): "YOUR CHOICE: 1"
10100 ACCEPT AT(I+1,14)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE<1 OR CHOICE>COUNT THEN 10100
10120 SUBEND
```

AGREGADO DE SUFIJOS A LAS PALABRAS HABLADAS

M

Se pueden usar los subprogramas en cualquier programa una vez que se ha determinado el número de bytes a truncar. El siguiente programa usa el subprograma DEF1NG en las líneas 1000 a 1130 para que la computadora diga DRAWING usando DRAW mas el sufijo 1NG. Note que se encontró que se debían truncar 41 caracteres de DRAW para poder producir un sonido más natural para DRAW1NG. El subprograma DEFING en las líneas 1000 a 1130 es el programa que se almacenó con la opción merge.

```
100 CALL DEFING(ING$)
110 CALL SPGET("DRAW",DRAW$)
120 L=LEN(DRAW$)-3-41! 3 BYTES OF SPEECH OVERHEAD, 41 BYTES TRUNCATED
130 DRAW$=SEG$(DRAW$,1,2)&CHR$(L)&SEG$(DRAW$,4,L)
140 CALL SAY("WE ARE",DRAW$&ING$,"Al SCREEN")
150 GOTO 140
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55: :READ A: :A$=A$&CHR$(A): :NEXT I
1130 SUBEND
```

(Presione **FCTN 4** para detener el programa)

Errores

APENDICE

N

La siguiente es una lista de los errores producidos por TI BAS1C Extendido. La primera lista tiene los mensajes ordenados alfabéticamente, y la segunda por número de error. Este número de error es el que devuelve CALL ERR. Si ocurre un error durante la ejecución de un programa, el mensaje a menudo está seguido por IN número-de-línea.

ORDENADO POR MENSAJE

Mensaje — Descripción de los posibles errores

74 **BAD ARGUMENT**

- Se ha dado un valor incorrecto en ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQR, TAN o VAL.
- Se especificó un elemento de un arreglo en una instrucción SUB.
- Está mal el primer parámetro de un LINK o hay demasiados parámetros.

61 **BAD LINE NUMBER**

- Número de línea menor que 1 o mayor que 32767.
- Se omitió el número de línea.
- El número de línea producido por RES está fuera del rango 1-32767.

57 **BAD SUBSCRIPT**

- Se está usando un subíndice muy grande o muy pequeño en un arreglo.
- Subíndice incorrecto en DIM.

79 **BAD VALUE**

- Se ha dado un valor incorrecto en AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, 1NPUT, MOTION, NOT, OR, POS, PR1NT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, o XOR.
- El subíndice de un arreglo es mayor que 32767.
- Número de archivo mayor que 255 o menor que cero.
- Se ha especificado para SOUND más de tres tonos y un ruido.
- Un valor que se ha pasado a un subprograma no es aceptable. Por ejemplo, una velocidad de sprite menor que -128 o un valor de caracter mayor que 143.
- Un número de línea especificado para ON…GOTO o ON…GOSUB es mayor que el número de líneas del programa.
- Se ha dado una posición incorrecta luego de la cláusula AT en un ACCEPT o D1SPLAY.

67 **CANT CONTINUE**

- Se ha editado el programa luego de un "breakpoint".
- El programa no se detuvo por un "breakpoint".

69 **COMMAND ILLEGAL IN PROGRAM**

- Se intentó usar BYE, CON, LIST, MERGE, NEW, NUM, OLD, RES o SAVE en un programa.

84 **DATA ERROR**

- Faltan datos para un READ o RESTORE, o bien se ha especificado una variable alfanumérica donde se esperaba una variable numérica. El número de lima después de un RESTORE es mayor que el número de línea mas grande del programa.
- Error en un archivo objeto en un LOAD.

109 **FILE ERROR**

- Tipo de datos erróneo en una instrucción READ.
- Se intentó usar CLOSE, EOF, INPUT, OPEN, PRINT, PRINT USING, REC, o RESTORE con un archivo que no existe o no tiene los atributos apropiados.
- No hay suficiente memoria para usar un archivo.

44 **FOR NEXT NESTING**

- Las instrucciones FOR y NEXT de algún ciclo no se corresponden correctamente.
- Falta una instrucción NEXT.

130 **I/O ERROR**

- Se detectó un error al tratar de ejecutar un CLOSE, DELE- TE, LOAD, MERGE, OLD, OPEN, RUN, o SAVE.
- No hay suficiente memoria para listar un programa.

16 **ILLEGAL AFTER SUBPROGRAM**

- No puede haber nada que no sea END, REM o SUB después de un SUBEND

36 **IMAGE ERROR**

- Se detectó un error en la ejecución de un DISPLAY USING, IMAGE o PRINT USING.
- En un formato hay mas de 10 (formato E) ó 14 (formato numérico) dígitos significativos.
- El formato de un IMAGE es mayor de 254 caracteres.

28 **IMPROPERLY USED NAME**

- Se usó un nombre de variable ilegal en un CALL, DEF, o DIM.
- Se está usando una palabra reservada de TI BASIC Extendido en una instrucción LET.
- Se está usando una variable con subíndice o una variable alfanumérica en un FOR.
- Se está usando un arreglo con un número incorrecto de dimensiones.
- Se está usando un nombre de variable distinto del que se asignó originalmente. Una variable puede ser solamente un arreglo, una variable numérica o alfanumérica o una función definida por el usuario.
- Se ha colocado el nombre de una función definida por el usuario a la izquierda del signo igual en una instrucción de asignación.
- Se ha dimensionado un arreglo dos veces.
- Se usó el mismo nombre de variable dos veces en la lista de parámetros de una instrucción SUB.

81 **INCORRECT ARGUMENT LIST**

- Los argumentos de CALL y SUB no coinciden.

83 **INPUT ERROR**

- Se detectó un error en un INPUT.

60 **LINE NOT FOUND**

- Se encontró un número de línea incorrecto en un BREAK, GOSUB, GOTO, ON ERROR, RUN o UNBREAK o luego de un THEN o ELSE.
- No se encontró la línea que se solicita para ser editada.

62 **LINE TOO LONG**

- Se intenta ingresar una linea demasiado larga en un programa.

39 **MEMORY FULL**

- El programa es demasiado grande para poder ejecutar uno de los siguientes: DEF, DELETE, DIM, GOSUB, LET, LOAD, ON . . . GOSUB, OPEN o SUB.
- El programa es muy grande para agregar una nueva línea, insertar una línea o evaluar una expresión.

49 **MISSING SUBEND**

- Falta SUBEND en un subprograma.

47 **MUST BE IN SUBPROGRAM**

- Falta SUBEND o SUBEXIT en un subprograma.

19 **NAME TOO LONG**

- Mas de 15 caracteres en un nombre de variable o subprograma.

43 **NEXT WITHOUT FOR**

- Falta la instrucción FOR, NEXT está antes que FOR, hay un anidamiento incorrecto FOR-NEXT, o se intenta bifurcar dentro de un ciclo FOR-NEXT.

78 **NO PROGRAM PRESENT**

- Se pidió un listado de un programa que no está presente, o bien se pidió RESEQUENCE, RESTORE, RUN o SAVE.

10 **NUMERIC OVERFLOW**

- Hay un número muy grande o muy pequeño como resultado de una operación *, + , / , o en un ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN, o VAL.
- En una instrucción PEEK o LOAD hay un número fuera del rango -32768 a 32767.

70 **ONLY LEGAL IN A PROGRAM**

- Se usó como comando una de las siguientes instrucciones: DEF, GOSUB, GOTO, IF, IMAGE, INPUT, ON BREAK, ON ERROR, ON . . . GOSUB, ON. . . GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND, o SUBEXIT.

25 **OPTION BASE ERROR**

- Se intentó ejecutar OPTION BASE mas de una vez, o con valores distintos de cero o uno.

97 **PROTECTION VIOLATION**

- Se intentó almacenar, listar o editar un programa protegido.

48 **RECURSIVE SUBPROGRAM CALL**

- Un subprograma intentó llamarse a si mismo directa o indirectamente.

51 **RETURN WITHOUT GOSUB**

- Hay un RETURN sin GOSUB o hay un error en la ejecución previa de una instrucción ON ERROR.

56 **SPEECH STRING TOO LONG**

- La cadena devuelta por SPGET tiene mas de 255 caracteres.

40 **STACK OVERFLOW**

- Hay un exceso de paréntesis.
- No hay suficiente memoria para evaluar una expresión o asignar un valor.

54 **STRING TRUNCATED**

- Una cadena creada por RPT$, o por concatenación ("&"), o por una función definida por el usuario, tiene más de 254 caracteres.

24 **STRING -NUMBER MISMATCH**

- En un subprograma o función provista por TI BASIC Extendido, se dio un valor numérico donde se esperaba uno alfanumérico o viceversa.
- Se intentó asignar un valor numérico a una variable alfanumérica o viceversa.
- Se intentó concatenar un número. (Con "&").
- Se intentó usar una variable alfanumérica como subíndice.

135 **SUBPROGRAM NOT FOUND**

- El subprograma que se está llamando no existe, o no está cargado el subprograma assembler LINK.

14 **SYNTAX ERROR**

- Se detectó uno de los siguientes errores en un comando, instrucción, función o subprograma de TI BASIC Extendido: falta o sobra una coma o un paréntesis, los parámetros están desordenados, falta una palabra clave o está mal escrita, o está fuera de lugar.
- DATA o IMAGE no son las primeras o las únicas instrucciones en una línea.
- Siguen datos luego de un ")" .
- Falta el signo # en un sprite.
- Falta dar ENTER, un símbolo !, o un separador de instrucciones (: :).
- Falta THEN después de un IF.
- Falta TO después de un FOR.
- No hay nada después de un CALL, SUB, FOR, THEN, ELSE.
- Hay dos E en una constante numérica.
- Hay una lista de parámetros equivocada en un subprograma provisto por TI BASIC Extendido.
- Se intentó entrar o salir de un subprograma con un GOTO, GOSUB, ON ERROR, etc.
- Se llamó a INIT sin tener conectada la Expansión de Memoria.
- Se llamó a LINK o LOAD sin haber llamado antes a LINK.
- Se está usando una constante donde se requiere una variable.
- Hay mas de 7 dimensiones en un arreglo.

17 **UNMATCHED QUOTES**

- Hay un número impar de comillas en una línea de entrada.

20 **UNRECOGNIZED CHARACTER**

- Un caracter no reconocido tal como ? o % no está entre comillas.
- Hay un campo incorrecto en un archivo objeto accedido por LOAD

ORDENADOS POR NUMERO DE ERROR

MENSAJE

10 **NUMERIC OVERFLOW**
14 **SYNTAX ERROR**
16 **ILLEGAL AFTER SUBPROGRAM**
17 **UNMATCHED QUOTES**
19 **NAME TOO LONG**
20 **UNRECOGNIZED CHARACTER**
24 **STRING-NUMBER MISMATCH**
25 **OPTION BASE ERROR**
28 **IMPROPERLY USED NAME**
36 **IMAGE ERROR**
39 **MEMORY FULL**
40 **STACK OVERFLOW**
43 **NEXT WITHOUT FOR**
44 **FOR-NEXT NESTING**
47 **MUST BE IN SUBPROGRAM**
48 **RECURSIVE SUBPROGRAM CALL**
49 **MISSING SUBEND**
51 **RETURN WITHOUT GOSUB**
54 **STRING TRUNCATED**
56 **SPEECH STRING TOO LONG**
57 **BAD SUBSCRIPT**
60 **LINE NOT FOUND**
61 **BAD LINE NUMBER**
62 **LINE TOO LONG**
67 **CAN'T CONTINUE**
69 **COMMAND ILLEGAL IN PROGRAM**
70 **ONLY LEGAL IN A PROGRAM**
74 **BAD ARGUMENT**
78 **NO PROGRAM PRESENT**
79 **BAD VALUE**
81 **INCORRECT ARGUMENT LIST**
83 **INPUT ERROR**
84 **DATA ERROR**
97 **PROTECTION VIOLATION**
109 **FILE ERROR**
130 **I/O ERROR**
135 **SUBPROGRAM NOT FOUND**

Indice

Las páginas que aparecen en itálicas muestran los lugares en que los elementos del lenguaje se usan en programas ilustrativos.

IMPORTANTE INFORMACION ACERCA DE TI BASIC EXTENDIDO

TI BASIC Extendido ha sido ampliado y modificado para poder ser usado por las computadoras TI-99/4 y TI-99/4A. Es importante remarcar algunas diferencias importantes de acuerdo con el modelo de computadora que poses. Se recomienda leer este apunte y marcar los cambios apropiados en su copia del Manual del Usuario de TI BASIC Extendido.

Si bien las computadoras TI-99/4 y TI-99/4A son parecidas, se puede reconocer fácilmente la TI-99/4A por su teclado stándard que permite escribir tanto con mayúsculas como con minúsculas. Si se presiona la tecla ALPHA LOCK, las teclas alfabéticas se bloquean para escribir siempre con mayúsculas. Para liberar el ALPHA LOCK se deberá presionar la tecla nuevamente.

Cuando tienen colocado el módulo de TI BASIC Extendido ambas computadoras tienen en común varias ampliaciones. Sin embargo, cada computadora tiene sus únicas características propias. Esas son las características que se discuten en los párrafos siguientes.

CARACTERISTICA DE AUTO REPETICION

Cuando se usa TI BASIC Extendido en cualquiera de las dos computadoras, si se sostiene una tecla durante más de un segundo, el símbolo se repetirá automáticamente hasta que se libere la tecla.

TECLAS DE FUNCIONES ESPECIALES

La computadora TI-99/4A tiene las mismas funciones especiales que la TI-99/4. Sin embargo, esas funciones en general, están asignadas a otras teclas en la computadora TI-99/4A. El siguiente diagrama compara las teclas de función en las dos computadoras.

TECLAS DE FUNCION

NOMBRE DE LA TECLA	TECLAS PARA TI-99/4A	TECLAS PARA TI-99/4A
AID	SHIFT A	FCTN 7
CLEAR	SHIFT C	FCTN 4
DELete	SHIFT F	FCTN 1
INSert	SHIFT G	FCTN 2
QUIT	SHIFT Q	FCTN =
REDO	SHIFT R	FCTN 8
ERASE	SHIFT T	FCTN 3
Flecha izq.	SHIFT S	FCTN S
Flecha der.	SHIFT D	FCTN D
Flecha abajo	SHIFT X	FCTN X
Flecha arriba	SHIFT E	FCTN E
PROC'D	SHIFT V	FCTN 6
BEGIN	SHIFT W	FCTN 5
BACK	SHIFT Z	FCTN 9
ENTER	ENTER	ENTER

Además de estas funciones, la computadora TI-99/4A tiene funciones representa-das como símbolos en la cara frontal de algunas teclas.

Estas funciones se pueden acceder presionando FCTN y la tecla apropiada al mismo tiempo.

TECLAS DE CONTROL

La computadora TI-99/4A tiene también caracteres de control que se usan en tele-comunicaciones. Para ingresar un caracter de control, presione la tecla CTRL junto con la letra, número o símbolo apropiado.

CONJUNTO DE CARACTERES AMPLIADOS - TI-99/4A

Tal como se explica en el manual de TI BASIC Extendido, los códigos 32 a 95 son caracteres ASCII standard predefinidos en la computadora TI-99/4. El cursor y los caracteres de borde, con códigos ASCII 30 y 31, están asignados al conjunto de caracteres es 0.

Los códigos de caracter no definidos (128 a 135 y 136 a 143) están asignados a los conjuntos 13 y 14 respectivamente.

Estos códigos y los caracteres correspondientes se muestran en el APENDICE C del manual. Los códigos de caracter para CALL KEY también están en el APENDICE

C. El APENDICE E del manual contiene los 15 conjuntos de códigos de caracteres que pueden usarse para graficas en color.

Debido a la inclusión de las mayúsculas en el conjunto de caracteres de la computadora TI-99/4A, los caracteres definidos tienen códigos de 32 a 127. El siguiente cuadro lista esos caracteres y sus códigos.

CODIGO ASCII	CARACTER		CODIGO ASCII	CARACTER	
30	■	(cursor)	55	7	
31		(caracter de borde)	56	8	
32		(espacio)	57	9	
33	¡	(signo de exclamación)	58	:	(2 puntos)
34	"	(comillas)	59	;	(punto y coma)
35	#	(numeral)	60	<	(menor que)
36	$	(dolar)	61	=	(igual)
37	%	(por ciento)	62	>	(mayor que)
38	&	(ampersand)	63	?	
39	'	(apóstrofe)	64	@	(at sign)
40	(	(paréntesis apertura)	65	A	
41	)	(paréntesis cierre)	66	B	
42	*	(asterisco)	67	C	
43	+	(más)	68	D	
44	,	(coma)	69	E	
45	-	(menos)	70	F	
46	.	(punto)	71	G	
47	/	(barra)	72	H	
48	0		73	I	
49	1		74	J	
50	2		75	K	
51	3		76	L	
52	4		77	M	
53	5		78	N	
54	6		79	O	

CODIGO ASCII	CARACTER		CODIGO ASCII	CARACTER	
80	P		104	h	
81	Q		105	i	
82	R		106	j	
83	S		107	k	
84	T		108	l	
85	U		109	m	
86	V		110	n	
87	W		111	o	
88	X		112	p	Aparecen en la
89	Y		113	q	pantalla como
90	Z		114	r	letras minúsculas
91	[	(corchete abierto)	115	s	
92	\	(barra invertida)	116	t	
93	]	(corchete cerrado)	117	u	
94	^	(exponencial)	118	v	
95	_	(subrayado)	119	w	
96	`	(grave)	120	x	
97	a		121	y	
98	b		122	z	
99	c	Aparecen en	123	{	(llave izq.)
100	d	la pantalla	124	|	(línea vert.)
101	e	como letras	125	}	(llave der.)
102	f	minúsculas	126	~	(tilde)
103	g		127	DEL	(aparece en la pantalla como un blanco)

SUBPROGRAMA CALL KEY

La información dada sobre el subprograma KEY en el Capítulo 4 del manual de TI BASIC Extendido es exacta para la computadora TI-99/4. Los valores 3, 4 y 5 no son accesibles como unidades tecla.

Sin embargo, la TI-99/4 usa las unidades tecla entre 0 y 5 para especificar modos de operación. Si la unidad-tecla es 0, el teclado estará en el modo especificado por el último CALL KEY que se ejecutó.

Si unidad-tecla es 1, la entrada se tomará desde el lado izquierdo del teclado.
Si unidad-tecla es 2, la entrada se tomará desde el lado derecho del teclado.

Una unidad-tecla igual a 3, define el teclado stándard para computadora TI-99/4. Tiene caracteres minúsculos solamente, y los códigos 1 a 15 están activos pero no se devuelve el control de caracteres.

Una unidad-tecla de 4 coloca la computadora en modo Pascal, con caracteres mayúsculos y minúsculos activos. También están activos los códigos de función 129 a 143 y los códigos de control de caracter de 1 a 31.

La unidad-tecla 5 coloca a la computadora TI 99/4A en modo BASIC. Están activos los caracteres mayúsculos y minúsculos. Los códigos de función activos son del 1 al 15, y los códigos de control de caracter activos son de 128 a 159 (y 187).

Además, se asignan códigos a las teclas de función y control para que las pueda utilizar el subprograma CALL KEY. Las teclas siguientes muestran las asignaciones de códigos.

CODIGOS DE TECLAS DE FUNCION

Códigos TI-99/A& MODO BASICO	*Modo Pascal*	*Nombre Función*	*Tecla Función*
1	129	AID	FCTN 7
2	130	CLEAR	FCTN 4
3	131	DELete	FCTN 1
4	132	INsert	FCTN 2
5	133	QUIT	FCTN =
6	134	REDO	FCTN 8
7	135	ERASE	FCTN 3
8	136	Flecha izq.	FCTN S
9	137	Flecha der.	FCTN D
10	138	Flecha abajo	FCTN X
11	139	Flecha arriba	FCTN E
12	140	PROC'D	FCTN 6
13	141	ENTER	ENTER
14	142	BEGIN	FCTN 5
15	143	BACK	FCTN 9

CODIGOS PARA TECLAS DE CONTROL

Modo BASIC	*Códigos Modo Pascal*	*Código Mnemónico*	*Presione*	*Comentarios*
129	1	SOH	**CONTROL A**	Com. de encabezamiento
130	2	STX	**CONTROL B**	Corn. de texto
131	3	ETX	**CONTROL C**	Fin de texto
132	4	EOT	**CONTROL D**	Fin de transmisión
133	5	ENQ	**CONTROL E**	Consulta
134	6	ACK	**CONTROL F**	Reconocimiento
135	7	BEL	**CONTROL G**	Campana
136	8	BS	**CONTROL H**	Espaciado hacia atrás
137	9	HT	**CONTROL I**	Tabulación horizontal
138	10	LF	**CONTROL J**	Alimentación de líneas
139	11	VT	**CONTROL K**	Tabulación vertical
140	12	FF	**CONTROL L**	Alimentación de formul.
141	13	CR	**CONTROL M**	Retorno del carro
142	14	SO	**CONTROL N**	Cambio afuera
143	15	SI	**CONTROL 0**	Cambio adentro
144	16	DLE	**CONTROL P**	
145	17	DC1	**CONTROL Q**	Disp. de control 1 (X—ON)
146	18	DC2	**CONTROL R**	Disp. de control 2
147	19	DC3	**CONTROL S**	Disp. de control 3 (X—OFF)
148	20	DC4	**CONTROL T**	Disp. de control 4
149	21	NAK	**CONTROL U**	Reconocimiento negativo
150	22	SYN	**CONTROL V**	Sincrónico desocupado
151	23	ETB	**CONTROL W**	Fin bloque de transm.
152	24	CAN	**CONTROL** X	Cancelación
153	25	EM	**CONTROL Y**	Fin de medio
154	26	SUB	**CONTROL Z**	Substituto
155	27	ESC	**CONTROL .**	Escape
156	28	FS	**CONTROL ;**	Separador de archivos
157	29	GS	**CONTROL =**	Separador de grupos
158	30	RS	**CONTROL 8**	Separador de registros
159	31	US	**CONTROL 9**	Separador de unidades

En el Manual de Referencia del Usuario de la computadora TI-99/4A se puede obtener información detallada sobre el uso del subprograma.

SUBPROGRAMA CALL VERSION

El subprograma VERSION (discutido en el capítulo 4 de su Manual de TI BASIC Extendido) devuelve ahora un valor 110 en ambas computadoras.

INSTRUCCION DATA

La computadora lee toda la información que viene a continuación de una instrucción DATA, considerándola parte de esa instrucción. Por lo tanto, en una línea con varias instrucciones, la instrucción DATA no debe ir seguida por otra instrucción.

NOTACION CIENTIFICA

Si usa notación científica o exponential, debe asegurarse que la "E" sea un caracter en mayúsculas. Una "e" podría ocasionar un mal funcionamiento del programa.

BUSQUEDA PREVIA -- ! P - y ! P ≤

Cuando se pide la ejecución de un programa mediante RUN, puede notarse que hay una pequeña demora antes de que el programa comience a ejecutarse realmente. Esta pausa es el tiempo que se toma la computadora para reservar espacio para las variables, arreglos y datos. Luego la computadora recorre instrucción por instrucción, establecimiento el valor de las funciones y variables. Dado que el tiempo requerido para esta búsqueda previa depende de la longitud del programa, puede desearse disminuir esa pausa, particularmente si el programa es muy largo.

TI BASIC Extendido tiene dos nuevos comandos ! P y ! P +, que le permiten determinar a qué instrucciones no se les aplicará este procedimiento. Dado que el propósito de esta búsqueda previa es reservar espacio en memoria para las variables, sólo las instrucciones que contienen la primera referencia a la variable necesitan intervenir en la búsqueda.
Por lo tanto habrá muchas otras instrucciones que podrán quedar fuera del procedimiento.

Se necesita una programación muy cuidadosa para minimizar las instrucciones que intervendrán en la búsqueda. Cuando en un programa se usan ciertos tipos de instrucciones (como se explica aquí) se deberán incluir en la búsqueda previa los siguientes procedimientos.

— Ingrese su primera instrucción DATA con búsqueda previa.
— Incluya la primera aparición de cada variable o arreglo. (También incluya la instrucción OPTION BASE, si la usa).
— Incluya la referencia a la primera llamada para cada instrucción CALL de cualquier subprograma.
— Incluya las instrucciones DEF para funciones definidas por el usuario.
— Incluya las instrucciones SUB y SUBEND.

Note que una variable perteneciente a un subprograma definido por el usuario (SUB) es considerada única, aún cuando existan en el programa otras variables con el mismo nombre y/o valor. Por lo tanto cada variable de un subprograma definido por el usuario deberá incluirse en la búsqueda previa.

Para usar la opción de búsqueda previa, asegúrese primero que su programa trabaja correctamente. Luego, al comienzo de un grupo de instrucciones coloque ! P para desactivar la búsqueda. Esto hará que su programa se ejecute más rápidamente. Cualquier instrucción referida a nombres de variable (no referenciada previamente durante la búsqueda previa) provocarán un error de sintaxis si la búsqueda está en "off'. Note que ! P no puede ir seguida por otra instrucción en una línea con múltiples instrucciones.

Para retomar la búsqueda use el comando ! P + . Este comando hace que la búsqueda se active y se defina espacio en la memoria para las variables. Recuerde incluir el comando ! P + antes de una instrucción SUB o SUBEND y no incorpore este comando como parte de una instrucción

Se puede utilizar la característica de búsqueda previa varias veces en un programa Activándola y desactivándola el programa puede comenzar a ejecutarse mas eficazmente. La efectividad de la búsqueda previa se notará más en los programas externos que en los pequeños. Note que si está usando esta característica con la computadora TI-99/4A, los comandos ! P y ! P + pueden también ingresarse con "p" minúscula.

Los ejemplos siguientes muestran cómo incluir instrucciones de búsqueda previa en un programa ya existente. El ejemplo final muestra un uso mas eficaz de esta característica utilizando una instrucción GOTO.

Ejemplos:

Programa original:

```
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 DATA 3
220 .
230 .
```

Con el agregado del control de búsqueda previa:

```
 10 DATA 3
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
125 !@P-
130 .
140 .
150 .
155 !@P+
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
185 !@P-
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Note que la primera instrucción DATA ha sido movida al comienzo del programa para que pueda ser incluída en la búsqueda. Con la inclusión de las instrucciones ! P y ! P + en las líneas 125, 155 y 185, la búsqueda se desactiva, se activa y vuelve a desactivarse. Esto hace que el programa se ejecute más rápidamente.

Con el agregado del GOTO:

Se agregó la habilidad de "engañar" a la computadora en lo referente a la reserva de espacio para CALL y variables, sin incluir realmente esas instrucciones. Para poder hacerlo, se puede usar la instrucción GOTO. El ejemplo siguiente muestra el programa original adaptado con búsqueda previa e instrucción GOTO.

```
 10 DATA 3
 20 GOTO 100::DELAY::CALL CHAR::CALL CLEAR::CALL HCHAR::CALL
    VCHAR::!@P-
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Note que el método del GOTO hace que se reserve el espacio de memoria necesario en la línea 20. Sin embargo, estas instrucciones no se ejecutarán hasta que se las encuentre mas tarde en el programa. Así, como se muestra en este ejemplo y los anteriores, se puede poner todas las referencias a las variables juntas, y las llamadas a los subprogramas no necesitan ser sintácticamente correctas. Este puede ser un uso más eficaz de la opción de búsqueda previa.

```
100 GOTO 180::X,Y,ALPHA,BETA,Z=DELTA::DIM B(10,10)
110 CALL KEY::CALL HCHAR::CALL CLEAR::CALL MYSUB
120 DATA 1,3,STRING
130 DEF F(X)=1-X*SIN(X)
140 .
150 .
160 .
170 !@P-
180 .
190 .
200 .
```

PROGRAMACION CON LETRAS MAYUSCULAS

Los nombres de dispositivos deben ser ingresados con letras mayúsculas únicamente. Un nombre correcto es DSK1, Dsk1 no lo es. Cualquier referencia a un dispositivo con letras minúsculas provocará un mensaje de error.

Los nombres de archivo también deben ser específicos. No solamente en su deletreo exacto, sino también en la combinación de mayúsculas y Por ejemplo el nombre de archivo MYFILE no es el mismo que Myfile.
Cualquier nombre de archivo que contenga una combinación de mayúsculas y minúsculas no será reconocido por la computadora TI-99/4. Solo la TI-99/4A admite las letras minúsculas.

Las minúsculas incluidas en instrucciones DATA o en expresiones alfanuméricas entre comillas funcionan correctamente y ofrecen una amplia variedad en técnicas de programación para la computadora TI-99/4A.
Sin embargo, las minúsculas incluidas en DATA y expresiones entre comillas, no aparecerán si se ejecuta el programa en la TI-99/4. Si planea ejecutar un programa en ambas computadoras, TI-99/4 y TI-99/4A, tenga especial cuidado con el uso de las letras mayúsculas.

Para poder mostrar letras mayúsculas cuando un programa escrito para la TI-99/4A quiere ejecutarse en una TI-99/4 se deberá incluir las instrucciones siguientes:

```
100 FOR 1=65 TO 90
110 CALL CHARPAT(I,A$)
120 B$="0000-&SEG$(A$,1,4)&SEG$(A$,7,4)&SEG$(A$,13,4)
130 CALL CHAR(I+32,B$) 140 NEXT I
```

La inserción de las anteriores líneas de programa en su programa, harán que la computadora TI-99/4 muestre letras mayúsculas.

COMANDO SIZE

El ejemplo para SIZE en el capítulo 4 de su manual de TI BASIC Extendido le informará ahora que tiene 24488 "BYTES OF PROGRAM SPACE FREE" (bytes libres de programa).

COMENTARIOS

Si tiene en su programa comentarios idénticos a las instrucciones de búsqueda previa (! P y ! P +) éste no se ejecutará correctamente.
Ahora estos grupos de caracteres se consideran palabras reservadas.

CORRECCION AL APENDICE C

El código ASCII 12 en el Apéndice C del manual de TI BASIC Extendido debe definirse como "PROC'D" en lugar del caracter CMD.

ARCHIVOS DE PROGRAMA EXTENSOS

Algunos programas escritos con TI BASIC pueden ser demasiado extensos para ejecutarse con TI BASIC Extendido, dado que este último requiere una mayor sobrecarga del sistema que TI BASIC. Si intenta cargar tal programa, el sistema se bloqueará y será necesario apagar la computadora durante algunos segundos y encenderla nuevamente.

Ingresando CALL FILES (1) o CALL FILES (2) antes de cargar su programa se podrá liberar suficiente memoria para poder ejecutar el programa con TI BASIC Extendido. (Se puede encontrar una explicación más completa sobre los comandos CALL FILES en el manual del Sistema de Discos).

Si el comando CALL FILES no libera suficiente memoria, deberá acortar el programa TI BASIC eliminando algunas instrucciones hasta que se adapte a la cantidad de memoria disponible.

Si tiene la Unidad Expansión de Memoria se puede ejecuta el programa completo usando el siguiente procedimiento:

1. Como medida de seguridad, haga una copia back-up de su programa TI BASIC en cassette o diskette.
2. Con la Expansión de Memoria conectada y encendida, cargue su programa con TI BASIC. A continuación borre varias instrucciones, y almacene el programa acortado en un cassette o diskette. Luego trate de cargar este programa acortado con TI BASIC Extendido.
3. Mecanografíe nuevamente las instrucciones que había borrado, en los lugares apropiados.
4. Almacene su programa sólo en undiskette. Ahora está listo para ejecutarlo con TI BASIC Extendido y la Expansión de Memoria.

Nota: Los programas convertidos de esta manera, sólo pueden ser ejecutados con TI BASIC Extendido y con la Expansión de Memoria conectada y encendida. No se almacenan en formato PROGRAM.

LA UNIDAD EXPANSION DE MEMORIA Y LOS PROGRAMAS EN CASSETTE

La unidad Expansión de Memoria agrega 32 k bytes de memoria de acceso al azar (RAM) a la memoria que viene con la computadora. Sin embargo, aún con la Expansión de memoria disponible el programa TI BASIC Extendido mas grande que puede almacenarse en cassette es de 12 k bytes.

Note que, a pesar de que la longitud real del programa es limitada, el uso de la Expansión de, Memoria proporciona otras ventajas. Por ejemplo, con la unidad conectada y encendida, un programa (que puede tener hasta 12 k bytes de longitud) se almacena en la expansión RAM. Los datos generados por el programa se almacenan en la Expansión de Memoria y las características en la memoria de la computadora. Sin la unidad, el programa debe ser más corto y tanto los datos numéricos como los alfanuméricos se almacenarán en la memoria de la computadora.

COMANDO CONTINUE

El comando CONTINUE se usa para retomar el programa cuando fue interrumpido con el comando BREAK o la tecla CLEAR. Sin embargo, si el último comando (antes del comando CONTINUE) contiene un error, el programa podría no continuar apropiadamente. El último comando antes del CONTINUE deberá ser correcto. Si recibe un mensaje de error, asegúrese de ingresar un comando correcto, por ejemplo un PRINT, antes de retomar la ejecución del programa.

ERRORES DEL MANUAL

Página 39
La segunda frase en el tercer párrafo de "Constantes numéricas" debe corregirse para que se lea " ... el número es mayor que 99 o menor que -99, luego ... "

Páginas 79 y 150
La cadena usada en Expresión-alfanumérica con DISPLAY ... USING y PRINT .. . USING pueden ser mas generales que los ejemplos que se muestran en el manual. Por ejemplo, los siguientes son instrucciones válidas.

```
PRINT USING A$:X,Y
DISPLAY USING RPT$("#",5)&V$:A(12)
```

Páginas 89, 133 y 135
Las instrucciones GOSUB, ON GOSUB, y ON GOTO no deberían usarse para transferir el control entre subprogramas.

Página 114
Si se presiona CLEAR cuando se está usando el comando LIST, el listado se detiene y no se puede reiniciar.

Páginas 118 y 119
Las figuras gráficas al final de la página 118 y al principio de la pag. 119 deben estar invertidas.

Página 185
La función TAB no debe usarse con las instrucciones PRINT ... USING o DISPLAY ... USING. También el segundo párrafo de esta explicación debe corregirse para que se lea como sigue: "Si el número de caracteres ya impreso en el registro corriente es menor que expresión-numérica, el próximo item de impresión se imprimirá en el comienzo de la posición indicada por expresión-numérica. Si el número de caracteres ya impreso en el registro corriente es mayor que o igual a la posición indicada por expresión-numérica, el próximo item de impresión se imprime en el registro siguiente comenzando en la posición indicada por expresión-numérica.

Página 200
En el Apéndice H, Combinaciones de Color, los códigos de color para las dos últimas líneas de la mejor categoría, deben ser como sigue:

14, 10	Magenta sobre rojo claro
3, 16	Verde mediano sobre blanco

En la cuarta "mejor" categoría, la tercera combinación de la segunda columna debe decir:

6, 2	Azul claro sobre negro.

Lineas de formato

La lista siguiente da las correcciones que deben hacerse a los formatos indicados, y también muestra la actual información sobre formatos.

Instrucción DIM (pág. 76)
Formato correcto: (entero 1, entero 2... entero 7) ...
Formato actual : (entero 1, entero 2... entero 7) …

Instrucción DISPLAY (pág. 77)
Formato correcto: SIZE (expresión-numérica): lista-de-impresión.
Formato actual: SIZE (expresión-numérica): lista-de-variables

DISPLAY . . . USING (pág. 79)
Formatos correctos: USING expresión-alfanumérica: lista-de-impresión
USING número-línea: lista-de-impresión
Formatos actuales: USING expresión-alfanumérica: lista-de-variables
USING número-línea: lista-de-variables

LINPUT (pág. 113)
Formato correcto: número-archivo, REC número-registro:
Formato actual: número-archivo, REC número-registro:

PRINT ... USING (pág. 150)
Formato correcto: número-archivo, REC número-registro,
Formato actual: número-archivo, REC número-registro

Subprograma SPRITE (pág. 173)
Formato correcto: punto-col., veloc.-col. …
Formato actual: punto-col., veloc.-col. …

TEXAS INSTRUMENTS
ARGENTINA S. A. I. C. F.

Impreso en la Argentina 1983

TEXAS INSTRUMENTS
ARGENTINA S.A.I.C.F.

Impreso en Argentina