Beta Basic

Es imprescindible leer el contenido de este manual antes de empezar a utilizar el programa.

VEA EL APARTADO SOBRE CONSULTAS EN LA CONTRAPORTADA.

CONTENIDO

INTRODUCCION

Beta Basic añade 30 nuevos comandos y más do 29 nuevas funciones a las ya disponibles en el Basic del Spectrum. Además, algunos comandos existentes han sido considerablemente reforzados, y se han añadido algunas posibilidades convenientes, como un «apuntador de línea en curso» parpadeante, un BREAK más potente y la posibilidad de mover el cursor de edición en todas direcciones.

Los nuevos comandos se obtienen cambiando al modo GRAPHICS y presionando una tecla (a veces con SHIFT). Las nuevas funciones se obtienen entrando la palabra clave

FN, después una letra y “(“ o “$”. Se verán y actuarán como palabras clave después de entrarlas.

Para cargar el programa desde el casette, escribe LOAD “” o LOAD “Beta Basic”. Tres líneas de Basic (líneas 0, 1 y 2) serán cargadas y la línea 2 se ejecutará automáticamente, cargando así la parte central del Beta Basic, que es un bloque de código máquina de 9,3 K de longitud. RAMTOP se eleva a 55800 para proteger el código máquina en la parte alta de la memoria. La carga habrá terminado cuando veas el mensaje de copyright en la parte inferior de tu pantalla. Si esto no ocurre, ha habido un error de carga. Inténtalo de nuevo. (Utiliza un volumen diferente o prueba con la segunda copia del programa en tu cassette). Las líneas 1 y 2 serán automáticamente borradas una vez aparezca el mensaje de copyright, y sólo quedará la línea 0. La línea 1 es una rutina de SAVE que grabará el programa actual Basic, seguido del código máquina del Beta Basic. Si deseas usarlo, haz MERGE del cargador Basic para detener la auto ejecución, para el casette (y quizás rebobina un poco) después entra:

CLEAR rt:LOAD “” CODE

Y pon en marcha el cassette. Cuando la segunda parte del programa se haya cargado, RUN hará que el programa sea salvado en cinta. Para salvarlo en el Microdrive, entra RANDOMIZE USR 58419 (esto inicializa el Beta Basic) y luego modifica las líneas 1 y 2 como sigue antes de utilizar RUN.

```
1 LET rt = DPEEK(23730):RANDOMIZE USR 59904: SAVE*0«m»;1; «run»
  LINE 2: SAVE*«m»;1;«BB»CODE rt+1,65367 rt: STOP

2 CLEAR rt: LOAD *«m»;1;«BB»CODE : CLS : RANDOMIZE USR 58419
  DELETE 1 TO 2
```

NOTA IMPORTANTE: La facilidad de salvar Beta Basic ha sido añadida para permitirte hacer copias para tu propio uso, o para transferirlo al Microdrive. Proveer a otras personas de copias es una violación del copyright. Intentamos suministrar soporte a los clientes; por favor apóyanos cumpliendo nuestro copyright.

La línea 0 contiene definiciones de función para las nuevas funciones Beta Basic, y está siempre presente, aunque no es normalmente listada a menos que sea la única línea del programa. La otra señal inmediata de que Beta Basic reside en memoria es el «puntero de línea actual». El tímido «click» del teclado ha sido alargado. Si esto te desagrada, POKE 23609,0 lo apagará. Cualquier programa ya existente que «fundas» (MERGE) ahora con el Beta Basic, se ejecutará normalmente, excepto cuando tengas que entrar «KEYWORDS 0» por ejemplo si quieres imprimir gráficos definidos por el usuario en vez de palabras clave del Beta Basic. (No cargues con LOAD un programa que no haya sido escrito bajo Beta Basic o perderás la línea 0 —.en vez de esto, usa MERGE.)

Observaras algún incremento de velocidad en el programa comparado con el Basic del Spectrum, particularmente en programas que usan un montón de GO TO’s y GOSUB's: (Estos comandos trabajarán más rápido que antes, cuando la línea de destino está en la parte inferior del programa.) RETURN es también mucho más rápido, — un RETURN a la última línea de un programa es ahora tan rápido como un RETURN a la primera línea (no como en el Basic del Spectrum).

NEW es menos drástico de lo que usualmente suele ser — borra cualquier programa que esté presente, excepto la línea 0, y ejecuta CLEAR. (Si quieres deshacerte del Beta Basic, el efecto de la desconexión del ordenador puede ser suplido por RANDOMIZE USR 0.)

Una vez hayas escrito un programa que contenga alguna de las nuevas palabras clave, puedes grabarlo en cinta o microdrive de forma normal. Cuando desees cargarlo de nuevo, antes de nada carga el Beta Basic si no está ya presente en el ordenador, después carga tu programa. (Una copia de la línea 0 seré grabada, con el programa, por lo tanto no es necesario usar MERGE).

Si olvidas cargar el Beta Basic primero, las nuevas palabras,clave aparecerán como caracteres simples los cuales darán mensajes «Nonsense in BASIC» cuando ejecutes el programa. Puedes usar:

MERGE «Beta Basic»:GO TO 2

para cargar Beta Basic en este caso. El «GO TO 2» es necesario para hacer el «auto run» de la línea 2 (el auto run normal es detenido por MERGE).

Todos menos uno (EDIT) de los nuevos comandos se obtienen presionando «shift GRAPHICS». Una vez en modo gráfico, la mayoría de las teclas darán una palabra clave, en vez de un gráfico. Se sale automáticamente de modo gráfico después de apretar una tecla. Las asignaciones de tecla son mostradas en el diagrama del teclado de Spectrum, y cerca de la parte superior de la entrada para cada palabra clave de este manual.

Las nuevas funciones se obtienen entrando la palabra clave FN, seguida por una letra y «$» o «(». Más detalles en la segunda sección de este manual.

El funcionamiento de cada nuevo comando está explicado en las siguientes páginas. La sintaxis requerida se muestra al principio de cada entrada — los paréntesis como estos « < > » significan que lo que está encerrado puede ser omitido si se desea. El chequeo completo de sintaxis se hace en la entrada como es usual. Los errores, durante la ejecución del programa pueden producir la impresión de uno de los mensajes del Beta Basic, o de uno de los mensajes del Basic Spectrum, en el caso apropiado.

En los ejemplos dados en las páginas que siguen, se ha hecho un esfuerzo por evitar el uso de más de uno o dos comandos nuevos en el mismo ejemplo. Aunque esto ayuda a entenderlo, no significa que la aproximación más indicada no sea a menudo utilizada. Cuando te sientas capaz de usar muchos comandos nuevos a la vez, hazlo, por favor.

Hemos hecho to posible pars asegurarnos de que este programa y esto manual estén libres de errores. Si encuentras algún problema, escribe por favor a la dirección abajo indicada, describiendo las circunstancias lo más ampliamente posible. No podemos asumir ninguna responsabilidad por las pérdidas en que se incurra debido a la programación o a los errores de documentación; sin embargo, intentaremos corregir la mayoría de los problemas. Agradeceremos también las sugerencias para futuras ampliaciones, u otros comentarios.

Si has comprado el programa en una tienda, por favor, guarda tu casette original como evidencia de compra para adquirir por un precio reducido futuras versiones ampliadas del Beta Basic.

VENTAMATIC

Avda. Rhode, 253

ROSES

GIRONA

ALTER (descripción de atributo) TO descripción de atributo:

Tecla A

ALTER permite una manipulación extensiva del fichero de atributos de pantalla (que guarda la información de INK, PAPER, FLASH y BRIGHT para cada posición de carácter). De esta forma tan simple, ALTER puede cambiar la totalidad del color de INK o PAPER sin limpiar la pantalla:

```
100 PRINT AT 0,0;«TEST»: PAUSE 50: ALTER TO PAPER 1
```

cambiará el color de PAPER en todas las posiciones a color azul. También puedes cambiar la totalidad de la pantalla a alguna combinación de atributos:

```
ALTER TO PAPER 2, INK 7, FLASH 1
```

lo hará. Es posible hacer una selección de las posiciones de caracteres que serán afectadas incluyendo una descripción de sus atributos antes del TO:

```
ALTER INK 7 TO INK 0
```

cambiará cualquier carácter en tinta blanca a tinta negra. Es posible crear un gráfico complicado usando tinta del mismo color que el papel, y hacerlo aparecer instantáneamente usando ALTER para cambiar el color de la tinta (o del papel). Son posibles sentencias complejas como ésta:

```
ALTER INK 3, BRIGHT 1, PAPER 7 TO INK 5, FLASH 1
```

que afectará sólo a las posiciones de caracteres con los atributos INK 3, BRIGHT 1 y PAPER 7.

El siguiente programa muestra alguna de las posibilidades - crea un gran “tablero” parpadeante. Prueba cambiando las sentencias ALTER para variar el efecto. (Puedes cambiar la duración del parpadeo cambiando las líneas 170 y 220.

```
100 LET A = 2: LET B = 4
110 FOR L = 1 TO 5
120 FOR N = 1 TO 16
130 PRINT lNK A; PAPER B; «XXXX»: PAPER A; INK B;«0000»;
140 NEXT N
150 LET C = A: LET A = B: LET B = C
160 NEXT L
170 LET T = 30
180 ALTER INK K TO INK B: PAUSE T
190 ALTER PAPER A TO INK A: PAUSE T
200 ALTER INK A TO PAPER B: PAUSE T
210 ALTER INK B TO PAPER A: PAUSE T
220 LET T= T - T/10 + 1; GO TO 180
```

AUTO (número de línea) (,Intervalo)

Tecla: 6

AUTO activa una numeración de líneas automática que facilita la entrada de programas — el ordenador entra los números de línea por ti. Si se omite el intervalo, se usará diez. Si AUTO es entrado solo, el número de línea asumido será el de la línea actual (con el

cursor « > »), más diez. AUTO se desactiva cuando el número de línea es menor que 10 o mayor de 9983, o cuando se imprime algún mensaje. Un método conveniente de dejar AUTO es apretar BREAK durante algo más de un segundo.

Si deseas saltar un bloque de números de líneas mientras usas AUTO, borra el número de línea ofrecido y entra el tuyo. Teclea el resto de la línea y éntrala. El siguiente número de línea ofrecido será el número de línea que proporcionaste, más el intervalo.

AUTO	desde la línea actual + 10, intervalo 10
AUTO 100	desde la línea 100, intervalo 10
AUTO 100,5	desde la línea 100, intervalo 5

BREAK

Tecla: Shift-space (no en modo gráfico).

(Break no es una palabra clave.) El Break normal ofrecido por el Basic Sinclair es perfectamente adecuado para la mayoría de los propósitos; sin embargo, mucha gente interesada en código máquina habrá experimentado la desesperación cuando su código entra en un bucle sin fin que no puede ser detenido con BREAK. Beta Basic añade un sistema de BREAK - si presionas shift-space durante algo más de un segundo, el sistema decide que debes estar «colgado» y ejecuta Break si la rutina normal no funciona. Esto puede ser útil si descuidadamente has impedido «STOP IN INPUT» o «BREAK into program» usando ON ERROR. También te sacará de una INPUT LINE.

I) Si has visto el mensaje de copyright de Sinclair, BREAK no te podrá ayudar.

II) (Para programadores de código máquina) - no debes prohibir las interrupciones si deseas usar esta forma de BREAK. (El modo de interrupción 2 está siempre activado.)

CLOCK número o string

Tecla: C

CLOCK controla el funcionamiento de un reloj «interrupt driven» de 24 horas, que puede imprimir el tiempo actual en la esquina superior derecha en horas, minutos y segundos. Cuando un tiempo preseleccionado es activado, una alarma audible puede hacerse sonar y/o una rutina dada puede ser accedida por GOSUB. «Interrupt driven» significa que el reloj sigue corriendo siempre mientras escribes o ejecutas algún programa. Sólo durante las operaciones con cinta, Microdrive o Interface 1 o durante BEEP el reloj parará. Para controlar cuál de las posibilidades mencionadas está operando, CLOCK deberá ir seguido por una expresión numérica en el rango de 0 a 7, que activará el reloj al modo especificado. El resultado será como sigue:

MODO	ALARMA GOSUB	ALARMA AUDIBLE	DISPLAY
0	NO	NO	NO
1	NO	NO	SI
2	NO	SI	NO
3	NO	SI	SI
4	SI	NO	NO
5	SI	NO	SI

6	SI	SI	NO
7	SI	SI	SI

El reloj empezará a correr tan pronto como Beta Basic es cargado, empezando en «00:00:00» si ia carga fue desde la cinta original. Entra CLOCK 1 para ver el display. No hay duda de que el tiempo es incorrecto. Para seleccionar una hora, CLOCK deberá ir seguido por una cadena (string):

CLOCK «09:29:55»

por ejemplo. No necesitas realmente los separadores «:» — CLOCK (con una excepción) simplemente cogerá los seis primeros dígitos de la cadena como tiempo actual, e ignorará cualquier otro. Si el string tiene menos de seis dígitos, el resto son asumidos como ceros:

CLOCK «xyz10»

dará un display de 10:00:00. La excepción mencionada anteriormente es «a» (o «A»). Esta letra provoca que los siguientes dígitos sean usados como hora de alarma

CLOCK «A06:20»

Activará la alarma a las seis y veinte. Cuando la hora actual alcance la hora de alarma, la alarma audible sonará brevemente si ja sido activada usando CLOCK 2, 3, 6 o 7. (Ver tabla anterior)

También es posible para una subrutina especifica ser accedida por GOSUB cuando la hora de alarma es alcanzada, si se ha ejecutado CLOCK 4, 5, 6 o 7. Esto sólo ocurrirá si algún tipo de programa se está ejecutando al mismo tiempo (no interrumpirá la escritura de tu programa).

El programa puede ser tan simple como: 10 GO TO 10 o tan complicado como un procesador de textos o un juego. Una vez la hora de alarma ha sido alcanzada, CLOCK espera a que acabe la sentencia corriente (que podría tardar un poco si es un INPUT o PAUSE) y después ejecuta una subrutina especifica. No se producirá ningún GOSUB desde un programa que está enteramente en código máquina. La rutina que será accedida por CLOCK número de línea, siendo el número entre 8 y 9999. Dentro de la subrutina deberás usar nombres de variables diferentes de las usadas por el programa principal, excepto que quieras alterar su funcionamiento. Si necesitas que la subrutina grabe datos, y el programa principal usa CLEAR o RUN, tendrás que hacer un POKE de los datos en un área apropiada de memoria, o lo perderás.

Las posibles aplicaciones de la subrutina incluyen variaciones de la alarma normal - p. e. una canción con gráficos, el campaneo de las horas (la hora actual es facilitada como una función y puede ser usado por la rutina — ver TIME$). Los entusiastas de la electrónica pueden recoger datos del mundo exterior cada minuto u hora o cualquier intervalo de tiempo con algo como esto:

```
8999 STOP
9000 PRINT «subrutina activada»
9010 LET pointer = DPEEK(USR «a»):POKE pointer, IN 127
9020 LET pointer = pointer+1:IF pointer > 65535 THEN LET pointer=USR «a» + 2
9030 DPOKE USR «a», pointer:LET Z$ = TIME$()
9040 LET horas = VAL Z$(1 TO 2): LET mins = VAL Z$(4 TO 5)
9050 LET mins = mins+1: IF mins = 60 THEN LET horas =horas+1: LET mins =0
9060 CLOCK «a» + USING$(«00», horas) + USING$(«00», mins): RETURN
```

¡No ejecutes esto! En vez de hacerlo, entra como un comando directo:

```
DPOKE USR «a», USR «a»+2: CLOCK 9000: CLOCK 5
```

El comando directo inicializa el apuntador (almacenado en las posiciones de memoria USR «a» y USR «a» + 1) para apuntar a la siguiente posición en el área de gráficos de usuario. La línea 9000 es seleccionada para ser accedida por el GO SUB cuando se alcance la hora de alarma, y la alarma Go sub es activada. Activa la alarma para irte en un futuro cercano usando:

```
CLOCK «AXXXX»
```

donde «XXXX», es algo más tarde de la hora actual. Ahora ejecuta otro programa. La subrutina de CLOCK se activará cada minuto; coge un valor del port 127 el cual es almacenado en una posición del área de gráficos de usuario indicado por el regularmente incrementado apuntador (el cual vuelve a su posición inicial si alcanza el tope.de memoria), para su posterior impresión. Si no tienes ningún proyecto sobre ports, puedes usar mucho del programa anterior .(líneas 9040:9060) para hacer alguna otra cosa a intervalos regulares. Estas líneas activan la alarma para un minuto después de la hora actual, antes de volver al programa principal.

CODIGOS DE CONTROL DE CURSOR

CHR$ 8 (cursor izquierda)

CHR$ 9 .(cursor derecha)

CHR$ 10 (cursor abajo)

CHR$ 11 (cursor arriba)

Beta Basic permite a todos los caracteres anteriores trabajar como deben cuando se imprimen (ej. PRINT CHR$ 10 moverá la posición de impresión abajo una línea, PRINT CHR$ 9; moverá la posición de impresión un carácter a la derecha). Estos códigos de control también trabajan cuando usamos PLOT para situar string$
en la pantalla.

El carácter CHR$ 8 del Basic Spectrum (cursor izquierda) tiene una pega: no es posible retroceder un espacio con el cursor a la línea superior del display desde una línea inferior, y si empiezas en la línea superior, puedes retroceder fuera de la pantalla (¡dentro del programa!). Esto ha sido corregido ahora.

El carácter CHR$ 9 del Basic Spectrum (cursor derecha) tiene una pega diferente: que se detiene trabajando en conjunto. Esto se ha corregido.

La impresión del Basic Spectrum de los caracteres CHR$ 10 y CHR$ 11 se hace como «?», lo que no es muy práctico. Ahora funcionan de la forma esperada.

Como ejemplo, los cuatro caracteres de control pueden incluirse en strings para permitir la impresión de complejas figuras rápida y fácilmente (una vez que el string ha sido creado).

```
10 LET a$ - «1234» + CHR$ 8 + CHR$ 10 + «5» + CHR$ 8 + CHR$ 10+ «678» +
CHR$ 8+ CHR$ 11 + «9»

20 PRINT AT 10,10; a$

30 PAUSE 100:CLS

40 FOR n = 32 TO 255

50 PLOT n,n/2; a$: NEXT n
```

Esto puede ser muy efectivo con gráficos diseñados por el usuario; te dejamos diseñarlos.

DEF KEY strings de una letra; string

o DEF KEY string de una letra: sentencia: sentencia:…

Tecla: Shift 1 (igual que DEF FN)

Beta Basic permite strings o líneas de programa para ser producidas por cualquier tecla de letra o número, con los caracteres producidos siendo entrados en el ordenador o bien permaneciendo en la parte baja de la pantalla hasta que se presione ENTER. El último caso se selecciona haciendo que el último carácter del string sea «:» o haciendo seguir a la última sentencia de la línea con «:». (El signo «:» no se usa excepto para decir al ordenador lo que hay que hacer). Prueba:

```
DEF KEY «1»; «HOLA:»
```

Ahora aprieta «symbol shift» y «space» juntos. El cursor cambiará a una estrella parpadeante. Si aprietas «1», aparecerá «HOLA» en la parte baja de tu pantalla. Puesto que las otras teclas no las has definido, si has apretado cualquier otra tecla habrás obtenido sólo el valor normal.

```
DEF KEY «a»: PRINT «Adiós»
```

En este ejemplo, la parte de línea de programa después de DEF KEY «a»: se asigna a «a» (o «A» - son tratadas de la misma forma). No es ejecutado cuando la línea es introducida. También, ya que la última sentencia no va seguida por «:», cuando se presiona symbol shift/space, y se aprieta «a», la sentencia asignada será entrada y ejecutada. Si hemos usado:

```
DEF KEY«a»; «10 REM Hola»
```

(usando THEN temporalmente para obtener la palabra clave REM) la línea habrá sido añadida al listado, después de haber pasado el chequeo de sintaxis.

Si usas un Microdrive, esto te dará algunas ideas:

```
DEF KEY «i»: INPUT «nombre?»;a$: LOAD *«m»;1;a$
```

A una tecla se le puede asignar un valor diferente en cualquier momento — la definición anterior será sobreescrita. Si se usa un string nulo, o no sigue ninguna instrucción a la definición de tecla, entonces la tecla no contendrá definición alguna. DEF KEY ERASE borrará todas las definiciones de tecla, las cuales están almacenadas por encima de RAMTOP y por otra parte protegidas también de NEW. La rutina SAVE del cargador Basic salvará todas las definiciones con e! programa. (Salva desde RAMTOP hasta el final de Beta Basic).

Lo siguiente es probablemente de interés sólo par los usuarios del código máquina.

RAMTOP es automáticamente bajado para acomodar las definiciones de tecla; si usas CLEAR tu mismo para cambiar RAMTOP esto probablemente impedirá que las asignaciones de tecla que hayas hecho sean encontradas. La siguiente rutina moverá RAMTOP y cualquier definición de tecla hacia debajo de la memoria por un número dado de espacios para prevenir este problema. El espacio está hecho entre el final de las definiciones de tecla y el RAMTOP original, p. e. 55800, pero esto puede ser cambiado si se desea.

```
10 INPUT «Espaclos?»;espacio
20 LET –oldrt = DPEEK (23730)
30 DPOKE 23728,oldrt
40 CLEAR oldrt-espacio
```

```
50 LET oldrt = DPEEK (23728)
60 LET a$ = MEMORY$ () (oldrt TO 56800)
70 LET newrt = DPEEK (23730)
80 LET space oldrt-newrt
90 POKE newrt,a$
100 POKE 55801-space,STRING$(space,CHR$ 0)
```

Deja siempre un byte a cero después de la última definición como marcador de final.

DEF PROC nombre de procedimiento

Tecla: 1 (la misma que DEF FN).

Ver también: PROC, END PROC

Empieza la definición de un procedimiento nombrado,— ver PROC. DEF PROC debe ser la primera palabra clave de una línea (aunque se permiten espacios y códigos de control de color precediéndole). El nombre de procedimiento utilizado debe seguir las reglas de los nombres de variable — el primer carácter debe ser una letra, seguida por un string de letras o números, o $. Los espacios no son significativos, y mayúsculas y minúsculas son equivalentes. Un procedimiento puede tener el mismo nombre que una variable sin ninguna confusión.

DELETE (número de línea) TO (número de línea)

Tecla: 7 (la misma que ERASE)

Borra todas las líneas en el bloque especificado del programa. Si el primer número de línea se omite, se asumirá la primera línea después de la línea 0; si se omite el segundo número se asumirá la última línea del programa.

DELETE TO 100	borra todas las líneas después de la 0 hasta la 100 inclusive.
DELETE 100 TO	borra la línea 100 y todas las siguientes.
DELETE 100 TO 100	borra solamente la línea 100.
DELETE 0 TO 0	borra solamente la línea 0.
DELETE TO	borra todo el programa excepto la línea 0.

El último ejemplo es diferente de NEW, en que no limpia las variables. Cualquier número de línea especificado debe existir u obtendrás el error U, «No such line». DELETE puede incluirse en un programa, con ciertas reservas. Cuando partes más altas del listado de un programa son borradas por una instrucción DELETE que es parte de una subrutina, PROCedimiento o DO - LOOP, el programa normalmente parará ya que las direcciones almacenadas ya no corresponderán a la propia situación del programa. Si la instrucción DELETE se usa para borrarse ella misma del programa, deberá ser la última instrucción del bloque a borrar.

Una posible aplicación de DELETE dentro de un programa sería el borrado de sentencias DATA una vez han sido leídas, liberando memoria extra para variables (los números en las sentencias DATA ocupan, como mínimo, 8 bytes por valor).

DO

o DO WHILE condición

o DO UNTIL condición

Tecla: D (WHILE es la tecla J, UNTIL la tecla K)

Ver también: LOOP, EXIT IF

DO y LOOP, junto con sus calificativos WHILE y UNTIL proporcionan una estructura de control que posee algunas ventajas sobre las proporcionadas normalmente por el Basic. Por sí misma, DO simplemente sirve como una marca a la cual una instrucción LOOP emparejada puede regresar:

```
10 DO
20 PRINT «Hola »; PAUSE 50
30 LOOP
```

Esto se mantiene imprimiendo hasta que se presiona Break. DO puede ser cualificado por WHILE (alguna condición). Si la condición especificada es cierta, las sentencias y líneas de programa después del DO será ejecutado hasta que se encuentre un LOOP. La parte de programa entre el DO y el LOOP es ejecutada mientras la condición es verdadera. Si la condición especificada es falsa, la siguiente parte del programa es ignorada hasta que se encuentre un LOOP. DO UNTIL (alguna condición) es exactamente lo contrario — la parte del programa entre el DO y el siguiente LOOP será ejecutada sólo si la condición es falsa (o por decirlo de otra manera, es ejecutada hasta que la condición sea verdadera).

```
10 LET total = 0
20 DO UNTIL total >100
30 INPUT «Entra un número »;x
40 LET total = total + x: PRINT total
50 LOOP
60 PRINT «Esto es superior a cien»
```

La línea 20 puede reemplazarse por:

```
20 DO WHILE total < = 100
```

Las parejas DO - LOOP deben anidarse de la misma forma que los bucles FOR – NEXT. Por ejemplo:

```
DO ------  |
DO ---- |  |
LOOP -|  |
LOOP---  |


DO ------  |
DO ---- |  |
LOOP -|  |
          |
DO -----|  |
LOOP -|  |
LOOP---  |
```

Puesto que la dirección del DO es almacenada, no debes saltar fuera de un DO LOOP a menos que uses EXIT IF o POP (de lo contrario desordenará la pila del ordenador). Si DO no va seguido en algún punto por LOOP, obtendrá el mensaje S, «Missing LOOP», si un DO condicional intenta saltar fuera del DO LOOP y no puede encontrar dónde acaba.

Estructuras de control muy similares a los DO LOOP existen en otros lenguajes de programación bajo nombres distintos:

REPEAT (instrucciones) UNTIL (condición)	=	DO (instrucciones) LOOP UNTIL (condición)
REPEAT (instrucciones) UNTIL FALSE	=	DO (instrucciones) LOOP
WHILE (condición) (instrucciones) ENDWHILE (o WEND)	=	DO WHILE (condición) (instrucciones) LOOP

DPOKE dirección, número

Tecla: P

Ver también: DPEEK (dirección) función

DPOKE significa Doble POKE . el equivalente en Basic es:

POKE dirección, número — INT(número/256)*256
POKE dirección + 1, INT(número 256)

En otras palabras, el byte menos significativo del número se «POKEa» en la dirección, y el byte más significativo se «POKEa» en la siguiente (superior) dirección.
El código máquina almacena números entre 0 y 65525, de esta forma, muchas variables del sistema tienen este formato y son más fácilmente alteradas con DPOKE. La función DPEEK proporciona el equivalente al doble PEEK.

EDIT < número de línea >

Tecla: 0 (cero; no en modo gráfico)

¡Esto no es lo mismo que presionar Shift 1! EDIT es una palabra clave – es un intento de perfeccionar la irritante secuencia: LIST (alguna línea), BREAK, shift 1. Para ser una mejora, EDIT realmente ha de ser una palabra clave sin Shift, pero todas las teclas de letras son usadas ya de esta forma, y las teclas numéricas son necesarias para los números de línea. Sin embargo, un número de línea no comienza normalmente con cero, por tanto puede usarse esta tecla. La palabra clave EDIT se obtiene si «0» es la primera tecla apretada después de un ENTER precedente. Si se le añade un número de línea y se pulsa ÉNTER, la línea especificada estará inmediatamente lista para editar, si se omite el número de línea, obtendrás la línea actual. La comodidad de editar (iniciada ya sea por shift 1 o EDIT) ha sido perfeccionada por la posibilidad de usar las teclas de cursor arriba y abajo para moverlo dentro de la línea. El cursor no irá a través de las palabras, pero coge el primer espacio siguiente. Si intentas mover el cursor más atrás del principio de la línea o más adelante del final, el cursor saltará al final de la línea. Usa EDIT, luego flecha hacia arriba, como rápido, y fácil sistema de añadir más instrucciones al final de una línea.

ELSE (instrucciones)

Tecla: E

ELSE es parte de la estructura IF - THEN. Normalmente, cuando la instrucción siguiente a IF es falsa, la ejecución del programa saltará a la siguiente línea. Sin embargo, cuando la pareja IF - THEN tiene un ELSE asociado posteriormente en la línea, el programa continuará con las instrucciones que siguen al ELSE. Cuando la condición siguiente al IF es cierta, en otra pasada, la línea sólo ejecutará lo anterior al ELSE — después saltará a la siguiente línea. Por ejemplo:

```
10 INPUT «Dame un número »; x
20 PRINT «Es este número = 1 ?»
30 PAUSE 50
40 IF x = 1 THEN PRINT «cierto»:ELSE PRINT «falso»
50 GOTO 10
```

(Fíjate que ELSE siempre va precedido por dos puntos «:»). La situación es más complicada si tienes varios IFs y ELSEs en la misma línea - aquí tienes algunos ejemplos que te enseñan que ELSE se utilizará para los diferentes IFs cuando su condición especificada es falsa.

```
                    _____________
IF-THEN inst.: IF-THEN inst: ELSE inst.

    ____________________________________
                    _________________
IF-THEN inst.: IF-THEN inst.: ELSE inst.: ELSE inst.

    _______________   _________________
IF-THEN inst.: ELSE inst.: IF-THEN inst.: ELSE inst
```

END PROC

Tecla: 3

Ver también: DEF PROC, PROC

Marca el final de un procedimiento nombrado, permitiendo al ordenador retirar la ejecución de las instrucciones entre DEF PROC y END PROC, a menos que el procedimiento sea invocado por PROC. Cuando un procedimiento está en uso, END PROC termina su ejecución y provoca la ejecución del programa para volver a la instrucción siguiente a PROC, (Ver PROC para más detalles). Usando END PROC sin DEF PROC se da el mensaje W, «Missing DEF PROC».

EXIT IF (condición)

Tecla: I

Ver también: DO LOOP

Esto es parte de la estructura DO – LOOP. ¡Lee acerca de ella primero! EXIT IF se usa para dejar un DO - LOOP desde cualquier lugar de dentro en vez de en el DO o el LOOP. Si la condición especificada es cierta, la ejecución del programa salta a la instrucción siguiente al LOOP; en cualquier otro caso no sucede nada.

```
100 DO: PRINT «línea 100». PAUSE 20
110 EXIT IF INKEY$ = « STOP
120 PRINT «línea 120»: PAUSE 20: LOOP
130 PRINT «Fuera del LOOP»
```

Esto se repetirá hasta que se pulse «STOP» (shift A). Fíjate que «línea 120» no se imprime cuando se sale del DO LOOP. Si no encuentra LOOP, obtendrás el informe S. «Missing LOOP»

FILL x,y

o FILL color INK: x,y

o FILL color PAPER : x,y

Tecla: F

Llena un área de PAPER con INK si se usa FILL o FILL INK, o llena un área de INK con PAPER si se usa FILL PAPER, empezando en las coordenadas x,y. Si intentas llenar un área con INK en el punto x,y y sus alrededores ya están en INK, no pasará nada. De otra forma, que en el caso normal en el Basic Spectrum el número del color puede omitirse si se desea un FILL con el color actual de INK o PAPER. Por ejemplo FILL PAPER; x,y es correcto.

```
10 FOR N = 1 TO 6
20 CLS
30 CIRCLE INK N; 128,88,N * 10
40 FILL INK N;128,88
50 NEXT N
```

Es posible usar una forma compleja de FILL como esta:

```
FILL INK 2; PAPER 1, FLASH 1; x,y
```

En este caso, la primera palabra después de FILL determina si se usará INK o PAPER y las otras palabras sólo cambiarán los atributos de las líneas llenas.

Ya que no es posible para más que dos colores ocupar el mismo cuadro de carácter, coge un poquito para producir algunos cortes sin resultados confusos donde dos áreas de INK son adyacentes. Se hace necesario tener las áreas unidas a través de una separación de un cuadro de carácter.

```
10 LET x=128: LET y=88: LET rad=70
20 LET L = (SQR 2) + rad/2
30 CIRCLE x,y,rad
40 PLOT x,y: DRAW L, L
50 PLOT x,y: DRAW L, L
60 PLOT x,y: DRAW 0, rad
70 FILL INK 2; x 5,y
80 FILL INK 4; x + 5,y
```

FILL trabaja con cualquier figura. Como ejemplo de figura compleja, prueba esto, que llenará la pantalla completa excepto las áreas de dentro de las «Q»s.

```
PRINT STRING$ (704,«Q»): FILL 0,0
```

El método usado trabaja más rápidamente si se dispone de cantidad de memoria. Si llenas una gran área blanca tal como la pantalla completa verás pausas hacia el final según FILL chequea sus datos almacenados para no olvidar los puntos.

El número de píxels llenados durante el último uso de FILL se puede obtener mediante la función FILLED()

Un FILL puede ser detenido en cualquier momento usando BREAK.

GET variable numérica o variable string

Tecla: G

Como STRING$, GET es una forma de leer el teclado sin el uso de ENTER. La diferencia es que GET espera a que se apriete una tecla antes de continuar. Usado con una variable string, GET obtiene un string de un carácter.

```
10 GET A$; PRINT A$; GO TO 10
```

Esto trabaja un poco como una máquina de escribir. Puedes seleccionan entre mayúsculas y minúsculas de la forma usual. Los controles de cursor mueven la posición de impresión en todas las direcciones, y ENTER es equivalente a PRINT. Una versión más sofisticada del programa anterior es:

```
10 GET A$: PRINT A$; FLASH 1; «B»; FLASH 0; «.»; CHR$ 8; CHR$ 8;: GO TO 10
```

Si se usa GET con una variable numérica (p. e. GET x o GET key) ,entonces la variable será igual a 1 si se teclea «1», hasta 9 si se teclea «9». El valor será 10 para «A», 11 para «B», etc. GET es particularmente útil para programas conducidos por menú (ver ON).

JOIN número de línea

Tecla: Shift 6 (modo gráfico misma tecla que &)

Ver también SPLIT

Une la línea especificada (o, si no se especifica, la línea que señale el apuntador de línea actual), y la de abajo, si existe. La línea siguiente perderá su número de línea, y estará separada de la primera por «:».

JOIN puede usarse después de utilizar SPLIT si deseas trasladar parte de una línea de programa y añadirla a otra.

KEYIN string

Tecla: Shift 4

Inserta un «string» aun cuando se haya escrito desde teclado. Esto permite que los programas sean auto escritos; la totalidad de implicaciones de esto están bajo el radio de acción de este manual (y su autor). Es posible automatizar la producción de sentencias DATA:

```
10 LET a$ = «(X) DATA»: REM usa la palabra clave DATA
20 FOR n = 0 TO 9
30 LET a$ = a$ + STR$ (PEEK n) + «,»
40 NEXT n
50 LET a$ = a$ (1 TO LEN a$ -1): REM quita la última coma
60 KEYIN a$
```

Verás que una nueva línea ha sido añadida al programa después de haberlo ejecutado. (Una línea claramente inútil, ya que da información de los 10 primeros bytes de la ROM, pero ilustra el motivo).

Alguien sentirá el deseo realmente intrépido de usar SCREEN$ y KEYIN para escribir un editor de pantalla en Basic. Nota: Puede ser un poco lento: tendrás que convertir las palabras clave en caracteres simples.

NOTA: KEYIN debe usarse como parte de un programa, no como un comando directo. Esto es debido a que el string especificado es entrado en el ordenador por el usuario; sobreescribirá cualquier comando directo y provocará un mensaje de error.

KEYWORDS 1 o 0

Letra: B

Controla cuándo son utilizados los gráficos definibles (KEYWORDS 0) o las palabras clave Beta Basic (KEYWORDS 1), inicialmente el sistema está en el estado KEYWORDS 1; cuando quieres usar gráficos definibles ejecuta KEYWORDS 0. Esto también te permite permanecer en modo gráfico indefinidamente, mientras que KEYWORDS 1 salta después de que se entra cada palabra clave. Aunque tus listados pueden aparecer un poco confusos, los programas se ejecutarán normalmente en este estado. («KEYWORDS» es la única palabra clave que no puede ser desactivada — el gráfico de esta tecla es en este caso equivalente a «space»).

LIST número de línea TO número de línea

o LLIST número de línea TO número de línea

Esta sintaxis es permitida ahora, como mejora a la versión normal. El bloque específico de líneas será LISTado o LLISTado El primero o el segundo número de línea puede ser omitido. Si el primer número es omitido, entonces se asumirá la primera línea del programa después de la línea 0. Si se omite el segundo número se asumirá la última línea del programa (esto es equivalente a la acción de la sintaxis normal de LIST). Algunos ejemplos:

```
LIST 20 TO 100
LIST TO 200
LLIST 100 TO 180
```

Si el primer número de línea existe, será listado con el cursor « > ». Si no existe se utilizará el siguiente número de línea, pero no aparecerá ningún cursor.

Si el segundo número de línea no existe, el siguiente número de línea, o el final del programa será utilizado.

Si los dos números de línea son iguales sólo se listará una línea.

LOOP

o LOOP WHILE condición

o LOOP UNTIL condición

Tecla: L (WHILE, Tecla J; UNTIL, Tecla K)

Ver también: DO, EXIT IF

Es parte de la estructura DO – LOOP. Lee primero acerca de DO: LOOP por si mismo causa simplemente que la ejecución del programa salte-a la instrucción DO emparejada. Los calificadores WHILE y UNTIL permiten un salto de regreso condicionado - LOOP WHILE (condición) vuelve solamente si la condición especificada es cierta; en caso contrario se ejecutarán las instrucciones siguientes a LOOP. LOOP UNTIL es lo contrario: el bucle no se hará si la condición; especificada es cierta y se hará si es falsa. Obtendrás el mensaje T «LOOP without DO» si usas LOOP sin una instrucción DO emparejada.

ON

Como en: GO TO o GOSUB. ON número; línea no, línea no, línea no

Tecla: O

ON permite acceder mediante GOSUB o GO TO, un número da línea particular de una lista de números de línea, de acuerdo con el valor de la expresión numérica situada inmediatamente después de ON. (La sintaxis más usual es ON número GO TO número

de línea, numero de línea, pero el sistema de teclado del Spectrum hace difícil esto). ON es más flexible que la alternativa usual del Spectrum, p. e.:

```
10 INPUT opción: GO TO opción 100 + 100
```

ya que los números de línea no han de estar, en ninguna secuencia particular:

```
10 INPUT opción: GO TO ON opción;90,135,60,40
20 PRINT «Entra de 1 a 4» GO TO 10
```

En el ejemplo anterior, se irá a la línea 90 si opción 1, a la línea 135 si opción 2, etc. (Si opción es negativo, se usará su valor absoluto). Si no está en el rango de 1 a 4, no se producirá ningún GO TO; en su lugar, el programa continuará con la siguiente instrucción o línea, la cual en este caso fuerza al usuario a probar de nuevo. INPUT puede ser reemplazada por GET en el ejemplo anterior para implementar de una forma mas elegante los programas conducidos por menús.

ON ERROR numero de línea

Tecla: N

Después de la ejecución de este comando, se accederá a la rutina especificada mediante GOSUB si se produce un error. Cualquiera de los informes listados en el manual Sinclair o en el manual Beta Basic aparte del informe 0, «OK» y del informe 9, «STOP statement» producirá el dicho GO SUB. La ayuda es desconectada mediante ON ERROR 0, pero es también desconectada mientras ejecuta la rutina de manipulación de errores, y conectada de nuevo cuando vuelve al programa principal. (¡Una rutina de manipulación de errores que ha sido accedida por si misma puede dar lugar a confusiones!). Tres variables especiales están disponibles para la subrutina; no son palabras clave y deben ser entradas por completo. LINE y STAT son el número de línea y el número de instrucción donde se produce el error, y ERROR se deduce del código del informe del error producido — ver Apéndice C para más detalles. Puedes usar estos nombres de variables, pero los valores serán sobreescritos si se activa ON ERROR o TRACE. Es deseable tener todos los informes arreglados excepto uno o dos normalmente, o podrás confundirte con lo que esté ocurriendo. ¡En cualquier caso, tus programas no sufrirán múltiples errores! Un ejemplo (que será sólo parte de un programa) se da en aquellas series de puntos dibujados con PLOT y la rutina de error simplemente salta aquellos que están fuera de pantalla:

```
100 ON ERROR 5000
110 FOR n = 1 TO 10: lNPUT «coord x»;x;«coord y»;y
120 PLOT x,y: NEXT n

4990 STOP
5000 IF error = 11 AND line = 120 THEN RETURN : ELSE POP: CONTINUE
```

Puntos importantes: La instrucción STOP es un intento de prevenir que la subrutina sea ejecutada accidentalmente. El valor de línea y error son testeados porque «Integer out of range» es claramente un informe muy común. El RETURN de la rutina de manipulación de errores en la línea 5000 volverá a la instrucción siguiente a la que produjo el error — por tanto RETURN es a «NEXT n». Si la línea o el error no está especificado, se usará CONTINUE. Esto hace que la instrucción que produjo el error sea re-ejecutada (a menos que el mensaje sea «BREAK into program» - ver-manual del Spectrum Apéndice B) y puesto que CONTINUE no restaura ON ERROR, esta vez es hecha la respuesta normal al error (El POP quita la dirección de retorno al programa principal — de lo contrario esto desordenaría la pila del ordenador, si no se usa RUN o CLEAR). Un error que ha de ser manipulado de forma algo diferente al resto es «BREAK into program». Ya que ON ERROR se desconecta cuando se accede a la rutina de manipulación de errores, el

resultado normal de apretar BREAK es que el programa para después de la primera instrucción de la subrutina (porque todavía estás presionando BREAK). Por lo tanto, si quieres efectuar alguna acción cuando se pulsa BREAK, deberás introducir un retardo en la primera instrucción de tu subrutina, así tendrás tiempo de dejar de presionar BREAK. BEEP sirve muy bien - puedes usar una casi inaudible frecuencia.

Por ejemplo:

```
100 ON ERROR 5000
110 PRINT «vuelve y»; PAUSE 10: GOTO 110

4990 STOP
5000 IF error 21 THEN BEEP 1,69: BORDER RDN 7: RETURN: ELSE POP:
CONTINUE
```

Este es un ejemplo de ON ERROR usado en operaciones con Microdrive:

```
100 INPUT: «FICHERO A CARGAR ? »; f$
110 ON ERROR 130
120 LOAD «m»;1;f$:G0 TO 140
130 POP: IF error = 61 THEN PRINT «fichero »;f$; « no encontrado»: GO TO 100:
ELSE STOP
140 ON ERROR 0
150 REM resto del programa
```

Date cuenta de que esta rutina de error sólo es utilizada por esta parte del programa y tiene una acción muy específica - previene la rotura del programa Si se hace un intento de cargar un fichero inexistente. ON ERROR es conectado justo antes de la operación de carga y es desconectado después.

PLOT coord X, coord V < ;string >

Tecla: la tecla normal para PLOT (no en modo gráfico)

Como puedes ver por la sintaxis de arriba, Beta Basic te permite «PLOTear» un string, al igual que los usuales pixeles. Las coordenadas de las instrucciones PLOT se refieren a la posición en que la esquina superior izquierda del primer carácter del string va a ser situada. Todos los calificativos usuales del PLOT pueden ser utilizados, como INVERSE, OVER, INK, etc. Si el string se extiende fuera de la parte derecha de la pantalla, será movido hacia la parte izquierda. Si la parte alta de un carácter excede la parte superior o inferior de la pantalla, se producirá un mensaje «Integer out of range». (Es posible sin embargo, para los siete pixeles inferiores de un carácter extenderse dentro del área de edición en la parte inferior de la pantalla.) La posición de plot que utiliza DRAW como posición inicial, no es alterada cuando hacemos un PLOT de un string.

Cambiando las coordenadas apropiadamente para hacer un PLOT de un carácter, puedes conseguir muchos movimientos mejores de los que son posibles con PRINT AT:

```
100 FOR X = 16 TO 224. PLOT X X/2;«<  >»: NEXT X
```

Prueba añadiendo STEP 2, 3 o más a la sentencia FOR para mayor velocidad y un efecto diferente. Puesto que « < > ». tiene un borde como mínimo de un pixel de amplitud, borrará automáticamente su posición previa si es movido de pixel en pixel. Algunas letras como la «T» tienen pixeles en INK que se extienden hasta el borde del cuadro de carácter. Esto significa que pueden dejar un rastro si se mueven en ciertas direcciones a menos que explícitamente se borre con un sobre PLOT de la posición anterior. Si estás diseñando tus propios caracteres, es una buena idea dejar alrededor un borde a PAPER de un pixel.

Los códigos de control de cursor CHR$8-CHR$11 pueden incluirse en cadenas a «PLOTear» para darles complejas formas. Ver CÓDIGOS DE CONTROL DEL CURSOR para más detalles.

La posibilidad de hacer PLOTs de strings es útil para etiquetar gráficos y diagramas ya que además de su gran precisión, es conveniente el uso de un sistema de coordenadas común. Es muy simple etiquetar los ejes de una gráfica directamente opuestos a las marcas indicadoras.

El siguiente ejemplo enseña a utilizar el comando PLOT para proporcionar una capacidad de superescritura y subescritura en el Spectrum.

```
100 LET a$ = «Saber cuántas moléculas tiene cm + 3 - NA -2 + PO-4 + ?»
110 LET x = 0: LET y = 160
120 FOR c = 1 TO LEN a$
130 IF a$(c) = «-» THEN LET y = y-3: NEXT c
140 IF a$(c) -= «+» THEN LET y = y+3: NEXT c
150 PLOT x,y; a$(c): LET x = x +8
160 IF x > = 248 THEN LET x = 0: LET y = y-12
170 NEXT c
```

El programa usa « + » y « - » como códigos de control no-imprimibles que mueven la posición de plot arriba o abajo tres pixels. A menudo se desearán códigos de control más inusuales,-para prevenir operaciones accidentales). Prueba jugando con la línea 150 del ejemplo para que X sea incrementado por más o por menos de 8 espacios (esto variará el espaciado). Puedes querer utilizar PLOT OVER 1 con un espaciado muy limitado (esto evita que los bordes colocados a PAPER de cada letra se impriman sobre parte de las letras anteriores). Un espaciado de 5 pixels por carácter da 51 caracteres por línea moderadamente legibles.

Una sugerencia para los entusiastas como tú; usa PLOT para proporcionar una justificación de texto proporcional espaciada por la derecha en un programa de tratamiento de texto. Necesitarás colocar espacios en el texto y alargar o contraerlos (o el texto entero) así cada línea será exactamente llenada con un número entero de palabras.

POKE dirección, string

(palabra clave normal)

Beta Basic te permite hacer un POKE con strings al igual que con números, lo que en combinación con la función MEMORY$ (ver suplemento) permite la manipulación rápida de grandes áreas de memoria. (Quizás hay que decir que si tu puedes romper un ordenador con un POKE mal interpretado, esto se hace muchas veces superior si haces POKEs con largos strings).

Vamos a hacer un POKE con un string en el que podremos ver algunos efectos:

```
10 LET pantalla = 16384
20 POKE pantalla, STRING$(6144, «U»
```

(STRING$ es explicado en la sección de funciones). Hemos llenado la memoria de pantalla con «U»s, que en su posición no son solamente «U»s, sino datos. Ya que «U» es «01010101»en binario, el carácter es «destripado». El siguiente ejemplo copia el principio de la ROM en el fichero de atributos, lo que produce algunos efectos de color.

```
30 LET atributos = 22528
40 POKE atributos, MEMORY$()(1 TO 704)
```

Ahora escribiremos algo muy simple para colocar una figura en la pantalla, almacenarla en su string y luego colocarla de nuevo:

```
10 CIRCLE 128,88,70
20 FILL 128,88
30 LET a$ = MEMORY$ () (16384 TO 23295): REM pantalla completa
40 CLS: PRINT «Pulsa alguna tecla»: PAUSE 0
50 POKE 16384, a$
```

La memoria del Spectrum almacenará varios dibujos, permitiéndote cambiarlos o mostrarlos en secuencia. Para más dibujos, un tercio de la pantalla puede ser leído convenientemente en un string, si todo lo que se quiere es la información de los caracteres (también puedes añadir un tercio del fichero de atributos al mismo string o a otro, por supuesto). Para almacenar los tres tercios de la pantalla en Springs, usa una de estas:

```
SUPERIOR: LET a$ = MEMORY$()(16384 TO 18431)
MEDIA: LET a$ = MEMORY$()(18432 TO 20479)
INFERIOR: LET a$ = MEMORY$()(20480 TO 22527)
```

Hay suficiente memoria para hacer una caricatura razonable usando POKEs secuencialmente de los distintos Springs. Puedes usar un array (p. e. DIM a$(10,2048)) para almacenar la información.

Por supuesto, hay otros muchos usos potenciales además de jugar con la memoria de pantalla. Podemos limpiar (CLEAR) una gran área de memoria, y almacenar allí un programa completo:

Primero, CLEAR 33900; después ejecuta esto:

```
10 POKE 34000, MEMORY$()(23552 TO 33800)
20 REM resto del largo programa
```

Ahora puedes usar NEW, después devuelve el programa a su sitio con:

```
POKE 23552, MEMORY$()(34000 TO 44248)
```

Lo anterior podría esconder de NEW en una tecla definida por el usuario (ver suplemento) la cual es definida después de la instrucción CLEAR y antes de que el programa se ejecute:

```
DEF KEY «j»; POKE 23552, MEMORY$()(34000 TO 44248)
```

Cuando el programa es devuelto a su sitio, continuará ejecutándose donde lo dejó, ya que todas las variables del sistema fueron salvadas, al igual que las variables normales. Es ligeramente más difícil coordinar el trueque de programas; necesitarás disponer de dos áreas de almacenamiento, así como espacio para ejecutar un programa.

Una última idea para el POKE: salvar código máquina con un programa Basic, asignado a un string usando MEMORY$, y grabar el programa Basic incluyendo el string de forma que en la auto-ejecución un POKE mueva el código a su lugar al cargarlo. Esto es considerablemente más rápido que la carga separada del código.

Nota: No serás capaz de utilizar CLEAR sin la pérdida del string, por lo tanto usa CLEAR antes de LOAD, o evita la sobreescritura de algo importante. Podrás hacer un POKE a la pantalla o al buffer de la impresora sin problemas.

POP variable numérica

Tecla: Q

Saca una dirección de la pila de GOSUB/DO-LOOP/PROC. El número de línea indicado es asignado a una variable si ésta se utiliza. Esta palabra hace posible saltar fuera de las subrutinas, DO-LOOPs y DEF PROCs sin desordenar la pila con direcciones inutilizadas.

POP usado solo, simplemente desecha el valor, pero POP loc (por ejemplo), sirve para disponer del valor en la variable «loc». La subrutina o procedimiento sabe desde dónde fue llamada. Si te arrepientes del POP es todavía posible hacer algo parecido a RETURN por GO TO loc + 1 (esto no es idéntico a RETURN ya que no puedes especificar el número de instrucción).

```
100 GOSUB 500
110 STOP
500 POP loc
510 PRINT« Subrutina llamada desde la línea »;loc
520 GO TO loc + 1
```

Si reemplazas la línea 520 por:

```
520 RETURN
```

obtendrás «RETURN without GOSUB» ya que la dirección requerida de RETURN ya no está presente en la pila.

El uso de POP cuando no hay datos en la pila para extraer dará el informe V «No POP data».

PROC nombre

Tecla: 2 (la misma que FN)

También se trata aquí: DEF PROC, END PROC

PROC es una abreviación de procedimiento (procedure). Estos son algo parecidos a los llamados GOSUBs, pero tienen la ventaja de que no tienes que preocuparte del lugar del programa donde esté la definición del procedimiento; el ordenador lo buscará por todo el listado, previniendo que el DEF PROC sea la primera instrucción de una línea (esta restricción acelera un poco las cosas). El nombre del PROCedimiento puede ser cualquier nombre válido de variable — los espacios y el estado de las letras no importan. La DEFinición del PROCedimiento puede ser de cualquier número de líneas de longitud; se termina en cualquier punto por medio de END PROC. De forma similar a DEF FN, las definiciones de procedimientos se ignoran si no son llamadas por PROC — la ejecución del programa simplemente se lo salta. (El ordenador no es capaz de hacer esto de forma correcta si has usado múltiples END PROCs con una sola DEF PROC; si quieres hacer esto, tendrás que saltarte las DEF PROC tú mismo). Todas las variables usadas por el programa principal son utilizables por el procedimiento, y cualquier variable que el procedimiento crea o altera son utilizables por el programa principal.

```
10 FOR n = 1 TO 20: PROC dibujar cuadro: NEXT n
200 DEF PROC dibujar cuadro: LET lado = RND * 20 + 20
210 PLOT RND * 215, RND * 135: DRAW lado,0: DRAW 0,lado
220 DRAW -lado,0: DRAW 0, -lado: END PROC
230 PRINT «Se acabó»
```

Un programa idealmente estructurado de gran potencia consiste en una serie de procedimientos definidos, cada uno de los cuales hace un trabajo particular y puede ser probado separadamente. Estos son llamados por el cuerpo principal del programa, el cual puede ser algo así:

```
100 PROC colocar tablero
110 PROC jugar
120 PROC enseñar puntuación: STOP
```

Si no has definido los procedimientos, usando PROC se producirá el informe W, «Missing DEF PROC»; esto también pasará normalmente si usas END PROC sin DEF PROC. (La

excepción es si hay una dirección de RETURN disponible en la pila END PROC no puede decir que no es la dirección de una PROC, y volverá felizmente a la línea indicada). Si olvidas incluir un END PROC, el programa principal dará el informe X, «No END PROC», cuando intente salir de la definición del procedimiento y no pueda encontrar dónde acaba.

RENUM < inicio TO final > < LINE nuevo inicio > < STEP intervalo >

Tecla: 4

RENUM usado por sí solo renumera el programa entero tomando la línea 10 como nueva primea línea y 10 como intervalo entre líneas. Se puede especificar la renumeración de un bloque de líneas con un tipo de expresión «recortado» siguiendo a RENUM:

> RENUM (130 TO 220) - renumera el bloque especificado.
> RENUM (130 TO) renumera la línea 130 y todas las siguientes líneas.
> RENUM (TO 100) renumera todas las líneas hasta la 100 inclusive, excepto la línea 0

si es imposible renumerar el bloque que se requiere sin que alguno de los nuevos números de línea se extiendan en el rango de números de línea ya utilizados por números de línea fuera del bloque, o si el orden de las líneas fuera cambiado, obtendrás el mensaje G, «No room for line» (el cual en este contexto no tiene el significado dado en el manual Sinclair). Un nuevo número de línea de comienzo distinto del valor por defecto 10 puede darse usando LINE (la palabra clave normal). Un incremento del valor entre líneas distinto de 10 puede darse por medio de STEP (la palabra clave). El bloque específico, LINE y STEP puede ser incluido todo u omitido como se requiera, pero deben usarse en el orden especificado. Ejemplos:

> RENUM
> RENUM LINE 100 STEP 20
> RENUM (1540 TO) LINE 2000
> RENUM (100 TO 176) LINE 230 STEP 5

El programa entero se revisa antes que nada para las expresiones que necesitan forzosamente renumeración p. e. «GO TO L * 100». Estas expresiones podrían referirse a una línea que va a ser renumerada; sin embargo, son muy problemáticas para cualquier rutina normal de renumeración, por tanto RENUM frustra el intento con el mensaje Y, «Too hard» y da la línea y número de sentencia de la expresión. Esto te da la oportunidad de alterar la expresión, quizás utilizando «ON» (p. e.). La línea problemática será la línea actual por tanto sólo pulsando EDIT te será mostrada; la línea puede alterarse temporalmente para realizar un RENUM con éxito; por ejemplo, la sentencia que contenga la expresión problemática puede ser temporalmente encerrada entre comillas — p. e. PRINT «GO TO L * 100». Una vez que todas las expresiones han sido alteradas o extraídas, re-entra el comando RENUM. Las referencias al bloque renumerado en cualquier parte del programa son alteradas como se requiere, incluyendo GOTO, GOSUB, RESTORE, RUN, ON, ON ERROR, TRACE, LIST, LLIST, LINE y DELETE. CLOCK no es modificado — ¡tendrás que hacerlo tú mismo! (Esto es debido a que CLOCK puede ir seguido por un número de línea o un modo de selección).

Nota: RENUM crea una tabla de datos en la pantalla de memoria — esto es limpiado tan pronto como la renumeración se haya completado, y no es motivo de alarma.

ROLL código de dirección < ,pixels > <;x,y; anchura, longitud >

Tecla: R

Ver también: SCROLL

ROLL mueve la pantalla entera o una ventana de pantalla definida, arriba, abajo, izquierda o derecha. Cualquier cosa movida fuera de los limites de la ventana ROLL reaparecerá en la parte opuesta de la pantalla. En otras palabras, el comando no destruye nada de lo que hay en la pantalla, simplemente la reordena (no como SCROLL).

Como puedes ver al principio de la página, ROLL puede tener una compleja sintaxis. Sin embargo, la mayoría de ésta se necesita sólo para definir el área a girar. Si quieres mover la pantalla completa un pixel, puedes usar simplemente:

```
10 ROLL código de dirección
```

Para un ROLL de la información de caracteres solamente, la dirección de ROLL se especifica usando 5, 6, 7 u 8 después del comando. Las flechas de dirección están impresas en el teclado — 5 es izquierda, 6 es abajo, 7 es arriba y 8 es derecha. Debido a que cada uso de ROLL sólo causa un pequeño movimiento, el comando se suele utilizar en un bucle. Dibuja un gran gráfico, o lista un programa, después prueba:

```
100 FOR d = 5 TO 8: FOR p = 1 TO 100
110 ROLL d
120 NEXT p: NEXT d: STOP
```

El ROLL diagonal se puede conseguir, por ejemplo, moviendo repetidamente un pixel a la izquierda, y luego uno arriba. Para mayor rapidez en el movimiento (con menos suavidad), cambia la línea 110 a:

```
110 ROLL d,4
```

la cual moverá 4 pixels a la vez. Este número de «pixels a mover» debe ser menor de 256 para movimiento horizontal, o menor de 177 para movimiento vertical. Si no se especifica, se asume que es uno. Obviamente, cuanto mayor sea el número, más lejos moverá la pantalla el bucle FOR-NEXT. En dirección vertical, la velocidad del movimiento es directamente proporcional al número de pixels movidos a la vez. En dirección horizontal, los movimientos de 4 y 8 pixels darán las mejores velocidades, ya que emplean las instrucciones de movimiento de medio byte y un byte que posee el chip Z-80.

Para mover la información de atributos (color) con la información de caracteres, añade 4 al código de dirección; para mover sólo los atributos, resta 4. (Ver tabla en pag. siguiente).

Los atributos pueden moverse sólo de 8 en 8 pixels (el número de pixels a mover es ignorado). Cuando los códigos de dirección 9-12 son usados para mover los atributos y los datos de caracteres al mismo tiempo, los resultados serán más satisfactorios si los pixels a mover son 8; en otro caso los atributos y los caracteres perderán la alineación.

CÓDIGO DE DIRECCION	DIRECCIÓN	APLICADO A
1	IZQUIERDA	ATRIBUTOS
2	ABAJO	ATRIBUTOS
3	ARRIBA	ATRIBUTOS
4	DERECHA	ATRIBUTOS
5	IZQUIERDA	CARACTERES
6	ABAJO	CARACTERES
7	ARRIBA	CARACTERES
8	DERECHA	CARACTERES
9	IZQUIERDA	AMBOS
10	ABAJO	AMBOS
11	ARRIBA	AMBOS
12	DERECHA	AMBOS

Una ventana específica de la pantalla puede ser «enROLLada» siguiendo el código de dirección con cuatro parámetros: las coordenadas X e Y de la esquina superior izquierda de la ventana (el mismo sistema de coordenadas que PLOT y DRAW), después la anchura de la ventana en caracteres (no pixels — esto es posible pero costaría mucha memoria o rapidez) después la longitud de la ventana en pixels. La anchura puede ser 1-32, y la longitud 1-176. La especificación de la ventana para movimiento de atributos sólo puede ser acatada con una precisión de cuadros individuales de carácter, no pixels.

ROLL puede ser muy efectivo en juegos para el perfeccionamiento de movimientos planos de jugadores o de fondo — «Frogger» puede ser fácil. Se pueden obtener efectos extremadamente interesantes — aunque ligeramente antipáticos — efectos activando varias ventanas ROLL sobrepuestas para que la pantalla sea retorcida y fragmentada en complicados caracteres.

La rutina de debajo, fragmenta sus propios listados (o cualquier cosa de la pantalla) crea algunos efectos interesantes.

```
100 LIST: LIST: LIST
110 LET pixels = 4
120 ROLL 5,pixels;0,175;32,88
130 ROLL 6,pixels;0,175; 16,176
140 ROLL 8,pixels;0,87;32,88
150 ROLL 7,pixels; 128,175; 16,176
160 GO TO 120
```

Prueba usando pixels = 1 u otros valores, o cambia la línea 100 a:

```
100 KEYWORDS 0: PRINT STRING$(704,« END PROC »): KEYWORDS 1
```

Otro ejemplo.

```
200 FOR N = 1 TO 7: LIST: NEXT N
210 FOR L = 1 TO 175: ROLL 5;0,175;32,L. NEXT L
```

SCROLL < dirección > < , pixels > < ;x,y; anchura, longitud >

Tecla: S

Ver también: ROLL

SCROLL tiene una sintaxis muy similar a ROLL, la cual habrá sido leída primero. Una diferencia es que SCROLL puede usarse solo, en cuyo caso simplemente mueve la pantalla entera una línea arriba (como el ZX81). Si SCROLL va seguido por 5, 6, 7 u 8, la pantalla entera se moverá un pixel en la dirección indicada por la flecha de la tecla. Cualquier elemento colocado fuera de los límites de la pantalla será destruido; la nueva pantalla que está siendo “movida” quedará en blanco por el lado contrario. Un área dada de pantalla puede moverse siguiendo el código de dirección con las coordenadas X e Y de la esquina superior izquierda de la ventana deseada (en el mismo formato que PLOT y DRAW) y la anchura de la ventana (en posiciones de carácter) y su longitud (en pixels). Ambos, SCROLL y ROLL son de mucha ayuda para la creación de juegos y en interesantes displays de gráficos. Prueba los ejemplos dados para ROLL con SCROLL y contrata los resultados. Aquí tienes un programa que acepta la entrada de un string y lo mueve a través de la pantalla:

```
100 LET A$ = «UNA BONITA Y LARGA CADENA…»
110 FOR C = 1 TO LEN A$
120 PRINT AT 10,31; INK 7; A$(C)
130 FOR P = 1 TO 8: SCROLL 5; 0,95; 32,8: NEXT P
140 NEXT C
```

```
150 FOR P = 1 TO 255: SCROLL 5; 0,95; 32,8: NEXT P
```

El string imprime carácter a carácter en la misma posición de la pantalla en tinta blanca (o INK del mismo color que el papel). Esto hace que cada letra aparezca suavemente ya que es "SCROLLada" fuera de la posición con "tinta invisible". El bucle FOR-next inferior mueve los caracteres una posición a la izquierda antes de que se imprima el siguiente carácter usando una ventana de cobertura de sólo una línea de caracteres. La línea 150 mueve el string fuera de la pantalla una vez que ha sido impresa en su totalidad; alternativamente podrías añadir 32 blancos como rastro. (Si encuentras confusa la mezcla de sistemas de coordenadas de PRINT AT y PLOT, puedes usar el PLOT mejorado de Beta Basic y reemplazar la línea 120 por:

```
120 PLOT 248,95; a$(c)
```

El mismo principio puede ser utilizado para embellecer la presentación de textos (p.e. instrucciones de programas).

```
200 DATA «HACE MUCHO, EN UNA LEJANA GALAXIA…»
210 DATA «MUCHO MÁS TEXTO. MUCHO MÁS…»
300 FOR L = 1 TO 2: READ A$
310 PRINT AT 21,0; INK 7; A$
320 FOR P = 1 TO 8: SCROLL 7: NEXT P: NEXT L
330 FOR P = 1 TO 176: SCROLL 7: NEXT P
```

SORT

o SORT INVERSE array de strings o array numérico o string

Tecla: M

SORT ordena strings, números o letras en orden ascendente o descendente. Su uso con arrays de strings será explicado en primer lugar. Aquí hay un programa para generar un array de 100 strings aleatorios de 10 letras. (Acelerarás el proceso usando la función RNDM de Beta Basic mejor que la RND).

```
100 DIM A$(100,10)
110 FOR S = 1 TO 100: FOR L = 1 TO 10
120 LET A$(S,L) = CHR$(RND * 25 + 65)
130 NEXT L: NEXT S: GOTO 200
140 SORT A$
200 FOR S = 1 TO 100: PRINT A$(S): NEXT S
```

Tan pronto como el array haya sido creado —lo que tardará un rato— saldrá impreso para ti. Ahora GO TO 140 y el array será ordenado y de nuevo se imprimirá (evita RUN o perderás tu array). El tiempo empleado para ordenar el array va sobre un quinto de segundo; este tiempo se incrementará relativamente poco si usas cadenas más largas. El número de strings es más importante — 200 tardará 0.7 seg. y 400 alrededor de 3 segundos.

Los strings son ordenados de acuerdo a sus códigos — el SORT normal tiene «space» antes que «A», la cual está antes que «a» — ver la lista del Apéndice A del manual Sinclair. Si la línea 140 se cambia a SORT INVERSE A$ el orden será invertido, ¡pruébalo! Podemos seleccionar cualquier bloque dado de strings para ordenarlo con una expresión:

```
SORT A$ (1 TO 20)
```

ordenará sólo los 20 primeros strings y

```
SORT A$ (30 TO )
```

clasificará todos los strings desde el 30 en adelante. También es posible ordenar de acuerdo a partes particulares de los strings:

SORT A$()(2 TO)

clasificará el array entero sobre la base de la segunda y subsiguientes letras de cada string - la primera letra no se tiene en cuenta en el cómputo, aunque es movido con el resto del string. (Nota que hemos tenido que usar dos paréntesis aún cuando hemos querido ordenar el array completo. Esto se debe a que SORT cuenta con el segundo de los dos paréntesis para especificar la parte del string a considerar).

SORT hace muy fácil el desarrollo de una rápida y flexible base de datos. En este contexto, es común llamar al array «fichero» y a sus elementos «registros». Áreas de cada string serán posiblemente reservadas para tipos particulares de información, y serán llamados «campos». A menudo querrás utilizar strings de longitud variable; un fichero de nombres y direcciones y otros datos, digamos edad —, puede ser creado como los 20 primeros caracteres de cada registro (string) sean el nombre, los 20 siguientes la dirección y los últimos su edad. Ya que la edad es limitada estará en el rango de 0 a 255, y podemos usar algo como:

```
LET A$(S,41) = CHR$ edad
```

si deseas situar la edad en el registro S. Tal almacenamiento de números es simple y ahorra memoria, pero suponte que necesitamos almacenar algo más complejo, como un balance de banco. Si usas:

```
LET A$(S,41 TO 46) = STR$ balance
```

la información será almacenada en el string, pero quedará justificada por la izquierda (el «9» de un balance de 9$ quedará en la posición 41, como el «1» de un balance de 100$). Esto impedirá que SORT trabaje correctamente en este campo. La pregunta consiste en formatear cada cosa limpiamente para que todos los puntos decimales queden alineados, y que tengamos todas las unidades, decenas y centenas en la misma posición para cada string. Esto es muy simple con la función de formateo, USING$:

```
LET A$(S,41 TO 46) = USING$(«000.00», balance)
```

(Ver USING$ y USING para una completa descripción de esta función, y CHAR$ para información sobre enteros codificados en strings). Ahora puedes hacer cosas como ordenar un conjunto de registros basándote en la edad, o en el balance del banco, y luego ordenar los 20 primeros por orden alfabético. Fíjate que ya que el código de «1» viene primero que el de «2», SORT dará primero los números más pequeños, y SORT INVERSE dará primero los números mayores, cuando los números son representados como strings de esta forma. Necesitarás tener cuidado con la distribución en campos de tus datos — p. e. asegúrate de que la primera letra de todos los apellidos empiece en la misma posición en cada string.

SORT también trabajará con strings normales y arrays unidimensionales de strings:

```
INPUT s$: SORT s$: PRINT s$
```

dará « BFdegglors» si se ha entrado «Fred Bloggs». Esto no parece muy útil, pero permite a SORT trabajar con ciertas clases de datos numéricos que pueden ser almacenados más eficientemente en strings usando, por ejemplo:

```
LET S$(posición) = CHR$(dato)
```

También trabaja con arrays convencionales numéricos de una o dos dimensiones, usando la misma sintaxis que para arrays de strings. Un array bidimensional numérico puede ser

considerado como una tabla en la cual la primera dimensión son las filas y la segunda las columnas:

SORT B(1 TO 20) (2)

clasificará las primeras 20 filas del array B tomando como base los números de la segunda columna. (Serán movidas filas completas). Fíjate que siempre necesitamos usar como mínimo un paréntesis en un SORT numérico para distinguir el array B() de la variable simple B. El segundo paréntesis, si se usa, debe tener siempre una longitud de uno — no como en el caso de strings. El ordenamiento de números es aproximadamente cuatro veces más lento que el de strings, porque SORT ha de revisar dos formatos diferentes de números (ver página 170 del manual Spectrum) y por la posibilidad de incorporar números positivos y negativos.

Ya que no es una clasificación alfabética, sino numérica, los números mayores vendrán en primer lugar a menos que se use sort inverse.

SPLIT - no es una palabra clave. Actualmente entrarás « < > »

Tecla: symbol-shift W (no modo gráfico)

Si una línea que has editado o que acabas de escribir es entrada con « < > » como primer carácter de alguna instrucción, sólo la parte de la línea anterior a será puesta en el listado. El resto quedará en la línea editada en la parte, inferior de la pantalla. El signo « < >» será extraído y reemplazado por una copia del número de línea original. El cursor estará justo a la derecha del número de línea, listo para que la alteres antes de presionar enter (a menos que desees sobreescribir la primera parte de la línea en el listado). Si entras:

10 PRINT «hola»: GO TO 10: <> PRINT «adiós»

Entonces

10 PRINT «hola»: GO TO 10

aparecerá en el listado, y

10 (cursor)PRINT «adiós»

quedará en la parte baja de la pantalla. Puedes mover esta parte de la línea a cualquier lugar-del programa cambiando el número de línea, o también añadirla a una línea ya existente usando JOIN.

TRACE número de línea

Tecla: T

Esta es una posibilidad que permite la depuración de programas Basic imprimiendo la línea actual, instrucción y variables seleccionadas, y velocidad reducida o paso simple. TRACE provoca que la subrutina especificada por el comando sea accedida por GOSUB inmediatamente antes de que cada sentencia sea ejecutada. La subrutina dispone de las variables especiales (no palabras claves) LINE y STAT que son el número de línea y de sentencia de la parte del programa que va a ser ejecutada. TRACE es desconectada durante la ejecución de la subrutina pero es restablecida por el RETURN con el que termina. El contenido de la subrutina depende de ti - un simple ejemplo podría ser:

9000 PRINT INVERSE 1;line;«:»;stat: RETURN

Añade esta rutina a un programa e inserta la instrucción:

TRACE 9000

en el programa en el punto donde quieras empezar la depuración. Si quieres desconectar TRACE en algún punto del programa inserta

TRACE 0

La utilización de la rutina anterior producirá el listado de las instrucciones según sean ejecutadas, en el formato utilizado por los «informes». INVERSE se usa para distinguir esta salida de alguna propia que pueda efectuar el programa. Para ver el texto de la línea que está siendo ejecutada, incluye en la rutina:

LIST line TO line

o LIST line - 1 TO line

(La versión siguiente evitará los cursores de línea actual en todos sitios, siempre que hayas usado números de línea separados por más de 1.) RUN, CLEAR o TRACE 0 desconecta la utilidad. Si deseas añadir un simple intervalo o enlentecimiento, usa PAUSE u otros métodos de tu agrado. Las variables también se pueden imprimir — si haces esto será mejor declarar al principio todas las variables prontamente o podrás encontrarte con el mensaje «Variable not found». Para evitar la confusión de las salidas de TRACE con las del programa, se puede usar PRINT AT, pero será necesario salvar la posición de impresión actual o la impresión del programa principal estará interferido con:

```
9000 LET col = PEEK 23688: LET fila = PEEK 23689
9010 PRINT AT 0,0;line;«:»;stat;«      »,«a$ =   »;a$;«      »
9020 POKE 23688, col: POKE 23689, fila: RETURN
```

La posición del programa y la variable a$ se imprimen después de cada sentencia, en la parte superior de la pantalla. (Arrastrando espacios para asegurarse de que los valores anteriores son sobreescritos). La subrutina salva las variables del sistema importantes por lo que la posición de impresión sólo es alterada temporalmente por la instrucción PRINT AT.

Tu rutina TRACE favorita puede ser convenientemente asignada a una tecla definible (DEF KEY) y grabada junto a Beta Basic. Después simplemente escribe un número de línea y usa la tecla definida.

UNTIL condición

como en: LOOP UNTIL condición

o: DO UNTIL condición

Permite la ejecución condicional de DO y LOOP - ver estos comandos para más detalles.

USING

Tecla: U

Como en: PRINT USING string formato; número

También discutida aquí: función USING$

Ambas USING y USING$ permiten la especificación del formato de los números a imprimir. USING sólo puede ser usado como un modificador de números impresos por medio de PRINT o LPRINT, mientras que USING$ es una función. Todas las aplicaciones de USING pueden ser reemplazadas por USING$ si se desea. USING$ devuelve el string que PRINT USING imprimiría; esto permite formatear los números para usar con LET, PRINT y otros comandos que pueden aplicarse a un string. Ya sea con USING o USING$, el formato deseado se especifica por medio de un string en que los signos (#) se quedarán como espacios, los ceros quedarán como ceros, y también pueden usarse para mostrar el número de dígitos después del punto decimal:

```
100 FOR n = 1 TO 20: LET x = RND * 100
110 PRINT x, USING «# # #.# #»;x: NEXT n
```

Notarás lo limpio que quedan los números formateados. Comparando las dos columnas, también verás que USING redondea al número más cercano. Experimenta con diferentes strings para formatear (puedes usar una variable string si lo deseas). Algunos formatos posibles, y su salida (con los espacios representados por «s») para el número 12.3456 son:

« # #. # »	12.3
« # # # # . # # »	ss12.35
«000.00»	012.35
«00»	12
«$00.00»	$12.35
«0. 00»	%..3

El penúltimo ejemplo muestra que es posible incluir al principio caracteres en el string de formato distintos de « # » y «0». El último ejemplo demuestra una salida con el signo «%» que indica overflow (desbordamiento) del formato especificado.

Nota: USING no trabaja con notación científica.

La función USING$ (entrada como FN U$) se usa de la misma forma pero en vez de:

```
PRINT USING a$; número
```

usaremos

```
PRINT USING$(a$. número)
```

WHILE condición

Tecla: J

Permite la ejecución condicional do DO y LOOP ver estos comandos para más detalles.

XOS, XRG, YOS y YRG

Ver también: Apéndice D

Estas cuatro palabras no son palabras clave - son una clase especial de variable que permite el cambio de escala y de origen usado por PLOT, DRAW, CIRCLE y FILL. XOS se encarga de la medición del eje X (del origen), YOS es lo mismo para el eje Y, y XRG y YRG se encargan del rango de los ejes X e Y respectivamente. Una forma en la que estas variables son especiales es que son asignadas a unos valores particulares por CLEAR y RUN, en vez de ser borradas. Haz CLEAR, después PRINT XOS (o xos) y obtendrás «0», no «variable not found». Las dos mediciones tienen el valor 0 (es decir no se le ha añadido nada al origen normal) excepto si utilizas LET para asignarle un valor diferente. RUN o CLEAR los devolverá a cero. Cambiar la posición de origen es más conveniente que alterar varias instrucciones PLOT — por ejemplo:

```
LET XOS = 128: LET YOS = 88
```

moverá el origen al centro de la pantalla y te permitirá hacer un PLOT de cualquier valor para X entre -128 y 127, y cualquier valor para Y entre -68 y 87.

XRG normalmente tiene el valor 256 (puedes hacer PLOT a 256 sitios distintos en el eje X) y YRG es normalmente 176 Cambiando XRG y YRG altera la escala en la que PLOT y DRAW trabajaran:

```
10 GOSUB 100: REM normal
20 LET XRG = 128: GOSUB 100
30 LET YRG = 88: GOSUB 100
40 LET XRG = 256: GOSUB 100: STOP

100 CLS: PLOT 0,0: DRAW 50,0: DRAW 0,50
110 DRAW 50,0: DRAW 0,50: PAUSE 100: RETURN
```

El cuadrado dibujado por la subrutina es al principio normal, luego es desviado a través del eje X, después a través de ambos ejes X e Y, y finalmente a través sólo del eje Y según son escaladas las instrucciones DRAW de acuerdo a los rangos especificados. El siguiente ejemplo muestra el seno de una ola siendo dibujado utilizando ambos cambios de mediciones y rangos. (Utiliza la función del Beta Basic SINE para mayor rapidez).

```
100 LET XRG = 2*PI: REM 360 grados
110 LET YRG = 2.2: REM seno varía entre -1 y 1
120 LET YOS = 1.1: REM origen de ordenadas en el centro de la parte superior de
pantalla
130 FOR n = 0 TO 2*PI STEP 2*PI/256
140 PLOT n, SIN n: NEXT n
```

Nota que el valor de coordenadas es escalado de acuerdo con el rango dado.

El punto final cuando es dibujada una curva será situado exactamente sobre el sistema alterado de coordenadas creado por el uso de estas variables especiales; sin embargo, la curva misma siempre será parte de un círculo y no es necesario desviar al original el nuevo sistema de coordenadas. De forma parecida, a pesar de que el centro de un círculo será exactamente situado, el radio no es escalado por XRG o YRG; no es posible producir círculos retorcidos cambiando XRG o YRG.

FUNCIONES

GENERAL

Más de 20 funciones han sido añadidas al Beta Basic usando las definiciones de función de la línea 0 (que no es normalmente visible en el listado) para apuntar al código máquina superior de la memoria que hace el trabajo práctico. Aunque las funciones se entran como funciones definidas por el usuario normales, aparecerán como palabras claves una vez entradas; por ejemplo:

```
si tecleas:     PRINT FN s$
obtendrás       PRINT STRING$
```

tan pronto como sea entrado el signo «$». El cursor actuará no obstante como que STRING$ es una palabra clave - una simple presión de tecla y será sobrepasada. Las funciones definidas por el usuario que no sean parte del Beta Basic actuarán normalmente.

Las nuevas funciones no funcionarán (de hecho, fracasarán) si se intenta utilizarlas cuando la parte principal (código máquina) del Beta Basic no reside en memoria. Por otro lado, si la línea 0 no está presente, simplemente obtendrás «FN without DEF» si intentas usarlas. En este caso el resto del Beta Basic todavía trabajará, pero habrás perdido la capacidad de utilizar las nuevas funciones. Al salvar un programa también salvará la línea 0, por eso, si cargas un programa escrito bajo Beta Basic, la línea 0 estará presente. Sin embargo, cargar un programa que no ha sido escrito bajo Beta Basic borrará la línea 0 —

para evitar esto, utiliza NEW para librarte de algún programa que puedes tener en tu ordenador (esto dejará la línea 0 intacta) y haz MERGE del nuevo programa (reteniendo la línea 0). Es posible extraer la línea 0 de un programa usando DELETE 0 TO 0.

Abajo tienes un sumario de las funciones, de cómo aparecen y de cómo se entran. Son dadas en orden alfabético de la forma en que aparecen. En las páginas siguientes cada nueva función se describe con detalle.

DISPLAY	ENTRADA
AND	FN A(
BIN$	FN B$
CHAR$	FN C$
COSE	FN C(
DEC	FN D(
DPEEK	FN P(
FILLED	FN F(
HEX$	FN H$
INSTRING	FN I(
MEM	FN M(
MEMORYS	FN M$
MOD	FN V(
NUMBER	FN N(
OR	FN O(
RNDM	FN R(
SCRN$	FN K$
SINE	FN S(
STRING$	FN S$
TIME$	FN S$
USINGS	FN U$
XOR	FN X(

AND (número, número)

FN A (número, número)

Esta función se deletrea de la misma forma que la palabra clave normal, pero se puede distinguir en un listado de programa por la distinta sintaxis utilizada. Da un AND bit a bit de dos números, los cuales deben estar entre 0 y 65535. Sólo si un bit particular es un «1» en el primer número y (AND) en el segundo, el mismo bit del resultado será «1». Un «0» en cualquiera de los dos números dará un «0» en esa posición del resultado. La nueva función BIN$ es de una gran ayuda para comprender lo que sucede.

BIN$(254)	«11111110»
BIN$(120)	«01111000»
BIN$(AND(254,120))	«01111000»

Puedes utilizar AND para «desenmascarar» bits no deseados, por ejemplo:

```
PRINT AND (BIN 00000111 ATTR (línea , columna))
```

dará el color de INK para la posición (línea, columna) desenmascarando los otros bits (Podríamos haber usado «7» en lugar de «00000111». El ejemplo siguiente imprimirá «Bang!» si se aprieta «B»,«aunque se aprieten otras teclas al mismo tiempo. (Ver página 160 del manual del Spectrum para información del teclado considerado como series de ports).

```
10 IF AND(BIN 00001000, IN 65022) = 0 THEN PRINT «Bang! »;
20 GOTO 10
```

BIN$ (número)

FN B$ (número)

Da el equivalente binario de «número» como un string de ocho caracteres (si «número» es menor que 256) o como un string de dieciséis caracteres (si «número» está entre 256 y 65535)

Esta función es muy útil en el conocimiento del código máquina y de las funciones bit a bit AND, OR y XOR

También puede ser muy útil cuando examinamos el generador de caracteres de la ROM, el área de gráficos definibles por el usuario, el fichero de atributos, las variables del sistema o el teclado. Lo último es mostrado más adelante (ver página 160 del manual Spectrum).

```
10 PRINT AT 10,10; BIN$ SIN 65022): GO TO 10
```

Si deseas algo distinto a «1» y «0» en el string haz POKE 62865 o 62869 con el carácter deseado.

CHAR$ (número)

FN C$ (número)

Ver también función NUMBER (string)

Esta función convierte enteros (todos los números entre 0 y 65535) en un string de dos caracteres permitiendo el almacenamiento de muchos datos numéricos con un muy importante ahorro de memoria. El equivalente en Basic es:

```
LET A = INT (número 256): LET B = número A * 256
LE T C$ = CHR$ A * CHR$ B
```

El string resultante dará a menudo un informe K, «Invalid colour», si se intenta imprimirlo, ya que puede obtener códigos de control de impresión, NUMBER (string de dos caracteres; se utilizará normalmente para traducir los enteros codificados en carácter a números. Dado que sólo se utilizan dos bytes por número contra cinco en e! formato normal, es de importancia considerar esta función si tienes un montón de datos numéricos a almacenar. Los datos no necesitan ser enteros; en un principio puedes multiplicar algo como 87.643 para que de 8764.3. CHAR& trabajará en la parte 8764 para producir un string de dos caracteres, el subsiguiente uso de NUMBER y la división por 100 nos dará 87.64, lo que es de una precisión aceptable para algunos propósitos. El siguiente ejemplo muestra la implementación de lo que es un array de enteros:

```
100 DIM A$(500,2)
110 FOR E = 1 TO 500: LET A$(E): CHAR$(E * 10): NEXT E
120 PRINT « Array creado   pulsa alguna tecla para imprimirlo»
130 PAUSE 0
```

```
140 FOR E = 1 TO 500 PRINT E, NUMBER(A$(E)): NEXT E
```

Fíjate que este array utiliza sólo 1K, contra los 2.5K de un array standard del mismo número de elementos. SORT actuará correctamente con estos arrays.

COSE (número)

FN C (número)

Da el coseno de «número», con un grado de aproximación menor que COS (aunque utiliza normalmente una aproximación de cuatro dígitos decimales), pero es unas seis veces más rápido.

DEC (string)

FN D (string)

Ver también función HEX$(número)

Esta función da el equivalente decimal de un string de 1 a 4 caracteres que representa un número hexadecimal válido El modo de las letras no importa

```
DEC(«FF»)     =     255
DEC(«10»)     =     16
DEC(«4000)    =     16384
DEC(«e»)      =     14
```

Para «POKEar» la memoria, usando entradas hexadecimales puedes utilizar:

```
INPUT A$ POKE dirección, DEC (A$)
```

La utilización con un string nulo o con un string de más de cuatro caracteres, o el uso de un string que contenga caracteres distintos a 0-9. A F o a f dará el mensaje «Invalid argument».

DPEEK (dirección)

FN P (dirección)

Ver también: comando DPOKE

DPEEK es un doble PEEK de la dirección especificada y la siguiente. El equivalente en Basic es:

```
LET valor PEEK(dirección) + 256 * PEEK (dirección + 1)
```

Fíjate que el byte menos significativo se asume que viene primero, lo que es la práctica usual para las variables del sistema y el código máquina Por ejemplo:

```
100 LET nxt DPEEK(23637): POKE nxt + 5, 65
110 REM xxxxxx
```

leerá la dirección de la línea 110 desde la variable del sistema «NXTLIN». El primer carácter después de REM es alterado a «A» por POKE. (POKE nxt + 5,«Ha!» también funcionará.) El « + 5» se necesita para saltar el número de línea y el apuntador de bytes de línea, y la instrucción REM.

DPOKE permite un doble POKE de la misma forma que DPEEK permite un doble PEEK.

FILLED ()

FN F()

Ver también: comando FILL

Da el número de pixels rellenados por el último comando FILL. Por ejemplo:

```
10 PLOT 0,0: DRAW 9,0: DRAW 0,9
20 DRAW -9,0: DRAW 0,-9
30 FILL 5,5
40 PRINT FILLED()
```

La longitud del lado de la caja es de 10 pixels. (Recuerda, si hemos usado 1 en vez de 9 en las instrucciones DRAW, la longitud del lado habrá sido de 2). Esto es porque aunque las líneas pueden teóricamente ser infinitamente cortas, las líneas reales de un ordenador son de un pixel de anchura; puedes imaginar las líneas teóricas como la mitad de las líneas de pixel reales. Esto da un borde de medio pixel por cada cara, por tanto las dimensiones externas de nuestra caja del ejemplo es incrementada por dos medios pixels en anchura y longitud. De este modo, tiene una longitud de lado de 10 en vez de nueve pixels. Internamente, anchura y longitud son reducidas a 8 pixels, por tanto FILLED() nos dio 64. Si usamos FILL PAPER,5,5 para quitar la caja, PRINT FILLED() nos dará 100. La diferencia entre 64 y 100 son los 36 pixels usados para formar e! perímetro).

HEX$ (número)

INH$ (número)

Ver también: función DEC(string)

El argumento numérico se convierte en un string hexadecimal. Será de dos caracteres de longitud si el número estaba entre 255 y 6255, y de cuatro caracteres si el valor absoluto del número era mayor que eso. Valores superiores a 65535 darán un mensaje «Integer out of range».

```
HEX$(32)        «20»
HEX$(255)       «FF»
HEX$(512)       «0200»
HEX$( 64)       «C0»
HEX$( 1024)     «FC00»
```

La capacidad de trabajar con números negativos será de gran utilidad a los usuarios de código máquina para el cálculo de saltos relativos hacia atrás.

Para imprimir la memoria en hexadecimal puedes utilizar:

```
100 INPUT «Dirección inicial? », direc
110 PRINT HEX$(direc);«     »;HEX$(PEEK direc)
120 LET direc     direc     + 1: GO TO 110
```

Para especificar la dirección inicial en hexadecimal, cambia la línea 100 a:

```
100 INPUT «Dirección inicial? »;A$: LET direc     DEC(A$)
```

INSTRING (inicio, string1, string2)

FN (inicio, string1, string2)

Ver también función MEMORY$

INSTRING busca en el string1 el string2, empezando en el carácter «inicio» del string1. Si se encuentra el string, el resultado será la posición en el string1 del carácter del string2; en otro caso el resultado será 0.

String1 puede ser de cualquier longitud, pero string2 debe ser menor de 256 caracteres (o recibirás el informe «Invalid argument). Si «inicio» es cero, obtendrás el mensaje «Subscript wrong». Se devolverá cero si el strings es más largo que string1 o «inicio» es mayor que LEN string1 y si cualquiera de los dos strings tiene longitud cero.

Es posible reemplazar algunos de los caracteres del string a buscar por el signo # , que significa «no importa». Por ejemplo:

```
PRINT INSTRING(1.A$.«SM # TH»
```

Encontrará la posición de «SMITH», «SMYTH», o «SMATH», etc. en cualquier lugar de A$. La única forma de que el signo # (literal) sea buscado, será cuando éste sea el primer carácter del string a buscar. La capacidad de especificar la posición de inicio para la búsqueda es útil cuando esperas encontrar más de una aparición del string muestra. El siguiente ejemplo buscará en A$ todas las apariciones de «TEST»

```
100 DIM A$(1000)
110 FOR N = 1 TO RND  *   10+ 3
120 LET pos RND * 995
130 LET A$ (pos 10 * pos + 3) «TEST»
140 NEXT N
150 PRINT «TESTs escondidos en A$. Pulsa alguna tecla para encontrarlos»
160 PAUSE 0
170 LET loc = 1
180 LET loc = INSTRING(loc,A$,«TEST»)
190 IF loc < > 0 THEN PRINT « Muestra encontrada en la posición »;loc: LET loc =
loc + 1: GO T0 180
200 PRINT «Esto es el final!»
```

El string A$ es buscado primero desde la posición 1 (loc 1) y posteriormente desde justo pasado cada «TEST» que encuentra. Cuando INSTRING devuelve «0», todas las apariciones han sido localizadas.

Nota: la línea 190 podría leerse «190 IF loc THEN PRINT ..» etc. ya que «0» es el equivalente a «no cierto». Si crees que el ejemplo anterior es muy fácil (después de todo, el string es en su mayoría blanco) cambia la línea 100 a

```
100 LET A$ = FN S$(250,«TESS»)
```

que requerirá que cada «TESS» sea chequeado hasta el cuarto carácter antes de que pueda ser distinguido de «TEST».

La totalidad de la memoria o una zona en particular de ella puede ser explorada por INSTRING usando la función MEMORY$. Un complejo, pero útil ejemplo se da más adelante. Un string dado es buscado en el array A$, y los elementos en que éste aparece se imprimen. Si tienes un array manejable, omite la línea 30 y modifica las líneas 10 y 20 (utiliza una letra mayúscula en la última). La rutina construye un string de tres caracteres el cual precederá al array en el área de variables (líneas 60 a 80 ver página 168 del manual Spectrum). El área de variables es entonces rastreada (línea 110) y la dirección del primer carácter del array es asignada a st. Utilizando esta información, el array completo es entonces rastreado para encontrar el string requerido (línea 120) y el elemento (o elementos) del array que contengan el string se imprimirán.

```
10 LET r = 100: LET c = 20: REM dimensiones
20 LET b$ = CHR$ (CODE«A» * 128: REM letra del array
30 DIM a$(r,c)
40 INPUT (Buscar que? »: s$
50 LET a$(RND r + 5): s$
60 LET lenght = r * c + 5
70 LET c$ = CHAR$(lenght)
80 LET b$ = b$ + c$(2)  + c$(1)
90 LET char = 1
```

```
100 LET vars = DPEEK(23627)
110 LET s8t = INSTRING(vars, MEMORY$(),b$) + 8
120 LET char = INSTRING(char,MEMORY$()(st TO st + r * c), s$)
130 lF: char <>0 THEN PRINT INT (char/c) + 1: LET char = char + 1: GO TO 120
```

INSTRING puede ser utilizado para revisar un string de entrada en programas educativos o de juegos. Por ejemplo, suponte que el Spectrum ha preguntado algo, cuya respuesta requerida, almacenado en C$, es «NAPOLEON». La gente que haya entrado «NAPOLEON » (fíjate en el espacio) o «NAPOLEON BONAPARTE» y les han dicho que estaban equivocados, se sentirán un poco molestos pero por supuesto, esto pasa a menudo cuando el string de entrada y C$ son comparados de forma simple. Puedes resolver este problema en la mayoría de los casos con algo como lo que viene a continuación, que imprime «Correcto» si la respuesta está en algún lugar del string de entrada:

```
INPUT A$: IF INSTRING(1,A$.C$) <> 0 THEN PRINT «Correcto!»
```

Aún otra aplicación para esta versátil función es implementar el empaquetado de múltiples strings de longitud variable o registros en un gran string. Un paso sería reservar todos los caracteres de un rango particular (p. e. CHR$ 1 - CHR$ 31) como caracteres «indicadores». CHR$ 1 podría marcar el principio del primer substring de la larga cadena, CHR$ 2 el segundo, etc. Entonces:

```
PRINT A$(INSTRING(1,$,CHR$ n) + 1 TO I
INSTRING(1, A$,CHR$(n + 1)) - 1)
```

imprimiría el substring «n» de A$. Son posibles muchos programas alternativos, pero todos tienen la ventaja de que el string principal puede ser rastreado para localizar rápidamente un substring particular, y se gana espacio si los string son de longitud irregular.

MEMO

FN M()

MEM() devuelve el número de bytes de memoria libre disponible. No ha de ir nada entre los paréntesis Prueba:

```
PRINT MEM(): DIM A$(100): PRINT MEM()
```

Esta es una simple función que consiste principalmente en una llamada a la ROM y puede ser duplicada en ausencia de Beta Basic por PRINT 65535 USR 7962.

MEMORY$()

FN M$()

Ver también: POKE (en este manual)

Devuelve la totalidad de la memoria como un string. Actualmente, el primer byte del ordenador, posición 0, no se incluye, ya que CODE MEMORY$()(1) es lo mismo que PEEK 1. Los últimos tres bytes de memoria son también excluidos (por razones técnicas) por tanto la función tiene de longitud 65533. Obviamente te quedarás sin espacio si utilizas:

```
LET a$ = MEMORY$()
```

pero puedes utilizar

```
LET a$ = MEMORY$()( 16384 TO 22527)
```

En combinación con la capacidad del Beta Basic para hacer POKEs de cadenas, esta función da al programador Basic el poder de mover grandes áreas de memoria muy rápidamente. Para una más amplia descripción de este aspecto, ver POKE en este manual.

Otra utilización de MEMORY$ es que permite rápidas búsquedas de memoria utilizando INSTRING. Aunque una sección especifica de memoria puede ser localizada usando la notación entre paréntesis (p. e. MEMORY$()(23759 TO)) INSTRING es tan rápido que puede a menudo ser más simple buscar en la totalidad de la memoria.

```
10 REM asdfg
20 PRINT INSTRING(1,MEMORY$(),«asdfg»)
```

encontrará la posición de «asdfg» en la instrucción REM. Si omites la línea 10, el «asdfg» será encontrado en la línea 20. Si haces LET a$ = «asdfg» y buscas por a$, la encontrarás en el área de variables. Siempre encontrarás como mínimo una aparición del string en cualquier parte de la memoria.

En lugar de «1» podríamos haber usado «DPEEK(23635)» que es la variable del sistema PROG, para empezar la búsqueda al principio del programa en lugar de la dirección 1 en la ROM.

Para buscar todas las apariciones de un string puedes usar:

```
10 LET adr = 1
20 LET adr = INSTRING(adr,MEMORY$(),a$)
30 IF adr <> 0 THEN PRINT adr: LET adr = adr + 1: GO TO 20
```

Ya que Beta Basic permite el POKE de strings, la búsqueda de un string y reemplazarlo por otro fácilmente, convendrá que sepas lo que estás haciendo A menudo deberás evitar que el string «reemplazador» sea más largo que el primero.

MOD (número, número)

FN V (número, número) V

Da el resto sobrante cuando el primer número es dividido por el segundo número (la letra de función está sobre la tecla «/» esto te ayudará a recordarlo). A esto se le llama tomar el primer número «módulo» segundo número.

```
MOD(10,3)      =    1
MOD(66,16)     =    2
MOD(125,35.5)  =    18,5
```

El siguiente ejemplo previene que PLOT se salga de pantalla:

```
10 FOR n = 0 TO 400
20 PLOT MOD(n,256), MOD(n,176)
30 NEXT n
```

NUMBER (string)

FN N (string) N

Ver también: CHAR$ (número)

Convierte un string de dos caracteres en un número entero (un número entero entre 0 y 65535). El equivalente en Basic es:

```
LET número = 256 * CODE C$(1) CODE C$(2)
```

Obtendrás un mensaje «Invalid argument» si el string no es de longitud 2. Con la función complementaria CHAR$; NUMBER facilita la implementación de «arrays de números enteros» (ver CHAR$ para ejemplo)

OR (número, número)

FN O (número, número) O

Esta función se llama de la misma forma que la palabra clave normal, pero puede distinguirse en un listado de programa por la diferente sintaxis utilizada. Da un OR bit a bit de dos números, que deberán estar entre 0 y 65535 Si un bit es un«1» en el primer número o (OR) en el segundo número, entonces será «1» en el resultante. El bit debe ser «0» en ambos números para ser «0» en el resultado.

RNDM (número)

FN R (número)

Si «número» es 0, RNDM da un número aleatorio entre 0 y 1, como RDN. Es sobre unas dos veces y media más rápido. Sin embargo, Si «número» no es cero. RNDM da un número aleatorio entero entre 0 y «número» inclusive. Esto también es unas dos veces y media más rápido de lo sería RDN * «número».

```
10 PLOT RNDM(255), RNDM(175)
20 GO TO 10
```

RANDOMIZE (número) llevará a RNMD a un lugar particular en su secuencia de números pseudo aleatorios, justo como hace RDN.

SCRN$(línea, columna)

FN K$(línea, columna)

Tecla: la misma que SCREEN$

Funciona de forma similar a SCREEN$, excepto que reconoce los gráficos definidos por el usuario como caracteres normales. Un fallo que afecta al SCREEN$ del Spectrum también ha sido evitado. Primero, entra KEYWORDS 0. Luego prueba el ejemplo siguiente que transforma los gráficos definibles en caracteres aleatorios, y después lee algunos desde la pantalla.

```
10 FOR a = USR «a» TO USR «u» = 7
20 POKE a, RND * 255: REM RNDM(255) es más rápido
30 NEXT a
40 PRINT «(algunos gráficos definibles por el usuario)»
50 LET a$ x ««
60 FOR c = 0 TO 31
70 LET a$ = a$ + SCRN$(0,c)
80 NEXT c
90 PRINT a$
```

El conjunto de gráficos del Spectrum no es reconocido. Si necesitas hacer esto, programa algunos de los gráficos definibles para verlos como el bloque de gráficos.

SINE(número)

FN S(número)

Da el seno de «número», con un menor grado de precisión que SIN, aunque normalmente se aproxima a los cuatro decimales), pero es unas seis veces más rápido.

STRING$(número, string)

FN S$(número, string)

Esta función repite «número» veces el «string».

STRING(32,«-»)	32 signos menos
STRING$(4, «AB»)	«ABABABAB»
PRINT STRING$(704,«X»)	pantalla llena de «X»s
PRINT STRING$(3, «A», CHR$ 13)	A
	A
	A

STRING$ es más rápido que un bucle FOR NEXT y más corto que entrar el string explícitamente, siempre que el string sea más largo de unos 14 caracteres. Puede ser útil en combinación con el total de strings del Beta Basic para llenar un área de memoria (como el fichero de atributos) con los valores deseados

TIME$()

FN T$()

Ver también: CLOCK

Esta función devuelve la hora actual, tal como está almacenada el reloj digital del Beta Basic (ver CLOCK), como un string de ocho caracteres. El formato es HR:MN:GS. TIME$() devolverá constantemente valores cambiantes:

```
100 CLOCK 1
110 LET M$ = TIME$(): PRINT M$
120 PRINT «Horas    »;N$(1 TO 2);«Mins    »;N$(4 TO 5)
130 GOTO 110
```

Es usualmente una buena idea pasasr el resultado de TIME$ a una variable, para «congelar» por un momento el tiempo.

USING$ (string formato, número)

FN U$ (string formato, número)

Ver también: USING

Devuelve el string equivalente de «número», formateado según el «string formato» especificado. El número de dígitos antes y después del punto decimal puede ser especificado, y el redondeo se hace al último signo impreso. La palabra clave USING de la tecla «U» proporciona una función equivalente que sólo puede ser aplicada a PRINT, mientras USING$ puede usarse con cualquier comando que permita la utilización de un string, p. ej. LET. Ver USING para una explicación más completa del formateo con USING/USING$

XOR(número, número)

FN X(número, número)

Da un OR exclusivo bit a bit entre dos números que deben tener un valor entre 0 y 65535. Si un bit es un «1» en ambos números o un «0» en ambos números será «0» en el resultado. Si un bit es un «1»en sólo uno de los números, será «1» en el resultado.

EL JUEGO DE CARACTERES APENDICE A

Las siguientes enmiendas al Apéndice A del «Spectrum handbook» son a la fuerza en estado «KEYWORDS 1».

CODIGO	TECLA	CARACTER
128	8	KEYWORDS
129	1	DEF PROC
130	2	PROC
131	3	END PROC
132	4	RENUM
133	5	-----
134	6	AUTO
135	7	DELETE
136	shift-7	----
137	shift-6	JOIN
138	shift-5	EDIT
139	shift-4	KEYIN
140	shift-3	----
141	shift-2	----
142	shift-1	DEF KEY
143	shift-8	----
144	A	ALTER
145	B	B (no función)
146	C	CLOCK
147	D	DO
148	E	ELSE
149	F	FILL
150	G	GET
151	H	H (no función)
152	I	EXIT II
153	J	WHILE
154	K	UNTIL
155	L	LOOP
156	M	SORT
157	N	ON ERROR
158	O	ON
159	P	DPOKE
160	Q	POP
161	R	ROLL
162	S	SCROLL
163	T	TRACE
164	U	USING

INFORME G ZX INDICE B

El informe G se usa, indiferentemente por el Beta Basic y por el Basic del Spectrum. Los demás mensajes que siguen son nuevos y se usan como extensión de los del Basic Spectrum.

CODIGO	SIGNIFICADO	SITUACIONES
G	No room for line	RENUM

	La renumeración del programa especificado provocaría que una o más de las nuevas líneas resultaría en uno de los bloques que no esta siendo renumerado o es mayor de 9999	
S	Missing LOOP	DO, EXIT IF
	EXIT IF o un DO condicional (seguido de WHILE o UNTIL) intenta ir al final de un DO LOOP y falla al buscar una instrucción LOOP.	
T	LOOP without DO	LOOP
	Se ha usado LOOP sin una instrucción DO emparejada.	
U	No such line	DELETE
	Se ha utilizado DELETE con un número de línea que no existe en el programa.	
V	No POP data	POP
	Se ha producido un intento de extraer datos de la pila de GOSUB/DO-LOOP/PROC cuando la pila está vacía; p. ej. no había ningún GOSUB, DO-LOOP o PROC en operación.	
W	Missing DEF PROC	PROC, END PROC
	Se ha utilizado PROC sin que exista ninguna definición de procedimiento del mismo nombre en el programa, o se ha utilizado END PROC sin DEF PROC.	
X	No END PROC	DEF PROC
	El intento de saltarse un DEF PROC (no durante PROC) ha fallado al no encontrar un END PROC.	
Y	Too hard	RENUM
	Al renumerar un programa se ha encontrado una referencia a un número de línea en forma de expresión.	

CODIGOS DE ERROR APÉNDICE C

La siguiente lista da el valor de la variable ERROR (producida por la utilización del comando ON ERROR) para los diferentes estados de error. La primera sección es para los mensajes standard de error, la segunda es para los mensajes de Beta Basic, y la tercera es para los «errores de Interface» y sólo se aplica si el Interface 1 está conectado. Los mensajes «0» y «9» no son interceptados por ON ERROR.

VALOR ERROR	CODIGO	MENSAJE
0	0	OK (no interceptado)
1	1	NEXT without FOR

2	2	Variable not found
3	3	Subscript wrong
4	4	Out of memory
5	5	Out of screen
6	6	Number too big
7	7	RETURN without GOSUB
8	8	End of file
9	9	STOP statement (no interceptado)
10	A	Invalid argument
11	B	Integer out of range
12	C	Nonsense in Basic (ver informe 44)
13	D	BREAK - CONT repeats
14	E	Out of DATA
15	F	Invalid file name
16	G	No room for line
17	H	STOP in INPUT
18	I	FOR without NEXT
19	J	Invalid I/O device
20	K	Invalid colour
21	L	BREAK into program
22	M	RAMTOP no good
23	N	Statement lost
24	O	Invalid stream
25	P	FN without DEF
26	Q	Parameter error
27	R	Tape loading error

INFORMES BETA BASIC

28	S	Missing LOOP
29	T	LOOP without DO
30	U	No such line
31	V	No POP data
32	W	Missing DEF PROC
33	X	No END PROC
34	Y	Too hard

INFORMES DEL INTERFACE 1

Son dados en el orden en que existen en la ROM del Interface 1. Dos informes, «m header mismatch error» y «u Hook code error», no son documentados en el manual del Interface 1. El informe «c Nosense in BASIC» es un duplicado del informe normal, pero causado por operaciones del Interface. Los códigos con letras minúsculas sólo se producen cuando Beta Basic está presente.

VALOR ERROR	CODIGO	MENSAJE
43	b	Program finished
44	c	Nosense in BASIC
45	d	Invalid stream number
46	e	Invalid device expression
47	f	Invalid name
48	g	Invalid drive number
49	h	Invalid station number
50	i	Missing name
51	j	Missing station number
52	k	Missing drive number
53	l	Missing baud rate
54	m	Header mismatch error
55	n	Stream already open
56	o	Writing to a «read» file
57	p	Reading to a «write» file
58	q	Drive «write» protected
59	r	Microdrive full
60	s	Microdrive not present
61	t	File not found
62	u	Hook code error
63	v	CODE error
64	w	MERGE error
65	x	Verification has failed
66	y	Wrong file type

VARIABLES ESPECIALES APENDICE D

Beta Basic posee un número de variables especiales, que no toman el mismo camino que las ordinarias Sus nombres y propiedades son dadas abajo.

VARIABLES GRAFICAS

Estas variables siempre existen - se les asigna un valor particular por RUN o CLEAR en vez de ser destruidas. Utilizando LET para cambiar sus valores afectará al funcionamiento de DRAW, PLOT, CIRCLE y FILL.

NOMBRE	VALOR ASIGNADO POR RUN/CLEAR	SIGNIFICADO
XOS	0	valor eje X
XRG	256	rango eje X
YOS	0	valor eje Y
YRG	176	rango eje Y
ANGLE	0	(NO UTILIZADO POR LA VERSIÓN 1.8)

VARIABLES TRACE Y ON ERROR

Estas variables normalmente no existen - son creadas por ON ERROR si se produce un error mientras este se utiliza, o por TRACE antes de que se ejecute cada instrucción, si TRACE está en funcionamiento.

NOMBRE	SIGNIFICADO
ERROR	- valor del código para el último error (ver Apéndice C)
LINE	- Para TRACE, número de línea a ejecutar.
	- Para ON ERROR, número de línea donde ocurre el error.
STAT	- Para TRACE, número de instrucción a ejecutar.
	- Para ON ERROR, número de instrucción donde se produce el error.

UTILIZACION DE IMPRESORAS APENDICE E

Los programas Beta Basic «LLISTarán» correctamente en la impresora ZX o en una impresora serie conectada al canal «t» del Interface 1.

En otros casos, los muchos tipos de Interface de impresoras y de software de control pueden crear un problema de compatibilidad. Beta Basic necesita el control del canal «p» si es para «LLISTar» las nuevas palabras claves. Antes de coger el control (durante la inicialización del programa, una vez la carga ha finalizado) el programa almacena la dirección que encuentra en el canal LPRINT, y después mueve toda la salida de impresora a esa dirección después de modificarla.

Debes entonces cargar e inicializar tu software de control de impresora antes de cargar el Beta Basic. Has de prever que no exista ningún problema de utilización de memoria, y prever que el software de control no rellene subsecuentemente su propia dirección al canal LPRINT, y todo irá bien.

Para ayudarle a comprender cualquier problema, aquí tienes alguna información:

La dirección de la rutina actual de LPRiNT puede ser leída del canal «p» por:

```
PRINT DPEEK(DPEEK(23631) + 15)
```

El valor es 64423 si Beta Basic lo controla; cualquier otro valor indicará que tu software de impresora lo ha alterado.

PRINT DPEEKI61081)

Da la dirección a la cual la salida de impresora es conducida después de que Beta Basic la haya modificado; esto normalmente es 2548, una dirección en ROM. Contiene la dirección de tu software de control de impresora si has inicializado tu impresora antes de cargar Beta Basic. Puedes conseguir que Beta Basic libere el canal «p» como hace durante la inicialización por:

RANDOMIZE USR 58419

Nota:

El ROLL y SCROLL horizontal utiliza el buffer de la impresora para almacenamiento temporal, por tanto cualquier código situado allí será sobreescrito si se utilizan estos comandos.

INFORMACION COMPLEMENTARIA

1.- Para que Beta Basic trabaje correctamente con las nuevas ROMs del Interface 1 (nos. de serie del 87316 en adelante) han de hacerse los siguientes POKEs. Sólo los dos primeros son importantes a menos que utilices el canal «t».

```
POKE 56843,102
POKE 61212,183
POKE 61224,191
POKE 57488,195
POKE 57489,68
POKE 57490,12
POKE 57531,62
POKE 59326,58
```

2.- Si tienes un Interface 1 con un número de serie menor de 87316, Beta Basic proporciona un TAB en el canal «t». La longitud de línea de la salida de este canal puede ser activada con POKE 57500 con el valor deseado (el valor normal es 80).

Si tienes un Interface 1 con un número de serie mayor de 87316, el Interface proporciona el TAB por sí mismo, y la longitud de línea es activada por POKE 23729

Nota: Si tienes alguna duda sobre qué versión del Interface 1 tienes, entra CAT sin número de drive. PEEK 23738 será 123 en la primera versión y 131 en la nueva versión.

3.- CHR$ 12 (delete) trabaja correctamente, cuando se imprime en la pantalla el último carácter es borrado, en vez de producir un «?».

4.- Cuando utilizamos impresoras quo no sean ZX directamente conectadas al Spectrum, es a veces conveniente prevenir que los códigos de control tengan su efecto usual. (Los usuarios del Interface 1 pueden utilizar el canal «b»). Por ejemplo, si estabas mandando información «bit-image» a una impresora de matriz de puntos no querrás arriesgarte a que los códigos de control TAB envíen series de espacios, como hacen normalmente. Puedes asegurarte de que Beta Basic no responderá a ningún código de control durante el LPRINT entrando KEYWORDS 0. Todos los caracteres serán enviados directamente al software de control de impresora que controlaba el canal de LPRINT cuando el Beta Basic fue cargado.

SUMARIO DE SINTAXIS

Cualquier cosa encerrada entre < esto > puede ser omitida si se desea.

ALTER < atributos > TO atributos

AUTO < línea de comienzo > <, intervalo >

BREAK (mejorado) shift-space

CLOCK número o string

CURSOR CONTROL CODES: CHR$8 CHR$9 CHR$10 CHR$11

DEF KEY carácter: instrucciones<: > o carácter; string

DEF PROC nombre

DELETE < num línea > TO < num línea >

DO o DO WHILE o DO UNTIL

DPOKE dirección, número (0-65535)

EDIT < núm línea >

ELSE instrucción

END PROC

EXIT IF condición

FILL < INK o PAPER color; > x,y

GET variable numérica o string

JOIN < núm línea >

KEYIN string

KEYWORDS 1 o 0

LIST < núm línea > TO < núm línea >

LLIST < núm línea > TO < núm línea >

LOOP o LOOP WHILE o LOOP UNTIL

ON como en GO TO ON n;línea,línea...
GO SUB ON n;línea,línea...

ON ERROR num línea

PLOT x,y;string

POKE dirección, string

POP < variable numérica > PROC nombre

RENUM < rebanada > <LINEIin> < STEP s >

ROLL dir <,pixels > <;x,y;anch, long >

SCROLL < dir > <,pixels > <;x,y;anch,long >

SORT string o array de strings < parent > < parent >
o array numérico < parent > < parent >

SPLIT entrado como< > (shift-W)

TRACE número de línea

UNTIL como en DO UNTIL o LOOP UNTIL

USING como en PRINT USING. string;num

WHILE como en DO WHILE o LOOP WHILE

XOS como en LET XOS = número

XRG como en LET XRG = número

YOS como en LET YOS = número

YRG como en LET YRG = número

FUNCIONES

FN	A	AND(número, número)
FN	B$	BIN$(número)
FN	C$	CHARS(número)
FN	C	COSE(número)
FN	D	DEC(hex$)
FN	P	DPEEK(dirección)
FN	F	FILLEDO
FN	H$	HEX$(número)
FN	I	INSTRING(inic,a$,b$)
FN	M	MEM()
FN	M$	MEMORY$()
FN	V	MOD(número, número)
FN	N	NUMBER(string 2 car)
FN	O	OR (número, número)
FN	R	RNMD(número)
FN	K$	SCRN$(lin,col)
FN	S	SINE(número)
FN	S$	STRING$(rep, string)
FN	T$	TIME$()
FN	U$	USING$(format, num)
FN	X	XOR(número,número)