CAPÍTULO TERCERO

Programación en BASIC con el ZX Spectrum

¿QUÉ ES UN PROGRAMA?

Anteriormente ya he mencionado en alguna parte la palabra programa, y no dudo de que usted ya la habrá oído en alguna otra parte. ¿Pero, qué es exactamente un programa? Pues bien, un programa es una serie de instrucciones usando palabras del BASIC como las que hemos ido viendo hasta ahora. La diferencia importante es que cada instrucción tiene un **número de líneas,** como esto:

```
10 PRINT "Mi nombre es Spectrum"
20 PRINT "¿Cuál es el suyo"
```

Escriba ambas líneas y pulse ENTER después de cada una. Inmediatamente observará que al contrario de lo que ocurría antes cuando usaba PRINT, el mensaje no aparece en la pantalla en el momento en que usted pulsa ENTER, sino que la línea sube arriba con una flecha ('>') delante. El interrogante se obtiene usando SYMBOL SHIFT y pulsando la tecla C.

Por tanto, la diferencia más grande que existe entre los **comandos directos** que utilizábamos hasta ahora, y las líneas de programa, es que estas últimas son almacenadas en la memoria del ordenador para ejecutarse en cuanto reciban nuestra orden. Esta orden es **RUN** que se obtiene pulsando la tecla R seguida de ENTER. Hágalo, y en la pantalla obtendrá:

```
Mi nombre es Spectrum
¿Cuál es el suyo?
```

Así, ¿qué función tienen los números de línea? Esencialmente le dicen al Spectrum que cualquier instrucción que vaya precedida de ellos debe ser almacenada y no ejecutada inmediatamente, y en segundo lugar, también sirven para indicar el orden en que deben ser ejecutadas las instrucciones. Por lo tanto, cuando usted hace RUN y ENTER el Spectrum busca el programa y empieza a ejecutar la instrucción con el número de línea más bajo. Una vez hecho esto busca el número siguiente, ejecuta la línea que le corresponde y así sucesivamente hasta que se llega al final del programa o bien se le indica que salte a un número de línea anterior, pero sobre esto ya hablaremos más adelante.

En el ejemplo se han numerado las líneas con los números 10 y 20, pero hubieran podido tener cualquier otro par de números entre 1 y 9999 mientras que el segundo sea más alto que el primero. Por ejemplo, se podrían haber usado los números 1 y 2. Pero por regla general es bueno numerar las líneas en intervalos de 10. De esta manera se podrá insertar cualquier línea que se haya olvidado sin necesidad de hacer correr todos los números de línea. Quizá la única excepción a esta regla es cuando hay que hacer un programa muy largo en el Spectrum de 48K. En este caso es posible que a intervalos de 10 se llegue a superar el máximo de 9999 antes de que se haya completado el programa. Es difícil estimar la longitud que va a tener un programa, pero intente hacerlo para determinar el intervalo más adecuado. El numerar de 5 en 5 es una buena alternativa para los programas más largos.

En mi ejemplo anterior parece que el Spectrum esté esperando una respuesta. Intente responder y vea lo que ocurre. Simplemente, intente escribir su nombre. De buen principio, ya va a tener dificultades pues la primera tecla que pulse producirá una palabra entera y no la letra correspondiente. Pero si lo consigue, al pulsar ENTER aparecerá el mensaje de error: "2 Variable not found, 0:1" que se produce cuando el Spectrum intenta interpretar su nombre o lo que haya escrito como un número. El problema está en que no hemos incluido nada en el programa que le permita al Spectrum aceptar una respuesta, y él no hará nada que no le hayamos dicho previamente mediante una línea de programa. Vamos a hacer que pueda aceptarnos una respuesta a nuestra pregunta. Esto se consigue con la

instrucción INPUT que se obtiene pulsando la tecla "I" cuando el cursor está en modo K. Escriba lo siguiente:

```
10 PRINT "Mi nombre es Spectrum'
20 PRINT "¿Cuál es el suyo"
30 INPUT n$
```

El símbolo del dólar se obtiene pulsando SYMBOL SHIFT y la tecla 4. Haga RUN y observe la diferencia. Esta vez, en lugar de dar el mensaje de que la tarea ha terminado, el Spectrum presenta el cursor L parpadeante entre dos comillas en la parte inferior izquierda de la pantalla. Escriba su nombre y pulse ENTER. Su nombre apareció entre las comillas pero tan pronto como pulsó ENTER desapareció y fue reemplazado por el mensaje "0 OK" otra vez. Ahora observe, escriba esto tal como está aquí:

PRINT n$ (y pulse ENTER)

Su nombre aparecerá justo debajo de la última cosa que hay escrita en la pantalla (que probablemente era la frase "¿Cuál es el suyo"). ¿Ha visto lo que ha ocurrido?, el Spectrum ha asignado su nombre a n$ en la instrucción PRINT y cuando le hemos dicho que escribiera su nombre lo ha hecho obedientemente. A n$ se le llama **variable de cadena** y puede ser cualquier letra del alfabeto seguida del símbolo del dólar. El símbolo del dólar detrás de una letra le indica al Spectrum que con ese nombre (por ejemplo n$) reconocerá de ahora en adelante una **cadena** de caracteres. En lugar de su nombre usted podía haber escrito cualquier otra cosa, inténtelo, pero antes añadiremos una línea más al programa:

```
40 PRINT "Hola"; n$      (indica un espacio)
```

Cuando haga RUN el Spectrum le dirá hola a usted, o a cualquier cosa que ponga en el momento de preguntarle su nombre. Experimente con distintos nombres, por ejemplo, "123 mmsgdter4tre" o cualquier otra cosa así de extraña y verá que el Spectrum no distingue, todo lo que le escribe lo enseña luego como si fuera su nombre. Vamos a ver rápidamente qué es lo que hace este programa y cómo lo hace. Primero pulse ENTER para obtener el listado del

programa, que sustituirá a los mensajes que había en la pantalla.

Las primeras dos líneas simplemente escriben en la pantalla dos mensajes. La línea 30 hace que el Spectrum se espere hasta que usted entre una cadena de caracteres (que debería ser su nombre, pero ya hemos visto que acepta cualquier cosa). Todo lo que usted escriba en la respuesta hasta que pulsa enter es tomado por el Spectrum y etiquetado con el nombre n$. En la línea 40 el Spectrum escribe "Hola" e inmediatamente después, escribe la cadena que ha sido etiquetada con el nombre n$, y no los caracteres n y $ (para hacer esto último, deberían estar entre comillas ¿recuerda?). Ahora vamos a añadir algunas líneas al programa que lo harán más divertido, o quizá más embarazoso, depende de la edad que tenga usted.

```
40 PRINT "Yo soy joven".                    (Escriba las
   "¿Cuántos años tiene usted?"             dos comillas.)
50 INPUT edad
60 PRINT n$; "tiene"; edad; "años"
```

Haga RUN y observe los cambios, esta vez, el Spectrum enseña cosas como éstas:

```
Mi nombre es Spectrum
¿Cuál es el suyo?
Yo soy muy joven
¿Cuántos años tiene usted?
Tim tiene 189 años
```

Y ahora, para hacerlo más interesante añada estas dos líneas:

```
55 CLS          (se encuentra en la tecla V)
70 GOTO 60      (pulse G para obtener GOTO
```

Intente adivinar qué es lo que ocurrirá, para ello recuerde que CLS borra la pantalla y que **GOTO** significa "IR A", es decir, produce un salto a la línea que se le indica. Haga RUN del nuevo programa.

¿Sorprendido? Ha aparecido un "¿scroll" en la parte de abajo de la pantalla. Si quiere continuar pulse cualquier tecla que no sea SPACE o N. Ha observado también que

le he hecho escribir la línea 40 dos veces. Pulse ENTER y mire cuál de las dos es la que está. Sólo está la última que se escribió. Recuerde esto, pues es muy útil saber que al escribir una misma línea dos veces se borra la que había anteriormente. Si usted comete un error y la línea aún está abajo entonces usted puede usar DELETE (CAPS SHIFT y 0) para corregirla. Si ya ha pulsado ENTER entonces puede escribirla de nuevo para hacer la corrección.

¿Cómo trabaja el programa? En la línea 40 el Spectrum pregunta por su edad y en la línea 50 se espera a que la introduzca y la carga en una *variable* numérica que yo he querido que se llamara "edad", veremos que las variables numéricas (que son aquéllas que sirven para almacenar números en lugar de caracteres en general) no llevan detrás el signo del dólar. Es posible utilizar una palabra entera como nombre de *variable numérica,* mientras que las de cadena sólo pueden tenerlo de una sola letra del alfabeto (seguida de $). Comprenderá que he elegido n$ para el nombre para recordar que era para el Nombre. Es muy provechoso que coja el hábito de nombrar las variables con nombres que recuerden a su contenido. El hecho de usar "edad" para la edad está mucho mejor que haber usado "e", y esto último es mejor que usar "x", que no tiene nada que ver con el contenido y hará que no se comprenda el programa si se repasa algunos meses más tarde.

Por último, la línea 60 escribe el nombre que el Spectrum tiene almacenado en n$, inmediatamente escribe "tiene" con un espacio en cada lado, luego escribe el contenido de la variable edad y seguidamente la palabra años con un espacio en cada lado. ¿Sabe por qué he puesto un espacio después de "años"? ¿y un punto y coma? Pues bien, el punto y coma hace que la próxima vez que se escriba algo lo haga a continuación de lo anterior, y el espacio es para que quede un hueco entre los dos campos que se imprimen. La línea 70, como no dudo que habrá adivinado, envía al Spectrum de vuelta a la línea 60 una y otra vez para escribir las mismas series de palabras y números.

Vamos ha intentar un programa más difícil. En él se pregunta por un año del pasado y el Spectrum le da una idea de cuántos años, meses, horas y minutos han pasado hasta el año actual. Escriba el programa y haga RUN:

```
  10 REM "CUANTO TIEMPO?"
  20 INPUT "Anno en el pasado";
y1
  30 INPUT "Anno actual";y2
  40 LET yrs=y2-y1
  50 LET mth=yrs*12
  60 LET dys=yrs*365
  70 LET hrs=dys*24
  80 LET mins=hrs*60
  90 PRINT "Han pasado ";yrs;" annos,
"'';mth;" meses,"'';dys;" dias,"'';
hrs;" horas,"'"y ";mins;" minutos"
```

La línea 90 es la más difícil, especialmente hasta que usted no domine todos los pequeños trucos que encierra. En todo caso, intente por favor reescribirla exactamente tal y como está aquí, con cada espacio, cada comilla, cada punto y coma y cada par de apóstrofes. Todo es necesario (el apóstrofe se obtiene con SYMBOL SHIFT y 7). Ahora, si usted ha hecho RUN del programa obtendrá una salida en la pantalla más o menos como esto:

Anno en el pasado 1604	(parte baja de la pantalla)
Anno Actual 1982	(parte baja de la pantalla)
Han pasado 378 annos,	(arriba)
4536 meses,	
1378705 días,	
3311280 horas	
y 1.986768E + 8 minutos.	(Esto puede parecerle un poco extraño, pero ya explicaré la manera que tiene el Spectrum de escribir los números más grandes.)

En este programa he introducido muchas cosas nuevas, intencionadamente, así que no se preocupe si lo encuentra un poco difícil. Vamos a verlo línea por línea.

De entrada tenemos una nueva palabra en la línea 10. **REM** es una palabra del BASIC que es una abreviación de la palabra inglesa REMARK que significa comentario. Esta línea no hace nada más que dar una aclaración sobre lo que hace, o de qué trata el programa o una parte del programa. En una sentencia REM se puede poner tanto un título, como una breve descripción, un aviso, etc. Es fácil escribir una sentencia REM. La palabra REM se obtiene pulsando la tecla E y ya se tiene en la pantalla. El Spectrum no hace ningún caso de esta línea cuando se ejecuta el programa. Para probarlo escriba:

10 y pulse ENTER

Esto provoca que la línea 10 sea borrada del programa. Haga RUN para provar que no hay ninguna diferencia. Es una buena práctica usar sentencias REM en los programas que ayuden a recordar qué es lo que hace, cómo lo hace, etc. ¡Es como un salvavidas encontrarse sentencias REM en un programa largo y complicado cuando se vuelve a él después de un tiempo!

¿Qué hay de las líneas 20 y 30? Parecen un cruce entre las líneas con INPUT que vimos antes y las sentencias con PRINT. De hecho esto es una buena descripción de lo que son en realidad. Cuando usted quiere que su Spectrum le pregunte algo por pantalla, tiene que usar previamente una sentencia PRINT para que aparezca la pregunta. Pero para obtener la información hay que usar una sentencia INPUT en la que el Spectrum se espera a que escriba un número o una palabra, que es asignado a una variable numérica o de cadena. Gracias a este nuevo tipo de sentencia INPUT podemos poner juntas en una misma línea las dos cosas, la aparición de la pregunta, y el tratamiento de la respuesta. Este método es más elegante y ocupa menos espacio, pero ésta no es la única diferencia. Para ilustrarlo, escriba una nueva línea 20 y añada una línea 21, dejando la línea 30 tal como está para poder comparar los dos métodos. Escriba:

```
20 PRINT "Anno en el pasado?"
21 INPUT Y1
```

Ahora haga RUN del programa otra vez para ver los cambios. En primer lugar la pregunta "año en el pasado" aparece arriba de la pantalla en vez de abajo, pero en cambio la respuesta se escribe igualmente abajo. Es cuestión de opinión decidirse por uno de los dos métodos en un momento determinado, pero el hecho de disponer de los dos da más flexibilidad a la programación. El método de INPUT "..." deja la pantalla limpia mientras que el otro mantiene la pregunta en la pantalla (aunque también puede colocar un CLS en la línea 80 más o menos, para deshacerse de ella). También puede escoger entre poner una coma o un punto y coma después del mensaje o pregunta de una línea INPUT. Debe ir después de las últimas comillas y antes de la variable. Pruebe esto como comandos directos:

```
INPUT "Cuánto es", q$      (y pulse ENTER)
INPUT "Cuánto es", q$      (y pulse ENTER)
```

Como puede ver, la única diferencia, es que el cursor L (y por lo tanto, el lugar donde se escribirá su respuesta) está en el primer caso en el lado derecho de la parte baja de la pantalla mientras que en el segundo caso, está justo detrás de la pregunta. Usted puede usarlo como quiera, aquí sí que no hay reglas.

Ahora usted ya debe haber comprendido algo de lo que hacen la coma y el punto y coma en las sentencias PRINT e INPUT. El punto y coma hace que lo próximo que se vaya a imprimir lo haga justo detrás de lo último que se imprimió. Por otro lado la coma indica que el próximo campo a imprimir debe hacerlo en la mitad derecha de la pantalla, pero si ésta ya tiene alguna cosa escrita entonces lo hace en la siguiente línea. El punto y coma coloca la información seguida mientras que la coma va muy bien para poner texto o números en dos columnas, derecha e izquierda. Por ejemplo:

```
PRINT "Hola", "Aquí"      (y ENTER)
PRINT "Hola"; "Aquí"      (y ENTER)
```

Volvamos al programa. Las líneas de la 40 a la 80 calculan el número de años, meses, días, horas y minutos que han pasado desde el año que usted da en la línea 20 hasta el que introduce en la línea 30. He dado a cada variable un nombre que es una abreviación de lo que contienen. Como puede ver el programa simplemente calcula el número de meses multiplicando por 12 el número de años, los días multiplicando por 365 el número de años, y así sucesivamente.

La línea 90 es la más difícil de introducir y requiere plena concentración para ser entendida completamente. En esencia, lo que hace es escribir en la pantalla el número de años, meses, días, etc., que se han calculado. Pero como he pensado que será mucho más fácil leer la respuesta, si ésta está en líneas separadas, ha habido que incluir todos estos signos de puntuación: comas, puntos y comas, apóstrofes. El apóstrofe hace que el próximo campo a imprimir lo haga en la próxima línea, por lo tanto, dos apóstrofes saltarán dos líneas. Ahora observe con atención la línea 90 y podrá ver fácilmente por qué cada respuesta se escribe en una línea separada.

Problema: Usando la información que se ha dado en esta última parte, usted debe ser capaz de mejorar un poco el programa. Por ejemplo, el programa no permite tener en cuenta los días, ni los meses de los dos años. ¿Puede añadir algunas líneas para arreglarlo? El programa también ignora los años bisiestos, pues haga algo para que se tengan en cuenta en los cálculos. También puede hacer que calcule el número de segundos que han pasado.

Finalmente, estará un poco sorprendido de haber obtenido los minutos que han pasado en una forma como ésta:

1,986768E + 8

¿Qué significa esto? Pues bien, si usted ha hecho matemáticas, habrá reconocido la **notación científica.** Esta manera de escribir los números divide la información en dos partes, la primera es siempre un número decimal con un solo dígito antes de la coma; la segunda parte, indica la potencia de 10 por la que hay que multiplicar a la primera para obtener el número en cuestión. Por si no ha quedado claro, lo veremos con un ejemplo. Tomemos el número 1000. En notación científica se escribe:

1,0E + 3

Esto significa que hay que multiplicar 1,0 por 10 elevado a tres. La E mayúscula le dice al Spectrum que lo que le sigue es un exponente. El signo más indica que la potencia es positiva. Como probablemente ya sabrá usted, es lo mismo decir que elevamos 10 una potencia X que poner X ceros detrás de un 1. Así 10 elevado a 3 es 10^3 que es 1000. Por otro lado, 10^{-3} es 0,001. ¿Cuántos minutos hemos obtenido antes?, pues bien, hay que multiplicar el número 1,986768 (casi es 2) por un 1 seguido de ocho ceros (ya que tenemos E + 8), es decir, 1,986768 × 100.000.000.

¡GUARDÉMOSLO!

Ahora tiene un programa en la memoria del Spectrum. ¿Cómo guardarlo en un cassette? Después de todo supongo que usted no querrá escribir el programa de nuevo cada vez que vaya a utilizarlo. Es muy simple guardar un programa en el cassette. Primero asegúrese de que dispone del cable con dos enchufes (en cada lado), uno gris y el otro negro. Enchufe el gris (podía haber sido el otro pero he escogido el gris) en la entrada MIC (para grabar) de su cassette: Ahora enchufe el otro gris en la salida marcada MIC que está en la parte de atrás del Spectrum. Asegúrese de que dispone de una cinta virgen y córrala hacia atrás hasta el principio. Ahora córrala un poco hacia adelante hasta que la parte marrón de la cinta esté situada delante de los cabezales (esto puede verlo sacando la cinta). Su cassette probablemente dispone de nivel automático de grabación, pero en caso contrario sitúe el volumen a tres cuartos aproximadamente del máximo. Ahora todo está listo para efectuar la grabación del programa. Con la cinta ya preparada pulse la tecla S y la palabra **SAVE** aparecerá en la pantalla. Después de SAVE el Spectrum espera el nombre de su programa encerrado entre comillas. El comando SAVE debe parecerse a esto:

SAVE "Nombre"

Cuando pulse ENTER el Spectrum escribirá un mensaje en la pantalla que le indica que debe poner su cassette en

marcha para grabar. Seguidamente aparece el mensaje "press any key" que significa que pulse cualquier tecla. Hágalo y en el borde de la pantalla aparecerán líneas rojas y azules corriendo hacia arriba. Esto ocurre dos veces, primero cuando se graba el nombre del programa y luego cuando se graba el programa mismo, pero ahora con líneas amarillas, azules y negras. En ese momento el Spectrum también hace un zumbido. Cuando se termina la grabación aparece el mensaje "0 OK". El programa está ahora guardado en la cinta y ya puede parar el cassette.

Su programa se habrá grabado bien (al contrario que el ZX81, el Spectrum es muy fiable a la hora de grabar programas). Pero además usted puede comprobarlo usando el comando **VERIFY.** Rebobine la cinta hasta el punto en que se encuentra el principio del programa que acaba de grabar. Ahora enchufe dos de las clavijas del mismo color, una detrás del Spectrum en la entrada marcada EAR y la otra en el cassette en la salida marcada EAR o MONITOR, en caso de que no disponga pruebe con la salida para auriculares o altavoz externo. Ahora escriba VERIFY, pulsando CAPS SHIFT y SYMBOL SHIFT al mismo tiempo y luego la tecla R. Detrás escriba el nombre de su programa entre comillas. Este nombre debe ser exactamente el que usted le dio al programa cuando lo grabó, incluso los espacios, las minúsculas y las mayúsculas. Tampoco puede contener más de 10 caracteres. Ahora pulse ENTER y pulse la tecla Play de su cassette. El Spectrum irá mostrando el nombre de todos los programas que encuentre en la cinta. Cuando llegue al programa que se desea verificar mostrará de nuevo aquellas rayas amarillas y azules que aparecieron cuando lo grabó. Cuando haya terminado aparecerá un mensaje que normalmente es el ya conocido "0 OK". Si algo anda mal, probablemente dirá "Tape loading error", en caso de que esto ocurra compruebe que el nombre que ha dado sea el correcto así como el nivel de volumen. Haga la misma operación en caso de que no esté seguro, y si hay problemas, de nuevo, grabe el programa otra vez.

¿Cómo recuperar el programa desde la cinta al ordenador? Bien, esto también es fácil. Esta vez debe usar el comando **LOAD.** Primero rebobine la cinta hasta el principio del programa (sin dunda es mejor que vaya más hacia atrás). Una vez esté la cinta colocada, enchufe las clavijas

en las hembras marcadas EAR (lo mismo que en VERIFY). Ahora pulse la tecla J y la palabra LOAD aparecerá, ponga detrás el nombre exacto del programa entre comillas. Pulse ENTER y ponga en marcha la cassette como si fuera a escuchar lo que hay en ella. El borde de la pantalla se llenará de líneas rojas y azules hasta que el programa cuyo nombre ha dado sea localizado. Cuando esto ocurre el color de las líneas cambia a amarillo y azul. Al terminar aparece el informe "0 OK" y ya puede apagar el cassette. Observe que las líneas que se escriben para verificar (VERIFY) y para recuperar (LOAD) son muy similares:

VERIFY "programa"
LOAD "programa"

Recuerde que su Spectrum es muy exigente: Si ha usado letras mayúsculas cuando hizo SAVE debe también usarlas en LOAD y VERIFY. Lo mismo ocurre con las minúsculas.

SI NO PUEDE RECORDAR EL NOMBRE DEL PROGRAMA

Entonces use esta expresión para recuperarlo:

LOAD ""

No deje espacios entre las comillas. Al pulsar ENTER entonces el Spectrum recuperará el primer programa completo que encuentre. Si no es el que desea entonces pulse NEW (aunque no es necesario) y vuelva a intentar lo mismo con el siguiente. Por supuesto que si no puede recordar el nombre de su programa ni el lugar de la cinta en que éste se encuentra es bastante engorroso ir repitiendo LOAD "" cada vez. Aquí tiene un truco para evitarlo:

Para obtener un catálogo de lo que hay en una cinta, rebobine ésta hasta el principio y escriba lo siguiente:

VERIFY "CAT" (y ponga en marcha la cassette igual que en LOAD)

Usted puede usar cualquier otra palabra que no sea CAT pero que no coincida con el nombre de ningún programa de la cinta. El Spectrum recorrerá la cinta entera mostrando los nombres de los programas, pero sin recuperar ni verificar ninguno de ellos. Una razón por la que he utilizado la palabra CAT para este propósito es que otros ordenadores disponen de un comando CAT que hace exactamente lo mismo.

MERGE

Probablemente ha visto esta palabra en el teclado y se ha preguntado para qué sirve. Pues bien, con **MERGE** usted puede recuperar de la cinta dos o más programas diferentes al mismo tiempo en memoria. Si todos tienen distintos números de línea y juntos no exceden del total de memoria disponible, entonces todos los programas recuperados con MERGE podrán "convivir" juntos en la memoria del Spectrum. Por ejemplo, supongamos que escribimos estos tres programas:

```
1 REM PROGRAM
10 PRINT "Hi"

2 REM PROGRAM 2
20 PRINT "There"

3 REM PROGRAM 3
30 PRINT "Fred"
```

Si recuperamos el programa 1 y luego hacemos MERGE con los programas 2 y 3 (por supuesto habiéndolos grabado previamente en la cinta como programas separados) nos encontremos con esto en la memoria del Spectrum:

```
1 REM PROGRAM 1
2 REM PROGRAM 2
3 REM PROGRAM 3
10 PRINT"Hi"
20 PRINT"There"
30 PRINT"Fred"
```

Si por ejemplo hubiéramos usado la línea 2 en más de un programa, entonces la última línea dos que entrara en memoria provocaría que se borrara la anterior. Por lo tanto, si quiere unir dos programas procure que tengan números de línea diferentes. Un uso muy útil de MERGE es para añadir rutinas cortas al final de la memoria (digamos de la línea 9000 en adelante). Por ejemplo, más adelante en este libro habrá una rutina de renumeración que permitirá cambiar los números de líneas de los programas.

LOS BUCLES - ALGO NUEVO

Volvamos a programar. Pero usted probablemente tenga algún programa en memoria. Para deshacerse de él y empezar de nuevo use **NEW.** El uso de NEW es bastante drástico, así que es importante el vigilar no escribirlo por error. Pulsando la tecla A cuando el cursor está en modo K, aparecerá NEW en la pantalla. Si luego pulsa ENTER la pantalla ennegrecerá durante unos instantes y luego estará limpia, con el mensaje "Sinclair Research" que aparece cuando se enchufa. Por lo que al programa se refiere, el uso de NEW equivale a desenchufar y volver a enchufar, por lo que hay que tener cuidado cuando quiera utilizar la letra A o el comando STOP. Recuerde: al hacer NEW se pierde el programa.

Ahora vamos a por los bucles. Recordará que hace poco hicimos un programa que escribía varias veces su nombre y edad. Para ello usamos GOTO que puede enviar el programa a una línea anterior. Pero ¿qué pasa si queremos poner un nombre en la pantalla exactamente quince veces? O bien, ¿si queremos que se pregunten siete cuestiones y que el Spectrum almacene las respuestas? Bien, pues para hacer esto usamos las palabras **FOR** y **TO** junto con **NEXT** para crear un bucle. Aquí tenemos un ejemplo:

```
10 REM CONVERTIDOR DE LITROS
20 FOR N = 1 TO 20
30 INPUT "¿CUANTOS GALONES?", G
40 LET L = 4.546*G
50 PRINT G;"␣GALONES SON␣";L;"␣LITROS"
60 NEXT N
```

La línea 10 es una sentencia REM que nos recuerda qué es lo que hace el programa. La línea 20 es el principio del bucle en el que se usan las palabras FOR y TO para decirle al Spectrum cuántas veces queremos que se repita el proceso que está en la línea 30 a 50. Como puede ver he decidido que ocurra 20 veces, ya que en la línea del FOR... TO la variable N toma valores del 1 hasta el 20. Las líneas 30 a 50 comprenden la conversión de galones a litros. En la línea 30 se usa la ya familiar sentencia INPUT con una frase que pregunta cuál es el volumen en galones que se quiere convertir a litros. La línea 40 calcula el número de litros, para ello crea una nueva variable llamada L (por Litros, por supuesto) y le dice al Spectrum, mediante la instrucción LET, que L es igual a G (volumen en galones) multiplicado por 4,546. La línea 50 simplemente escribe el resultado en una frase. La línea 60 es la otra parte del bucle y hace que el valor de N aumente en uno hasta un máximo de 20. Esta línea dice "vuelve a la línea en que está el FOR (la 20 en este caso), cogiendo el próximo (en inglés, NEXT) valor de N a no ser que ya haya llegado al máximo (que en este caso es 20)". Cuando se llega a 20 veces, entonces se habrán convertido 20 valores en galones a su equivalente en litros y el programa se detendrá con el informe:

0 OK, 60:1

Supongo que habrá visto que es muy fácil conseguir que el Spectrum le haga la pregunta una sola vez o 100 veces, o cualquier número de veces que usted quiera. Lo único que hay que hacer es cambiar el 20, que hay al final de la línea 20, por otro número.

EDICIÓN

Vamos a cambiar este número por cinco para ver lo fácil que es hacerlo. La manera más simple es usar una nueva función llamada **EDIT,** que le permite traer una de las líneas del programa a la parte baja de la pantalla, cambiar o borrar cualquier trozo de línea y luego devolverla al programa.

Para encontrar EDIT pulse CAPS SHIFT y la tecla 1 al mismo tiempo. La última línea de programa que escribió,

bajará a la parte inferior de la pantalla. Ahora pulse ENTER y aparecerá un listado del programa sin ningún cambio. Para hacer un cambio en la línea 20, primero tiene que buscar en qué línea está el cursor'>' que estará junto a la última línea que introdujo. Si hacemos EDIT nos bajará la línea que tiene este cursor por eso debemos colocarlo antes en la línea 20. Para hacerlo se usan las flechas de dirección que las encontrará sobre las teclas 5 a 8. De hecho la que tendrá que utilizar es la flecha hacia arriba que se encuentra sobre la tecla 7. Para mover el cursor (de ahora en adelante lo designaré como *cursor de línea actual)* pulse CAPS SHIT y la tecla 7 simultáneamente. Cada pulsación provoca que el cursor de línea actual suba hacia la línea inmediatamente superior. Unas pocas pulsaciones y ya estará sobre la línea 20 (o simplemente mantenga la tecla apretada, pues también dispone de autorrepetición). Antes de bajar la línea 20 haga pruebas moviendo el cursor de línea actual arriba y abajo (con la tecla 6).

Con el cursor de línea actual al lado de la línea 20 pulse CAPS SHIFT y 1 y la línea en cuestión bajará. Ahora, para mover el cursor L sobre la línea pulse CAPS SHIFT y 8 hasta llegar justo detrás del 20. Borre el 20 con CAPS SHIFT y 0 (DELETE). Dos pulsaciones con DELETE y el 20 desaparecerá. Para poner el 5 en sustitución del 20 simplemente escríbalo y pulse ENTER. Inmediatamente aparecerá un nuevo listado en el que la línea 20 estará así:

```
20 FOR N=  TO 5
```

Haga RUN del programa para comprobar que ahora el bucle sólo se realiza 5 veces.

Aquí hay dos consejos para utilizar EDIT:

CONSEJO

Primero: Si usted tiene una línea muy larga y acaba de cambiar algo del final de la línea, pero luego se da cuenta de que hay un error en el principio entonces es más rápido pulsar ENTER y volver a hacer EDIT que no correr el cursor hacia atrás con la tecla 5. Por supuesto que esto

sólo funciona si el error que hay le permite pulsar ENTER sin que aparezca el error de sintaxis.

CONSEJO

Segundo: Habrá observado que cuando tiene un programa muy largo del cual quiere editar una línea, para colocar el cursor de línea actual en dicha línea puede usar LIST × (donde × es el número de línea). Pero esto presenta el problema de que produce un listado que llena la pantalla y entonces el Spectrum pregunta si queremos continuar ("Scroll"). Para evitar el ser preguntado coloque el cursor en la línea deseada escribiendo un número de línea menos y pulsando ENTER. Esto suponiendo que dicho número no exista pues sino se borraría (de todas maneras es improbable si usted me ha hecho caso y ha numerado sus líneas de 10 en 10). Esto produce un listado con el cursor de línea actual en la línea deseada y sin la pregunta "scroll". Por ejemplo:

```
800 PRINT A$
810 LET C = S + 2
```

Si tiene estas líneas en medio de un programa y ha visto que C debe ser igual a "S + 3" y no a "S + 2" entonces escriba:

```
809          (y pulse ENTER)
```

Como que la línea 809 no existe entonces el cursor de línea actual se coloca en la siguiente que es la 810 que ya puede ser bajada rápidamente con EDIT.

SI... ENTONCES...

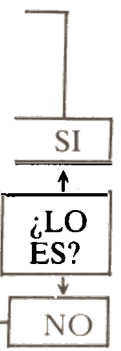
¿Qué ocurre si usted no está seguro de cuántas veces quiere convertir galones a litros? O de ¿cuántas veces quiere repetir cualquier proceso? Pues bien, es muy fácil resolver esto utilizando las **sentencias condicionales.** Para ello

se usan las instrucciones **IF** (que en español significa "si" condicional) y **THEN** (que significa "entonces"). Estas dos palabras clave junto con GOTO resuelven el problema fácilmente. Observe esta versión del programa de conversión, que esta vez convierte grados Fahrenheit a grados Centígrados.

```
10 REM CONVERTIDOR DE TEMPERATURA
20 INPUT "GRADOS FAHRENHEIT?", F
30 LET C = 5/9*(F — 32)
40 PRINT "LA TEMPERATURA EN
 CENTIGRADOS ES DE ";C;" GRADOS"
50 INPUT "OTRA CONVERSIÓN?", Q$

60 IF Q$ = "YES" THEN GOTO 20

70 STOP
```



En el lado derecho he puesto un pequeño esquema para que vea qué es lo que el Spectrum decide en cada momento. Del mismo modo que el programa anterior, éste pregunta por la temperatura en grandos Fahrenheit, y luego, en la línea 30 calcula su equivalente en grados Centrígrados. La respuesta se imprime en la lína 40. Observe que he vuelto a utilizar "␣" para indicar los espacios.

La línea 60 es nuestra sentencia condicional y actúa según la respuesta que demos en la línea 50 en que se pregunta si queremos hacer otra conversión. En respuesta usted puede contestar YES o NO. Esta respuesta es asignada a la variable Q$ (recuerde que el signo "$" le indica al Spectrum que la variable es de cadena, es decir que contiene palabras o letras). En la línea 60 se pregunta si Q$ es YES o no lo es, y en caso de que lo sea vuelve a empezar (THEN GOTO 20).

Si usted no responde YES, entonces continúa y se encuentra en la línea 70 con la instrucción **STOP,** que como ya puede suponer produce la detención del programa. De hecho el programa se hubiera detenido de todas maneras, ya que no había más líneas. Al usar STOP se produce el siguiente mensaje:

9 STOP statement, 70:1

En algunos ordenadores es obligatorio terminar un programa con STOP o con END pero el Spectrum no lo requiere. De todos modos, STOP puede ser muy útil en algunas ocasiones, pues no sólo provoca la detención del programa sino que también nos dice dónde tuvo lugar (en este caso ha sido en la línea 70, primera parte).

Recordará que anteriormente ya he dicho que los ordenadores pueden ser muy exigentes. Al comparar cadenas tal como hemos hecho en el programa es uno de los puntos en que la exigencia del Spectrum puede llegar a confundirle. Por ejemplo, si en vez de escribir "YES" hubiera escrito "^YES", es decir, con un espacio antes de la palabra, usted o yo hubiéramos dicho que se trataba de la palabra YES mientras que el Spectrum que es muy estricto diría que no lo es. Para comparar la respuesta con las letras YES el Spectrum coloca una al lado de otra para ver si son exactamente lo mismo:

Su respuesta: | | Y | E | S | ''

Comparada con: | Y | E | S |

Ya puede ver que mientras cualquiera de nosotros diríamos que las dos palabras son lo mismo, para el Spectrum no se parecen en absoluto ambas cadenas de caracteres. Lo primero que observa es que la respuesta tiene longitud cuatro mientras que YES tiene longitud tres. Y además, donde debería encontrar una Y se encuentra con que hay un blanco, donde debería haber una E encuentra una Y, etc. En definitiva intente recordar siempre lo exigentes que son los ordenadores o perderá la cabeza enfrentándose con problemas de fácil solución.

¿Qué pasa si usted, no contesta ni YES ni NO? Pues bien, la sentencia condicional de la línea 60 sólo comprueba si la respuesta es YES o no es YES, por lo tanto, cualquier otra respuesta, incluso ninguna respuesta hará que el Spectrum actúe como si la respuesta fuera NO. Piense cómo lo haría para añadir una línea, la 65 por ejemplo, que imponga la condición de que si la respuesta no es ni YES ni NO vuelva a efectuarse la pregunta.

Hemos visto cómo hacer que un bucle realice un proceso un cierto número de veces y también cómo hacerlo de manera que finalice en el momento en que queramos. Pero supongamos, por ejemplo, que queremos escribir un símbolo en la pantalla un cierto número de veces pero cada símbolo separado por una distancia. Este problema puede ser resuelto bajo algunas circunstancias usando la instrucción **STEP** en los bucles FOR NEXT. Aquí tenemos a STEP en acción:

```
10 BORDER 4
20 INK 2
30 PAPER 6
40 CLS
50 FOR N = 1 TO 31 STEP 2
60 PRINT TAB N; "↑";
70 NEXT N
```

Haciendo RUN de este programa se obtiene una línea de flechas en la parte de arriba de la pantalla, separadas cada una por un espacio. El secreto de todo esto está en el uso de la función **TAB.** TAB x, hace qué lo que se imprima lo haga en la columna x (x espacios a la derecha) de la línea en que se imprime. El valor máximo de x es 31, pues las columnas se numeran desde 0 a 31. Si usted da un número mayor de 31 entonces el Spectrum le resta 32, de manera que siempre se imprime en la línea en cuestión. Así, TAB 32, no imprimirá en la primera posición de la línea siguiente sino en la primera de la línea presente, a condición de que no haya nada escrito en dicha línea.

```
PRINT TAB 32 = PRINT TAB 0
PRINT TAB 33 = PRINT TAB 1
```

Continuemos, cambie la línea 50 por:

```
50 FOR N = 1 TO 31 STEP 4
```

Haga RUN del programa y compruebe lo que ocurre al cambiar el paso (STEP en inglés).

Para colocar algo en una posición determinada de la

pantalla también puede usar **PRINT AT,** que tiene la ventaja de que puede imprimir en cualquier columna de cualquier fila, y no sólo en cualquier columna de la fila en que se está imprimiendo como TAB. Aquí tiene como una sentencia PRINT AT reemplaza a la sentencia TAB del programa anterior:

```
60 PRINT AT 0,N;"↑"
```

PRINT AT, siempre va seguido de dos números separados por una coma. El primero especifica la fila (de 0 a 22) y el segundo, la columna (de 0 a 31). Como puede ver, aunque en el programa se ha especificado la línea 0, hubiera sido muy fácil, poner cualquier otra línea cambiando el primer número detrás de PRINT AT. Intente hacer un bucle FOR NEXT que imprima una línea de flechas en cualquier otra fila de la pantalla.

¿Puede imaginarse el efecto que causaría el borrar el punto y coma de la línea 60? Bórrelo usando EDIT, para ver lo que pasa.

Para los incansables, aquí hay otra versión del programa que usa TAB otra vez. Mire si puede ver lo que hace.

```
 5 LET X = 1
10 FOR N = 1 TO 31 STEP X
20 POKE 23692, -1 (esto provoca un scroll automático)
30 PRINT TAB N;"↑"
40 NEXT N
50 LET X = X + 0.5
60 GOTO 10
```

Se encontrará con que tiene que parar el programa con **BREAK,** ya que la línea 60 provoca un bucle continuo hacia la línea 10. Para hacerlo pulse CAPS SHIFT y la tecla SPACE/BREAK simultáneamente.

LÍNEAS CON VARIAS INSTRUCCIONES

En el programa de la línea de flechas, empezamos cambiando la tinta (INK), el papel (PAPER), y el borde (BORDER). En total, este proceso, junto con el obligado CLS nos ocupaba cuatro líneas de programa. Es posible

hacerlo todo en una línea. El Spectrum permite varias instrucciones en una sola línea separadas por dos puntos ":". En efecto, cada instrucción es como una línea de programa, pero distintas instrucciones tienen el mismo número de línea. Observe cómo se coloca en una línea las asignaciones de color del programa que hablábamos:

```
10 INK 2:PAPER 6: BORDER 4: CLS
```

Recuerde que este tipo de sentencias no es *exactamente* lo mismo que tener varias líneas en una. No se puede por ejemplo ir (GOTO) a una instrucción de una línea múltiple sin ejecutar las otras.

Hay que vigilar que tipos de instrucciones se mezclan en una línea. No todas las combinaciones funcionarán satisfactoriamente. Por ejemplo, es una buena idea colocar sentencias REM que indiquen lo que hace una línea o una parte del programa, pero nunca coloque una sentencia REM al principio de una línea múltiple, pues el resto de la línea no se ejecutará nunca ya que será tomado como parte del comentario. Compare por ejemplo estas dos líneas:

```
10 REM no lo haga así: PRINT "MAL"
20 PRINT "BIEN": REM así se debe hacer
```

Si hace RUN de este programa, sólo se imprimirá "BIEN". Las sentencias condicionales, también pueden causar problemas. No coloque sentencias IF... THEN al principio o en medio de líneas múltiples, ya que el Spectrum ignorará las instrucciones que le siguen si la condición no se cumple. Compare éstas:

```
10 IF 2>3 THEN PRINT "YES" :PRINT "hi"
20 PRINT "hello": IF 2<3 THEN GOTO 40
30 PRINT " at 30":STOP
40 PRINT "at 40" :STOP
```

Como verá al ejecutar el programa, sólo se imprime la palabra "hello" y el programa se para con el STOP de la línea 40; demostrando que si la condición se cumple, entonces se ejecutan las demás instrucciones de la línea. Tenga en cuenta que el uso de las líneas múltiples dificultará el

seguimiento del programa para otras personas o incluso para usted al cabo de un tiempo, así que úselas con moderación y claridad.

IR Y VOLVER

Hemos usado GOTO en varias ocasiones, pero tiene un hermano llamado **GOSUB,** que hace una función similar y puede ser muy útil en ocasiones. Primero veamos a GOTO de nuevo en acción. Escriba el siguiente programa y ejecútelo:

```
10 GOTO 100
20 GOTO 200
30 GOTO 300
40 STOP
100 PRINT "EN LINEA 100"
110 GOTO 20
200 PRINT "EN LINEA 200"
210 GOTO 30
300 PRINT "EN LINEA 300"
310 GOTO 40
```

Al hacer RUN deberá obtener:

```
EN LINEA 100
EN LINEA 200
EN LINEA 300
```

Ahora crearemos una rutina muy similar usando GOSUB y RETURN en lugar de GOTO:

```
10 GOSUB 100
20 GOSUB 200
30 GOSUB 300
40 STOP
100 PRINT "EN 100": RETURN
200 PRINT "EN 200": RETURN
300 PRINT "EN 300": RETURN
```

La diferencia más importante es que GOSUB salta a un número de línea y continúa el programa hasta que encuen-

tra un RETURN (en español "vuelta atrás"). En cuanto el RETURN es localizado, el programa vuelve a la línea siguiente a GOSUB. Dado que RETURN no incluye ninguna dirección de vuelta, usted puede intercambiar las líneas 10 y 20 y el programa trabajará igualmente. Use EDIT para efectuar el cambio y cuando ejecute la nueva versión obtendrá:

```
EN 200
EN 100
EN 300
```

Con esto se demuestra que efectivamente el GOSUB hace que el Spectrum guarde la dirección de retorno. Aquí tenemos un programa que usa GOSUB de un modo más práctico:

```
10 BORDER 2: INK5: PAPER 1: CLS
20 INPUT "¿Cuanto son 3 × 6?", ×
30 IF × 18 THEN GOSUB 100
40 IF × <> 18 THEN GOSUB 200
50 INPUT "¿cuánto es 9/3?", y
60 IF y = 3 THEN GOSUB 100
70 IF y <> 3 THEN GOSUB 200
80 STOP
100 PRINT "BIEN": RETURN
200 PRINT "MAL": RETURN
```

Hay varias cosas nuevas en este programa, pero escríbalo exactamente y ejecútelo para ver qué pasa cuando usted contesta correctamente a la pregunta "¿cuánto es 3 × 6?" y cuando contesta mal.

Concentrémonos en los GOSUBs. Como puede ver, GOSUB 100 es la subrutina de la "respuesta correcta" (GOSUB significa "ve a la SUBrutina"), y GOSUB 200 es la subrutina de la respuesta mala. No cuesta mucho de ver que hubiera sido bastante difícil escribir este programa usando GOTOs en vez de GOSUBs, sino lo cree inténtelo. Volveremos a los **operadores lógicos** más tarde, pero ahora necesita saber que el símbolo "< >" significa "es diferente de" o bien "no igual a"; aquí, se usa para ver si las respuestas son correctas comprobando si "x" e "y" valen 18 y 3 respectivamente. A propósito, observe que GOSUB es uti-

lizado como parte de una **sentencia condicional,** un uso muy frecuente de esta particular instrucción. Las sentencias IF... THEN pueden tomar muchas formas:

```
IF A = B THEN LET A = 20
IF S$ = "para" THEN PRINT "parado"
IF a > 6 THEN RUN ('>' significa "mayor que")
```

Y más aun, de hecho, se pueden poner ecuaciones enteras en este tipo de sentencias. Intente escribir el programa anterior de un modo distinto.

LET

Ya nos encontramos con la instrucción LET en aquel programa que convertía temperaturas, y vimos que se utilizaba para crear una nueva variable. Vuelva a él un momento y échele una rápida mirada. En este caso se creó una nueva variable C que era el resultado de una operación matemática en la que entraba la variable F. Se pueden crear variables mucho más sencillas. Observe cómo se usa LET de distintas maneras:

```
10 LET A = 24
20 LET N$ = "FRED"
300 LET K = M + 34
26 LET T = T + 1
9999 Let A$ = B$ + C$
```

En todos los casos LET hace lo mismo, si pudiéramos decirlo en palabras sería algo así como "coloca en... el resultado de...". Diciéndolo así, queda mucho más claro que decir "haz... igual a...". Esto puede parecer complicado, sobre todo al ver el ejemplo de la línea 26. ¿Cómo puede ser T igual a ella misma más 1? La respuesta es que LET crea (o vuelve a crear) una variable, la que está en la izquierda, con el resultado de las operaciones realizadas en la parte derecha del "=".

El ejemplo de la lína 26 es una manera muy común de incrementar una variable. Lo que hace es reemplazar T por lo que valía T más 1.

El ejemplo dado en la línea 10 consiste simplemente en asignar un número a una *variable numérica.* En tales casos la frase correcta para LET sería: "La variable... toma el valor de..." Algo semejante ocurre en el ejemplo siguiente con una variable de cadena, N$ que toma el valor "FRED". También se puede crear una nueva variable dependiendo de una, ya conocida (vea la línea 300); y asimismo se puede crear una nueva con la suma de dos, ya conocidas. En la línea 9999, se muestra un ejemplo de lo que llamaremos **concatenación.** Que significa que se unen dos cadenas para formar otra más larga. Veamos un ejemplo de esto:

```
10 LET B$ = "HI THERE"
20 LET C$ = "MAUD"
30 LET A$ = B$ + C$
40 PRINT A$
```

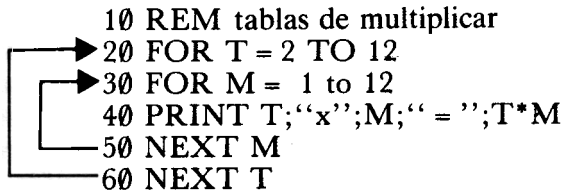
La línea 10 hace que la variable B$ tome el valor "HI THERE", la línea 20 hace C$ igual a " MAUD" (¡atento al espacio!), y la línea 30 hace A$ igual a la suma de las otras dos. Al contrario que con los números, la suma de cadenas es la unión literal de las dos partes. Si escribe este programa y lo ejecuta, obtendrá:

HI THERE MAUD

La concatenación es muy útil a veces, y volveremos a ella muchas veces a lo largo del libro.

BUCLES DENTRO DE BUCLES

Hemos visto cómo usar los bucles FOR NEXT para repetir una cosa un número determinado de veces. Hay ocasiones en las que es necesario poner bucles dentro de otros bucles. Un ejemplo muy claro es cuando queremos que el Spectrum escriba las tablas de multiplicar. Para cada tabla (haremos desde el 1 hasta el 12) hay que calcular los múltiplos de cada número, por lo tanto necesitamos poner otro bucle. Aquí tiene el programa de las tablas de multiplicar:



Observe que el bucle interior debe ser completado a cada paso del bucle exterior. No sería bueno intercambiar las líneas 50 y 60.

CONJUNTOS

Un uso muy bueno para los bucles FOR NEXT consiste en asignar valores a los conjuntos (vectores y matrices para los matemáticos). Aquí, un conjunto es como una lista de variables puestas todas en un mismo paquete, por lo que reciben el mismo nombre, y se distinguen entre sí por uno o varios números que les siguen y que se llaman subíndices. Por ejemplo, en vez de tener las variables A, B, C y D se puede crear un conjunto A (4). Para hacerlo hay que usar el comando **DIM,** que es una abreviación de DIMensión, para decirle al Spectrum cuántos elementos tendrá el conjunto (o cuántas variables habrá en el cajón). Este es un típico comienzo de programa:

```
10 DIM A(20)
20 FOR N = 1 to 20
30 PRINT "Nota para el estudiante"; N; "?"
40 INPUT A(N)
50 PRINT A(N)
60 NEXT N
```

Este programa crea un conjunto llamado A de 20 elementos. Seguidamente pregunta las notas de unos estudiantes (podían haber sido las puntuaciones de un equipo de fútbol, o cualquier otra cosa), escribiendo al lado el valor de cada respuesta (línea 50).

Se puede pensar que un conjunto está hecho de diversas variables. Por ejemplo, en este caso puede pensar que el conjunto A está hecho de las variables A(1), A(2)... hasta A(20). Como puede ver a cada uno de estos valores se le asigna la nota de un estudiante. Tiene muchas ventajas el

poder crear conjuntos como éste, en lugar de tener que crear un número elevado de variables. Es importante dominar el modo de darles valores como hace el programa anterior. Otra ventaja es que un conjunto puede ser manipulado mucho más fácilmente que una serie de variables. Por ejemplo, aquí tiene un programa que le pide 5 números cualesquiera y luego los imprime en orden descendente:

```
5 DIM A(5)
10 FOR N = 1 TO 5
20 INPUT A(N)
30 NEXT N
40 FOR X = 1 TO 5
50 FOR Y = 1 TO 4
60 IF A(Y)<A(Y + 1) THEN GOSUB 200
70 NEXT Y
80 NEXT X
90 FOR J = 1 TO 5
100 PRINT A(J);"^";
110 NEXT J
120 STOP
200 LET T = A(Y)
210 LET A(Y) = A(Y + 1)
220 LET A(Y + 1) = T
230 RETURN
```

Esto puede parecer un poco complicado, pero vamos a recorrerlo línea por línea. Las primeras líneas crean la variable A, dimensionada con cinco elementos y hacen un bucle para la introducción de los números. Las líneas 40 a 80 hacen la comprobación de si A(1) es mayor que A(2), A(2) mayor que A(3), etc. Si algún número no sigue el orden descendente entonces se va a la subrutina de la línea 200 donde los dos números intercambian sus puestos en el conjunto. Para ver cómo este proceso conduce a que los números queden ordenados tal como queremos, vamos a seguir dos valores a través de todo el proceso. Supongamos que A(1) es 6 y que A(2) vale 11. En la línea 60, a la primera vuelta del bucle el programa detecta que A(Y) (aquí vale 6) es menor que A(Y + 1) (que aquí vale 11), y, por lo tanto, salta a la línea 200. En esta línea, se crea lo que se llama una **variable falsa** y se le asigna el valor de A(1). Así que T = 6. Esto se hace para retener temporal-

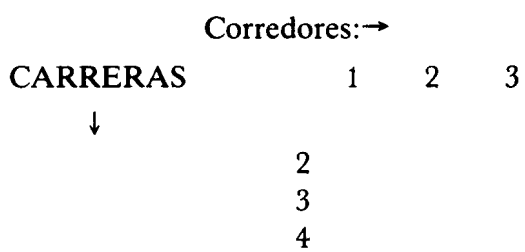
mente el valor de A(1) para poder intercambiar los valores de A(1) y A(2). En la línea 210 A(1) toma el valor de A(2) [por lo tanto ahora A(1) vale 11], y en la línea 220 A(2) toma el antiguo valor de A(1) al hacerlo igual a T. Así se llega al RETURN de la línea 230 con los dos elementos del conjunto cambiados de lugar.

Observará que la línea 50 hace un bucle desde 1 hasta 4 y no hasta 5. ¿Por qué es correcto? Si hubiera tomado valores desde el 1 hasta el 5 llegaría un momento en que la línea 60 intentaría comparar A(5) con A(6), pero A(6) no existe, y el programa se detendría con un mensaje de error. Hablando de detenciones observe el STOP estratégicamente colocado para evitar que el programa continúe ejecutando la subrutina una vez haya terminado lo anterior. Por último, intente averiguar usted mismo por qué únicamente se necesita recorrer el conjunto 5 veces comparando sólo los valores adyacentes para llegar a ordenarlos de mayor a menor.

Con el Spectrum también es posible construir conjuntos multidimensionales. Estos conjuntos se escriben como esto: A(3,4). En este caso, el conjunto es como un tablero de ajedrez rectangular de 3 de ancho por 4 de alto:

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

Como puede ver, el conjunto tiene 12 elementos, que pueden ser pensados como los nudos de una red de 3 × 4. Para ver un uso inmediato de los conjuntos de dos dimensiones piense en tres corredores que participan en cuatro carreras. La red podría ser algo como esto:



A(1,1) sería la posición del corredor 1 en la carrera 1. A(1,2), su posición en la segunda carrera. Y así seguiríamos para todos los corredores, y todas las carreras.

También se pueden dimensionar conjuntos más grandes. Por ejemplo, el conjunto A(2,2,2,2) es perfectamente aceptable. El único límite es la cantidad de memoria de que dispone en su Spectrum. Un Spectrum de 16K no podrá tener tantos conjuntos como el de 48K. Cuando recibe un comando DIM, el Spectrum ya reserva el espacio para todos los elementos del conjunto. Así, por ejemplo, en el Spectrum de 48K usted puede entrar el comando DIM A(8000), pero un DIM A(9000) dará como resultado el mensaje de error "out of memory", que significa que se ha excedido la capacidad de la memoria.

CONJUNTOS DE CARACTERES

Del mismo modo que hemos creado conjuntos cuyos elementos son números, se pueden crear conjuntos de caracteres, palabras y frases. Lo mismo que con las variables de cadena, estos conjuntos son reconocidos por el símbolo del dólar:

```
DIM A$(30)
DIM C$(5,10)
```

Tomemos como ejemplo el conjunto C$(5,10) que es de dos dimensiones. Este conjunto puede pensarse como si estuviera hecho de 5 líneas de 10 caracteres, lo mismo que si tuviéramos 5 variables de cadena, sólo que ninguna de ellas puede exceder, en este caso, de 10 caracteres:

```
C$(1,1) C$(1,2) C$(  3)      C$(1,10)
C$(2,1) ..........           C$(2,10)


C$(5,                        C$(5,10)
```

Un conjunto de caracteres de una sola dimensión es muy similar a una variable de cadena, con la única diferencia de que debe tener una longitud fija. Esto puede ser muy

útil, por ejemplo al tomar las respuestas de un jugador en un juego. En el momento en que se dimensiona un conjunto, como A$(3), se crea una cadena A$ de tres espacios. Para probarlo, escriba lo siguiente:

```
10 DIM A$ (3)
20 PRINT "  """;A$;'
```

Ejecútelo, y verá tres espacios entre las comillas, a propósito, ¿sabe por qué hay cuatro comillas a cada lado de A$? Recuerde que lo que hay en el interior de las comillas, se escribe exactamente como está.

La cadena dimensionada de longitud 1 puede usarse para aceptar la respuesta de un jugador en que un solo carácter es necesario, por ejemplo Y en lugar de YES. Recordará que si el Spectrum busca la cadena «YES», no reconoce ni la más mínima variación (incluso YES PLEASE, que significa «sí, por favor», será tomado como NO). Utilizando una variable dimensionada para asignarle la respuesta se puede solventar en parte el problema:

```
10 DIM A$ (1)
20 INPUT "¿DESEA CONTINUAR?", A$
30 IF A$ = "Y" THEN GOTO 50
40 STOP
```

Esto hace que A$ sólo contenga la primera letra de la respuesta. Por lo tanto, si la primera letra de la respuesta es una Y será considerada como YES. Esto no resuelve el problema completamente, pues puede darse una respuesta YES con un espacio delante, o bien una que empiece con Y, pero que su significado sea NO.

MANIPULACIÓN DE CADENAS

¿Se le ocurre algún truco que permita buscar en cualquier lugar de una cadena la palabra YES? Esto sería muy interesante para analizar la respuesta a la pregunta? «¿Desea continuar?». Para resolver esto fácilmente se necesitan algunos conocimientos más sobre las variables de cadena,

también llamadas string (*). Hay un gran número de funciones para el tratamiento de las cadenas. Entre ellas se encuentran **LEN** (que devuelve la longitud de la cadena) y **VAL** (que da el valor que contiene la cadena en caso de que ésta sea un número o una expresión numérica). Es la primera de estas dos, la que usaremos para buscar la palabra YES dondequiera que esté en el interior de una cadena.

Escriba este programa y ejecútelo:

```
10 INPUT A$
20 PRINT LEN A$
```

Cuando ejecute el programa usted debe poner cualquier palabra o frase e inmediatamente le aparecerá escrita en la pantalla la longitud de la cadena que ha dado como respuesta. Por ejemplo, si usted hubiera escrito «buenas noches» el resultado sería 13, que es la longitud de la cadena «buenas noches». Volvamos a nuestro problema de saber si alguien ha incluido la palabra YES en cualquier lugar de la cadena. Observe este programa, a ver si puede averiguar lo que hace. Para ello tenga en cuenta lo que acabo de decir sobre la función LEN.

```
10 PRINT "¿DESEA CONTINUAR?"
20 INPUT A$
30 FOR X = 1 TO (LEN A$ — 2)
40 IF A$ (X TO X + 2) = "YES" THEN GOTO 100
50 NEXT X
60 STOP
100 PRINT "CONTINUACION"
```

Este programa incluye una nueva posibilidad del Spectrum (que es particular del BASIC del Spectrum) en la línea 40. Se trata del troceado de cadenas, y es una herramienta extremadamente útil. Permite tratar o imprimir un trozo determinado de cadena. Aquí tiene las formas que puede tener esta operación:

N.T. El vocabulario informático español, crea palabras como: "variable string", "tratamiento string", etc. que son muy usadas.

```
A$ (1 TO 4)
A$ (1 TO)
A$ (TO 5)
A$ (5 TO)
```

Supongamos que la cadena A$ tiene longitud diez, es decir, se trata de un string de 10 caracteres. Entonces, con el primer ejemplo, se designan los 4 primeros caracteres del string. El segundo, designa a la cadena entera (desde el 1 hasta el final). El tercero, sólo tratará los cinco primeros caracteres. Mientras que el cuarto ejemplo se refiere a los seis últimos (del 5 al 10 ambos inclusive). Para verlo en acción escriba lo siguiente:

```
10 INPUT A$
20 PRINT A$(1 TO)
30 PRINT A$ (1 TO 4)
40 PRINT A$(TO 5)
50 PRINT A$ (5 TO)
```

Veamos de nuevo el programa que nos permitía detectar la palabra YES en una respuesta. Como puede ver, la línea 40 actúa de modo que toma grupos de tres letras adyacentes y los compara con la cadena «YES». Si uno de estos grupos de caracteres adyacentes es «Y», «E» y «S» entonces salta a la línea 100, donde continuará el programa, en caso de que termine el bucle sin que esto suceda, el programa se detiene en la línea 60. Observe que el límite al que puede llegar la variable de control del bucle (X), viene determinado por la longitud de la cadena A$, que se calcula mediante la función LEN.

Esta habilidad de poder manipular las cadenas puede tener otros muchos usos, aquí sólo daremos unos cuantos ejemplos, pero con un poco de imaginación, se puede llegar a hacer cosas bastante sorprendentes. Con un programa bastante más complicado, se puede conseguir que el Spectrum examine una frase hecha de un determinado vocabulario, y le conteste después.

El ejemplo que doy a continuación, no es nada complicado, le pide que introduzca dos frases y luego le dice cuál de ellas es más larga.

```
10 REM COMPARACION DE FRASES
20 INPUT "FRASE 1", X$
30 INPUT "FRASE 2", Y$
40 IF LEN X$>LEN Y$ THEN GOTO 70
50 IF LEN Y$>LEN X$ THEN GOTO 80
60 PRINT "¡SON IGUALES!": STOP
70 PRINT "LA PRIMERA ES MAS LARGA" STOP
80 PRINT "LA SEGUNDA ES MAS LARGA" : STOP
```

Observe que LEN también cuenta los espacios. Puede comprobarlo comparando «Buenasnoches» con «Buenas noches».

Este nuevo ejemplo ya es un poco más complicado, ya que compara y ordena tres cadenas en lugar de dos:

```
 10 REM ORDENACION DE PALABRAS
 20 PRINT "ESCRIBA 3 PALABRAS"
 40 INPUT A$, B$, C$
 50 FOR T = 1 TO 3
 60 IF LEN A$ > LEN B$ THEN GOSUB 200
 70 IF LEN B$ > LEN C$ THEN GOSUB 300
 80 NEXT T
 90 PRINT A$: PRINT B$: PRINT C$
100 STOP
200 LET X$ = A$: LET A$ = B$: LE  B$ = X$
210 RETURN
300 LET Y$ = B$: LET B$ = C$: LE  C$ = Y$
310 RETURN
```

Primero de todo, observe cómo en la línea 40 se introducen a la vez tres variables string en un solo comando INPUT. Todo lo que tiene que hacer es separar las variables mediante coma o punto y coma. Si usa el punto y coma, las comillas de la pregunta aparecerán una al lado de la otra, mientras que si usa las comas entonces aparecerán una en cada parte de pantalla. Esto es mucho más fácil que hacer:

```
INPUT A$: INPUT B$: INPUT C$
```

La forma en que trabaja el programa es muy simple, y puede ser ampliado fácilmente para poder hacer frente a problemas más complicados. Las líneas 50 a 80 comparan dos cadenas de las tres que se introducen, si la primera es

mayor que la segunda, entonces se intercambian si no se compara la segunda con la tercera y esto se hace tres veces, hasta que quedan ordenadas de menor longitud a mayor. Los intercambios se hacen de manera semejante al programa que ordenaba números, es decir, utilizando una variable falsa.

Un uso mucho más simple de lo que hemos visto nos lo muestra el siguiente programa:

```
10 INPUT "DIME TU NOMBRE ENTERO", N$
20 FOR X = 0 TO LEN N$
30 PRINT N$ (1 TO LEN N$ — X)
40 NEXT X
```

Que produce cosas como ésta:

```
Fred Bloggs
Fred Blogg
Fred Blog
Fred Blo
Fred Bl
Fred B
Fred
Fre
Fr
F
```

También pueden leerse palabras al revés:

```
10 REM INVERSOR DE PALABRAS
20 INPUT "PALABRA", W$
30 FOR J = LEN W$ to 1 STEP — 1
40 PRINT W$ (J):
50 NEXT J
```

Encontrará divertido ver cómo se escriben nombres al revés, o intentar encontrar palabras que se lean igual del derecho que del revés.

También puede desarrollar todo esto para ayudarle a resolver rompecabezas. Por ejemplo, uno muy común, consiste en averiguar cuántas palabras de tres letras se pueden sacar de otra más grande, o colocar una palabra dentro de otra. Para hacer todo esto, le irá bien saber cómo se

puede conseguir que el Spectrum produzca números aleatorios. Esto se consigue con la función **RND.**

NÚMEROS ALEATORIOS

Los números aleatorios, son el alma de muchos juegos, tal como se ha dicho arriba también pueden ayudar a resolver otros problemas. RND produce números entre 0 y 0,999999999. Intente escribir PRINT RND varias veces (seguido de ENTER) y verá lo que quiero decir. RND se obtiene pulsando la tecla T con el cursor en modo E.

Como puede ver, esto produce números con 10 cifras decimales. Tal como está no es muy útil, pero se puede emplear la función **INT,** para conseguir números ENTEROS. INT es una abreviación de la palabra inglesa INTEGER, que significa entero. La función INT, aplicada a un número decimal, da como resultado los dígitos que hay antes de la coma. Por ejemplo:

```
PRINT INT 6,4356752346 da 6
PRINT INT 234,987987876 da 234
```

Usando el programa siguiente, se obtienen números entre 1 y 6 (por ejemplo, para usarlos como si se tratara de un dado):

```
10 REM TIRAR DADOS
15 DIM Q$(1)
20 LET N = INT (6* RND) + 1
30 PRINT N;",";
40 INPUT "¿OTRA TIRADA?", Q$
50 IF Q$ = "S" THEN GOTO 20
60 STOP
```

Multiplicar RND por un número determinado, devuelve valores entre 0 y dicho número. Así, 6*RND produce números aleatorios, entre 0 y 5,999999999. Para obtener números entre 1 y 6 le añadimos 1 a 6*RND. Finalmente, para generar números enteros entre 1 y 6 le aplicamos la función INT que elimina los decimales. Dado que 6*RND + 1, produce números entre 1,0 y 6,99999999, usando INT se reducen a números entre 1 y 6, no entre 1 y 7 como

usted podría esperar. Recuerde que INT no redondea los números al entero más próximo, sino que simplemente ignora la parte decimal.

Con estos resultados, sabemos lo suficiente para intentar hacer un programa que busque palabras de tres letras que se pueden hacer con las letras de otra más grande. Inténtelo, tomando como base este programa que aquí propongo.

```
10 REM BUSCA PALABRAS
20 INPUT "PALABRA LARGA", W$
30 LET X = INT ((LEN W$*RND) + 1)
40 LET Y = INT((LEN W$*RND) + 1)
45 IF Y = X THEN GOTO 40
50 LET Z = INT ((LEN W$*RND) + 1)
55 IF Z = Y OR Z = X THEN GOTO 50
60 PRINT W$ (X) ;W$(Y);W$(Z)
70 GOTO 30
```

Ante todo observe que las líneas 30, 40 y 50 son muy similares. Le será muy útil utilizar EDIT para cambiar la línea 30, de modo que se transforme en la 40, ya que es mucho más rápido que escribir la línea 40 de nuevo. Lo mismo se puede aplicar para obtener la línea 50.

Todo lo que hace el programa en las líneas 3, 4 y 5, es encontrar tres números aleatorios diferentes entre sí (esto se controla en las líneas 45 y 55) y luego usarlos para escribir tres letras de la palabra larga. Las líneas 30, 40 y 50 también aseguran que el número hallado, se encuentre entre 1 y la longitud máxima de la palabra, para ello, se usa la función LEN.

Una innovación, se encuentra en la línea 55. Aquí, he usado un operador lógico llamado **OR** (que corresponde en español a la conjunción disyuntiva "o"). Esto simplemente hace lo que cabe esperar. Permite al Spectrum volver a la línea 50, tanto si Z es igual a Y *o* si Z es igual a X. Volveremos pronto a hablar de los operadores lógicos.

Habrá podido observar fácilmente, cómo se puede ir reordenando una frase sin parar para encontrar palabras en ella.

Si lo que reordenamos no es una palabra sino una frase, habrá que decirle al programa que ignore los espacios. No quiero ahora alargar este programa, pero le voy a dar unos consejos para que lo intente usted mismo. Esencial-

mente, lo que se pretende es generar números aleatorios, que vayan desde 1 hasta la longitud de la frase o palabra (restando los espacios). Luego, debe asegurarse de que cada número aparece sólo una vez. Esto se consigue generando números y comprobando constantemente si ya han aparecido. Cuando se ha generado un número de ellos igual a "LEN W$", entonces ya tenemos suficientes. Lo único que se necesita ahora es escribirlos en el orden aleatorio que ha surgido del proceso y volver a empezar de nuevo para efectuar otro intento.

Como ejemplo final sobre manipulación de cadenas, aquí tiene una para ver del popular juego del "Ahorcado", pero sin gráficos. Contiene diferentes comandos y funciones que no se han visto aún, pero intente escribirlo para ver lo que hace.

```
   5 REM EL AHORCADO
  10 DIM Q$(1)
  15 CLS
  20 LET X=INT (5*RND)+2
  30 RESTORE 100*X
  40 READ W$
  45 PRINT AT 10,5;
  50 FOR Y=1 TO LEN W$
  60 PRINT "-";" ";
  70 NEXT Y
  75 LET T=0
  80 INPUT "PRUEBA !";L$
  81 LET T=T+1: IF T=LEN W$+4 THEN  GO T
O 1000
  85 IF L$=W$ THEN  GO TO 2000
  90 FOR Z=1 TO LEN W$
 100 IF L$=W$(Z) THEN  GO TO 130
 120 NEXT Z: PRINT AT 2,5;"LO SIENTO, NO
": GO TO 80
 130 PRINT AT 10,3+(2*Z);L$
 140 GO TO 80
 200 DATA "CAMPO"
 300 DATA "MUELA"
 400 DATA "RATON"
 500 DATA "LOGICA"
 600 DATA "BIDON"
1000 REM PERDER
1010 PRINT "LO SIENTO, PERDISTE"
```

```
1020 PRINT "ERA ";W$
1030 INPUT "SIGUES ?";Q$
1040 IF Q$="S" THEN  GO TO 10
1050 STOP
2000 REM GANAR
2010 PRINT "SI ES ";W$
2020 INPUT "OTRA ?",Q$
2030 IF Q$="S" THEN  GO TO 10
2040 STOP
```

Lo que le será más difícil de entender es la parte en que aparecen los comandos READ, DATA y RESTORE. Juntos, proporcionan una gran facilidad para la obtención de información.

READ, DATA Y RESTORE

Existen tres funciones relacionadas que le permiten al Spectrum almacenar datos (en inglés, **DATA**) en una parte separada del programa, después se leen (en inglés, **READ**) los datos y se pueden asignar a conjuntos o bien a variables simples con gran facilidad y versatilidad. Aquí hay un ejemplo en el que se usan los comandos READ y DATA:

```
10 DIM A(10)
20 FOR X = 1 TO 10
30 READ A(X)
40 NEXT X:STOP
60 DATA 23,4,53,8,65,2,0,231,5,0
```

Este programa asigna el número 23 a la variable A(1), 4 a A(2), 53 a A(3), etc. En la línea 30 el comando READ le dice al Spectrum que busque la primera línea que empiece por DATA (en este caso la 60) y que lea de ella el primer trozo, este primer trozo, es asignado a la variable A(1). Cuando el bucle FOR NEXT da una nueva vuelta, y se carga a la variable de control X con su próximo valor, se encuentra de nuevo con la instrucción READ A(X). Esta vez, el siguiente dato de la línea DATA es asignado a la variable A(2), y así continuamente. Cada vez que el programa pasa por la línea READ el Spectrum incrementa en uno un contador interno para controlar qué trozo o ítem

de datos es el próximo en la lista DATA. Las líneas DATA, se pueden colocar en cualquier parte del programa, pero es aconsejable colocarlas junto a la instrucción READ o bien marcarlas con una sentencia REM por razones de claridad.

RESTORE, es la otra palabra que va con READ y DATA y hace que el contador interno vuelva de nuevo al principio de la línea DATA. Aquí hay un ejemplo en el que se usa RESTORE:

```
  5 REM READ A$
 10 DIM A$(12,5)
 20 FOR X = 1 TO 12
 40 READ A$(x)
 50 IF X/4 = INT (X/4) THEN RESTORE
 60 NEXT X
 70 REM PRINT A$
 80 FOR X = 1 TO 12
 90 PRINT A$(X);" ";
100 NEXT X
200 DATA "FRED", "BERT", "MARY", "JANE"
```

Como puede observar, los distintos datos, se separan por comas en la línea DATA, y pueden ser tanto números como cadenas de caracteres. Pero recuerde que la instrucción READ, debe leer datos numéricos, sobre variables o conjuntos numéricos, y datos string o alfanuméricos, sobre cadenas. Esto no puede ser de otra manera, y obtendrá el mensaje "Nosense in BASIC", indicándole el error. Cuando ejecute el programa obtendrá:

```
fred bert mary jane fred be
rt mary jane fred bert mary
jane
```

Como puede ver cada dato se lee tres veces sobre el conjunto A$ que contiene doce cadenas de 5 caracteres cada una. Cada vez que se cumple la condición de la línea 50 es que se ha terminado de leer todos los campos de DATA (esto ocurre cada 4 READs), y el contador interno se vuelve a colocar marcando al primer dato ("fred").

RESTORE también puede llevar un número detrás, que le indica el número de línea en que se encuentra DATA. Aquí tenemos un ejemplo:

```
10 RESTORE 100
20 READ X
30 RESTORE 200
40 READ Y
50 RESTORE 300
60 READ z
70 PRINT X;",";Y;",";Z
80 STOP
100 DATA 23,43,54
200 DATA 1,0,3
300 DATA 200,300,400
```

Una vez ejecutado se obtiene:

23,1,200

Observe cómo los otros datos de las líneas 100, 200 y 300 son ignorados completamente. Para ver más claro el efecto que produce el hecho de colocar un número de línea detrás de RESTORE, pruebe a borrar las líneas 10, 30 y 50, y vea lo que ocurre.

Ahora ya tiene los conocimientos suficientes para ver cómo actuaban READ, DATA y RESTORE en el programa del Ahorcado. Las palabras a adivinar estaban almacenadas como cadenas en las líneas DATA (de la 200 a la 600) y la línea a la que leía READ se determinaba aleatoriamente mediante la instrucción RESTORE 100 X en la línea 30 (siendo X un número aleatorio entre 2 y 6). Para añadir nuevas palabras para adivinar, sólo tiene que cambiar la línea 20, aumentando el rango de los números aleatorios X que se generan (hasta 7 para añadir una palabra más, hasta 8 para otra, etc.). La nueva palabra debe ser colocada entre comillas detrás de una instrucción DATA en la línea 700, la otra en la 800, etc. Para aumentar la dificultad, puede hacer que un amigo o familiar introduzca las palabras de manera que usted no las vea. También puede alterar la línea 81 para que el programa varíe el número de intentos permitidos. En este momento permite dos intentos más que el número de letras que tiene la palabra.

La línea DATA puede situarse en cualquier parte del programa, incluso antes de la instrucción READ que la ha de leer. Si hay más líneas DATA de las que tiene que leer el programa no ocurre nada. pero si en cambio se colocan

más READs que DATAs obtendrá un mensaje de error indicándole en qué línea READ se han excedido los datos.

Existe un uso para READ y DATA, que una vez aprendido le servirá para crear los **caracteres definidos** por el **usuario,** que se examinan en el próximo capítulo.