CAPÍTULO QUINTO

Algunos cabos sueltos

ES LÓGICO

En muchas ocasiones, nosotros queremos comparar un número con otro, o una palabra con otra. Los símbolos necesarios para tales comparaciones son: "<", ">", "< =", "> =", "< >"; que respectivamente significan: "es menor que", "mayor que", "menor o igual que", "mayor o igual que" y "distinto que". Su utilidad está muy clara.

```
10 INPUT "ESCRIBA UN NUMERO"; A
20 LET B = 100
30 IF A > B THEN PRINT "DEMASIADO GRANDE"
40 IF A < B THEN PRINT "DEMASIADO PEQUEÑO"
50 GOTO 10
```

Esto es la esencia del juego "Adivine mi número" (aunque usted podría naturalmente añadir la condición de "si A = B entonces..." o bien hacer que B sea un entero cualquiera usando RND). El uso de "menor o igual" y de "mayor o igual" tampoco presenta ningún problema. Estos signos, también se pueden usar para comparar palabras, en este último caso el orden tomado es el alfabético. Aquí tiene algunos ejemplos:

'A' es menor que 'B'
'Piedra' es mayor que 'Pera'
'ZZ' es menor que 'ZZZ'
'XY' es mayor que 'XD'

Como puede ver, si se comparan dos letras, la menor es la primera en el orden alfabético. Si las cadenas tienen más de una letra, primero se comparan entre sí las dos primeras, luego las dos segundas, y así hasta que una de ellas es mayor que la otra o bien se terminan las cadenas. Esto puede utilizarse para que la respuesta a una línea INPUT sea justamente una letra del alfabeto:

```
10 INPUT "UNA LETRA"; A$
20 IF A$< "A" OR A$> "Z" AND A$< "a" OR
   A$> "z" THEN GOTO 10
30 PRINT "ESCRIBISTE"; A$
```

He puesto la detección tanto para mayúsculas como para minúsculas, puesto que ya vimos que el Spectrum es muy exigente en este tipo de distinciones. Hay un ejemplo que demuestra claramente esto, cuando se pregunta cuál es el más pequeño de estos dos diablos:

DIABLO y diablo

La respuesta es por supuesto el primero de los dos, ya que las letras mayúsculas tienen el código más bajo que las minúsculas. Con esto se ve que las comparaciones no se efectúan con los caracteres en sí mismos, sino entre sus códigos.

OPERADORES LÓGICOS

En el programa anterior, he usado OR para decir "o si esto es verdadero". Así, la sentencia "IF A = 1 OR B = 2 OR C = 3 THEN PRINT "VERDAD" imprimirá la palabra "VERDAD" sólo si alguna de las tres cosas es verdadera. A esto, los matemáticos lo llaman disyunción. AND es la conjunción, es decir que la condición se cumple si todas sus partes son verdaderas:

IF A = 1 AND B = 2 THEN PRINT "VERDAD"

imprimirá la palabra "verdad" sí y sólo si el valor de A es 1 y el de B es 2. IF A THEN PRINT "VERDAD" imprimirá la palabra "verdad" si el valor de A no es cero. De un modo similar:

sólo imprimirá el número 2 (CHR$ 50) si A vale 1. En caso contrario, no imprimirá nada en absoluto (ni tan sólo un espacio). Esto es muy útil cuando usted desea que pasen cosas distintas según la tecla que se pulsa o bien dependiendo del valor de una variable. Sin embargo, no es posible hacer comparaciones lógicas directamente, tales como PRINT A AND B donde A y B tienen unos valores dados. En el Spectrum, esto provoca la impresión de A si B no es 0, pero *no* efectúa una conjunción (AND) lógica de los dos números. ¿Qué es un AND lógico? Bien, consideremos los siguientes números binarios:

01010101 y 11111111

Efectuar un AND lógico de estos dos números consiste en mirarlos por columnas, dando un 1 si los dos son unos, y cero en caso contrario.

Para efectuarlo, se pone un número debajo de otro:

```
01010101
11111111
--------
01010101
```

Por lo tanto dos unos (1) dan un 1, dos ceros (0) dan un cero, y uno de cada da 0:

```
  10
1|10
0|00
```

Nosotros podemos simular esta operación (que es posible en otros ordenadores) usando las comparaciones de cadenas:

```
 5 REM DECIMAL A BINARIO
10 INPUT N
20 PRINT N
30 LET X = N
40 LET A$ = "00000000"
50 FOR Y = 1 TO 8
```

```
 60 LET A$(Y) = ("1" AND X/(2↑(8 - Y))>=1)
 70 IF A$(Y) = "1" THEN LET X = X - (2↑(8 - Y))
 80 IF X = 0 THEN GOTO 110
 90 IF A$(Y)<> "1" THEN LET A$(Y) = "0"
100 NEXT Y
110 PRINT N; "es"; A$ "EN BINARIO"
```

y:

```
  10 REM PARA REPLICAR 'Y'
  20 INPUT "A?";X
  25 PRINT "A=";X
  30 INPUT "B?";Y
  35 PRINT "B=";Y
  40 DIM A$(8): DIM B$(8)
  50 FOR T=1 TO 8
  60 LET A$(T)="1" AND (X/(2^(8-T)))>=1)
  65 IF A$(T)=" " THEN  LET A$(T)="0"
  70 LET B$(T)="1" AND (Y/(2^(8-T)))>=1)
  75 IF B$(T)=" " THEN  LET B$(T)="0"
  80 LET X=X-(2^(8-T) AND A$(T)="1")
  90 LET Y=Y-(2^(8-T) AND B$(T)="1")
 100 NEXT T
 105 PRINT "A=";
 110 FOR W=1 TO 8
 120 PRINT A$(W);: NEXT W
 125 PRINT : PRINT
 126 PRINT "B=";
 130 FOR W=1 TO 8: PRINT "A Y B=";
 135 DIM C$(8)
 140 FOR J=1 TO 8
 145 LET C$(J)="0"
 150 IF A$(J)="1" AND B$(J)="1" THEN  LE
T C$(J)="1"
 160 PRINT C$(J);: NEXT J
 200 PRINT : PRINT "QUE ES>>";VAL ("BIN
"+C$)
```

Como puede observar las dos hileras de ceros y unos obtenidas al convertir los números decimales en binarios, son comparadas en el segundo programa. El resultado final puede ser pasado a decimal usando la función VAL, que

devuelve el valor numérico de una cadena, si ésta está compuesta de números o de variables. El complementario de VAL es STR$ que convierte cualquier número o línea de dígitos e incluso una ecuación en una cadena, y que, por lo tanto, puede ser manipulada como tal. Pero en nuestro programa no se permite la entrada directa de números en binario como ocurre aquí:

```
10 INPUT "INTRODUZCA EL PRIMER NUMERO BINARIO", N1
20 INPUT "Y EL SEGUNDO", N2
30 LET A$ = STR$ N1
40 LET B$ = STR$ N2
```

El problema es que los ceros de la izquierda serán eliminados al entrar los valores en variables numéricas.

LA IMPRESORA

La impresora Sinclair puede ser un excelente complemento de su Spectrum. Aunque en un principio fue designada para trabajar con el ZX81, también funciona razonablemente bien con el Spectrum. De todos modos, ocurre que el Spectrum la hace ir demasiado deprisa con lo que a veces hay que hacer varias copias para obtener una que esté bien. Pero no hay otro remedio si quiere tener sus programas listados.

El hecho de poseer un listado de su programa le permite revisarlo para buscar y localizar los posibles errores sin el inconveniente de ver sólo una parte al mismo tiempo. Más aún, el tener un catálogo listado de sus programas le permite recordar qué es lo que ha hecho y, quizá con un poco más de trabajo, en qué cinta y lugar los tiene registrados. El tener copias en papel puede ser útil por otras razones, desde copias de los resultados de los cálculos hasta complejos resultados científicos, pasando por el balance mensual de sus cuentas en el banco. Los comandos que hacen funcionar la impresora, son muy claros. LPRINT hace lo mismo que PRINT sólo que escribe la información en la impresora en vez de en la pantalla. LLIST trabaja como LIST, pero el listado lo hace en papel. Sin embargo, hay una ventaja, la impresora no se para y pregunta "scroll". Si sólo quiere una parte del programa listado, no habrá ningún problema dependiendo de qué parte quiera.

Si lo que quiere es el listado desde un número de línea haga LLIST n, donde n es el número de línea. Si en cambio desea una pantalla listada en papel entonces use el tercer comando para la impresora: COPY, que hace una copia en papel de toda la pantalla. Naturalmente, COPY es también muy útil para tener un recuerdo de los dibujos.

Si usted quiere listar menos de una pantalla en la mitad del programa entonces la cosa se complica un poco. Esto se puede hacer usando SCREEN$. Para hacer un listado parcial, lo primero que necesita es poner el listado en la pantalla, y luego escriba esto sin número de línea:

```
FOR A = 0 TO 31: LPRINT SCREEN$(0,A):NEXT A
```

Esto imprime la primera línea de la pantalla. La siguiente se hace de un modo similar cambiando el primer número de los paréntesis de SCREEN$ por un 1, y así sucesivamente. Si usted ya sabe que solamente quiere listar las cinco primeras líneas de la pantalla, entonces incluya esto en su programa:

```
 999 LIST
1000 FOR A = 0 TO 4
1010 FOR B = 0 TO 31
1020 LPRINT SCREEN$ (A,B)
1030 NEXT B
1040 NEXT A
```

Imprimir sólo un trozo de la pantalla es un poco complicado. El problema está en que el Spectrum almacena los caracteres a imprimir en un área de la memoria llamada PRINTER BUFFER, y espera hasta que está llena. Tal como podía esperar esta área mide 256 bytes, es decir, ocho bytes por cada carácter de los 32 que tiene una línea. Así que la intención es de que se imprima una línea cada vez.

Las únicas ocasiones en que el Spectrum no espera a que el BUFFER esté lleno es cuando se introduce un carácter de Newline (ENTER), una coma, un TAB o una comilla, que requieren un salto de línea. Es por esta razón que no podemos definir un trozo para imprimir como éste:

```
10 FOR A = 5 TO 15
20 FOR B = 4 TO 12
30 LPRINT SCREEN$ (A,B);
40 NEXT B: NEXT A
```

Esto simplemente almacena caracteres en el BUFFER de la impresora hasta que termina el programa o se llena con 32 caracteres. Con este programa se puede solventar todo esto:

```
    5 REM LA VENTANA IMPRESA
    6 REM vamos a confeccionar un texto q
ue va a escribirse en un cuadrado defini
do por R1, R2, C1, C2
   10 INPUT "PRIMERA FILA?",R1
   20 INPUT "ULTIMA FILA?",R2
   30 INPUT "PRIMERA COLUMNA?",C1
   40 INPUT "ULTIMA COLUMNA?",C2
   50 DIM T$((R2-R1)+1,32)
   60 FOR X=R1 TO R2
   70 FOR Y=C1 TO C2
   80 LET T$(X-R1+1,Y)=SCREEN$ (X,Y)
   90 NEXT Y: NEXT X
  100 FOR A=1 TO (R2-R1)+1
  110 LPRINT T$(A)
  120 NEXT A
```

Para imprimir caracteres en inverso, simplemente escriba INVERSE 1 antes de enviarlos e INVERSE 0 después de haberlo hecho. Esto no afecta a la pantalla para nada, y no trabajará cuando se use COPY. Por ejemplo:

```
10 LET A$ = "Inverse"
20 INVERSE 1: LPRINT A$: INVERSE
0
```

O bien un listado en inverso:

```
INVERSE 1: LLIST: INVERSE 0
```

También se obtiene el mismo efecto haciendo POKE 23697, 12 en lugar de INVERSE 1 y POKE 23697, 0 en lugar de INVERSE 0.

Lo que hace POKE es simplemente poner un trozo de información directamente en una posición de la memoria del ordenador. Usted puede imaginarse que la memoria es como una torre de cajas, cada una de ellas numerada desde 0 hasta 65.000 más o menos. A cada una de estas cajas se le llama posición de memoria o dirección. Mire el dibujo de las direcciones en el apéndice, llamado Mapa de la Memoria. Para ver lo que hay en una posición determinada se usa PEEK.

CLOSE# y OPEN# también trabajan con la impresora, aunque están pensados para el tratamiento de ficheros en diskette. Estos comandos, pueden conectar y desconectar la impresora. Si usted escribe OPEN# 2 "nombre" (la impresora ha sido designada como el dispositivo 2) entonces, todos los PRINTs o LISTs que haga a partir de ahora irán a parar a la impresora en vez de dirigirse a la pantalla. CLOSE# devuelve la situación a su estado natural. Obviamente, es más fácil usar OPEN# y CLOSE# para determinar dónde debe dirigirse un trozo de información, que escribir complicadas sentencias condicionales con PRINT o LPRINT.

Por último, le puede interesar una manera de escribir caracteres en la impresora de manera que tengan el doble de altura. Esto se hace utilizando de nuevo la instrucción SCREEN$. Esta vez se usa para encontrar el código del carácter que se encuentra en una posición determinada. Sabiendo el código, el programa se va al área de la ROM donde el Spectrum almacena los bytes correspondientes a ese carácter. Allí encuentra cuáles son los ocho bytes que corresponden a ese carácter, y los coloca (POKE) en el buffer de la impresora. Sin embargo, se coloca dos veces cada byte y el resultado de esto es que primero se imprime la mitad de arriba de todos los caracteres de la línea, y luego la mitad de abajo.

```
10 REM PONER EL CARACTER DE LA POSICION 90
EN LA IMPRESORA A MODO DE EJEMPLO
20 LET code = CODE SCREEN$ (0, 0)
30 LET X = 15616 + code *8        (tabla de caracteres)
40 FOR A = 23296 TO 23552 STEP 2
                                  (buffer de la impresora)
50 POKE A, PEEK X
```

```
60 POKE A + 32, PEEK X
70 LET X = X + 1
80 NEXT A
```

Evidentemente, habrá que escoger cada posición de la pantalla. La tabla de caracteres en la ROM, va desde la posición 15616 en adelante y los bytes de cada carácter se encuentran desde la posición 15616 + 8* code hasta la posición 15616 + 8* code + 8 (donde code representa el código del carácter en cuestión). Los caracteres se almacenan en el buffer del mismo modo que se imprimen en la pantalla, primero se pone el primer byte de los 32 caracteres, luego el segundo, etc. En la línea 60 del ejemplo se "POKEa" el mismo byte dos veces en el buffer unas 32 posiciones después de la última.

FUNCIONES

La última función que nos ofrece el Spectrum es DEF FN acompañada de FN. Estos dos comandos son muy potentes, ya que permiten definir una ecuación compleja con DEF FN A(xxx) y usarla en el programa simplemente llamándola mediante FN A(xxx). Por ejemplo, usted puede definir una función que sea el cubo de un número:

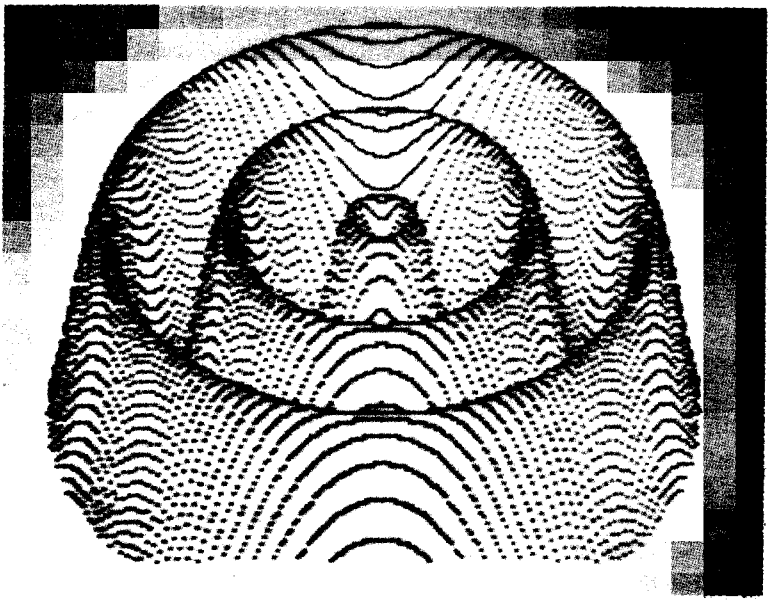
```
10 DEF FN a (z) = z*z*z*
20 PRINT FN a (37.9)
```

Este programa, calcula el cubo de 37,9 en la línea 20. También se pueden definir funciones de cadena. Por ejemplo, puede serle útil simular un comando que poseen otros ordenadores que permite obtener los últimos caracteres de una cadena. Por ejemplo:

```
10 DEF FN A$ (S$,n) = S$(N TO )
```

Veamos ahora una manera más útil de usar esta posibilidad de definir funciones. En la línea 5 hay una función que será representada por el programa. Usted puede probar de poner otras ecuaciones en la línea 5 para ver los resulta-

dos que se obtienen. En este ejemplo, existe la opción de escoger varias resoluciones (aunque sólo recomiendo entre 1 y 10). En el nivel 10 el programa tarda 10 minutos en representar la función, mientras que a nivel uno puede tardar varias horas. Este gráfico tridimensional, nos introduce a la nueva parte del libro dedicada al color, gráficos y sonido:



```
  2 REM TRES HONDONADAS
  5 DEF FN A(Z)=32*SIN (Z/6)
 10 INPUT "RESOLUCION?(1 TO 10)",R
 20 FOR X=-100 TO 100
 30 LET J=0: LET K=1
 40 LET V=R*INT (SQR ((10^4)-X*X)/R)
 50 FOR Y=V TO -V STEP -R
 60 LET Z=INT (80+FN A(SQR (X*X+Y*Y))-.
707*Y)
 70 IF Z<J THEN  GO TO 110
 80 LET J=Z
 90 PLOT X+110,Z
100 LET K=0
110 NEXT Y
120 NEXT X
130 STOP
```

La n en el primer ejemplo, y la z en este último son puramente *variables falsas*. Con esto quiero decir que es totalmente correcto definir una variable z en cualquier otra parte del programa sin que tenga nada que ver con la utilizada en DEF FN.